Kolmogorov Complexity

Last Updated February 16th, 2024

1 Introduction

The following are some uses of the word *complex*.

Complex Situation one for which there may be a large number of ways in which the situation could evolve, or in the number of possible past situations that resulted in the current one. Examples include the current configuration of a game, such as chess or go, or a negotiation between several parties.

Complex Problem one that requires a large amount of resources to solve. Examples include combinatorial optimization problems, such as finding a large clique in a graph, or finding a truth assignment that satisfies a number of logical constraints.

Complex System one that has a large number of interacting parts, each which behaves in accordance with one or more rules. Examples include the human body, the internet, or a computer's architecture.

In this lecture we use the word *complex* to reflect the size of the smallest program that can generate a given object, meaning that an object is complex to the degree that a large program is required to generate the object. In this case we are defining the **Kolmogorov complexity** (abbreviated as K-complexity) of the object x, denoted $\boxed{K(x)}$. In this lecture we limit the term "object" to mean a string of characters from some alphabet Σ. Unless otherwise noted, we assume that $\Sigma = \{0, 1\}$.

Example 1.1. Consider the string $x = \underbrace{01\ldots01}_{n\text{ times}}$. Then the program

```
for(i=1; i <= n; i++}
    print(01);
```

generates x. Thus $K(x) \leq \lfloor \log n \rfloor + c$ for some constant c, since the number n can be represented using $\lfloor \log n \rfloor + 1$ bits and the rest of the program requires a constant number of bits.

On the other hand, consider a string x that is created by tossing a fair coin n times. As n increases, it becomes more likely that the shortest program that can generate x is the one-line program

```
print(x);
```

in which case $K(x) \leq |x| + c$. □

1.1 Conditional Kolmogorov complexity

Definition 1.2. Given $x, y \in \Sigma^*$, the **conditional Kolmogorov complexity** of x given y, written as $K(x|y)$, is defined as the size of the smallest program which, on input y, outputs x.

Definition 1.3. The **length-conditional Kolmogorov complexity** of x, written as $K(x|n)$, is defined as the size of the smallest program which, on input $n = |x|$, outputs x.

Notice that $K(x)$ may be defined as a special case of conditional Kolmogorov complexity, namely when $y = \lambda$, where λ is the empty string.

Example 1.4. Recall the string $x = \underbrace{01\ldots01}_{n \text{ times}}$ from Example 1.1. That example suggests that $K(x||x|) \overset{\leq}{=} c$ since n may now be computed from $|x|$ which is as an input to the program. □

The proof of the following proposition is left as an exercise.

Proposition 1.5. The following statements are true.

1. There is a constant c such that, for all x, $K(x|x) \leq c$.
2. There is a constant c such that, for all $x, y \in \Sigma^*$, $0 \leq K(x|y) \leq K(x) + c$.
3. There is a constant c such that, for all $x \in \Sigma^*$, $\underline{K(x) \leq |x| + c}$.

2 The Robustness of Kolmogorov complexity

One attractive property of Kolmogorov complexity is that, up to an additive constant, it is invariant with respect to the model of computation that is used to write the programs that generate strings, assuming that the Church-Turing thesis hold for this model (e.g. the model has been shown to be equivalent to the URM model in terms of the functions it computes). To see this, consider two models of computation $\mathcal{M}_1$ and $\mathcal{M}_2$ and let $K_i(x)$ denote the Kolmogorov complexity of x when using a program from model $\mathcal{M}_i$ to generate x, $i = 1, 2$. Let P_1 be the length-$K_1(x)$ $\mathcal{M}_1$-program that generates x. Then by the Church-Turing thesis, there is an $\mathcal{M}_2$-program P_2 which works as follows.

P_2

Input $\mathcal{M}_1$-program Q.

Simulate the steps of Q.

If Q halts, then output the string that is output by Q.

By the Church-Turing thesis, the instructions of this program are independent of the input program Q. Let c denote the size of P_2. Now modify P_2 to obtain a new $\mathcal{M}_2$-program P_2' which is the same as P_2 but with the encoding of input Q placed in a lookup table within P_2'. Thus, we have

$$|P_2'| = c + |Q|.$$

$K_2(x) \leq K_1(x) + C$

In particular, if $Q = P_1$, then

$$K_2(x) \leq |P_2'| = c + |P_1| = |P_1| + c = K_1(x) + c,$$

where c is independent of x. Therefore, we have the following theorem.

Theorem 2.1. Consider two models of computation $\mathcal{M}_1$ and $\mathcal{M}_2$ for which the Church-Turing thesis holds for both models. Let $K_i(x)$ denote the Kolmogorov complexity of x when using a program from model $\mathcal{M}_i$ to generate x, $i = 1, 2$. Then there exist constants c and c', both independent of x, for which

$$c \leq |K_1(x) - K_2(x)| \leq c'.$$

Occam's Razor

3 A Universal Probability Distribution

In this section we develop a probability distribution for strings of length n, $n \geq 0$, for which the likelihood of selecting a length-n string x increases exponentially as $K(x)$ decreases. In other words, the distribution favors low-complexity strings. Then we show how sampling from this distribution can create an interesting and somewhat surprising result regarding the worst-case versus average case running times of an arbitrary program.

Proposition 3.1. If a string x having length n is randomly selected via the uniform distribution over $\{0,1\}^n$ (i.e. all strings of length n have the same probability of being selected), then the expected value of $K(x)$ satisfies

$$E[K(x)] \geq (n-2) + \frac{(n+2)}{2^n}.$$

Proof. Given the set $\{0,1\}^n$, the following statements are true:
at most 1 member has K-complexity equal to 0,
at most 2 members have K-complexity equal to 1,
⋮
at most 2^i members have K-complexity equal to i
⋮
at most 2^{n-1} members have K-complexity equal to $n-1$. In other words, there are a total of

$$1 + 2 + \cdots + 2^{n-1} = 2^n - 1$$

$$\underset{\text{complexity}}{i} \cdot \boxed{2^i} \quad \# \text{ of strings having } K(x) = i$$

members who have K-complexity less than n, meaning that at least one member has K-complexity equal to n. Therefore,

$\geq n$

$$E[K(x)] \geq \sum_{i=0}^{n-1} i \cdot 2^{i-n} + \frac{n}{2^n}.$$

$$2^{i-n} = \frac{1}{2^n} \cdot 2^i$$

Now, if we let

$$S = \sum_{i=1}^{n-1} i \cdot 2^{i-n},$$

then

$$2S = \sum_{i=1}^{n-1} i \cdot 2^{i+1-n}.$$

Then subtracting the former from the latter yields,

$$S = -2^{1-n} + (1 \cdot 2^{2-n} - 2 \cdot 2^{2-n}) + \cdots + ((n-2)2^{n-1-n} - (n-1)2^{n-1-n}) + (n-1)2^{n-n} =$$

$$-2^{1-n} + -2^{2-n} + \cdots + -2^{n-1-n} + (n-1) = (n-1) - \frac{1}{2^n}[2^1 + \cdots + 2^{n-1}],$$

and with further simplication we have

$$E[K(x)] \geq (n-2) + \frac{(n+2)}{2^n}.$$

And so the expected K-complexity of a randomly selected length-n string will be at least $n-2$. □

$2^{-2(K(x|n)+c')}$

Now consider the following probability distribution for length-n strings. For each $x \in \{0,1\}^n$,

$$\boxed{\mu_n(x) = c_n \cdot 2^{-2K(x|n)}}$$

$2^{-2c'}$

where c_n is a constant. This distribution is often referred to as the **universal distribution** since it is independent (up to a multiplicative constant) of the model of computation used to define $K(x|n)$ and reflects the intrinsic amount of information contained in x. Indeed, the average amount of information conveyed by x equals

$$H(\mu) = \boxed{\sum_{|x|=n} \frac{1}{c_n 2^{2K(x|n)}} \log\left(c_n 2^{2K(x|n)}\right)} = \sum_{|x|=n} \frac{1}{c_n 2^{2K(x|n)}} (\log c_n + \underline{2K(x|n)}) = \boxed{\log c_n} + \boxed{2E[K(x|n)]}$$

which is asymptotically proportional to the average K-complexity of a string (i.e. minimum number of bits needed to represent x).

We leave the proof of the following proposition as an exercise.

Proposition 3.2. There exists a constant D for which $c_n \geq D > 0$ for all $n \geq 0$, where c_n is the constant used to define the universal probability distribution $\mu_n(x)$ on strings of length n.

Hint: prove that

$$\boxed{\sum_{|x|=n} 2^{-2K(x|n)}}$$

has an upper bound that is independent of n.

We are now ready to prove the following remarkable result.

Theorem 3.3. Let A be an arbitrary algorithm that halts on all input strings and let $n \geq 1$ be given. Then

$$\text{AveSteps}(A,n) = \sum_{|x|=n} \mu_n(x)\text{Steps}(A,n) = \Theta(\text{WorstSteps}(A,n)),$$

meaning that when samples are drawn using the universal distribution, the average-case number of steps taken by A on a length-n input x differs by a multiplicative constant from the worst-case number of steps taken by A on some length-n input, and the constant is independent of A and n.

Proof. Let A be an algorithm that halts on all inputs and consider the following program.

```
Input n.
Initialize $S = 0$.
$x = \lambda$.
For each length-$n$ string $w$,
    Let $T'$ equal the number of steps needed to simulate $A$ on input $w$.
    If $T' > T$, then
        $T = T'$.        $T_{\max,n} = T'$
        $x = w$.         // input that gives worst-case running time
Return $w$ $= x_n$
```

Let c denote the length of the above program and x_n the program output on input n. Then $K(x_n|n) \leq c$ and thus

$$\mu_n(x_n) = c_n \cdot 2^{-2K(x_n|n)} \geq D \cdot 2^{-2c},$$

where D is the constant guaranteed from Proposition 3.2. Now, letting $\alpha = D \cdot 2^{-2c}$ and $T_{\max,n}$ denote the number of steps required by A on input x_n, we see that the average-case number of steps required by A on an input of size n is at most $T_{\max,n}$ (e.g. A requires the same number of steps for every input), but at least $(1-\alpha) + \alpha T_{\max,n}$ (e.g. aside from the worst-case, A halts in a single step for every other input). Thus we have

$$T_{\max,n} \leq \text{AveSteps}(A,n) \leq (1-\alpha) + \alpha T_{\max,n}.$$

In other words,

$$\text{AveSteps}(A,n) = \Theta(T_{\max,n}) = \Theta(\text{WorstSteps}(A,n)),$$

and the proof is complete. $\square$

$$(1-\alpha) + \alpha T_{\max,n} \leq \text{AveSteps}(A,n) \leq T_{\max,n}$$

4 Lower-bound Proofs using Kolmogorov Complexity

An important area of research interest is in the use of Kolmogorov complexity for proving various kinds of lower bounds. In this section we show how to use k-complexity to prove a lower bound on the number of prime numbers that are less than or equal to an integer n. In number theory, the exact value of this number is denoted by $\pi(n)$.

4.1 Prime Number Theorem

Theorem 4.1. (Prime Number Theorem) We have

$\pi(11) = 5$

$\pi(36) = 11$

$$\lim_{n\to\infty} \frac{\pi(n)}{\frac{x}{\log x}} = 1. = \operatorname{limit}_{n\to\infty} \frac{\pi(n)}{\frac{n}{\log n}}$$

Although proving the above theorem requires a significant amount of analysis and number theory, we may use a relatively simple Kolmogorov-complexity argument to prove the following weaker (but useful) result.

Theorem 4.2. For infintely many n,

$$\boxed{\pi(n) \geq \frac{n}{\log^2 n}.}$$

To prove the theorem, we'll need an efficient method for encoding a pair of words $\langle w_1, w_2\rangle$. To do so we use the concept of a **self-terminating code**, i.e. a method for encoding a word w that enables one to determine where w terminates within the context of a larger string. Given binary $w = w_1 w_2 \cdots w_n$, one such method is to encode w as

$$\overline{w} = \underline{w_1 0 w_2 0 \cdots w_n 1},$$

where the first 1 in an even position indicates the termination of w's encoding, and the odd bits allow one to recover w. One issue with this method is that the encoding length is twice the length of w and with Kolmmogorov-complexity applications, having an efficient encoding is often crucial to its success. For this reason we can instead encode w as

$$\overline{\mathrm{bin}(|w|)}w,$$

which only requires $2\log(|w|) + |w|$ bits. This encoding can be further improved by recursively applying the method as follows.

Recursive Self-Terminating Encoding Algorithm

```
If $|w| = 1$, then return $\underline{01w}$.
If $|w| = 2$, then return $\underline{11w}$.
$S = w$.
count = 1. //counts the number of concatenated words in the encoding
$L = |S|$.
While $L > 2$
    $S = L \cdot S$. //Prepend $S$ with the binary representation of $L$
    count ++.
    $L = |L|$.
Return $\overline{|\text{count}|} \cdot S$.
```

$\log^* n =$ # of log applications to n to get ≤ 1

$\log^* 2^{2^{16}} = 5$

$L = (20)_2 = 10100$

$|L| = 5$

$L = 5 \quad |L| = 101$

Example 4.3. Use the recursive self-terminating encoding described above to encode the binary representations of 3, 39, 175, and 286.

$\text{enc}(3) = \underline{11}\ \underline{11}$

$$\text{enc}(39) = \underbrace{1011}\ \underbrace{\overset{3}{11}}_{1}\ \underbrace{\overset{6}{110}}_{2}\ \underbrace{\overset{(39)_2}{100111}}_{3}$$

self-terminating encoding of count = 3

$1\,0\,1\,1$

$2^1 + 2^0 = 3$

$(286)_2$

$$10000111\ \underbrace{11}_{1}\ \underbrace{100}_{2}\ \underbrace{1001}_{3}\ \underbrace{100011110}_{4}$$

self-term. encoding of 100

Proof of Theorem 4.2. Letting p_m, $m \geq 1$, denote the m th prime number. It suffices to prove that, for infinitely many m,

$$\boxed{p_m \leq m \log^2 m.}$$

$m \log m \log(\log m)$

For suppose this is true. Then this would imply that

$$\underline{\pi(p_m)} = \underline{m} \geq \frac{p_m}{\log^2 m} > \frac{p_m}{\log^2 p_m}$$

$\boxed{\log^* m}$

since $p_m > m$ for all $m \geq 1$.

Now let $\underline{\text{bin}(n)}$ denote the binary representation of a natural number n. Note that

$$|\text{bin}(n)| = \lfloor \log n \rfloor + 1,$$

$\log K = \log n - \log p_m$

where we assume $\log 0$ equals 0. Choose n to be sufficiently large to where

$K = \frac{n}{p_m}$

$$\boxed{K(\text{bin}(n)) \geq \log n,}$$

and let p_m be the largest prime that divides n. Then we may reconstruct n from $\boxed{m \text{ and } k = n/p_m}$ by first effectively listing all prime numbers up to and including p_m and multiplying p_m with k. Therefore, there is a program whose size is a constant c plus the length of some encoding of the pair $\boxed{(m, k)}$ and that can generate n and hence $\text{bin}(n)$, where c is independent of n. Moreover if we use the recursive self-terminating code described above we see that

$$\log n \leq K(\text{bin}(n)) \leq \overset{\to \text{encode } k}{\boxed{\log n - \log p_m}} + \underset{\to \text{encode } m}{\boxed{\boxed{\log m} + \log(\log m) + \alpha \log(\log\log(m))}}$$

← rec. self-term. encoding

where α is indenpendent of n. Solving for p_m we have

$$\boxed{p_m \leq m \log m \log(\log^\alpha(m)).}$$

Of course, we could have extended our product out to any number of iterated log functions. This suggests that we can improve the statement of the theorem and replace one of the logs in the denominator with a product of iterated logarithms. We leave it as an exercise to state a stronger version of Theorem 4.2 based on this proof. $\square$

$\cancel{\log n} \leq \cancel{\log n} - \log P_m + \log m + \log(\log m)$

$\log_2 P_m \leq \log_2 (m \log m \log(\log(m)) \log(\log(\log(m))) + \alpha \log(\log(\log(m)))$