Excerpts from Chapter 2, Complexity Measures,
By Stephen R. Tate

From

Elsevier, 2003

April 21, 2008
Complexity Measures

STEPHEN R. TATE

OVERVIEW

In this chapter, an alternative measure of information, or string complexity, is introduced. The “Kolmogorov complexity” of a string is basically the length of the shortest program that can compute the string, which turns out to be a very fundamental and important way of defining information. We first discuss some problems with traditional information theory, as introduced in the previous chapter, and then give the basic definitions and properties of Kolmogorov complexity and discuss why this theory solves some of the problems inherent in traditional information theory. The chapter next explores the fundamental properties of this complexity measure, including the computational complexity of determining Kolmogorov complexity, showing that unfortunately the Kolmogorov complexity is uncomputable. Finally, it is shown that Kolmogorov complexity and traditional information theory agree in a very strong way in situations where both may be applied.

2.1 INTRODUCTION

A great deal of the progress in data compression owes its existence to the foundation and insights provided by Claude Shannon in his development of information theory, the basics of which were described in the previous chapter. While the very real and practical impact of Shannon’s ideas is obvious, there are some fundamental questions and problems that this theory does not adequately address. The material presented in this chapter shows that, in a very strong sense, there is an ultimate answer to questions about compressibility of data. To distinguish between the two fundamentally different approaches, in this chapter classical information theory will be referred to as “Shannon Information Theory,” and the approaches presented here will be referred
to as “Algorithmic Information Theory” or, by the more commonly used term, “Kolmogorov complexity.”

It should be pointed out here that the motivation for Algorithmic Information Theory is a solid understanding of what information is, and this theory can be used for reasoning about information and incompressibility. However, the practical uses (meaning uses that lead to implementation by programs) of this theory for data compression are somewhat limited. In the practical arena, Shannon Information Theory wins by virtue of its ability to be implemented, despite the less-than-perfect theory of information.

We continue this chapter with an overview of the shortcomings of Shannon Information Theory, followed by an overview of Algorithmic Information Theory and how it addresses these shortcomings, then continue with a more formal treatment of the subject, and finally end with some historical notes on the development of this theory.

Some topics in this chapter assume familiarity with basic computer science concepts and terminology, such as asymptotic notation and computability issues. Of course, this chapter just barely scratches the surface of Algorithmic Information Theory, with a concentration on those issues that relate to data compression. There are an astounding number of applications which are not obvious at all initially. For example, Algorithmic Information Theory can be used to provide an estimate of how many prime numbers are in a particular range, even though at first this question seems to have little to do with compression or even information theory in a standard sense. A book by Li and Vitányi [10] contains a thorough treatment of Algorithmic Information Theory for those who wish to pursue this area more fully. In this chapter, we use notation that mostly agrees with that in the Li and Vitányi book, although this chapter was written with a conscious effort to use terminology that is less formal and more familiar for non-experts when possible (e.g., we use “computable” instead of “recursive”).

2.1.1 An Aside on Computability

While non-computability is a well-known topic for people who have studied computer science, some results come as a surprise to those who have not studied the area. It is not surprising at all that there are functions which are so complex that it takes a long time to compute them on current computer hardware. For example, factoring the product of two large prime numbers (without knowing the primes, of course!) seems to be extremely difficult, and in fact much of the security of modern on-line commerce depends on this function being very difficult to compute. However, this is very different from saying that the function is non-computable—in fact, factoring is computable because there is a very simple algorithm to do so: Simply try dividing all possible divisors into the number in question. The fact that this algorithm, run on numbers used for cryptography today, would take longer than the lifetime of the universe to complete is very important for the efficiency of the procedure, but completely irrelevant for whether the function is computable, where only the existence of the algorithm is important.

If computers were suddenly to become $10^{30}$ times faster, then all of a sudden we could factor the numbers in question, and a lot of “security” would become not so secure. This is in stark contrast to the notion of non-computability, which means that no algorithm exists at all. Computers could become $10^{30}$ times faster, or even $10^{1000}$ times faster, and it would have no effect—non-computable is non-computable, regardless of time or speed or memory available. This fact is surprising for many people who seem to have an initial intuition that everything is computable, given enough time.

The fact that there are some functions that simply cannot be expressed by an algorithm is unusual, but has parallels in other areas of mathematics. For example, Gödel’s incompleteness theorem says that in any logical system (defined by a basic set of axioms), there are statements
that are true, but cannot be proved true—this is in fact very closely related to non-computability results, where the algorithmic language corresponds to the logical system, and the true statements that can be defined but not proved correspond to functions which can be defined but not computed.

In reading this chapter, keep in mind that when we refer to non-computable functions, we are not talking about simply a lack of power or speed of current machines, nor are we talking about a function which we simply have not been clever enough to invent an algorithm for. We are talking about a fundamental incomputability that exists on all conceivable computers.

### 2.2 CONCERNS WITH SHANNON INFORMATION THEORY

Shannon Information Theory provides some remarkable insight into the information content of data sources, but fails to satisfactorily answer some natural questions. We outline a few issues with Shannon Information Theory below, all of which will be addressed in a satisfactory way by Algorithmic Information Theory.

#### 2.2.1 Strings versus Sources

The most fundamental question in information theory is, given a string $x$, what is the information content of that string? Shannon Information Theory does not have the ability to address this question, as it must look at this in terms of a probabilistic data source. In particular, if we are considering only one possible string, you might say the probability of this string is 1, which leads to the result that the entropy is zero and the string contains no information. This contradicts common sense and argues for a theory that can describe the information content of individual strings.

#### 2.2.2 Complex Non-random Sequences

The digits of $\pi$ form an infinite sequence with many properties in common with truly random sequences, such as equal frequency of digits and strings of digits. By trying to apply Shannon Information Theory and forcing a probabilistic source on top of the stream of digits, the result is a source that offers no compression due to the apparent randomness of the digits. This again contradicts what we know to be true, since the digits of $\pi$ are extremely compressible. In fact $n$ digits can be represented in $O(\log n)$ space, since all we really need is a program that knows how to compute the digits of $\pi$ and an indication of the length of the string that we need to generate.

This argument extends to a different problem which turns out to be related to the same issue. Consider the output of a good pseudo-random number generator (by “good” we mean one where the output can pass all known statistical tests for randomness—cryptographic pseudo-random number generators certainly meet this requirement). Since the output appears random, Shannon Information Theory offers no help here, but again the data stream can be very highly compressed. The pseudo-random stream can be completely described by simply giving the length of the output and the seed required to start the pseudo-random number generator.

#### 2.2.3 Structured Random Strings

Consider a random binary data source in which each bit is uniformly and independently chosen (so each bit has probability $\frac{1}{2}$ of being 0 and probability $\frac{1}{2}$ of being 1). Shannon Information Theory tells us that data from such a source cannot be compressed on average and, in fact,
compression approaches that are based on Shannon Information Theory (such as Huffman coding or arithmetic coding) will code all data from this source using the same size code and will achieve no compression on any string. But is it accurate to say that no data from this source are compressible?

Consider two strings, "00000000000000000000000000000000" and "01010011011100111000." Most people would say the second string is "more random" and "less compressible" even though there is absolutely no justification for these statements in light of the data source, as both strings have exactly the same probability of occurrence (they both occur with probability $2^{-20}$). By Shannon's definitions, both strings have the same information content, although most people would disagree with this statement.

Each of the problems outlined above has a distinct flavor, but there is one common underlying theme: An information measure should reflect the difficulty of constructing or computing the data, not some probabilistic model that we try to fit to the data. With that in mind, we consider an information measure that addresses the descriptional complexity of an algorithm for computing the data and refer to this as Algorithmic Information Theory. This approach not only satisfactorily addresses the concerns we outlined above, but also matches up quite nicely with Shannon Information Theory in situations where that theory is applicable.

To this point, we have used the term Algorithmic Information Theory to emphasize that this is, indeed, a theory of information in the same sense as Shannon's theory. However, a more common term for this theory is Kolmogorov complexity, named after the Russian mathematician who was one of the originators of the theory. As will be described in the last section of this chapter, on the history of Kolmogorov complexity, this term is not without controversy since similar ideas were independently discovered at roughly the same time by Solomonoff and by Chaitin. However, to match common usage, in the remainder of this chapter we will use the term Kolmogorov complexity, as this has emerged as the most commonly used name.

### 2.3 KOLMOGOROV COMPLEXITY

The basic idea of Kolmogorov complexity is to associate information content with the difficulty of describing data, whether or not we can model that data as coming from a probabilistic source as in Shannon Information Theory. The notion of "describing data" can be made more concrete by allowing any process which can create the data in question, and a process is simply a computational procedure or algorithm. This then is the rough definition of Kolmogorov complexity:

The Kolmogorov complexity of a string $x$ is the length of the shortest program that generates $x$.

There are two important points to understand from this rough definition. First, only the size of a program is mentioned, but people familiar with compression generally think in terms of two items: a program (the decompressor) and its input (the compressed data). However, this is not really such an odd restriction. Since we are considering only a single instance (i.e., one output string), the input to a decompression program is fixed and can, in fact, be included in the program itself. Second, notice that every string does have a finite program that generates it. While we delve into the question of "what is a program" below, every reasonable programming language could produce any string with a single long "print statement," for example, "Print (0100111011000101000010010101)."

While many different types of computing machines are available, the notion of what is computable is extremely robust across all the options, from real physical machines to mathematical models (including more esoteric models such as quantum computers) and even to our understanding of how the human brain processes information. In fact, the "Church–Turing thesis" states that the intuitive notion of computability agrees exactly with what can be computed by current
formal universal models of computation, such as a Turing machine or a random access machine [6]. While the Church–Turing thesis has not been proved (and cannot be proved unless something more precise than an "intuitive notion" is involved), it is generally accepted to be true. So in considering a specific model of computation, such as a Turing machine, we do not restrict the power of what we can compute.

Associating the smallest description of data with a metric for the validity or information content of the data is not unique to Kolmogorov complexity. Other areas, including other branches of mathematics and philosophy, also express similar ideas. For instance, in statistics there is a theory known as the “minimum description length principle,” or just “MDL,” which states that if you are given a collection of theories that describe some data, the most appropriate one is the one that minimizes the sum of the sizes of the theory description and the encoding of the data under that theory. As such theories must be describable, and hence computable, this is clearly just a restatement of Kolmogorov complexity, and in fact Kolmogorov complexity has been put to great use in the area of statistics and inductive reasoning. Furthermore, these concepts correspond very nicely to the philosophical statement known as “Occam’s Razor”: given competing explanations for an event, the simplest one is usually right.

2.3.1 Basic Definitions

As described above, the difficulty of describing data can be identified with the difficulty of describing an algorithm that produces the data. Note that we are not concerned (at least for now) with the computational complexity of this process, but rather only with the difficulty of describing the algorithm. So how are algorithms expressed, and how can we measure the complexity of an algorithm description?

The choice of measurement is a standard one: We represent data with binary strings, and the complexity of such a string is simply its length. In this chapter, we use the notation $|x|$ to denote the length of a binary string $x$. While the choice of a binary coding alphabet is straightforward, the semantics of such a string are unclear—in other words, given a binary string $x$, what does it represent? There are many possible choices: perhaps $x$ is a binary encoding of a Turing machine, or $x$ is the ASCII representation of a LISP program, or $x$ is a binary executable for some specific machine architecture and OS.

We will use the term “descriptive language” to refer to such a language,¹ where the only condition is that it be possible to actually execute programs from this language. In particular, let $p$ be a program written in such a language, and $y$ be an input to $p$, and let $\phi$ be the function that maps the pair $(p, y)$ to the output when program $p$ is run with input $y$. Then the function $\phi$ must be computable (or more precisely, $\phi$ must be a partial recursive function). As the language and the function that simulates it ($\phi$) are really the same, these terms can be used interchangeably.

**Definition 2.3.1.** The Kolmogorov complexity of a string $x$ with respect to a descriptive language $L$ is the shortest program in $L$ that produces $x$ as output. We denote this complexity by $C_L(x)$.

An example should clearly demonstrate that selection of the descriptive language can make a very real difference. Consider the following base 10 number: 51090942171709440000. This number is exactly $21!$ (21 factorial), but this number requires over 65 bits to represent in binary.

¹ Note that we use the term “language” here in the sense of programming languages, which have both syntax and semantics. This should not be confused with the formal languages sense of a “language” which defines just a set of strings with no particular semantics.
Although you could write a program in C to compute this value, numbers this large cannot be directly represented, at least on current machines. So, in addition to the factorial program, you would have to write routines to do arithmetic on “big numbers.” As a result, the size of this program would be substantially larger than just printing out the number. On the other hand, if our descriptive language is the Mathematica programming language, then this value can be produced by the three-character program “2 !,” since Mathematica has intrinsic support for both the factorial function and for working with large numbers. In other words, for this string \( C_{\text{Mathematica}}(x) \) is much smaller than \( C_C(x) \).

Although the choice of representation does indeed make a difference, as just shown, the difference turns out not to be terribly great for an important and common class of languages called “universal languages.” In fact, by definition, any universal language (sometimes called a “Turing complete language”) can simulate any other universal language, and the size of such a simulation program is a constant that depends only on the two languages and not on the program that is being simulated. For example, we can certainly simulate full Mathematica functionality with a constant-size C program (in fact, Mathematica may very well be written in C). This leads us to what is called the “Invariance Theorem.”

**Theorem 2.3.1.** If \( f \) and \( g \) are two universal languages, then for all strings \( x \), \( |C_f(x) - C_g(x)| \leq c_{f,g} \), where \( c_{f,g} \) is a constant that depends only on \( f \) and \( g \).

At times (such as in the proof of Theorem 2.4.1 later in this chapter), it is convenient to consider arbitrary languages, even if they are not universal. In such a case we can still upper-bound the power of such a language, but since we do not require that all things are computable we cannot lower-bound the power of this language. Thus, the following theorem is a slight variation of the Invariance Theorem.

**Theorem 2.3.2.** If \( f \) is a universal language, and \( g \) is any language defined by a partial recursive function \( g \), then for all strings \( x \), \( C_f(x) \leq C_g(x) + c_{f,g} \), where \( c_{f,g} \) is a constant that depends only on \( f \) and \( g \).

Theorem 2.3.1 says that as long as the language is universal, it does not matter what it is, up to an additive constant. The example given previously (representing 21!) showed an extreme difference, but this was only because the data being represented were so small. For large strings, the additive constant difference in complexities becomes insignificant. This independence of language is what makes Kolmogorov complexity have a fundamental meaning that transcends issues of whim such as choice of language. Li and Vitányi [10] stress this point succinctly by noting that “The remarkable usefulness and inherent rightness of the theory of Kolmogorov complexity stems from this independence of the description method.”

Since the choice of language does not make a significant difference (as long as it is universal), we will remove the subscript from the complexity notation and simply refer to \( C(x) \), understanding that results are significant only up to an additive constant value. There is in fact an area of study known as “concrete Kolmogorov complexity” in which a specific language is chosen (such as an encoding of a Turing machine with a specific, preferably small, number of states, symbols, and tapes), and then the actual constants can be estimated and evaluated.

### 2.3.2 Incompressibility

It should be clear that any lossless compression technique can be viewed in the light of Kolmogorov complexity, as just described, since we must have a program that can reproduce the original data. Unlike Shannon Information Theory, we do not have to refer to a sometimes artificially created