Elements of Information Theory

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 500

Example

- Consider the strings:
  1. 0101010101010101010101010101010101010101010101010101010101010101
  2. 0110101000001001111001100110011111110011101111001100100100001000
  3. 1101111001110101111101101111101110101101111000101110010100111011
- The first sequence consists of thirty-two 01's.
- The second sequence looks random and passes most tests for randomness, but it is in fact the initial segment of the binary expansion of $\sqrt{2}-1$.
- The third again looks random, except that the proportion of 1's is not near $\frac{1}{2}$. We assume that it is otherwise random.

Example: Comparing Complexities

- One can give a description of the sequence in roughly
  $$\log n + nH\left(\frac{k}{n}\right)$$
  bits 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.
- This is substantially fewer than the n bits in the sequence.
- We conclude that the sequence, random though it is, is simple.
- However, it is not as simple as the other two sequences, which have constant-length descriptions.
- Its complexity is proportional to n.

Example: Random Sequences

- 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.
- This is tantamount to saying that there is no better description for such a sequence than simply saying

  "Print the following: 0101100111010...0".

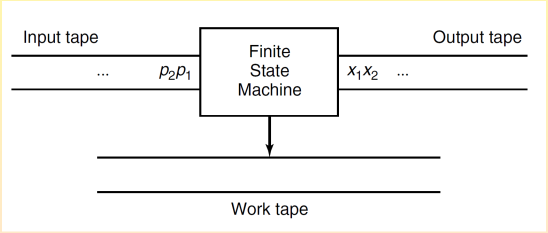
- The reason for this is that there are not enough short programs to go around.
- Thus, the descriptive complexity of a truly random binary sequence is as long as the sequence itself.

Subsection 1

Models of Computation

The Turing Machine Model of Computation

- We model computation using the Turing machine model.



- 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;
  - 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.

Partial Recursive Functions

- 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 $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of **partial recursive functions**.

Subsection 2

Kolmogorov Complexity: Definitions and Examples

Kolmogorov Complexity

- Let x be a binary string of finite length $\ell(x)$.
- Let $\mathcal{U}$ be a universal computer.
- Let $\mathcal{U}(p)$ be the output of $\mathcal{U}$ when the input is 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

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} \ell(p),$$

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 $\mathcal{U}$.

Conditional Kolmogorov Complexity

- If we assume that the computer already knows the length of x, we can define the **conditional Kolmogorov complexity** knowing $\ell(x)$ as

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

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

Universality of Kolmogorov Complexity

Theorem (Universality of Kolmogorov Complexity)

If $\mathcal{U}$ is a universal computer, for any other computer $\mathcal{A}$ there exists a constant $c_{\mathcal{A}}$, such that

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

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

- Assume a program $p_{\mathcal{A}}$ for computer $\mathcal{A}$ prints x, i.e., $\mathcal{A}(p_{\mathcal{A}}) = x$.

  We can precede this program by a simulation program $s_{\mathcal{A}}$ which tells computer $\mathcal{U}$ how to simulate computer $\mathcal{A}$.

Universality of Kolmogorov Complexity (Cont'd)

- Computer $\mathcal{U}$ will:
  - Interpret the instructions in the program for $\mathcal{A}$;
  - Perform the corresponding calculations and print out x.

  The program for $\mathcal{U}$ is $p = s_{\mathcal{A}} p_{\mathcal{A}}$.

  Its length is
  $$\ell(p) = \ell(s_{\mathcal{A}}) + \ell(p_{\mathcal{A}}) = c_{\mathcal{A}} + \ell(p_{\mathcal{A}}),$$
  where $c_{\mathcal{A}}$ is the length of the simulation program.

  Hence, for all x,
  $$\begin{aligned} K_{\mathcal{U}}(x) &= \min_{p:\mathcal{U}(p)=x} \ell(p) \\ &\leq \min_{p:\mathcal{A}(p)=x} (\ell(p) + c_{\mathcal{A}}) \\ &= K_{\mathcal{A}}(x) + c_{\mathcal{A}}. \end{aligned}$$

Comments on the Universality Property

- The constant $c_{\mathcal{A}}$ in the theorem may be very large.
  - $\mathcal{A}$ may be a large computer with many built-in functions;
  - $\mathcal{U}$ may be a simple microprocessor;
  - The simulation program will contain implementations of all 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, $c_{\mathcal{A}}$ will be small in comparison.
- In this sense Kolmogorov complexity may ignore the constants.
- If $\mathcal{A}$ and $\mathcal{U}$ are both universal, we have

$$|K_{\mathcal{U}}(x) - K_{\mathcal{A}}(x)| < c, \quad \text{for all } x.$$

- Hence, we will assume the unspecified computer $\mathcal{U}$ is some fixed universal computer.

Conditional Complexity and Length of the Sequence

Theorem (Conditional Complexity $<$ Length of the Sequence)

$K(x|\ell(x)) \leq \ell(x) + c$.

- A program for printing x is

  `Print the following` ℓ`-bit sequence:` $x_1 x_2 \ldots x_{\ell(x)}$.

  Note that no bits are required to describe ℓ since ℓ is given.

  The program is self-delimiting because $\ell(x)$ is provided and the end of the program is thus clearly defined.

  The length of this program is $\ell(x) + c$.

Unknown Length

- If the computer does not know $\ell(x)$, the method of the preceding theorem 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.
- A simple (inefficient) method uses a sequence 01 as a "comma".
- This will suffice to establish an upper bound.

Upper Bound on Kolmogorov Complexity

Theorem (Upper Bound on Kolmogorov Complexity)

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

- Suppose that $\ell(x) = n$.

  To describe $\ell(x)$:
  - Repeat every bit of the binary expansion of n twice;
  - Then end with a 01 to signal the end of the description of n.

  Example: The number 5 (binary 101) will be described as 11001101.

  This description requires $2\log n + 2$ bits.

  Thus, inclusion of the binary representation of $\ell(x)$ does not add more than $2\log \ell(x) + c$ bits to the length of the program.

  So we have the bound stated in the theorem.

A More Efficient Method

- A more efficient method for describing $\ell(x) = n$ uses recursion.
  - We specify the number $\log n$ of bits in the binary representation of n;
  - Then specify the actual bits of n.

  To specify $\log n$, we can use:
  - The inefficient method (length $2 \log \log n$);
  - The efficient method (length $\log \log n + \cdots$).

  If we use the efficient method at each level, we can describe n in

$$\log n + \log \log n + \log \log \log n + \cdots \text{ bits},$$

  where we continue the sum until the last positive term.

  This sum of iterated logarithms is sometimes written $\log^* n$.

  Thus, the theorem can be improved to

$$K(x) \leq K(x | \ell(x)) + \log^* \ell(x) + c.$$

Lower Bound on Kolmogorov Complexity

Theorem (Lower Bound on Kolmogorov Complexity)

The number of strings x with complexity $K(x) < k$ satisfies

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

- There are not very many short programs.
  If we list all the programs of length $< k$, we have
  $$\underbrace{\Lambda}_{1}, \underbrace{0, 1}_{2}, \underbrace{00, 01, 10, 11}_{4}, \ldots, \underbrace{\ldots, \overbrace{11\ldots1}^{k-1}}_{2^{k-1}}.$$
  The total number of such programs is
  $$1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1 < 2^k.$$
  Each program can produce only one possible output sequence.
  So the number of sequences with complexity $< k$ is less than 2^k.

Fixing Notation

- 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**

$$H_0(p) = -p \log p - (1-p) \log (1-p).$$

- Thus, when we write $H_0\left(\frac{1}{n}\sum_{i=1}^n X_i\right)$, we will mean

$$-\overline{X}_n \log \overline{X}_n - (1-\overline{X}_n) \log (1-\overline{X}_n)$$

  and not the entropy of random variable $\overline{X}_n$.
- When there is no confusion, we shall simply write $H(p)$ for $H_0(p)$.

Setup for Presentation of Examples

- We consider various examples of Kolmogorov complexity.
- The complexity will depend on the computer, but only up to an additive constant.
- We consider a computer that can accept unambiguous commands in English (with numbers given in binary notation).
- Stirling's Approximation Formula for the factorial is

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \le n! \le \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}.$$

- Using Stirling's Formula, we obtain a formula we use later,

$$\sqrt{\frac{n}{8k(n-k)}}2^{nH\left(\frac{k}{n}\right)} \le \binom{n}{k} \le \sqrt{\frac{n}{\pi k(n-k)}}2^{nH\left(\frac{k}{n}\right)}, \quad k \ne 0, n.$$

Example: A Sequence of n Zeros

- If we assume that the computer knows n, a short program to print

$$\underbrace{00\cdots 0}_{n \text{ zeros}}$$

is

```
Print the specified number of zeros.
```

The length of this program is a constant number of bits independent of n.

Hence, the Kolmogorov complexity of this sequence is c

$$K(\underbrace{00\cdots 0}_{n \text{ zeros}}|n) = c, \text{ for all } n.$$

Example: 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,

$$K(\pi_1\pi_2\ldots\pi_n|n) = c.$$

Example: Repeating Sequence of the form 010101. . .01

- A short program suffices.

```
Print the specified number of 01 pairs.
```

- Hence,

$$K(\underbrace{0101\cdots 01}_{n \text{ pairs}}|n) = c.$$

Example: Fractal

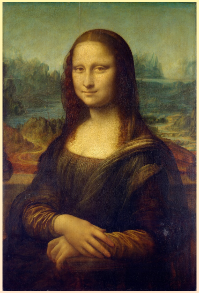
- A fractal is part of the Mandelbrot set and is generated by a simple computer program.
  - For different points c in the complex plane, one calculates the number of iterations of the map
    $$z_{n+1} = z_n^2 + c, \quad (z_0 = 0)$$
    needed for $|z|$ to cross a particular threshold.
  - The point 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 essentially zero.

Example: 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,

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

Example: Integer n

- If the computer knows the number of bits in the binary representation of the integer, 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 previously, we see that the Kolmogorov complexity of an integer is bounded by

$$K(n) \leq \log^* n + c.$$

Example: Sequence of n Bits With k Ones

Theorem

The Kolmogorov complexity of a binary string x is bounded by

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

- Consider a sequence of n bits with k ones.

  The following program will print out the required sequence:

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

  The only variables in the program are:
  - k, with known range $\{0,1,\ldots,n\}$;
  - i, with conditional range $\{1,2,\ldots,\binom{n}{k}\}$.

Example: Sequence of n Bits With k Ones (Cont'd)

- The total length of the program is
$$\ell(p) = c + \log n + \log \binom{n}{k}.$$
But
$$\binom{n}{k} \leq \frac{1}{\sqrt{\pi npq}} 2^{nH_0(p)}, \text{ for } p = \tfrac{k}{n},\ q = 1 - p \text{ and } k \neq 0, n.$$
Therefore,
$$\ell(p) \leq c' + \log n + nH_0\left(\frac{k}{n}\right) - \frac{1}{2}\log n.$$
We have used $\log n$ bits to represent k.
Thus, if $\sum_{i=1}^{n} x_i = k$, then
$$K(x_1, x_2, \ldots, x_n | n) \leq nH_0\left(\frac{k}{n}\right) + \frac{1}{2}\log n + c.$$

Subsection 3

Kolmogorov Complexity and Entropy

Kraft Inequality for Program Lengths

Lemma

For any computer $\mathcal{U}$,

$$\sum_{p:\ \mathcal{U}(p) \text{ halts}} 2^{-\ell(p)} \leq 1.$$

- Suppose the computer halts on some program.

  Then it does not look any further for input.

  This implies that there cannot be any other halting program with this program as a prefix.

  Thus, the halting programs form a prefix-free set.

  So their lengths satisfy the Kraft inequality.

Relationship of Kolmogorov Complexity and Entropy

Theorem (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, for all n,

$$H(X) \leq \frac{1}{n} \sum_{x^n} f(x^n) K(x^n|n) \leq H(X) + \frac{(|\mathcal{X}| - 1) \log n}{n} + \frac{c}{n}.$$

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

Kolmogorov Complexity and Entropy (Cont'd)

- Consider the lower bound.

  The allowed programs satisfy the prefix property.

  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,

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

Kolmogorov Complexity and Entropy (Cont'd)

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

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

  Hence,

$$\begin{aligned} & EK(X_1 X_2 \ldots X_n | n) \\ & \quad \leq nEH_0\left(\tfrac{1}{n}\textstyle\sum_{i=1}^{n} X_i\right) + \tfrac{1}{2}\log n + c \\ & \quad \leq nH_0\left(\tfrac{1}{n}\textstyle\sum_{i=1}^{n} EX_i\right) + \tfrac{1}{2}\log n + c \\ & \qquad \text{(Jensen's Inequality and Concavity of } H_0\text{)} \\ & \quad = nH_0(\theta) + \tfrac{1}{2}\log n + c. \end{aligned}$$

  So we have the upper bound for binary processes.

Kolmogorov Complexity and Entropy (Conclusion)

- We use the same technique for a nonbinary finite alphabet.
  - We first describe the type of the sequence (the empirical frequency of occurrence of each of the alphabet symbols) using $(|\mathcal{X}|-1)\log n$ bits (the frequency of the last symbol can be calculated from the rest).
  - 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) as shown before.

Therefore, the two-stage description of a string x^n has length

$$K(x^n|n) \leq nH(P_{x^n}) + (|\mathcal{X}|-1)\log n + c.$$

Taking the expectation and applying Jensen's inequality,

$$EK(X^n|n) \leq nH(X) + (|\mathcal{X}|-1)\log n + c.$$

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

Removing Conditioning

- By similar arguments, we can show that, for all n,

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

  - The lower bound follows from the fact that $K(x^n)$ is also a prefix-free code for the source.
  - The upper bound can be derived from the fact that

$$K(x^n) \leq K(x^n | n) + 2 \log n + c.$$

- Thus, we have

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

- So the compressibility achieved by the computer approaches the entropy.

Subsection 4

Kolmogorov Complexity of Integers

Kolmogorov Complexity of Integers

Definition

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

$$K(n) = \min_{p:\mathcal{U}(p)=n} \ell(p).$$

Theorem

For universal computers $\mathcal{A}$ and $\mathcal{U}$, $K_{\mathcal{U}}(n) \leq K_{\mathcal{A}}(n) + c_{\mathcal{A}}$.

Theorem

$K(n) \leq \log^* n + c$.

- Any number can be specified by its binary expansion.

Integers with High Complexity

Theorem

There are an infinite number of integers n, such that $K(n) > \log n$.

- We know by a previous lemma that $\sum_n \frac{1}{2^{K(n)}} \leq 1$.

  Moreover,
  $$\sum_n \frac{1}{2^{\log n}} = \sum_n \frac{1}{n} = \infty.$$

  Now suppose that $K(n) < \log n$, for all $n > n_0$.

  This would give
  $$\sum_{n=n_0}^{\infty} \frac{1}{2^{K(n)}} > \sum_{n=n_0}^{\infty} \frac{1}{2^{\log n}} = \infty.$$

  This is a contradiction.

Subsection 5

Algorithmically Random and Incompressible Sequences

Complexity of Sequences

- From the examples we looked at, 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 $2^{2^{2^{2^{2^{2^{2}}}}}}$ 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.
- It follows that, if we draw a sequence at random, we are likely to draw a complex sequence.

Probability of k-Bit Compressibility

- The probability that a sequence can be compressed by more than k bits is no greater than 2^{-k}.

Theorem

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

$$P(K(X_1X_2\ldots X_n|n) < n-k) < \frac{1}{2^k}.$$

$$\begin{aligned}
&P(K(X_1X_2\ldots X_n|n) < n-k)\\
&\quad = \sum_{x_1x_2\ldots x_n : K(x_1x_2\ldots x_n|n) < n-k} p(x_1, x_2, \ldots, x_n)\\
&\quad = \sum_{x_1x_2\ldots x_n : K(x_1x_2\ldots x_n|n) < n-k} \frac{1}{2^n}\\
&\quad = |\{x_1x_2\ldots x_n : K(x_1x_2\ldots x_n|n) < n-k\}|\frac{1}{2^n}\\
&\quad < 2^{n-k}\frac{1}{2^n} = \frac{1}{2^k}.
\end{aligned}$$

Algorithmic Randomness and Incompressibility

- Most sequences have a complexity close to their length.

  Example: The fraction of sequences of length n that have complexity less than $n-5$ is less than $\frac{1}{2^5} = \frac{1}{32}$.

Definition

A sequence $x_1, x_2, \ldots, x_n$ is said to be **algorithmically random** if $K(x_1 x_2 \ldots x_n | n) \geq n$.

- By the counting argument, there exists, for each n, at least one sequence x^n such that $K(x^n|n) \geq n$.

Definition

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

$$\lim_{n \to \infty} \frac{K(x_1 x_2 x_3 \ldots x_n | n)}{n} = 1.$$

Law of Large Numbers for Incompressible Sequences

Theorem (Strong Law of Large Numbers for Incompressible Sequences)

If a string $x_1 x_2 \ldots$ is incompressible, it satisfies the law of large numbers in the sense that

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

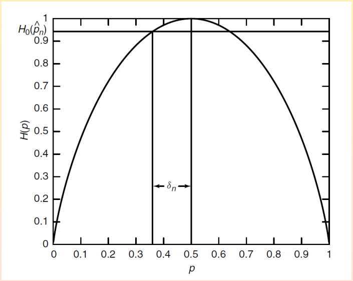
Hence the proportions of 0's and 1's in such a string are almost equal.

- Let $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denote the proportion of 1's in $x_1 x_2 \ldots x_n$.
  There is a program of length $nH_0(\theta_n) + 2\log(n\theta_n) + c$ printing x^n.
  Thus, $\frac{K(x^n|n)}{n} < H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}$.
  By incompressibility, we have, for large enough n,
  $$1 - \epsilon \leq \frac{K(x^n|n)}{n} \leq H_0(\theta_n) + 2\frac{\log n}{n} + \frac{c'}{n}.$$
  Thus, $H_0(\theta_n) > 1 - \frac{2\log n + c'}{n} - \epsilon$.

Law of Large Numbers (Cont'd)

- We have
$$H_0(\theta_n) > 1 - \frac{2\log n + c'}{n} - \epsilon.$$

Inspection of the graph of $H_0(p)$ shows that θ_n is close to $\frac{1}{2}$ for large n.



Specifically, the inequality above implies that
$$\theta_n \in \left(\frac{1}{2} - \delta_n, \frac{1}{2} + \delta_n\right),$$
where δ_n is chosen so that $H_0\left(\frac{1}{2} - \delta_n\right) = 1 - \frac{2\log n + c_n + c'}{n}$.
This implies that $\delta_n \to 0$ as $n \to \infty$. So $\frac{1}{n}\sum x_i \to \frac{1}{2}$ as $n \to \infty$.

Kolmogorov Complexity of Bernoulli(θ) Sequences

Theorem

Let $X_1, X_2, \ldots, X_n$ be drawn i.i.d. $\sim$ Bernoulli(θ). Then

$$\frac{1}{n} K(X_1 X_2 \ldots X_n | n) \to H_0(\theta) \text{ in probability.}$$

- Let $\overline{X}_n = \frac{1}{n} \sum X_i$ be the proportion of 1's in $X_1, X_2, \ldots, X_n$.
  Then
  $$K(X_1 X_2 \ldots X_n | n) \leq n H_0(\overline{X}_n) + 2 \log n + c.$$
  By the weak law of large numbers, $\overline{X}_n \to \theta$ in probability.
  It follows that
  $$\Pr\left\{ \frac{1}{n} K(X_1 X_2 \ldots X_n | n) - H_0(\theta) \geq \epsilon \right\} \to 0.$$

Complexity of Bernoulli(θ) Sequences (Cont'd)

- 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 nontypical 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 typical sequences can have complexity $< n(H_0(\theta)-c)$.

  We calculate the probability that the complexity of the random sequence is less than $n(H_0(\theta)-c)$.

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

Complexity of Bernoulli(θ) Sequences (Cont'd)

We continue

$$\begin{aligned}
&\leq \epsilon + \sum_{x^n \in A_\epsilon^{(n)}, K(x^n|n) < n(H_0(\theta)-c)} p(x^n) \\
&\leq \epsilon + \sum_{x^n \in A_\epsilon^{(n)}, K(x^n|n) < n(H_0(\theta)-c)} 2^{-n(H_0(\theta)-\epsilon)} \\
&\leq \epsilon + 2^{n(H_0(\theta)-c)} 2^{-n(H_0(\theta)-\epsilon)} \\
&= \epsilon + 2^{-n(c-\epsilon)}.
\end{aligned}$$

This 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

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

Subsection 6

Universal Probability

Universal Probability

Definition

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

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} \frac{1}{2^{\ell(p)}} = \Pr(\mathcal{U}(p) = x),$$

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 can be considered as the probability of observing the string in nature based on the implicit belief that simpler strings are more likely than complicated strings (Occam's Razor).

Universality of Universal Probability

Theorem

For every computer $\mathcal{A}$,

$$P_{\mathcal{U}}(x) \geq c'_{\mathcal{A}} P_{\mathcal{A}}(x), \quad \text{for every string } x \in \{0,1\}^*,$$

where the constant $c'_{\mathcal{A}}$ depends only on $\mathcal{U}$ and $\mathcal{A}$.

- Recall that, for every program p' for $\mathcal{A}$ that prints x, there exists a program p for $\mathcal{U}$ of length not more than $\ell(p') + c_{\mathcal{A}}$ produced by prefixing a simulation program for $\mathcal{A}$, which also prints x.

  Hence,

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} \frac{1}{2^{\ell(p)}} \geq \sum_{p':\mathcal{A}(p')=x} \frac{1}{2^{\ell(p')+c_{\mathcal{A}}}} = c'_{\mathcal{A}} P_{\mathcal{A}}(x).$$

Subsection 7

Noncomputability of Kolmogorov Complexity

Noncomputability of Kolmogorov Complexity

- The **halting problem** in computer science states that for any computational model, there is no general algorithm to decide whether a program will halt or not (go on forever).
- One of the consequences of the nonexistence of an algorithm for the halting problem is the noncomputability 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 or not they will halt and what they will print out.
- Hence, there is no effective way to find the shortest program to print a given string.

Subsection 8

Chaitin's Number Ω

Chaitin's Number Ω

- We introduce Chaitin's number Ω, which has some extremely interesting properties.

Definition

Chaitin's number Ω is defined by

$$\Omega = \sum_{p:\mathcal{U}(p)\text{ halts}} \frac{1}{2^{\ell(p)}}.$$

- Note that $\Omega = \Pr(\mathcal{U}(p)\text{ halts})$, the probability that the given universal computer halts when the input to the computer is a binary string drawn according to a Bernoulli $\left(\frac{1}{2}\right)$ process.
- Programs that halt are prefix-free, so their lengths satisfy Kraft.
- Hence, the sum above 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 noncomputable.

   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.

Property 2 of Ω

- The basic idea of the procedure using n bits of Ω is simple.

  We run all programs until the sum of the masses $\frac{1}{2^{\ell(p)}}$ contributed by programs that halt equals or exceeds $\Omega_n = 0.\omega_1\omega_2\ldots\omega_n$, the truncated version of Ω that we are given.

  Now we have $\Omega - \Omega_n < \frac{1}{2^n}$.

  So the sum of all further contributions of the form $\frac{1}{2^{\ell(p)}}$ to Ω from programs that halt must also be less than $\frac{1}{2^n}$.

  Thus, no program of length $\leq n$ that has not yet halted will ever halt.

  This enables us to decide the halting or nonhalting 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.

Property 2 of Ω (Cont'd)

- First, list all possible programs, starting with the null program, $\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots$.
  - 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.

So the computer can produce a list of all programs that halt.

Thus, we will ultimately know whether or not any program of less than n bits will halt.

So the computer will find a proof or a counterexample to the theorem if the theorem can be stated in less than n bits.

Property 3 of Ω

3. Ω is algorithmically random.

Theorem

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

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

- 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;
  - Generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs.

  Find the first string x_0 that is not on this list.

Property 3 of Ω (Cont'd)

- The string x_0 is the shortest string with Kolmogorov complexity
$$K(x_0) > n.$$
The complexity of the program, described in the preceding slide, to print x_0 is
$$K(\Omega_n) + c.$$
This must be at least as long as the shortest program for x_0.
Hence, for all n,
$$K(\Omega_n) + c \geq K(x_0) > n.$$
Thus,
$$K(\omega_1 \omega_2 \ldots \omega_n) > n - c.$$

Subsection 9

Universal Gambling

Gambling on Finite Binary Sequences

- Suppose a gambler is asked to gamble sequentially on a sequence
$$x \in \{0,1\}^*.$$
- He has no idea of the origin of the sequence.
- He is given fair odds (2-for-1) on each bit.
- We want to decide on the best gambling method.

Intuition

- If the gambler believes that the string occurred naturally, it seems intuitive that simpler strings are more likely than complex ones.
- Hence, he might bet according to the universal probability of the string.
- If the gambler knows the string x in advance, he can increase his wealth by a factor of $2^{\ell(x)}$ by betting all his wealth each time on the next symbol of x.

Betting Schemes and Universal Gambling

- A **betting scheme** is a mapping $b : \{0,1\}^* \to [0,\infty)$, such that
$$\sum b(x) = 1.$$
- The **wealth** S associated with betting scheme b is given by
$$S(x) = 2^{\ell(x)} b(x), \quad x \in \{0,1\}^*.$$
- The betting scheme
$$b(x) = \frac{1}{2^{K(x)}}, \quad x \in \{0,1\}^*,$$
is called **universal gambling**.

  Note that
$$\sum_x b(x) = \sum_x \frac{1}{2^{K(x)}} \leq \sum_{p:p \text{ halts}} \frac{1}{2^{\ell(p)}} = \Omega \leq 1.$$

Wealth Associated With A Bet

- Suppose the amount $b(0110)$ is bet on a sequence $x = 0110$.
- Then the **wealth associated with this bet** is the sum of:
  - $2^{\ell(x)} b(x) = 2^4 b(0110)$;
  - The amount won on all bets $b(0110\ldots)$ on sequences that extend x.

Wealth Achieved Using Universal Gambling

Theorem

The logarithm of the wealth a gambler achieves on a sequence using universal gambling plus the complexity of the sequence is no smaller than the length of the sequence:

$$\log S(x) + K(x) \geq \ell(x).$$

- Recall that, in the universal gambling scheme, $b(x) = \frac{1}{2^{K(x)}}$.

  So we have
  $$S(x) = \sum_{x' \sqsupseteq x} 2^{\ell(x')} b(x') \geq 2^{\ell(x)} \frac{1}{2^{K(x)}},$$
  where $x' \sqsupseteq x$ means that x is a prefix of x'.

  Taking logarithms establishes the theorem.

Universality of Universal Gambling

- For infinite sequences x, with finite Kolmogorov complexity,

$$S(x_1x_2\ldots x_\ell) \geq 2^{\ell-K(x)} = 2^{\ell-c}, \text{ for all } \ell.$$

- 2^ℓ is the most that can be won in ℓ gambles at fair odds.
- So this scheme does asymptotically as well as the scheme based on knowing the sequence in advance.

Examples

Example: Let $x = \pi_1\pi_2\ldots\pi_n\ldots$, the digits in the expansion of π.

Recall that $K(\pi_1\pi_2\ldots\pi_n|n) = c$.

For all n, the wealth at time n will be

$$S_n = S(x^n) \geq 2^{n-c}.$$

Example: Suppose the string is actually generated by a Bernoulli process with parameter p.

We know that, if $\sum_{i=1}^n x_i = k$, then

$$K(x_1, x_2, \ldots, x_n|n) \leq nH_0(\overline{X}_n) + 2\log n + c.$$

Thus,

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

Subsection 10

Kolmogorov Complexity and Universal Probability

Kolmogorov Complexity and Universal Probability

- Recall the definitions of Kolmogorov complexity and universal probability.

$$K(x) = \min_{p:\mathcal{U}(p)=x} \ell(p), \qquad P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} \frac{1}{2^{\ell(p)}}.$$

Theorem (Equivalence of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$)

There exists a constant c, independent of x, such that

$$\frac{1}{2^{K(x)}} \leq P_{\mathcal{U}}(x) \leq c\frac{1}{2^{K(x)}},$$

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

Equivalence of Complexity and Universal Probability

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

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

- 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.
- The theorem shows that $K_{\mathcal{U}}(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ are essentially equivalent complexity measures.

Kolmogorov Complexity and Entropy

- Notice the striking similarity between:
  - The relationship of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ in Kolmogorov complexity;
  - The relationship of $H(X)$ and $\log \frac{1}{p(x)}$ in information theory.
- The ideal Shannon code length assignment
$$\ell(x) = \log \frac{1}{p(x)}$$
achieves an average description length $H(X)$.
- In Kolmogorov complexity theory, the ideal description length
$$\log \frac{1}{P_{\mathcal{U}}(x)}$$
is almost equal to $K(X)$.
- We conclude that $\log \frac{1}{p(x)}$ is the natural notion of descriptive complexity of x in algorithmic as well as probabilistic settings.

Comments on the Lower Bound

- The upper bound in $K(x) - c' \leq \log \frac{1}{P_{\mathcal{U}}(x)} \leq K(x)$ is obvious from the definitions.
- 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 $\frac{1}{2^{K(x)}}$.

Comments on the Lower Bound (Cont'd)

- There is another way to look at the upper bound for $K(x)$ 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 for the source;
  - 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.
- It follows that this number is an upper bound on the Kolmogorov complexity $K(x)$.

Proof of the Equivalence Theorem

- For $\frac{1}{2^{K(x)}} \leq P_{\mathcal{U}}(x)$, let p^* be the shortest program for x. Then
$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} \frac{1}{2^{\ell(p)}} \geq \frac{1}{2^{\ell(p^*)}} = \frac{1}{2^{K(x)}}.$$
The inequality $P_{\mathcal{U}}(x) \leq \frac{c}{2^{K(x)}}$, can be rewritten $K(x) \leq \log \frac{1}{P_{\mathcal{U}}(x)} + c$.
We find a short program to describe the strings having high $P_{\mathcal{U}}(x)$.
  - An obvious idea is some kind of Huffman coding based on $P_{\mathcal{U}}(x)$. However, since $P_{\mathcal{U}}(x)$ cannot be calculated effectively, a procedure using Huffman coding is not implementable on a computer.
  - Similarly, the process using the Shannon-Fano code also cannot be implemented.

  But, if we have the Shannon-Fano code tree, we can reconstruct the string by looking for the corresponding node in the tree.

Idea of the Proof: Tree Construction

- 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 assign x successively 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.

Idea of the Proof: Assignment of Programs

- Consider the set of pairs $\{(p, x)\}$ consisting of:
  - Programs;
  - Corresponding outputs.
- We try to assign these pairs to the tree.
- But we immediately come across the problem of the existence of an infinite number of programs for a given string and the lack of enough nodes of low depth.
- We trim the list of program-output pairs to get a more manageable list that can be assigned to the tree.

Tree Construction Procedure: Programs and Outputs

- For the universal computer $\mathcal{U}$, we simulate all programs using the standard technique.
    - We list all binary programs:

$$\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots$$

    - 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.

If any program halts, it will, eventually, be discovered to halt.

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.

Tree Construction Procedure: Approximations

- For each pair (p_k, x_k), we calculate n_k, which is chosen so that it corresponds to the current estimate of $P_{\mathcal{U}}(x)$.
  - $\widehat{P}_{\mathcal{U}}(x_k) = \sum_{(p_i,x_i):x_i=x_k, i\leq k} \frac{1}{2^{\ell(p_i)}}$;
  - $n_k = \left\lceil \log \frac{1}{\widehat{P}_{\mathcal{U}}(x_k)} \right\rceil$.

  Note that $\widehat{P}_{\mathcal{U}}(x_k) \nearrow P_{\mathcal{U}}(x)$ on the subsequence of k, with $x_k = x$.

  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 purposes of the construction, we must ensure that all the n_i's corresponding to a particular x_k are distinct.
  - We remove from the list all triplets that have the same x and n as a previous triplet.
  - This ensures that there is at most one node at each level of the tree that corresponds to a given x.

Tree Construction Procedure: Assigning Nodes

- 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 (to keep the assignment prefix-free).

  Claim: 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.

Tree Construction Procedure: Proof of the Claim

- We start with the infinite sum in the Kraft inequality:

$$\sum_{k=1}^{\infty} \frac{1}{2^{n_k+1}} = \sum_{x\in\{0,1\}^*} \sum_{k:x_k=x} \frac{1}{2^{n_k+1}}.$$

We then write the inner sum as

$$\begin{array}{rcl}
\sum_{k:x_k=x} \frac{1}{2^{n_k+1}} & = & \frac{1}{2}\sum_{k:x_k=x}\frac{1}{2^{n_k}} \\
& \leq & \frac{1}{2}(2^{\lfloor \log P_{\mathcal{U}}(x)\rfloor} + 2^{\lfloor \log P_{\mathcal{U}}(x)\rfloor -1} + \cdots) \\
& & \text{(at most one node at each level for each } x\text{)} \\
& = & \frac{1}{2}2^{\lfloor \log P_{\mathcal{U}}(x)\rfloor}(1+\frac{1}{2}+\frac{1}{4}+\cdots) \\
& = & \frac{1}{2}2^{\lfloor \log P_{\mathcal{U}}(x)\rfloor}2 \leq P_{\mathcal{U}}(x).
\end{array}$$

Hence,

$$\sum_k \frac{1}{2^{n_k+1}} \leq \sum_x \sum_{k:x_k=x} \frac{1}{2^{n_k+1}} \leq \sum_x P_{\mathcal{U}}(x) \leq 1.$$

Using the Tree as a Code

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

  We identify an x by the path to the lowest depth node printing x.

  Call this node $\widetilde{p}$. By construction, $\ell(\widetilde{p}) \leq \log \frac{1}{P_{\mathcal{U}}(x)} + 2$.

  To use this tree in a program to print x:
  - We specify $\widetilde{p}$;
  - Ask the computer to execute the foregoing simulation of all programs;
  - The computer will construct the tree as described above and wait for the particular node; $\widetilde{p}$ to be assigned;
    - The computer executes the same construction as the sender;
    - So eventually $\widetilde{p}$ will be assigned.
  - Then, the computer halts and prints the x assigned to that node.
- The preceding is an effective (finite, mechanical) procedure for the computer to reconstruct x.

The Length of the Code Bounds the Complexity

- 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 $\left\lceil \log \frac{1}{P_{\mathcal{U}}(x)} \right\rceil + 1$.

  The length of the program to reconstruct x is essentially the length of the description of the position of the lowest depth node $\widetilde{p}$ corresponding to x in the tree.

  The length of this program for x is $\ell(\widetilde{p}) + c$, where

  $$\ell(\widetilde{p}) \leq \left\lceil \log \frac{1}{P_{\mathcal{U}}(x)} \right\rceil + 1.$$

  Hence the complexity of x satisfies

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