2 Plain Kolmogorov Complexity

In this section, we introduce plain Kolmogorov Complexity, prove the invariance theorem - that is, the complexity of a string does not depend crucially on the particular model of computation we choose. We then prove that the notion is uncomputable, and prove some theorems about upper and lower bounds. We give an estimate of the number of strings with high complexity, and conclude the section with a survey of the deficiencies of plain Kolmogorov Complexity.

We recall the fact that there is a computable enumeration $\phi_0, \phi_1, \ldots$ of the partial computable functions. We can now show that there is a universal partial computable function from strings to strings. For, there is a universal Turing machine U which takes inputs in the format $\langle s_n, x\rangle$, and computes $\phi_n(x)$. U's behavior in other cases is undefined. The partial computable function ψ computed by the universal Turing machine is a universal partial computable function.

We can now define the complexity of a string with respect to a given partial computable function ϕ.

Definition 2.0.1. Let x, y, p be strings. Any partial computable function ϕ, together with p and y, such that $\phi(\langle y, p\rangle) = x$ is a description of x. The *complexity* C_ϕ of x conditioned on y is defined by

$$C_\phi(x|y) = \min\{|p| : \phi(\langle y, p\rangle) = x\},$$

and $C_\phi(x|y) = \infty$ otherwise. We call p a program to compute x given y.

Example 2.0.2. Fix a string w and consider the string ww. Let L be the maximum length of any instruction in a programming language that we use.

We describe two programs with indices s_M and s_N which outputs strings. Let s_M be a program, that, given a string x, prints xx. This is a fairly simple program, and can be computed by the following code.

---

1. Input x.
2. Print xx.

The length of the program is essentially the length of the two instructions.

On the other hand, the second program s_N has w hardcoded into it, and prints ww. The code is as follows.

---

1. String $= w$.
2. Print ww.

The length of s_N is the length of the two instructions together with the length of w, since we hardcode w into the text of the program.

Let ϕ be a Turing machine capable of simulating s_M and s_N. (We do not necessarily say that ϕ is a universal partial computable function, i. e., it can compute any partial computable function whatsoever.)

That is, $\phi(\langle s_M, w\rangle) = ww$. Then $C_\phi(ww|w) \leq |s_M| \approx 2L$.

We can derive only the inequality because, there might be shorter programs which could produce ww given w.

On the other hand, $\phi(\langle s_N, \lambda\rangle) = ww$. That is, s_N is a program, which given no information, can output the string ww. Since there is such a program s_N, we get the estimate that

$$C_\phi(ww|\lambda) \leq |s_N| \approx 2L + |w|.$$

Thus the conditional complexity of ww depends on the string that we condition on. (End of example)

Example 2.0.3. Since there is a program which merely outputs its input, we have that $C_\phi(x|x) \leq 2L$.

We have defined complexity to be a function which depends on the partial computable function ϕ we choose. In what follows, we prove that a universal partial computable function causes at most an additive constant more than C_ϕ, where the constant depends on ϕ but not on x or y.

In order to ease the notational burden, we extend the pairing function to encode an arbitrary number of arguments. (Strictly speaking, we have an infinite number of tuple functions: for every n, there is a function $\langle\rangle^n : \Sigma^{*(n)} \to \Sigma^*$ which maps tuples having n strings, to a single string.) encode Thus, $\langle a, \langle b, c\rangle\rangle$ can be written as $\langle a, b, c\rangle$, with value $\mathrm{bd}(a)01\mathrm{bd}(b)01\mathrm{bd}(c)01$.

Theorem 2.0.4. *There is a universal partial computable function $\psi : \Sigma^* \to \Sigma^*$ such that for any partial computable function $\phi : \Sigma^* \to \Sigma^*$, there is a constant c_ϕ depending only on ϕ, such that for any strings x and y,*

$$C_\psi(x|y) \leq C_\phi(x|y) + c_\phi.$$

Proof. Let ψ be the partial computable function computed by a universal Turing machine U such that U on input $\langle s_n, y, p\rangle$ computes the n^{th} partial computable function, ϕ_n, on input $\langle y, p\rangle$.

Then,

$$\psi(\langle s_n, y, p\rangle) = \phi_n(\langle y, p\rangle).$$

Thus, the only additional input that ψ needs to compute ϕ_n on any input, is the number n. This, in our encoding requires $2|s_n| + 2$ bits. Therefore,

$$C_\psi(x|y) \leq C_{\phi_n}(x|y) + 2|s_n| + 2.$$

□

So, we can say that the complexity of a string as measured by a fixed universal Turing machine is at most a constant more than the complexity as measured by any other function, where the constant depends only on the other function, and not on the string. This helps us to

Definition 2.0.5.

2.1 Upper bound

We now prove an upper bound for the plain Kolmogorov complexity of any string. We know that for any finite string x, there is a program with x hardcoded inside it, with a single instruction of the form "print x". Let this program be s_M in the computable enumeration of Turing machines. Then $|s_M| \approx L + |x|$. Thus, we have the following theorem.

Theorem 2.1.1. *There is a constant c such that for any string x, $C(x)$ is at most $|x| + c$.*

We will later prove that this inequality is tight - that is, there are some strings which attain this maximum complexity. The shortest program which describes such a string x is "print x".

2.2 Uncomputability of plain Kolmogorov complexity

Berry's paradox considers the following description - "The smallest number which cannot be described with twelve words." The paradox arises as follows. We know that there are finitely many combinations of twelve words. Hence there are only finitely many numbers that are describable in twelve words. Since there are infinitely many numbers, there is such a smallest n which is not describable in twelve words. On the other hand, the above phrase used only ten words to describe n.

The resolution to the paradox comes from the fact that we were not careful with the allowable descriptions. If we fix the set of allowable descriptions, and then consider the numbers under that description scheme which is not describable in twelve words under that scheme, it does define a number. This is what we have achieved by fixing the allowed descriptions of strings - a partial computable function p, which, on some string y computes a string x, is considered a valid description of x. Thus the theory of computability helps us to avoid Berry's paradox and allows us a well-defined notion.

Once a concept is well-defined, computer scientists are interested to know if the concept is computable. However, Berry's paradox causes C to be uncomputable. Thus, even though we have evaded the paradox for one step by ending up with a well-defined notion, we cannot have a computable notion because of the paradox.

Theorem 2.2.1. *The unconditional Kolmogorov complexity of a string is uncomputable.*

The proof is inspired by Berry's paradox. We consider the phrase "the smallest string x with $C(x) > n$." Then we prove that if it is computable, then it is possible to compute x as follows: There is a program with n hardcoded in it. This program enumerates strings y in the standard enumeration of strings, querying $C(y)$ until it sees the first string with $C(y) > n$. This is the x that we want. However, this new program turns out to be substantially shorter than n in infinitely many cases, contradicting our claim that no program shorter than n bits produces x.

Proof. Let C be a computable function. Let n be an arbitrary number. Consider the first string x in the standard enumeration of strings, with $C(x) > n$. There is such a string, since C is *defined* for every string, and C eventually increases to ∞.

Consider the following program which computes x.

1. Set integer i to 0.
2. **while** $C(s_i) \leq n$ **do** Increment i **end while**
3. Output s_i.

First, we observe that, if C is total computable, then this program outputs the first string in the standard enumeration of strings, with $C(x) > n$.

Now we consider the length of the above program. We assume that any primitive instruction in our language has length L. If C is a computable function, there is a program s_M which computes it. So the length of the above program is approximately $4L + |0| + |s_M| + |n|$. We know that using the standard enumeration of strings, n can be represented by the string s_n, and the length of s_n is approximately $\log_2 n$. So the length of the above program is approximately

$$4L + |s_0| + |s_n| \approx 4L + 1 + \log n.$$

But we have the following inequality for n.

$$n < C(x) \leq 4L + 1 + \log n.$$

Since n is arbitrary, this implies that $n < 4L + 1 \log n$ for all n, which is a contradiction. □

2.3 Upper & lower bounds, and their computability

If a function is uncomputable, we can of course ask whether it is approximable - whether we can, given an error bound M, compute the complexity $C(x) \pm M$. This in itself is a very strong demand. We could weaken this further by asking, is it possible to upper bound or lower bound C? That is, can we prove that there are computable functions C^+ and C_- such that for any string x, $C_-(x) < C(x) < C^+(x)$? We know that theorem 2.2.1 provides a computable upper bound. The lower bound, however, turns out to be uncomputable, in a very strong sense.

Let us define the function $m : \Sigma^* \to \mathbb{N}$ by

$$m(x) = \min_{y \geq x} C(y).$$

Then, we have the following theorem.

Theorem 2.3.1. *For any total computable function $F : \Sigma^* \to \mathbb{N}$ monotonically increasing to ∞ from some x_0 onwards, we have $m(x) < F(x)$ for all large enough x.*

Proof. Suppose the assertion is false, and F is such a total computable function, with $F(x) \leq m(x)$ for an infinite set of points x. [1] Let $G : \Sigma^* \to \mathbb{N}$ be defined by

$$G(x) = \max\{F(x) - 1, 0\}.$$

---

[1] The negation of the statement "$P(x)$ holds for all large enough x", i.e., $\exists x_0 \forall x > x_0 P(x)$, is $\forall x_0 \exists x > x_0 \neg P(x)$, that is, "$P(x)$ is false for infinitely many x".

Since F is total computable and is monotone increasing to ∞, so is G. Also, $G(x) \leq m(x)$ for infinitely many x.

The only possibility of $G(x)$ being equal to $F(x)$ is when both are zero. However, we know that F monotonically increases to ∞, so $G(x)$ can be equal to $F(x)$ only for finitely many x. Thus, for infinitely many x, we have

$$G(x) < F(x) \leq m(x) = \min_{y \geq x} C(y).$$

We write

$$M(s_n) = \max_{C(x) \leq n} x.$$

That is, $M(s_n)$ is the last string in the standard enumeration whose complexity is at most n. It is easy to verify that the next string after the $M(s_n)^{\text{th}}$ string in the standard enumeration is

$$\min_{m(x) > n} x.$$

It follows that for infinitely many x,

$$\max_{G(x) \leq n} x \geq \min_{m(x) > n} x > M(s_n),$$

and the function $\Gamma(s_n) = \max_{G(x) \leq n} x$ is total recursive. Thus $\Gamma(s_n) > M(s_n)$. That is, $C(\Gamma(s_n)) > n$. But, by Theorem 2.0.4, there is a constant c such that

$$C(\Gamma(s_n)) \leq C_\Gamma(\Gamma(s_n)) + c < \log_2 n + c.$$

Thus, there is a constant c such that for all n, $n < \log_2 n + c$, which is impossible. □

In fact, even if the above function F is allowed to be partial, thus giving it the freedom not to be defined on some inputs, it is still not possible to lower bound C.

Theorem 2.3.2 (Kolmogorov)**.** *For any partial computable function $F : \Sigma^* \to \mathbb{N}$ monotonically increasing to ∞ from some x_0 onwards, we have $m(x) < F(x)$ whenever $F(x)$ is defined, for all large enough x.*

Proof. HW 4. □

Though the Kolmogorov complexity can vary between $|x| + c$ and $m(x)$, it does so fairly smoothly.

Let us fix the encoding: $\langle x, y \rangle = \text{bd}(x)01y$.

Lemma 2.3.3. *There is a constant c such that for all x and h*

$$|C(x + h) - C(x)| \leq 2|h| + c.$$

Proof. Let p_x be the shortest program for x. Therefore, $\psi(p_x) = x$. The word $x + h$ can be obtained from the tuple $\langle h, p_x\rangle$ by applying it to the function $F(\langle a, b\rangle) = a + \psi(b)$. F is total computable, and let us assume that it can be computed by the Turing Machine s_N. Therefore, there is a constant c such that

$$C_F(x + h) \leq 2|h| + |p_x| + c \approx 2|h| + C(x) + c.$$

Similarly, let p_{x+h} be the shortest program for $x + h$. There is a function $G : (\langle a, b\rangle) = \max\{\psi(b) - a, 0\}$ which obtains x from the tuple $\langle h, p_{x+h}$. We obtain

$$C(x) - C(x + h) \leq 2|h| + c.$$

□

2.4 Incompressible Strings

When we prove an upper bound, we would like to know whether the bound proved is tight - that is, are there instances of the problem where the inequality is actually an equality? We prove that the upper bound on plain Kolmogorov Complexity is indeed tight - in fact, most strings have very high complexity.

Lemma 2.4.1. *The number of strings of length n with complexity at most $n - c$ is at most 2^{n-c}.*

Proof. The number of programs of length at most $n - c$ is at most 2^{n-c}. So the number of strings produced by such programs is at most 2^{n-c}. □

A similar counting argument gives us:

Lemma 2.4.2. *Let c be a number. For each fixed y, every finite set A of cardinality m has at least $m(1-2^{-c})+1$ elements with $C(x|y) > \log m - c$.*

This is what we call "abundance of randomness".

The above lemmas don't give us the result that there are strings with complexity greater than their length - it seems entirely possible that all strings have complexity at most equal to their length. However, this is not true. The following argument is not merely a counting argument, it uses some properties of programming languages.

Lemma 2.4.3. *For every large enough n, there is a string of length n with complexity greater than n.*

Proof. Let n be large enough that both the following programs are shorter than n.

1. **Print** x	1.**for** i=1**to** 1**do** ;**done** 2. **Print** x.

Both the programs print x. Hence, there are at most 2^{n-1} strings produced by programs of length n. Thus, there is a string of length n with its shortest program longer than n bits. □

2.5 Problems with plain Kolmogorov Complexity

Even though this concept is useful, there are some nice properties which we would like a complexity notion to have, but are absent from Kolmogorov Complexity.

1. C is not subadditive: Suppose we have are given Turing machine descriptions p_x and p_y of the strings x and y, respectively. What can we say about the length of description of their concatenation xy? Intuition would tell us that the description of the concatenation would not be too much longer than the sum of the lengths of the descriptions of x and y. However, there is no constant c such that

$$C(x, y) \leq C(x) + C(y) + c.$$

To see this, we first consider an example encoding of the pair (p_x, p_y), with the constraint that it must be possible to parse out p_x and p_y uniquely from the encoding. The encoding $bd(p_x)01p_y$ obeys this property, and it is possible for a universal Turing machine to first execute p_x, and copy its output to the output tape, and then output the result of p_y. However, the length of this program is now $2|p_x| + |p_y|$.

However, we have to show that in general, there is no encoding scheme with the property that it causes at most an additive constant more than the sums of the complexities of x and y. This is HW 2c.

2. C is non-monotone over prefixes: It would be nice if $C(x) \leq C(xy)$. However, this is not always true!

To see this, let the complexity of "0" be a constant, say χ. Consider two strings s_m and s_n, where $s_m < s_n$, and s_m is an incompressible string with complexity at least $\log m > (2\chi)^2$ [2] and s_n be the first string beyond s_m which has Kolmogorov complexity at most $(\log m)/2$. Such strings do exist. [3]

We consider the string formed by repeating 0 for s_m times, and the string formed by repeating 0 for s_n times. Then we can give the loose estimates $C(0^{s_m}) \geq \log m$ and $C(0^{s_n}) \leq (\log m)/2 + 2M + c$ for some constant c.

To see the first inequality we show that, given a program to compute 0^{s_n}, it is easy to compute s_m. Let P be a shortest program that prints 0^{s_n}, and $\psi(P) = 0^{s_n}$ - that is, ψ is a computable function capable of "executing" P. Then the following program prints s_m:

---

1. **Set** s to $\psi(P)$.
2. **Print** length(s).

The second inequality is because, there is a following program.

---

1. **for** i = 1 **to** $\psi(p_m)$ **do**
2.   **print** 0.
3.   Increment i by 1.
4. **done**

---

[2] This ensures $(\log m)/2 > 2\chi$.

[3] If it did not, $x \mapsto (\log x)/2$ would be a total computable lower bound for C.

and the length of this program is at most $(\log m)/2 + 4L + |\psi| + 3$.

3. Comparison with Shannon Entropy The notion of Kolmogorov Complexity and that of Shannon entropy have both been defined to capture the notion of information content in a string, and they are the same up to a logarithmic factor in the length of any string. It would be more pleasant if complexity analogues of the classical information theoretic content is satisfied up to additive constants.