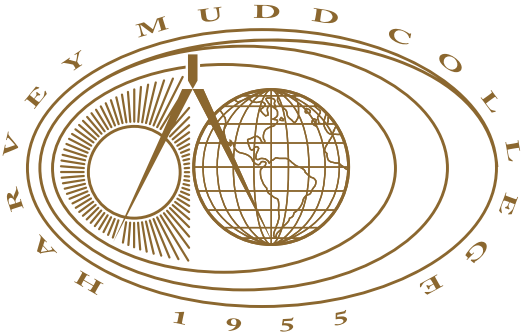
2006

Kolmogorov Complexity of Graphs

John Hearn
Harvey Mudd College



Applications of Kolmogorov Complexity to Graphs

John Hearn

Ran Libeskind-Hadas, Advisor

Michael Orrison, Reader

May, 2006

HARVEY MUDD
COLLEGE
Department of Mathematics

Abstract

Kolmogorov complexity is a theory based on the premise that the complexity of a binary string can be measured by its compressibility; that is, a string's complexity is the length of the shortest program that produces that string. We explore applications of this measure to graph theory.

Contents

Acknowledgments

Foremost, I would like to thank my advisor, Prof. Ran Libeskind-Hadas for his support, assistance, and direction throughout my research. I would also like to thank Prof. Michael Orrison, Prof. Art Benjamin, and Prof. Henry Krieger for their input and assistance at various sticking points during the year, and Prof. Ming Li at the University of Waterloo for allowing me to take his course on Kolmogorov Complexity and for his suggestions and feedback on my research. Last but not least, I would like to thank Prof. Lesley Ward for her dedication to making the Harvey Mudd Senior Thesis a rewarding experience for everyone involved.

Chapter 1

Introductory Material

When attempting to characterize the complexity of an object, a useful question to ask is: How much information does it contain? In other words, what is the shortest description we can give the object such that no information about that object is lost, that is, it can be accurately reproduced? Even establishing what constitutes a description poses some difficulties. For instance, consider the positive integer n, which is "the least natural number that cannot be discribed in less than twenty words." If n exists, we have just described it in thirteen, contradicting the statement of its definition. This rather upsetting paradox is not easily resolved and in fact serves as the basis of an elegant proof of Gödel's incompleteness result (Li and Vitányi 1997, 169-170). However, for our purposes, problems of this sort can be ignored. Even if n exists, the statement gives no information useful for finding it. We will circumvent the paradox by restricting ourselves to objects that can be fully characterized by finite strings and limit our descriptions to those which are *sufficient* to reproduce the desired object.

Kolmogorov complexity is a measure of the information contained in the description of an object. Specifically, the Kolmogorov complexity of an object is the length (literally the number of 1s and 0s) of the shortest binary string that is sufficient to replicate it. Hence, we have only countably many describable objects.

It is important to note the distinction here from information theoretic descriptions. Data transmission (from sender to recipient) relies upon an agreed upon context for interpretation. This reduces the amount of information that must be communicated to reproduce many objects and generally imposes some limitation on the time complexity of data interpretation algorithms. For instance, a sender would need only to communicate

2 bits to encode the integer 7, π, or a binary representation of the Oxford English Dictionary, provided the recipient knows in advance what those objects are, and that the sender will be selecting one of them and no others. Information theory allows selection of an object from a finite set, thus the information transmitted is a function of the set size, not the size of the objects themselves. We allow *any* encoding of a string so long as it can be decoded *eventually*. We will not generally think in terms of a sender and a recipient, but for the sake of comparison, we would say that the two speak the same language, but assume they have never communicated, thus a description must be completely self contained. An analogy more useful to us is that a description is a program which outputs the object.

The first objection one might raise at this point is that program length is dependent on language. For instance, some objects are more simply described using C++ than say FORTRAN. It turns out the difference in description length for an object programmed in different languages is bounded by an additive constant. We will show this result, but must first introduce some terms and notation useful for formalizing our descriptions and the measurement of their complexity.

1.1 Definitions and Notation

Remark. A brief treatment of Turing machines, (possibly finite state machines,) and regular expressions needs to go here.

1.1.1 Definition. By associating inputs and outputs, a Turing machine defines a partial function from n-tuples of integers onto the integers, with $n \geq 1$. We call such a function *partial recursive* or *computable*. If the Turing machine halts for all inputs, then the function computed is defined for all arguments and is called *total recursive*, or simply *recursive*.

Generally speaking, $\langle x, y \rangle$ will be used to denote a self-delimiting concatenation of the binary representation of x and y. There are many ways to do this. One method is to double each of the bits of the first string and place '01' between them. Thus, for $x = 1010$ and $y = 0110$ we have $\langle x, y \rangle = 11001100010110$. A computer can determine where x 'ends' and y 'begins' by finding the first string of zeros with odd length. Thus we use $\langle \cdot \rangle$ as the notation to denote a standard recursive bijective pairing function. The notation for concatenation of more than two strings is defined recursively by $\langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle$.

For some countable set of objects, S, we can assume some standard enumeration where $x \in S$ is associated with a natural number $n(x)$. We want to know if there exists another specification for x more space efficient than n. That is, a method f is a partial function over naturals where $n(x) = f(p)$. It is convenient to think of p as a program and f as the programming language, compiler, and computer. We denote length of a program by $l(p)$.

We say:

$$C_f(x) = \min\{l(p) : f(p) = n(x)\}$$

where p is the shortest program that generates x (with no input) with respect to some partial function f. We call $C_f(x)$ the *unconditional Kolmogorov complexity* with respect f. If no such p exists, we say $C_f(x) = \infty$.

If there exists a constant c such that for all $x \in S$, $C_f(x) \leq C_g(x) + c$, we say method f *minorizes* method g, and f and g are *equivalent* if they minorize each other. Each $x \in S$ might rely on any of the distinct methods $f_1, f_2, \ldots, f_r$ for a minimal Kolmogorov complexity. By reserving the first $\log r$ bits of p to indicate by enumeration of the functions which f_i is used for producing x from p we have a method f minorized by all f_i where $c \approx \log r$.

1.1.2 Definition. Let $\mathcal{C}$ be a subclass of the partial functions over $\mathbb{N}^+$. A function f is *universal* (or additively optimal) for $\mathcal{C}$ if it belongs to $\mathcal{C}$ and if for every function $g \in \mathcal{C}$ there is a constant $c_{f,g}$ s.t. $C_f(x) \leq C_g(x) + c_{f,g}$ for all x. Here $c_{f,g}$ depends on f and g but not x(Li and Vitányi 1997, 95).

Note that the above definition is given for single variable functions. We can extend the definition to encompass functions of multiple arguments by setting $f(x_1, x_2, \ldots, x_k) = f(\langle x_1, x_2, \ldots, x_k \rangle)$.

We say additively optimal methods f, g of specifying objects in S are equivalent in the following way:

$$\left|C_f(x) - C_g(x)\right| \leq c_{f,g}$$

for all x, where $c_{f,g}$ is a constant depending only on f and g.

There is no universal partial function f for all programs p. However, there does exist a universal element in the class of partial *recursive* functions. This is a modest and rather natural restriction of our descriptions, as there would be little use in attempting to define the information content of the non-existent output of programs which do not halt. We thus consider the class of description methods $\{\phi : \phi \text{ is a partial recursive function}\}$. We use ϕ_0 to denote the universal description method, which gives us the following definition (Li and Vitányi 1997, 95-97).

1.1.3 Definition. Let x, y, p be natural numbers. Any partial recursive function ϕ, together with program p and input y, such that $\phi(\langle y, p\rangle) = x$, is a *description* of x. The *complexity* C_ϕ of x *conditional* to y is defined by

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

and $C_\phi(x|y) = \infty$ if there is no such p. We call p a program to compute x by ϕ, given y.

‘

By selecting a fixed ϕ_0 as our reference function for C, we can drop the subscript to denote the *conditional Kolmogorov complexity* where $C(x|y) = C_{\phi_0}(x|y)$. Note the unconditional Kolmogorov complexity $C(x) = C(x|\epsilon)$.

1.2 The Invariance Theorem

Finally, we have sufficiently well defined the ideas and most of the notation necessary to see some powerful theorems and interesting results. The Invariance Theorem, along with the Incompressibility Theorem and a trivial upper bound given in the next section, though short, elegant and even simple to prove, form the basis for the whole study of Kolmogorov Complexity, and are sufficient for many important proofs.

1.2.1 Lemma. *There is a universal partial recursive function (Li and Vitányi 1997, 96).*

Proof. Let ϕ_0 be the function computed by a universal Turing machine U. Machine U expects input of the format

$$\langle n, p\rangle = \underbrace{11\ldots1}_{l(n)\text{ times}}0np.$$

The interpretation is that the total program $\langle n, p\rangle$ is a two-part code of which the first part consists of a self-delimiting encoding of T_n and the second part is the literally rendered program p. To encode T_n, it is suffices to provide U with n, where T_n is the n^{th} machine in the standard enumeration of Turing machines. This way U can parse the binary input into the T_n-part and the p-part, and subsequently simulate the computation of T_n started with program p as its input. What happens if U gets the program "$0p$"? By convention we can set $U = T_0$ and therefore $U(0p) = U(p)$. Altogether, if T_n computes partial recursive function ϕ_n, then

$$C_{\phi_0}(x) \leq C_{\phi_n}(x) + c_{\phi_n},$$

where c_{ϕ_n} can be set to $2l(n) + 1$. □

This result from computability theory generalizes to the Invariance Theorem, which considers the complexity of an object x facilitated by an already specified object y. Recall that Kolmogorov complexity for arbitrarily many conditionals can be defined by recursive use of the bijective pairing function.

1.2.1 The Invariance Theorem. *There is a universal partial recursive function ϕ_0 for the class of partial recursive functions to compute x given y. Formally this says that $C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi$ for all partial recursive functions ϕ and all x and y, where c_ϕ is a constant depending on ϕ but not x or y (Li and Vitányi 1997, 97).*

Proof. Let ϕ_0 be the function computed by a universal Turing machine U such that U started on input $\langle y, \langle n, p\rangle\rangle$ simulates T_n on input $\langle y, p\rangle$. That is, if T_n computes partial recursive function ϕ_n, then $\phi_0(\langle y, \langle n, p\rangle\rangle) = \phi_n(\langle y, p\rangle)$. Hence, for all n,

$$C_{\phi_0}(x|y) \leq C_{\phi_n}(x|y) + c_{\phi_n}.$$

By the proposed encoding of T_n, we have that $c_{\phi_n} \leq 2l(n) + 1$. □

Notice that the universal description method may not give the shortest description for all x, but no other method gives a shorter description for more than finitely many cases. We also note a trivial upper bound given by the following theorems (but omit the proofs).

1.2.2 Theorem. *There is a constant c such that for all x and y*

$$C(x) \leq l(x) + c \text{ and } C(x|y) \leq C(x) + c.$$

(Li and Vitányi 1997, 100)

In the case of objects conditionally belonging to finite sets, we can offer an improved upper bound with the following theorem and then explore some simple examples of how the Invariance theorem can be used.

1.2.3 Theorem. *Let $A \subset \mathbb{N} \times \mathbb{N}$ be recursively enumerable, and $y \in \mathbb{N}$. Suppose $Y = \{x : (x, y) \in A\}$ is finite. Then, for some constant c depending only on A, for all $x \in Y$, we have $C(x|y) \leq l(|Y|) + c$ (Li and Vitányi 1997, 103).*

1.2.1 Example. Show that $C(0^n|n) \leq c$, where c is a constant independent of n.

Proof. Given n, we can construct a Turing machine M which outputs 0^n regardless of input. By a canonical enumeration of turing machines, $M = T_m$, so $U(\overline{m}) = 0^n$ where $\overline{m}$ is the self delimiting string $1^{l(m)}0m$ giving us $C(0^n|n) = 2\log m + 1 \leq c$. □

1.2.2 Example. Show that there are infinite binary sequences ω such that the length of the shortest program for reference turing machine U to compute the consecutive digits of ω one after another can be significantly shorter than the length of the shortest program to compute an initial n-length segment $\omega_{1:n}$ of ω, for any large enough n.

Proof. Given program p such that $U(p) = \pi^*$, we have $C(\pi) = l(p)$. We can define infinitely many distinct infinite sequences by the function $\pi(m) = \pi_{m+1}\pi_{m+2}\pi_{m+3}\ldots$ where π_i denotes the i^{th} character of the sequence π. From p, we can construct a Turing machine M such that $M(m) = \pi(m)$ as follows. On input m, M runs $U(p)$ dovetailed with code to overwrite the left most non-blank character on the tape once that character is no longer necessary for further computation, and does so until the first m characters of the sequence have been overwritten, after which the output from $U(p)$ is unaltered. For some canonically enumerated turing machine, $M = T_k$, thus $U(\overline{k}m) = \pi(m)$, giving us a countably infinite set of programs, each of finite length but generating a distinct infinite sequence. We have $C(\pi(m)) \leq 2\log k + \log m + 1$.

Unlike the machines generating infinite sequences, a machine V that encodes the initial n-length segment $\pi(m)_{1:n}$ of $\pi(m)$ must cease writing characters to the input tape after the n^{th} character, or at least delimit the initial n-length segment from any other characters written. Hence, if $V(p_n) = pi(m)_{1:n}$ the self-delimiting description $1^{l(n)}0n$ must appear in p_n. So for $n \gg 2l(k) + l(m)$, $C(\pi(m)_{1:n}) > C(\pi(m))$. □

*We treat π as the sequence corresponding to its binary expansion (with decimal point omitted), rather than a real number. We also assume that the number of digits of the sequence written to the tape is proportional to the length of time the program has run, and that after some constant interval following a character being written to the tape, it is no longer necessary for computation of latter characters.

1.3 The Incompressibility Theorem

Now that we have established that a method exists for describing all but a finite number of x in S with maximal efficiency, what can we infer about those descriptions? Well, for each n there are 2^n binary strings of length n, but only $2^n - 1$ descriptions shorter than n. Thus there exists at least one binary string x of length n with $C(x) \geq n$. We then say x is *incompressible*.

1.3.1 Definition. For each constant c we say a string x is *c-incompressible* if $C(x) \geq l(x) - c$ (Li and Vitányi 1997, 109).

1.3.1 The Incompressibility Theorem. *Let $c \in \mathbb{N}^+$. For each fixed y, every finite set A of cardinality m has at least $m(1 - 2^{-c}) + 1$ elements x with $C(x|y) \geq \log m - c$ (Li and Vitányi 1997, 109).*

Proof. The number of programs of length less than $\log m - c$ is

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1$$

Hence, there are at least $m - \frac{2}{c} + 1$ elements in A that have no program of length less than $\log m - c$. □

What we see by this theorem is the fairly surprising result that of all binary strings of length n, at least half of them can only be compressed by no more than one digit. Another quarter or more of the strings can only be compressed by at most 2 digits, and so on. This itself has some rather counter intuitive results.

For instance, if x is an incompressible string, are all substrings in x also incompressible? Intuitively, the ability to compress a substring would seem to give us a means to compress x. We can place a lower bound on substring v given by $C(v) \geq l(v) - O(\log n)$ but cannot prove $C(v) \geq l(v) - O(1)$. If the latter were true, x could contain no long regular subsequences since, for example, a sequence of k zeroes has complexity $O(\log k)$. But for strings of length n, only a small subset have no regular substrings, which gives us an easy way to describe them. Thus, for x to be incompressible, it *must* have compressible substrings (Li and Vitányi 1997, 110).

Suppose that we know that x is an element of A, a subset of the natural numbers. We consider the complexity $C(x|A)$. When A has finitely many elements, it is fairly easily shown (recall the earlier discussion of Information theory and Theorem 1.2.3) that $C(x|A) \leq 2l(|A|) + c$ where c is a constant possibly dependent on A, but independent of x. On the other hand, $C(x|N) = C(x)$, since x is assumed to be a natural number.

1.3.2 Definition. The *randomness deficiency* of x relative to A is defined as $\delta(x|A) = l(|A|) - C(x|A)$. It follows that $\delta(x|A) \geq -c$ for some fixed constant c independent of x (Li and Vitányi 1997, 113).

1.3.2 Theorem. *(The above discussion is assumed.) Then,* $|\{x : \delta(x|a) \geq k\}| \leq |A|/2^{k-1}$ *(Li and Vitányi 1997, 113).*

Proof. There are fewer than 2^{l+1} programs of length less than or equal to l. □

These seemingly simple results prove surprisingly powerful. The Incompressibility theorem gives rise to the Incompressibility Method, an elegant and versatile proof technique we will use in sequel chapters. Here we show some more immediate results.

1.3.3 Example. We say x is an *n-string* if x has length n and $x = n00\ldots0$.

1. Show that there is a constant c such that for all n-strings x we have $C(x|n) \leq c$. (Where c depends on the reference Turing machine U used to define C.)

   Proof. We can build a Turing machine M which, given n, finds the n^{th} binary string given by the lexicographic indexing of all binary strings, prints the string followed by $n - l(n)$ zeros, and halts. For our canonical enumeration of Turing machines, $M = T_m$ and $C(x|n) = 2l(m) + 1 \leq c$. □

2. Show there is a constant c such that $C(x|n) \leq c$ for all x in the form of the n-length prefix of $nn\ldots n$.

   Proof. We can build a Turing machine M which, given n, finds s, the n^{th} binary string given by the lexicographic indexing of all binary strings, and prints the first n characters of the regular expression s^* and halts. For our canonical enumeration of Turing machines, $M = T_m$ and $C(x|n) = 2l(m) + 1 \leq c$, where c is dependent only on the reference Turing machine U. □

3. Let c be as in Item (1). Consider any string x of length n with $C(x|n) \gg c$. Let $y = x00\ldots0$ of length x. Prove that no matter how high its $C(x|l(x))$ complexity, for each string x, there exists string y with complexity $C(y|x) \leq c$ and $C(y|l(y)) < c$.

Proof. Given x, we can construct a Turing machine V that finds the index of the string that matches x given by the lexicographic indexing of all binary strings, runs machine M from Item (a) on the result, prints M's output, and halts. Thus, given x, our machine's output is y. Since $V = T_k$, some Turing machine in our canonical enumeration, $U(\bar{k}) = y$ and $C(y|x) = 2l(k) + 1 \leq c_v$. But we know from Item (1) that c_v is independent of x and y. Thus $c_v = c$, a constant such that each string x, no matter how high its $C(x|l(x))$ complexity, can be extended to a string y with $C(y|l(y)) < c$. □

1.3.4 Example. Prove that for each binary string x of length n there is a y equal to x but for one bit such that $C(y|n) \leq n - \log n + O(1)$.

Proof. For a binary string x of length n, let $\{y_1, y_2, \ldots, y_n\}$ be the set of strings where y_i is equal to x except at the i^{th} bit. At least one y_i is an element of a Hamming code of n-length strings.

Since the set of binary strings of length n constituting a Hamming code is recursive, there is a Turing machine H which will list them. We can enumerate the $2^n/n$ elements with $\lg\left(\frac{2^n}{n}\right) = n - \lg n$ bits. Thus given n, we can construct a Turing machine M which computes output y_i on input i by running H and returning the i^{th} element. Thus, $C_M(y_i|n) = l(i) \leq n - \lg n$. By Theorem 1.2.1, $C(y|n) \leq n - \log n + O(1)$. □

Chapter 2

Graph Complexity and the Incompressibility Method

Canonically, a graph $G = (V, E)$ with n vertices labeled $V = \{1, 2, \ldots, n\}$ is encoded as a $n(n-1)/2$ length string $E(G)$ where each bit corresponds lexicographically to a vertex pair. Thus $E(G) = e_{1,2}e_{1,3}\ldots e_{1,n}e_{2,3}e_{2,4}\ldots e_{n-1,n}$ where $e_{u,v} = 1$ if $(u, v) \in E$ and $e_{u,v} = 0$ otherwise. Thus by vertex relabeling we have $n!$ distinct strings, each encoding some member of an equivalence class of isomorphic graphs. However, the equivalence class has fewer than $n!$ members if there are automorphisms: multiple vertex labelings that produce the same string. Formally, we say an *automorphism* of $G = (V, E)$ is a permutation π of V such that $(\pi(u), \pi(v)) \in E$ if and only if $(u, v) \in E$ (Li and Vitányi 1997, 402). As a trivial example, consider the empty graph on n vertices: all labelings result in the same graph.

Often we encounter problems on unlabeled graphs, such as VERTEX COVER, that depend only on the graph's structure. Particularly in the case of the decision problem, any label information is irrelevant to the solution, but labels are necessary if we are to use our encoding scheme. (For clarity we will generally denote unlabeled graphs with Γ and labeled graphs with G. In some cases, however, we may indicate a graph Γ has been given label permutation π by Γ_π.) The label permutation can be arbitrarily selected, but it would be gratifying to have our string reflect the graph's complexity in some intuitive way. Kolmogorov Complexity is an attractive metric, as it would seemingly give us a measure based on the graph's compressibility. Unfortunately, the compressibility of the string $E(G)$ is clearly very dependent on our label permutation. Consider the labeled graphs G_1 and G_2 (Fig. 1) isomorphic to an unlabeled graph Γ.

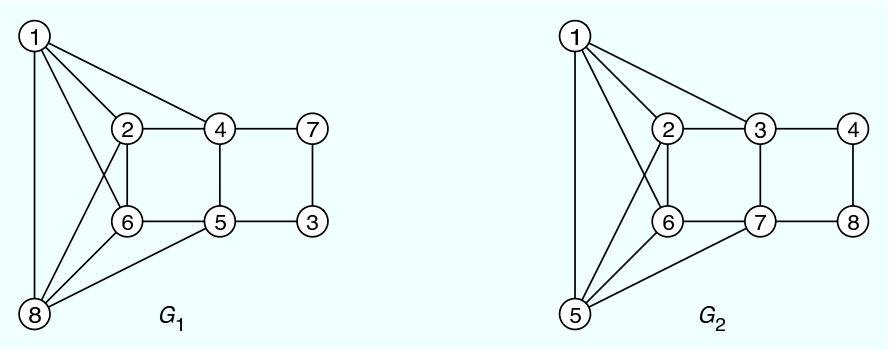


Figure 2.1: *Isomorphic graphs.*

The respective labelings give us $E(G_1) = 10101010101010101010101010 = (10)^{14}$ and $E(G_2) = 11011001011001001000011101 01$. (Note that we are using operations on regular expressions, not natural numbers.) While $E(G_2)$ may or may not be compressible, $E(G_1)$ is clearly *very* compressible and in all likelihood the most compressible string in the equivalence class Γ produces. (Admittedly, our chosen example is contrived. Most graph will be far less compressible.)

We would like to have the complexity of Γ, which we will abusively denote by $C(\Gamma)$, less than or equal to $C(E(G_1)) + O(1)$. First, though, we will consider some known properties of the complexity of labeled graphs.

2.1 Complexity of Labeled Graphs

Consider a class of finite objects (O) parametrized with n, such as n-node graphs or strings of length n.

2.1.1 Lemma. *Let P be a property holding for objects $O \in \mathcal{O}$ with randomness deficiency $\delta(n)$. Then P holds with probability at least $1 - 1/2^{\delta(n)-1}$ (Li and Vitányi 1997, 388).*

Proof. There are only $\sum_{i=0}^{\log|\mathcal{O}|-\delta(n)} 2^i$ programs of length less than or equal to $\log|\mathcal{O}| - \delta(n)$ and there are $|\mathcal{O}|$ objects. □

2.1.1 Corollary. *A fraction of* at least $1 - 1/2^{\delta(n)}$ *of all labeled graphs G on n vertices have a randomness deficiency no greater than $\delta(n)$ (or is $\delta(n)$-*random*) when $C(E(G)|n, \delta) \geq n(n-1)/2 - \delta(n)$ (Li and Vitányi 1997, 397).*

From these results, we can find asymptotic bounds on the complexity of unlabeled graphs. First, however, we must introduce a new proof technique.

2.2 The Incompressibility Method

The Incompressibility Theorem, introduced in the previous chapter, gives us the powerful and elegant proof technique called the *incompressibility method* which has applications in many fields. For example, it is known that any proof using a pumping lemma from formal language theory can be proven using the more intuitive incompressibility method (See Appendix A for examples).

The incompressibility method is also valuable in combinatorics and graph theory. Many proofs that rely on the probabalistic method can be proved using the incompressibility method, often yielding a more concise and intuitive proof. The two methods bear some resemblance, as they are both non-constructive. However, where the probabilistic method shows that some element with a particular property must exist, the incompressibility method shows that *most* elements must have the property. The methods are best explained by example, so we begin by considering the celebrated result from Ramsey Theory, proved by Paul Erdös in 1947, that first popularized the probabilistic method.

Ramsey Theory is an area of discrete mathematics concerned with combinatorial objects (such as graphs) in which particular properties must occur once the scale of the object grows beyond a certain threshold. The classical problem in Ramsey theory is the *party problem*, which asks the smallest number of people $R(j,k)$ that must be in a room to guarantee that at least j know each other or at least k do not know each other. Here, $R(j,k)$ is called the *Ramsey Number*. In graph theoretic terms, $R(j,k)$ is the smallest number n such that every graph on n vertices contains a clique of size j or an independent set of size k.

The following is a technical lemma necessary for the probabilistic proof of the subsequent theorem. The proof of lemma is not particularly instructive and is thus omitted. (See Propositions 14.1.5 and 14.1.6 in *Combinatorial Mathematics* by Douglas West for the proof.)

2.2.1 Lemma. *For $k \in \mathbb{N}$, $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$ (West 2006, 710).*

2.2.1 Theorem. *$R(k,k) \geq k2^{k/2}\left(\frac{1}{e\sqrt{2}} - o(1)\right)$ (West 2006, 711).*

Proof. (Probabistic method.) It is equivalent to show that any red-blue coloring of a K_n contains a red K_k or a blue K_k, where we think of the red edges as corresponding to edges in our graph, and blue edges to those not in the graph.

We randomly color the edges of the K_n by $\binom{n}{2}$ coin flips, with HEADS=RED and TAILS=BLUE, thus creating the probability space

$$P(e_{i,j} = \text{RED}) = P(e_{i,j} = \text{BLUE}) = \frac{1}{2}.$$

Let S be the set of edges of the graph induced by a k-subset of the n vertices. Let A_S be the *event* that the edges of S are monochromatic. Thus,

$$P(A_S) = 2 \cdot 2^{-\binom{k}{2}} = 2^{1-\binom{k}{2}}.$$

Consider the disjunction $\bigvee A_S$ over all possible S. For n very large, $P\left(\bigvee A_S\right)$ would be very difficult to calculate, but we need only to bound this quantity. Thus,

$$P\left(\bigvee A_S\right) \leq \sum_{S \subseteq [n]} P(A_S) = \binom{n}{k} 2^{1-\binom{k}{2}}$$

since there are $\binom{n}{k}$ summands. If $\binom{n}{k}2^{1-\binom{k}{2}} < 1$, then $P\left(\bigwedge \overline{A}_S\right) > 0$, that is, the event that there is a monochromatic K_k has positive probability. In other words, we are guaranteed to have a monochromatic K_k when the following condition is met:

$$\binom{n}{k} < 2^{\frac{k(k-1)}{2}-1}.$$

By Lemma 2.2.1, we have that $\binom{n}{k} < (ne/k)^k$. Thus it is sufficient to have that $ne/k \leq 2^{(k-1)/2}$, or equivalently $n \leq \frac{k2^{k/2}}{e\sqrt{2}}$, giving us the desired result. □

2.2.1 Definition. We say a labeled graph G on vertex set $[n]$ is an *incompressible graph* if the canonical binary string encoding $E(G)$ has $C(E(G)|n) \geq n(n-1)/2$.

2.2.2 Lemma. *Let G be an incompressible graph, and let k_G be the size of the largest clique (or independent set) in G.*

$$k_G \leq 1 + \lfloor 2 \log n \rfloor.$$

Proof. (Incompressibility method.) Choose an incompressible graph G such that

$$C(E(G)|n,p) \geq n(n-1)/2 \tag{2.1}$$

where p is a program that on input n and $E'(G)$ (a compression of $E(G)$ explained below) outputs $E(G)$.

Without loss of generality, we can assume the largest clique in G is at least as large as the largest independent set. (We can append a single bit at the end of our string to indicate whether we are encoding G or the complement of G. The cost of the additional bit drops during calculation as an $O(n^{-1})$ term.) Let S be the set of vertices of the largest clique in G. We attempt to compress $E(G)$ to $E'(G)$ as follows:

1. We add as a prefix to $E(G)$ the list of vertices in S, with each vertex using $\lceil \log n \rceil$ bits, adding $k_G \lceil \log n \rceil$ bits.

2. We delete all the redundant bits in the $E(G)$ part which represent edges between vertices in S, saving $k_G(k_G-1)/2$ bits.

Thus,

$$l(E'(G)) = l(E(G)) - \frac{k_G}{2}(k_G - 1 - 2\lceil \log n \rceil). \tag{2.2}$$

Given n and the program p, we can reconstruct $E(G)$ from $E'(G)$. Hence,

$$C(E(G)|n,p) \leq l(E'(G)). \tag{2.3}$$

Equations (2.1), (2.2), and (2.3) hold only when $k_G \leq 1 + \lfloor 2 \log n \rfloor$. □

Theorem 2.2.1 comes as a corollary to this lemma.

Proof. (Incompressibility method.) To describe a clique (or independent set) of size k in a graph G on $n = R(k,k)$ edges, we need $\log \binom{R(k,k)}{k}$ bits. By simple algebra, we find $\log \binom{R(k,k)}{k} \leq k \log R(k,k) - \log k!$ bits. Choose G to be incompressible. Then, $k \log R(k,k) - \log k! \geq k(k-1)/2$, since otherwise we can compress G as demonstrated in the proof of Lemma 2.2.2. We use Stirling's approximation $k! \approx \left(\frac{k}{e}\right)^k \sqrt{2\pi k}$ and algebraic manipulation to isolate $R(k,k)$ for the desired result. □

Remark. Here will be some exposition on the connection between incompressible and high probability objects and when one proof technique is likely to be more advantageous than the other. Will mention that this suggests a connection between probability and complexity, pointing toward the Coding Theorem introduced in Chapter 3.

2.3 Complexity of Unlabeled Graphs

Remark. The following lemma is my first original result. It relies on the Incompressibility method, a proof technique which takes the following form: a property holds for c-incompressible strings x, where $l(x) \gg c$; most strings are c-incompressible; thus the property holds for most strings.

We let $\langle \Gamma_\pi \rangle$ denote the maximally compressed string encoded from Γ under label permutation π and $\langle \Gamma_0 \rangle$ denote Γ encoded and compressed under the label permutation which produces the shortest string encoding.

2.3.1 Lemma. *There exist unlabeled graphs* Γ *such that* $C(\Gamma|n) \geq n(n-1)/2 - O(n \log n)$.

Proof. Consider a graph Γ on n vertices. Labeled under π, we have $l(E(\Gamma_\pi)) = n(n-1)/2$ and $C(E(G_\pi)|n) = n(n-1)/2 - \delta(n)$. We let π be a label permutation resulting in the string encoding with maximal randomness deficiency $\delta(n)$. Recall we denote the maximally compressed string by $\langle \Gamma_\pi \rangle = \langle G_0 \rangle$.

There are only $n!$ label permutations on n vertices, which we can enumerate using $O(n \log n)$ bits. By a self-delimiting concatenation of the compressed string $\langle G_0 \rangle$ with the enumeration of a desired permutation ρ, we have a compression mechanism for any labeled graph $G = \Gamma_\rho$ with $C(E(G)|n) \leq n(n-1)/2 - \delta(n) + O(n \log n)$. However, *most* strings are Kolmogorov-random, thus we know that for most graphs G we have an incompressible string, that is $C(E(G)|n) \geq n(n-1)/2 + O(1)$. So for all but very few G, we have that $\delta(n) \in O(n \log n)$. □

In other words, most graphs, even without labels, cannot be encoded using less than $2^{n(n-1)} - O(n \log n)$ bits. We would like to show that this is a tight bound. The following theorem will be necessary. The first proof of the theorem to use Kolmogorov complexity was found by Buhrman, Li, and Vitányi (Buhrman et al. 1999b, 596-597). Our proof is modified only slightly.

Remark. I developed a flawed proof of this theorem which followed similar reasoning before discovering it had already been proven. (I failed to show a proper bound on the number of automorphisms.) Upon presenting the failed proof, Prof. Li directed me to the paper where this proof was first published. I have tried to retain some of the original arguments I used, and as a result, the final upper bound on unlabeled graph compressibility is slightly less precise than in the paper. However, this gave better symmetry between the upper and lower bounds.

2.3.1 Theorem. *Let g_n denote the number of unlabeled graphs on n vertices. Then,*

$$g_n \approx \frac{2^{n(n-1)/2}}{n!}.$$

Proof. By inspection, it is easy to see $g_n \geq 2^{n(n-1)/2}/n!$. We can encode a graph Γ on n vertices labeled under permutation π with $n(n-1)/2$ bits. There are $2^{n(n-1)/2}$ strings of this length, but only $n! \approx \sqrt{2\pi n}(n/e)^n$ label permutations. However, because of automorphisms, there are graphs with fewer than $n!$ distinct string encodings. Thus, there are strictly more than $2^{n(n-1)/2}/n!$ equivalence classes.

Let $\mathcal{G}_n$ denote the set of all undirected graphs on vertices $V = \{0, 1, \ldots, n-1\}$. We partition the set of graphs by $\mathcal{G}_n = \mathcal{G}_n^0 \cup \mathcal{G}_n^1 \cup \ldots \cup \mathcal{G}_n^n$ where $\mathcal{G}_n^m$ is the set of all graphs for which each of $m \leq n$ vertices are mapped by some automorphism to a vertex other than itself. Thus, $\mathcal{G}_n^i \cap \mathcal{G}_n^j = \varnothing$ for $i \neq j$. For $G \in \mathcal{G}_n$, let $Aut(G)$ denote the automorphism class of G and $\overline{G}$ be the isomorphism class of G.

(1) For $G \in G_n^m$, $|Aut(G)| \leq n^m = 2^{m \lg n}$ since $|Aut(G)| \leq \binom{n}{m} m! \leq n^m$. Consider each graph $G \in \mathcal{G}_n$ to have probability $P(G) = 2^{-n(n-1)/2}$.

(2) By Corollary 2.1.1, if $G \in \mathcal{G}_n^m$ and $C(G|n,m) \geq \binom{n}{2} - \delta(n,m)$, then $\delta(n,m) \geq m\left(\frac{n}{2} - \frac{3m}{8} - \log n\right)$.

Let $\pi \in Aut(G)$ move m vertices. Suppose π is the product of k disjoint cycles of sizes $c_1, c_2, \ldots, c_k$. We can describe π with $m \log n$ bits. For instance, if π moves vertices $i_1 < v_2 < \ldots < v_m$, then we can list the sequence $\pi(i_1), \ldots \pi(i_m)$. By sorting the latter sequence, we can obtain π^{-1}, and thus π.

We select the least numbered vertex from each of the k cycles. For each of the $m-k$ vertices on the cycles, we can delete the $n-m$ bits encoding the edges connecting them to static vertices and the $m-k$ half-bits encoding edges to other cycle vertices. Thus we delete a total of

$$\sum_{i=1}^{k}(c_i - 1)\left(n - m + \frac{m-k}{2}\right) = (m-k)\left(n - \frac{m+k}{2}\right)$$

bits. Since $k \leq m/2$, we have the desired $\delta(n,m)$. The difference of bits added and bits deleted is $\frac{m}{2}(n - \frac{3m}{4}) - m \log n$, as claimed.

Continuing the proof of the theorem:

$$g_n = \sum_{G \in \mathcal{G}_n} \frac{1}{|\overline{G}|} = \sum_{G \in \mathcal{G}_n} \frac{|Aut(G)|}{n!} = \frac{2^{n(n-1)/2}}{n!} E_n,$$

where we define E_n to be $\sum_{G \in \mathcal{G}_n^m} P(G)|Aut(G)|$ is the expected size of the automorphism group of a graph on n vertices. Since $E_n \geq 1$, we have the lower bound for g_n. We note that $\mathcal{G}_n^1 = 0$ and use (1) and (2) to obtain the upper bound as follows:

$$\begin{aligned} E_n &= \sum_{m=0}^{n} P(G \in \mathcal{G}_n^m) \cdot \mathrm{AVE}_{G \in \mathcal{G}_n^m}\{|Aut(G)|\} \\ &\leq 1 + \sum_{m=2}^{n} 2^{-m\left(\frac{n}{2} - \frac{3m}{8} - 2\log n\right)} \\ &\leq 1 + 2^{-(n - 4\log n - 2)}, \end{aligned}$$

which proves the theorem: $\frac{2^{n(n-1)/2}}{n!} \leq g_n \leq \frac{2^{n(n-1)/2}}{n!}(1 + 2^{-(n-4\log n-2)})$. □

A corollary of this surprising theorem is our desired result, that our bound on the compressibility of Kolmogorov random graphs is tight.

2.3.1 Corollary. *For an unlabeled graph* Γ *on* n *vertices,*

$$C(\Gamma|n) \leq n(n-1)/2 - O(n \log n).$$

Proof. There are $g_n \approx \frac{2^{n(n-1)/2}}{n!}$ distinct undirected, unlabeled graphs on n vertices. We can enumerate them with $n(n-1)/2 - O(n \log n)$ bits. □

Remark. Here will be a segue leading into discussion of the coding theorem. (Recap discussion at end of section 2.2.)

Chapter 3

The Coding Theorem

The goal of this chapter is to prove the Coding theorem, which Li and Vitányi describe as the surprising result that three "quite different formalizations of concepts turn out to be equivalent ... [suggesting] an inherent relevance that transcends the realm of pure mathematical abstraction" (Li and Vitányi 1997, 253). But in order to understand the Coding theorem, it is necessary to lay considerable groundwork.

3.1 Prefix Complexity

Here we will briefly introduce prefix Kolmogorov complexity, which is defined slightly differently than the plain Kolmogorov complexity we have been using and has some advantageous properties (but some weaknesses as well). The difference in the theory lies primarily in the set of functions we use as the basis of our enumeration of Turing machines.

3.1.1 Definition. A *partial recursive prefix function* $\phi : \{0,1\}^* \to \mathbb{N}$ is a partial recursive function such that if $\phi(p)$ exists and $\phi(q)$ exists, then p is not a proper prefix of q (Li and Vitányi 1997, 192).

Recall our enumeration of all partial recursive functions $\phi_1, \phi_2, \ldots$, which clearly contains all partial recursive prefix functions. Let T be a Turing machine from our standard enumeration of Turing machines that computes a partial recursive function ϕ. If $\psi = \phi$ is a partial recursive prefix function, we define T' that computes ψ using T by the algorithm described below.

3.1.2 Definition. A *halting input*, for T' is an initial segment $x_1 x_2 \ldots x_k$ of a (potentially one-way infinite) binary string x such that T halts after reading x_k but before reading x_{k+1} (Li and Vitányi 1997, 192-193).

Algorithm:

① `SET` $p := \epsilon$.

② Dovetail all computations of T computing $\phi(pq)$, for $q \in \{0,1\}^*$.
`IF` $\phi(pq) < \infty$ is the first halting computation, `THEN GO TO` ③.

③ `IF` $q = \epsilon$, `THEN` output $\phi(p)$ and halt.
`ELSE SET` $x :=$ next input bit; `SET` $p := px$; `GO TO` ②.

By This construction, we have an effective enumeration of *prefix machines* $T'_1, T'_2, \ldots$ enumerating all, and only, the partial recursive prefix functions $\psi_1, \psi_2, \ldots$. This allows us to prove an invariance theorem for prefix complexity.

Recall from Definition 1.1.2 that a function is *universal* if it is additively optimal for a class of functions.

3.1.1 Theorem. *There exists a universal partial recursive prefix function ψ_0 such that for any partial recursive prefix function ψ, there is a constant c_ψ such that $C_{\psi_0}(x|y) \leq C_\psi(x|y) + c_\psi$, for all $x, y \in \mathbb{N}$ (Li and Vitányi 1997, 193).*

The proof is analogous to that of Theorem 1.2.1.

For each pair of universal partial recursive prefix functions ψ and ψ', $|C_\psi(x|y) - C_{\psi'}(x|y)| \leq c_{\psi,\psi'}$, for all $x, y \in \mathbb{N}$ and some constant $c_{\psi,\psi'}$. We fix one universal partial recursive prefix function ψ_0 as and universal prefix machine U such that $U(\langle y, \langle n, p\rangle\rangle) = T'_n(y, p)$ as reference and define the *prefix complexity* of x conditional to y as $K(x|y) = C_{\psi_0}(x|y)$ for all $x, y \in \mathbb{N}$ Analogously, $K(x) = K(x|\epsilon)$.

Remark. Put some exposition about the advantages and disadvantages of prefix complexity here.

3.2 Real-valued Functions

Recall Church's Thesis, that the class of algorithmically computable numerical functions coincides with the class of partial recursive functions. We consider the enumeration of partial recursive functions: $\phi_1, \phi_2, \ldots$ where ϕ_i is computed by Turing machine T_i in the canonical enumeration of Turing machines. Note that because there are many Turing machines that compute the same function, there are many $\phi_i = \phi_j$ where $i \neq j$. Frequently, when speaking in the context of this standard enumeration (such as in the diagonalization proof of the lemma below), we will refer to a partial recursive

function ψ. It is important not to confuse the *function* ψ with a name for ψ, where a name might be an algorithm that computes ψ, a Turing machine that implements the algorithm, or one of potentially countably infinitely many integers i where i is an index for ψ if $\psi = \phi_i$.

3.2.1 Lemma. *There is no total recursive function g such that for all x, y, we have $g(x, y) = 1$ if $\phi_x(y)$ is defined, and $g(x, y) = 0$ otherwise (Li and Vitányi 1997, 34).*

Proof. Suppose, by way of contradiction, that such a function g exists. Consider a partial recursive function ψ where $\psi(x) = 1$ if $g(x, x) = 0$ and is undefined otherwise. Let y be an index for ψ in our standard enumeration of partial recursive functions. Then, $\phi_y(y)$ is defined if and only if $g(y, y) = 0$, contrary to the given definition of g. □

Now, consider recursive functions of the form $g(\langle x, k\rangle) = \langle p, q\rangle$. We can write $g(x, k) = p/q$ and in this way interpret g as a rational-valued function, though it is in fact a proper recursive function over the integers. This provides a means of extending our definitions of *recursive* and *(recursively) enumerable* to real-valued functions.

3.2.1 Definition. A real-valued function f is *enumerable* if there exists a total recursive function $g(x, k)$, nondecreasing in k, with $f(x) = \lim_{k\to\infty} g(x, k)$. We say f is *co-enumerable* if $-f$ is enumerable. The real-valued function f is *recursive* if and only if there is a total recursive function $g(x, k)$ such that $|f(x) - g(x, k)| < 1/k$ (Li and Vitányi 1997, 35).

It may seem strange to use "enumerable" and "recursive" to describe real valued functions, and the definition for co-enumerable may even seem frivolous. Classically, the terms are used to describe countable sets that are recognizable or decidable by a Turing machine, and clearly, we could not enumerate the values of most real valued functions. However, consider that a function $f : \mathbb{N} \to \mathbb{R}$ is enumerable, as we have defined it, if the set $\{(x, r) : r \leq f(x), r \in \mathbb{Q}\}$ is enumerable in the classical sense.

The value of these definitions is that real-valued functions can be classified by their approximability by recursive functions over the natural numbers. An enumerable real-valued function can be "recursively enumerated from above," that is it can be approximated from one side, but possibly without knowing how precisely. True to the classical definition, real-valued functions that are enumerable and co-enumerable are recursive, and can be approximated (eventually) to any degree of precision. It may not be obvious that there are enumerable functions that are not recursive. As a trivial

example, consider the set $K_0 = \{\langle x, y\rangle : \phi_x(y) \leq \infty\}$ (known as the *halting set*). Clearly K is enumerable, but Lemma 3.2.1 could be restated as "The halting set is not recursive," and the same proof would suffice. The functions $C(x)$ and $K(x)$, it turns out, are co-enumerable functions but not recursive, so $-C(x)$ and $-K(x)$ would be non-trivial examples.

3.2.2 Definition. An enumerable function f is *universal* if there is an effective enumeration $f_1, f_2, \ldots$ of enumerable functions such that $f(\langle i, x\rangle) = f_i(x)$, for all $i, x \in \mathbb{N}$, or $f_i(x) = \infty$ if no maximum exists (Li and Vitányi 1997, 241).

3.2.2 Lemma. *There is a universal enumerable function (Li and Vitányi 1997, 241).*

Proof. Let $\phi_1, \phi_2, \ldots$ be the standard enumeration of partial recursive functions. Define for all i the function $f_i(x) = \max_{k \in \mathbb{N}} \{\phi_i(x, k)\}$.

(This proof is not terribly interesting. I'll come back to it.) □

The same argument holds for co-enumerable functions; however, there is no universal recursive function.

Define $f(x, y) = f(\langle x, y\rangle)$.

3.2.3 Definition. If $f(x, y) \geq C(x|y)$ for all x and y, we say $f(x, y)$ is a *majorant* of $C(x|y)$ (Li and Vitányi 1997, 241).

All co-enumerable majorants have the following property.

3.2.3 Lemma. *Let $f(x, y)$ be co-enumerable. For all x, y we have $C(x|y) \leq f(x, y) + O(1)$ if and only if $|\{x : f(x, y) \leq m\}| = O(2^m)$, for all y and m (Li and Vitányi 1997, 241).*

Proof. (This result is interesting, but not necessary for the Coding theorem. If time permits, I will explicate the proof.) □

3.3 Probability and Continuous Sample Spaces

In this and subsequent sections we extend some of the ideas of probability and complexity developed on the natural numbers to real valued functions. Recall Kolmogorov's Axioms of Probability:

1. If A and B are events, then so is the *intersection* $A \cap B$, the *union* $A \cup B$, and the *difference* $A - B$.

2. The *sample space* S is an event. We call S the *certain* event. The empty set $\varnothing$ is an event. We call $\varnothing$ the *impossible* event.

3. To each event E is assigned a non-negative real number $P(E)$ that we call the *probability* of event E.

4. $P(S) = 1$

5. If A and B are disjoint, then $P(A \cup B) = P(A) + P(B)$.

6. For a decreasing sequence $A_1 \supset A_2 \supset \cdots \supset A_n \supset \cdots$ of events with $\bigcap_n A_n = \varnothing$ we have $\lim_{n\to\infty} P(A_n) = 0$.

Remark. The appearance of Kolmogorov's name above is incidental. Andrei Kolmogorov was a prolific mathematician (and primarily a probabilist), and the above axioms are simply standard axioms of probability and not in some way specialized for complexity theory.

We want to apply the notion of probability for finite sample spaces, say the outcomes of sequences of fair coin tosses, to continuous sample spaces, say $S = \{0,1\}^\infty$. However, we have no proper definition for the probability of individual elements in S, since the likelihood of selecting an arbitrary element is necessarily 0 for all but finitely many elements. Thus, we are limited to defining probability for subsets of S. We begin by considering sets that are easily described and then can use union, intersection, complement, and countable union to define many more (though not all) subsets of S.

3.3.1 Definition. Using the binary expansion of real numbers ω on the half open interval $[0,1)$, a *cylinder* Γ_x is the set of all real numbers starting with $0.x$ where x is a finite binary string, that is $\Gamma_x = \{x\omega : x \in \{0,1\}^n, \omega \in \{0,1\}^\infty\}$. Where a real number has two binary expansions, such as $\frac{1}{2}$, which can be represented as $0.10000\ldots$ or $0.01111\ldots$, we use the representation with infinitely many zeros (Li and Vitányi 1997, 21).

There are countably many cylinders on a continuous interval. Each cylinder is an event. Closure of all events under pairwise union, intersection, and difference forms the set field $\mathcal{F}$. With probability distribution P, we have the probability field $(\mathcal{F}, P)$. Analogously, we can have probability *measure* $(\mathcal{F}, \mu)$. We denote the uniform distribution (*Lebesgue measure*) by λ where $\lambda(\Gamma_y) = 2^{-l(y)}$. An infinite probability field closed under all countable unions $\bigcup A_n$ of disjoint events A_n we call a *Borel* field.

Here we will introduce a slightly different notation than that classically used in measure theory. We wish to develop the ideas over the set of infinite binary sequences, rather than decimal expansion of real numbers. With basis $\mathcal{B} = \{0, 1\}$, we have $\mathcal{B}^*$ and $\mathcal{B}^\infty$ analogous to $\mathbb{N}$ and $\mathbb{R}$, respectively. A cylinder set $\Gamma_x \subseteq S$ is defined by $\Gamma_x = \{\omega : \omega_{1:l(x)} = x, x \in \mathcal{B}\}$. Let $\mathcal{G} = \{\Gamma_x : x \in \mathcal{B}^*\}$ be the set of all cylinders in S.

3.3.2 Definition. A function $\mu : \mathcal{G} \to \mathbb{R}$ defines a *probability measure* if

1. $\mu(\Gamma_\epsilon) = 1$
2. $\mu(\Gamma_x) = \sum_{b \in \mathcal{B}} \mu(\Gamma_{xb})$.

(Li and Vitányi 1997, 243-244)

Conventionally, we abusively let $\mu(x)$ denote $\mu(\Gamma_x)$.

3.3.3 Definition. A *semimeasure* μ is a defective measure with

1. $\mu(\epsilon) \leq 1$
2. $\mu(x) \geq \sum_{b \in \mathcal{B}} \mu(xb)$.

(Li and Vitányi 1997, 244)

3.3.4 Definition. We say a measure (or semimeasure) μ is recursive (enumerable) if and only if the function $f : \mathbb{N} \to \mathbb{R}$ given by $f(x) = \mu(\Gamma_x)$ is recursive (enumerable). (Li and Vitányi 1997, 245)

We can rectify a semimeasure μ to a proper measure ρ by adding an element $u \notin \mathcal{B}$ called the *undefined* element by concentrating the surplus probability on u by

1. $\rho(\epsilon) = 1$
2. $\rho(xu) = \rho(x) - \sum_{b \in \mathcal{B}} \rho(xb)$.

Thus for all $x \in \mathcal{B}^*$, we have $\rho(x) = \mu(x)$.

3.4 The Universal Discrete Semimeasure

3.4.1 Definition. A *discrete semimeasure* is an injective function $P : \mathbb{N} \hookrightarrow \mathbb{R}$ that satisfies $\sum_{x\in\mathbb{N}} P(x) \leq 1$. It is a *probability measure* if equality holds.

Practically speaking, there is no difference between a discrete measure and a normal probability measure over the sample space $\mathbb{N}$ (hence, the use of capital italics to denote both). The distinction lies purely in the theoretical interpretation of the domain. Using the language from measure theory, we can say the basic set $\mathcal{B} = \mathbb{N}$ and with sample space $S = \mathbb{N} = \{\Gamma_x : x \in \mathbb{N}^+, l(x) = 1\}$. The set of cylinders in the correspondence may seem odd, since there are not many integers of length $l(x) = 1$, but recall that l is a function defined on strings, not numbers, and here we are not using the binary encoding of x as we have in the past. We are encoding the integers with an alphabet with countably infinitely many symbols, thus every integer is encoded by a single distinct character. Consequently, no string representation of an integer is the prefix of another, and the second condition of our definition for probability measures, that $\mu(x) = \sum_{b\in\mathcal{B}} \mu(xb)$, has no meaningful interpretation for us, though the first condition is satisfied.

Measure theory primarily concerns measures defined on Borel sets, but henceforth we will be working merely on cylinder sets, and moreover, cylinders that are necessarily pairwise disjoint. The discrete sample space, in some sense, might be thought of as a first order approximation of the continuous domain. While not necessarily intuitive, the class of (enumerable) discrete semimeasures has an important feature: a universal element, as we will show later.

It is instructive to consider the discrete Lebesgue measure in contrast to the continuous Lebesque measure λ. With basic set $\mathcal{B} = \mathbb{N}$, we have the function $L : \mathbb{N} \to \mathbb{R}$ given by $L(x) = 2^{-2l(x)-1}$. We can verify that L is a probability measure as follows:

$$\sum_{x\in\mathbb{N}} L(x) = \sum_{n\in\mathbb{N}} 2^{-n-1} \sum_{l(x)=n} 2^{-l(x)} = \sum_{n\in\mathbb{N}} 2^{-n-1} = 1.$$

It remains that no element of the basic set is reused, so it isn't particularly useful to think of $l(x)$ as the number of symbols in x, but simply a place holder function.

The continuous Lebesgue measure λ is defined on basic set $\{0, 1\}$ with $\lambda : \{0, 1\}^* \to \mathbb{R}$, where the element $x \in \{0, 1\}^*$ with the interpretation that x can be a prefix of another element, or vise versa. Conversely, the discrete Lebesgue measure L has $\mathbb{N}$ for both the basic set and the domain, thus $x \in \mathbb{N}$, and no element can be a prefix of another.

3.4.2 Example. By our standard integer-string correspondence, $1, 5, 6 \in \mathbb{N}$ map to $0, 10, 11 \in \{0,1\}^*$. Thus, $L(1) = \frac{1}{4} > L(5) + L(6) = \frac{1}{16} + \frac{1}{16}$. In terms of cylinder sets, Γ_1, Γ_5, and Γ_6 are pairwise disjoint. However, for the continuous Lebesgue measure, we have $\lambda(1) = \lambda(10) + \lambda(11)$, and consistently, $\Gamma_1 = \Gamma_5 \cup \Gamma_6$ (Li and Vitányi 1997, 246).

A key point here is that the sum over all events in the continuous domain diverges while the sum over all events in the discrete domain converges to 1. That is, for continuous measure λ, we have for each n that $\bigcup_{l(x)=n} \Gamma_x = S$, thus $\sum_{l(x)=n} \Gamma_x = 1$ for each n, and $\sum_{x\in\{0,1\}^*} \lambda(x) = \infty$, while for discrete measure L, we have $\bigcup_{l(x)=n} \Gamma_x \subset S$ and $\sum_{l(x)=n} L(x) = 2^{-n-1}$ for each n, and $\sum_{x\in\mathbb{N}} L(x) = 1$.

3.4.3 Definition. Let $\mathcal{M}$ be a class of discrete semimeasures. A semimeasure P_0 is *universal* (or maximal) for $\mathcal{M}$ if $P_0 \in \mathcal{M}$, and for all $P \in \mathcal{M}$, there exists a constant c_p such that for all $x \in \mathbb{N}$, we have $c_p P_0(x) \geq P(x)$, where c_p may depend on P but not on x (Li and Vitányi 1997, 246).

We say that P_0 *multiplicatively dominates* each $P \in \mathcal{M}$. Clearly, there can be no universal semimeasure that dominates all semimeasures, but in fact, even the class of total recursive semimeasures has no universal element (Li and Vitányi 1997, 249). However, the class of *enumerable* discrete semimeasures does have a universal element.

3.4.1 Theorem. *There exists a universal enumerable discrete semi-measure. We denote it by* **m** *(Li and Vitányi 1997, 247).*

Proof. The proof occurs in two stages. We first demonstrate an effective enumeration of the class of enumerable discrete semimeasures. This is achieved by taking the enumeration of all real-valued partial recursive functions and converting them to discrete semimeasures. In the second stage of the proof, we construct an enumerable discrete semimeasure P_0 and show it to be universal.

STAGE 1: Consider an effective enumeration of all real-valued partial recursive functions $\psi_i, \psi_2, \ldots$.Without loss of generality, we can assume that each function ψ is approximated by a rational-valued two-argument function $\phi'(x,k) = p/q$. (Formally speaking, ϕ' is actually a single-argument function where $\phi(\langle x,k\rangle) = \langle p,q\rangle$, but it is useful to interpret it otherwise.) Without loss of generality, we can modify each ϕ' to a rational-valued two-argument partial recursive function by the following criteria. For all $x \in \mathbb{N}$ and $k > 0$,

- if $\phi(x,k) < \infty$, then $\phi(x,1), \phi(x,2), \ldots, \phi(x,k-1) < \infty$ (this property is achieved by dovetailing the computation of $\phi(x,1)'$, $\phi(x,2)', \ldots$ and assigning the computer values in enumeration order to $\phi(x,1)$, $\phi(x,2), \ldots$);
- $\phi(x,k+1) \geq \phi(x,k)$ (this property is achieved by essentially the same dove-tailing strategy used above);
- $\lim_{k\to\infty} \phi(x,k) = \lim_{k\to\infty} \phi'(x,k) = \psi(x)$.

The sequence of ψ given by the list of the approximators enumerates all enumerable real-valued partial recursive functions, and each approximating function ϕ defines a discrete semimeasure P via the algorithm below. Note that $P(x)$ in the algorithm is an array that stores the interim approximations of P during the computation, the nonzero part of which is always finite.

Algorithm:

① `SET` $P(x) := 0$ for all $x \in \mathbb{N}$; and `SET` $k := 0$.

② `SET` $k := k+1$, and compute $\phi(1,k), \ldots \phi(k,k)$. (If any $\phi(i,k)$ is undefined for $1 \leq i \leq k$, then P will not change any further and is trivially a discrete semimeasure.)

③ `IF` $\phi(1,k) + \cdots + \phi(k,k) \leq 1$ `THEN SET` $P(i) := \phi(i,k)$ for all $i = 1, 2 \ldots, k$, `ELSE` terminate. (This step guarantees that P satisfies the discrete semimeasure requirements.)

④ `GO TO` ②.

In the case that ψ is already a discrete semimeasure, then $P = \psi$ and the algorithm never terminates, but approximates P from below. If some x and k are encountered such that $x \leq k$ and $\phi(x,k)$ is undefined, then the last assigned values of P remain fixed though the computation runs forever. Because of the condition set by ③, P is a semimeasure. If the condition in ③ is violated, computation terminates and the approximation is a total recursive semimeasure.

Hence, by running the algorithm on the list $\phi_1, \phi_2, \ldots$, we have an effective enumeration $P_1, P_2, \ldots$ of all, and only, the enumerable discrete semimeasures.

STAGE 2: We define the partial recursive function P_0 as follows:

$$P_0(x) = \sum_{n\geq 1} \alpha(n) P_n(x),$$

where $\alpha(n) > 0$ and $\sum_n \alpha(n) \leq 1$ for all n. As such, P_0 satisfies the conditions for a discrete semimeasure, since

$$\sum_{x \geq 0} P_0(x) = \sum_{n \geq 1} \alpha(n) \sum_{x \geq 0} P_n(x) \leq \sum_{n \geq 1} \alpha(n) \leq 1.$$

The function P_0 is enumerable as we can approximate it by the universal partial recursive function ϕ_0 using the same construction as was used in STAGE 1, since the $P_n(x)$ are enumerable in n and x. Clearly, P_0 dominates each P_n since $P_0(x) \geq \alpha(x)P_n(x)$. Thus, there are in fact infinitely many universal enumerable semimeasures. We fix a reference universal enumerable discrete semimeasure and denote it by $\mathbf{m}$.

□

3.5 A Priori and Algorithmic Probabilities

3.5.1 A Priori Probability

Let $P_1, P_2, \ldots$ be the effective enumeration of all enumerable semimeasures constructed in the proof of theorem 3.4.1. We consider an alternate enumeration, as follows. We let a prefix machine T accept as input an infinitely long sequence of coin flips. The probability of generating an initial segment p is $2^{-l(p)}$. Thus if $T(p)$ halts, T will halt upon reading only the first $l(p)$ bits of input, since it is a prefix machine. We let $T_1, T_2, \ldots$ be the standard enumeration of prefix Turing machines.

For each prefix machine T in our canonical enumeration of prefix machines, the probability that T computes x on input provided by successive coin flips is

$$Q_T(x) = \sum_{T(p)=x} 2^{-l(p)}. \tag{3.1}$$

Note that $\sum_{x \in \mathbb{N}} Q_T(x) \leq 1$ where equality holds for T if every one-way infinite sequence contains an initial segment for which T halts. Thus $Q_T(x)$ is a discrete semimeasure, and a probability measure if equality holds. We can approximate $Q_T(x)$ by the following algorithm. Note that $Q(x)$ is a local variable used to store the current approximation of $Q_t(x)$.

Algorithm:

① SET $Q(x) := 0$ for all x.

② Dovetail computations of all programs on T such that at stage k step $k - j$ of program j is executed. `IF` the computation of some program p halts with $T(p) = x$, `THEN GO TO` ③.

③ `SET` $Q(x) := Q(x) + 2^{-l(p)}$ and `GO TO` ②.

The variable $Q(x)$ of our algorithm approximates Q_T as given in Equation (4.3) above for each x. Thus, $Q_T(x)$ is enumerable. By the canonical enumeration of prefix machines, our construction gives us an effective enumeration of only enumerable semimeasures $Q_1, Q_2, \ldots$. Hence, the P-enumeration given in the previous theorem lists all elements enumerated by the Q-enumeration. It has been shown that the Q-enumeration contains all of the elements in the P-enumeration, that is, all enumerable measures (Li and Vitányi 1997, 253), but we assume this result without proving it, as the proof is not necessary for understanding the Coding theorem.

3.5.1 Definition. The *universal a priori probability* on the positive integers is defined as

$$Q_U(x) = \sum_{U(p)=x} 2^{-l(p)},$$

where U is a universal prefix machine (Li and Vitányi 1997, 252).

It is this definition that necessitates the use of prefix complexity rather than plain Kolmogorov complexity in this discussion. The series $\sum_p 2^{-l(p)}$ converges (to ≤ 1) if the summation is taken over all halting programs p of any fixed prefix machine, but diverges if taken over all halting programs p of an universal plain Turing machine.

3.5.2 Algorithmic Probability

The whole of Kolmogorov complexity is premised on the idea that the complexity of an object is a function of how briefly it can be described. Clearly the brevity of the description depends upon the methods we allow. We require that a description be sufficient to completely reproduce the object, thus we quantify the the shortest self-delimiting description of an object x by $K(x)$. This intuitively leads to the assertion that one object being simpler than another is the same as having a higher probability of occurrence. Hence, algorithmic probability can be thought of as a mathematical formalization of Occam's Razor.

3.5.2 Definition. The *algorithmic complexity* $R(x)$ of x is defined as

$$R(x) = 2^{-K(x)}.$$

(Li and Vitányi 1997, 252)

Here we make some simple observations about algorithmic complexity. Of objects of length n, the simplest object is the string of n zeros. It can be shown that $K(0^n) \leq \log n + 2 \log \log n + c$, where c is a constant independent of n. Thus, for all x with $l(x) \geq n$, we have

$$R(x) \geq \frac{1}{cn \log^2 n}.$$

Conversely, we have for almost all binary sequences y generated by n consecutive coin tosses, $K(y) \geq n$ and $R(y) \leq 1/2^n$.

3.6 Proof of the Coding Theorem

3.6.1 The Coding Theorem. *There is a constant c such that for every x,*

$$-\log \mathbf{m}(x) = -\log Q_u(x) = K(x),$$

with equality up to the additive constant c (Li and Vitányi 1997, 253).

Proof. □

Appendix A

Applications of Kolmogorov Complexity to Algorithms and Computability

Claim: A 1 head 2-way DFA cannot accept $L = \{w\#w | w \in \{0,1\}^*\}$.

Proof. Suppose, by way of contradiction, that our two-way DFA accepts L. Set $x = (01)^n$ with $C(n) \geq l(n) = \log\log x + O(1)$. When run on $x\#x$, after $m \geq 0$ complete two-way passes, A enters state q while positioned at the start of the input string. With no further changes in direction, $\delta'(x\#x, q) = q_f$. Let $\delta'(x\#, q) = q'$. The length of a description of A initialized to q' is bounded by some constant c, but gives a constant upper bound for describing n, which is a contradiction. Thus, our DFA cannot accept L. □

Claim: The average case run-time for binary search has a lower bound in $\Omega(\log n)$.

Proof. Suppose we were performing a binary search for string s in a lexicographically sorted one-dimensional array A of n keys where $s = A[i]$ and $C(i) \geq \log n + O(1)$ (which we know to be true of *many* $i \leq n$ for large n). Without loss of generality, we say $s, A[i] \in \{0,1\}^*$. (We will use the notation $s < t$ to indicate that s precedes t lexicographically, and likewise, $s > t$ to indicate s follows t.) The binary search is implemented as follows. Set string $p := \varepsilon$. We initially compare s to $A[\lfloor n/2 \rfloor]$. If $s < A[\lfloor n/2 \rfloor]$, we set $p := p0$ and perform a binary search for s in $A[1 : \lfloor n/2 \rfloor]$. If $s > A[\lfloor n/2 \rfloor]$, we set $p := p1$ and perform a binary search for s in $A[\lceil n/2 \rceil : n]$. We

return i for $s = A[i]$ and 0 if s is not found. Thus p is a record of the comparison results from our search. Using some canonical encoding of Turing machines, we have the enumeration of a machine M which reads input string p as a decision record, constructs a balanced binary search tree with depth $l(p) + 1$, and returns the index of the node reached by p's decision path. Thus, $l(\langle M \rangle, p) \geq C(i) \geq \log n + O(1)$. Since $l(\langle M \rangle) = O(1)$, we have that $l(p) \geq \log n + O(1)$. □

Bibliography

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Modern Phys.*, 74(1):47–97, 2002. ISSN 0034-6861. Formative paper on small world graphs and networks. It seems probably that Kolmogorov complexity should be useful for proving some bounds on small world graphs, but we haven't yet found a way to approach it. Knowledge of them comes more in the form of statistical observations much more than theorems and proven properties.

Béla Bollobás. *Modern graph theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1998. ISBN 0-387-98488-7. Not really anything on graph complexity, but is the source of graph theory notation I use.

Harry Buhrman, Jaap-Henk Hoepman, and Paul Vitányi. Space-efficient routing tables for almost all networks and the incompressibility method. *SIAM J. Comput.*, 28(4):1414–1432 (electronic), 1999a. ISSN 0097-5397. Interesting article found while looking at small-world graphs. Connection to my research seems tenuous, and understanding the article would likely require more background study than I have time to invest.

Harry Buhrman, Ming Li, John Tromp, and Paul Vitányi. Kolmogorov random graphs and the incompressibility method. *SIAM J. Comput.*, 29(2): 590–599 (electronic), 1999b. ISSN 1095-7111. Contains essentially the same material as is covered in Section 6.4 of the Kolmogorov text, as well as the original proof of Theorem 2.2.1.

Qi Cheng and Fang Fang. Kolmogorov random graphs only have trivial stable colorings. *Inform. Process. Lett.*, 81(3):133–136, 2002. ISSN 0020-0190. Haven't really digested it, but it is primarily concerned with labeled graphs.

Bruno Durand and Sylvain Porrot. Comparison between the complexity of a function and the complexity of its graph. *Theoret. Comput. Sci.*, 271(1-2): 37–46, 2002. ISSN 0304-3975. Deals with graph complexity, but the ideas are pretty far removed from the work I have been doing.

Peter Eades, Charles Stirk, and Sue Whitesides. The techniques of Kolmogorov and Barzdin for three-dimensional orthogonal graph drawings. *Inform. Process. Lett.*, 60(2):97–103, 1996. ISSN 0020-0190. Other research done by Kolmogorov. Not directly related to graph complexity or Kolmogorov complexity.

Paul Erdös and Joel Spencer. *Probabilistic Methods in Combinatorics*. Probability and Mathematical Statistics. Akadémiai Kiadó, Budapest, 1974. ISBN 0-12-240960-4. This book provided some useful examples of the probabilistic method.

Junichi Fujii. Entropy of graphs. *Math. Japon.*, 38(1):39–46, 1993. ISSN 0025-5513. Necessary background for mfujii96.

Junichi Fujii and Yūki Seo. Graphs and tensor products of operators. *Math. Japon.*, 41(2):245–252, 1995. ISSN 0025-5513. Necessary background for mfujii96.

Junichi Fujii, Masatoshi Fujii, Hiromitsu Sasaoka, and Yasuo Watatani. The spectrum of an infinite directed graph. *Math. Japon.*, 36(4):607–625, 1991. ISSN 0025-5513. Necessary background for mfujii96.

Masatoshi Fujii, Masahiro Nakamura, Yuki Seo, and Yasuo Watatani. Graphs and Kolmogorov's complexity. *Math. Japon.*, 44(1):113–117, 1996. ISSN 0025-5513. Despite the promising title, this article proved to be a colossal waste of time. The article is poorly written, with a combination of grammatical nonsense, undefined notation, and vaguely referenced citations. Ultimately, we concluded that whatever the authors are calling 'Kolmogorov complexity' has little or nothing to do with what the rest of the world uses the term to refer to. Academic detritus.

W. W. Kirchherr. Kolmogorov complexity and random graphs. *Inform. Process. Lett.*, 41(3):125–130, 1992. ISSN 0020-0190. Need to revisit this article. As I recall, it builds upon ideas of (labeled) Kolmogorov random graphs as developed in buhrman99.

Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on internet-like graphs, 2003. URL

`http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/0308288`. Article that seems promising if we return to studying small world graphs.

Hoàng-Oanh Le and Van Bang Le. The NP-completeness of $(1, r)$-subcolorability of cubic graphs. *Inform. Process. Lett.*, 81(3):157–162, 2002. ISSN 0020-0190. Showed up in my key word searches, but does not seem to be related to my research.

Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997. This book is the foundational material of my research.

Ming Li, John Tromp, and Paul Vitányi. Sharpening Occam's razor. *Inform. Process. Lett.*, 85(5):267–274, 2003. ISSN 0020-0190. More or less redundant with some of the material from the Kolmogorov textbook. Not obviously relevant to graph complexity.

B. Litow and N. Deo. Graph compression and the zeros of polynomials. *Inform. Process. Lett.*, 92(1):39–44, 2004. ISSN 0020-0190. A very different approach to graph compression that I don't really understand.

Akemi Matsumoto and Yuki Seo. Graphs and Fibonacci numbers. *Math. Japon.*, 44(2):317–322, 1996. ISSN 0025-5513. Necessary background for mfujii96.

Masahiro Nakamura and Yasuo Watatani. An extension of the Perron-Frobenius theorem. *Math. Japon.*, 35(3):569–572, 1990. ISSN 0025-5513. Necessary background for mfujii96.

N. Rashevsky. Life, information theory, and topology. *Bull. Math. Biophys.*, 17:229–235, 1955. While this article was written well before the notion of Kolmogorov complexity was developed, and the connection seems strained at best, it does pertain to the motivations behind my research. The authors develop an idea of information content based on graph topological properties. If there is time, I hope to revisit this article.

Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, second edition, 2006. ISBN 0-534-95097-3. Primary reference for relevant theorems and definitions from computability theory.

R. J. Solomonoff. A formal theory of inductive inference. I. *Information and Control*, 7:1–22, 1964. ISSN 0890-5401. Original article on inductive inference. Not clearly written, and the material has been better developed in the Kolmogorov complexity textbook.

Joel Spencer. *Ten Lectures on the Probabilistic Method*, volume 64 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, 1994. ISBN 0-89871-325-0. This book provides a good introduction to the probabilistic method and some basic theorems.

Russell K. Standish. On complexity and emergence. *Complex. Int.*, 9:6 pp. (electronic), 2001/02. ISSN 1320-0682.

Russell K Standish. Complexity of networks, 2005. URL `http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0508075`. This article will be the jumping off point for work next semester, and has been the motivation for the past several weeks worth of work attempting to understand the coding theory. Deals with Kolmogorov complexity of unlabeled graphs. Still a fair amount of work before I really understand the article (which is not actually published in a journal yet).

Ernesto Trucco. A note on the information content of graphs. *Bull. Math. Biophys.*, 18:129–135, 1956. Builds on the ideas developed in rashevsky55.

Duncan J. Watts. *Small worlds: The dynamics of networks between order and randomness*. Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, 1999. ISBN 0-691-00541-9. Beginning chapters of the book is largely redundant with Albert's paper, but presented in a more narrative format. Somewhat easier to digest.

Douglas B. West. *Combinatorial Mathematics*. University of Illinois Mathematics Department, Chicago, preliminary (abridged) edition, 2006. The chapter from this book on the probabilistic method provided some useful theorems.