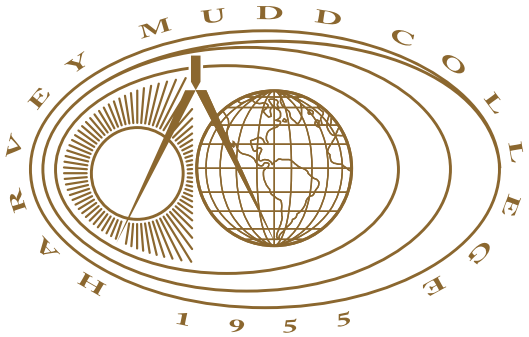

Applications of Kolmogorov Complexity to Graphs

John Hearn

Ran Libeskind-Hadas, Advisor

Michael Orrison, Reader

December 12, 2005

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 possible applications of this measure to graph theory.

Contents

List of Figures

Acknowledgments

To be filled in later.

Note to readers/graders: Writing this mid-year report posed some difficult problems. Much of what I have written about (most of the first two chapters of the report) is based on work that was done before the semester began. As I understand, this work will not be considered when grading my performance for the semester. This poses a problem, as all this work needs pretty serious treatment as it is necessary background material for all the work I have done since—more serious than it received when I initially approached it. As a result, considerable work has gone into presenting it here.

The first chapter contains no original work. It is based on a paper that I wrote for Math 104 last semester based of two weeks of research, which amounted to me trying to learn as much Kolmogorov complexity as I could. The paper was primarily my paraphrasing of material from a textbook. I have since taken a class on Kolmogorov complexity from the author of the textbook, and spent much of this semester learning concepts that were not covered in the class. Needless to say, my understanding is now much deeper, and I was able to rework the original paper into something more useful for understanding Kolmogov complexity, not only by correcting the many errors, but also treating some underlying concepts that previously got short shrift, but are necessary to subsequent work. Hopefully my remarks throughout the chapter will give some idea of what work is new and what is reused.

The second chapter is a mixture of paraphrasing of necessary background material and original work I did over the summer while taking the course on Kolmogorov complexity. My understanding was much deeper when I wrote it, and thus it required somewhat less reworking than the first chapter.

The third chapter, to the extent that I was able to finish it, is paraphrasing of material, mostly from the textbook, that Prof. Ran and I have been trying to digest as a prerequisite for a paper we want to study deeply in the spring.

Unfortunately, much of the work we have done this semester has just been reading chapters of the book and various articles until I had a depth of understanding that allowed us to make an educated guess as to whether the material would prove useful to further research. In many cases, we came to the conclusion that it would be better to move on to a new topic. Most of these efforts are not reflected in this paper, beyond possibly through the deeper grasp of Kolmogorov complexity I have gained in the process. I

hope this will be considered when evaluating my work for the semester.

Regards,
John Hearn

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? This often proves to be a challenging problem, but 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 20 words." If n exists, we have just described it in 13, 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 such a description. 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 tends to

reduce 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 would likely 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. The following and all other labeled definitions and theorems in this chapter on introductory material are taken directly from Li and Vitányi's text, except for changes of wording made for clarity. All my notation, with the exception of set cardinality, is consistent with theirs. Other remarks and explanations are my paraphrasing of material covered in Chapters 1 and 2. Items listed as "Example" include problem statements taken from the text, but the solutions are my own.

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 example 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 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 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 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 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 $\mathbb{C}$ be a subclass of the partial functions over $\mathbb{N}^+$. A function f is *universal* (or additively optimal) for $\mathbb{C}$ if it belongs to $\mathbb{C}$ and if for every function $g \in \mathbb{C}$ 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. The definition for a class of two-variable functions is had by replacing occurrences of x above with $\langle x, y \rangle$.

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.*

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

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

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.*

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*

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

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 $C(x|y) \leq l(|Y|) + c$.*

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 program 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 $U(\bar{k}m) = \pi(m)$, 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 $C(x) \geq l(x) - c$.

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 $C(x|y) \geq \log m - c$.*

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 $C(v) \leq l(v) - O(\log n)$ but cannot prove $C(v) \leq l(v) - O(1)$. 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, 112).

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.

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

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

Early Results

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: 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.

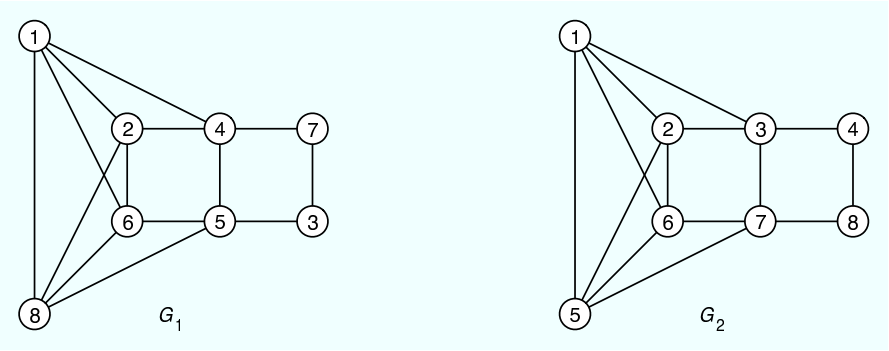

Figure 2.1: *Isomorphic graphs.*

The respective labelings give us $E(G_1) = 1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset1\emptyset = (1\emptyset)^{14}$ and $E(G_2) = 11\emptyset11\emptyset\emptyset1\emptyset11\emptyset\emptyset1\emptyset\emptyset1\emptyset\emptyset\emptyset\emptyset111\emptyset1\emptyset1$. (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. Hence we would like to have the complexity of Γ, which we will abusively denote by $C(\Gamma)$, less than or equal to $C(E(G_1)) + O(1)$. 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.

Admittedly, our chosen example is contrived. Most graph will be far less compressible.

2.1 Complexity of Labeled Graphs

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 $1 - 1/2^{\delta(n)-1}$.*

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 (Li and Vitányi 1997, 389). ❑

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

From this, we can use the incompressibility method to prove the following upper bound on the randomness deficiency of *most* unlabeled graphs.

2.2 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.
2.2.1 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.

2.2.1 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 Theorem 3:

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

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:

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

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.

2.2.1 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. The remainder of the material in this chapter will likely not prove useful in further research. I am retaining primarily because it is revealing of the motivations behind the research.

Now that we have proven that all unlabeled graphs *can* be compressed, it would be gratifying to find a practical strategy for actually doing so. We will, for now, leave the subject of Kolmogorov complexity. The compression methods based on enumeration we used as a basis for our proofs are space efficient but have some obvious weaknesses. Most prominently, they are not feasible under most reasonable time constraints. Secondly, enumeration is not a particularly intuitive representation of a graph. Ideally,

regularities in our graph would correlate to regularities in our standard encoding. With this purpose in mind, we will slightly modify our encoding method.

Recall graph G_1 from FIGURE 1. If we consider the adjacency matrix $A(G_1)$, we can find the string $E(G_1)$ by reading off the elements of the matrix 'above' the diagonal in row-major order.

$$A(G_1) = \begin{bmatrix} \emptyset & 1 & \emptyset & 1 & \emptyset & 1 & \emptyset & 1 \\ 1 & \emptyset & \emptyset & 1 & \emptyset & 1 & \emptyset & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset & 1 & \emptyset & 1 & \emptyset \\ 1 & 1 & \emptyset & \emptyset & 1 & \emptyset & 1 & \emptyset \\ \emptyset & \emptyset & 1 & 1 & \emptyset & 1 & \emptyset & 1 \\ 1 & 1 & \emptyset & \emptyset & 1 & \emptyset & \emptyset & 1 \\ \emptyset & \emptyset & 1 & 1 & \emptyset & \emptyset & \emptyset & \emptyset \\ 1 & 1 & \emptyset & \emptyset & 1 & 1 & \emptyset & \emptyset \end{bmatrix}$$

We can just as well characterize the graph by taking the elements 'below' the diagonal. We need only append one bit to the end of our string to identify the encoding scheme. We will denote the strings $E_\emptyset(G_1)$ and $E_1(G_1)$, where $E_\emptyset(G_1) = E(G_1) + \emptyset$ and $E_1(G_1) = E(G_1) + 1$.

$A(G_1)$ yields $E_\emptyset(G_1) = (1\emptyset)^{14}\emptyset$ and $E_1(G_1) = 1\emptyset\emptyset11\emptyset\emptyset\emptyset1111\emptyset\emptyset1\emptyset\emptyset11\emptyset\emptyset11\emptyset\emptyset11\emptyset1$. In this example, the new encoding method provides no clear benefit to the canonical one, and in fact increases both the string's length and complexity. However, we will see it offers us some conveniences. We can generalize our notion of graph complexity as follows.

2.2.1 Definition. Let Γ be an unlabeled graph on n vertices. We denote the complexity of Γ by

$$C(\Gamma) = C(\min(\{E_\emptyset(\Gamma_\pi) | 1 \leq \pi \leq n!\} \cup \{E_1(\Gamma_\pi) | 1 \leq \pi \leq n!\})) + O(1)$$

where the subscript π enumerates all permutations of the labels on the vertices of G, automorphisms inclusive.

One feature of E_1 is that we can find the encoding for the subgraph on vertices $\{0, 1, \ldots, m\}$ in the first $m(m-1)$ bits of the string. Likewise, we can add a vertex to our graph and by appending an additional n bits to the end of the string have a valid encoding for the new graph (always keeping the extra '1' at the end of the string). This property may or may not prove useful for actually finding a compression strategy, but is elegant nonetheless.

The more important property the encoding method gives us is the ability to encode either the largest clique or largest independent set as a monotonic substring at the beginning of our encoding. Regardless of the complexity of the rest of the string, having a long prefix of ones allows us some compression. Qualitatively speaking, having a high frequency of ones in the first half of the string and a high frequency of zeros in the second half makes our string 'less' random, and thus more compressible. With that in mind, we define the following strategy. We find the largest clique and largest independent set in our graph. Without loss of generality, we can assume that the maximal clique size is larger than the maximal independent set size. (We can always use an extra bit to indicate whether we are encoding the graph or its complement.) In the case of multiple cliques of maximal size, we select one maximally connected to vertices outside the clique. We label our clique of m vertices with $\{0, 1, \ldots, m-1\}$ with higher degreed vertices given lower labels. For vertices of equal degree, precedence is given to the vertices with neighbors more fully connected to the clique. Among these, precedence is given to the vertex whose neighbors are more fully connected to the clique vertices of highest degree. We sort the remaining $n - m$ vertices not in the clique in descending order by degree. From this list, we select the first vertex with a maximal number of connections to vertices already labeled and label it $m + 1$. We repeat this step until no vertices remain.

The unlabeled graph Γ of FIGURE 1 is not an ideal example for illustrating this strategy, since it has many automorphisms, but we use it for convenience. Where α is the label permutation given by our strategy, we get $E_1(\Gamma_\lambda) = 11111111\emptyset\emptyset\emptyset\emptyset111\emptyset\emptyset\emptyset\emptyset1\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset111$. Again, this is not very impressive when compared to the highly compressible $E_0(G_1)$, but does well when compared to most encodings of Γ. Of the first 15 (of 29) bits, 11 are ones while of the last 14 bits, only 4 are ones. In contrast, $E_1(G_1)$ has 8 of the first 15 and 7 of the last 14 bits as ones and $E_0(G_2)$ has 8 in the first 15 and 6 in the last 14. Comparatively, $E_1(\Gamma_\lambda)$ has a much less random distribution. Analysis for how much, on average, strings produced using this strategy can be compressed remains to be seen.

Though we now have a strategy to encode graphs as compressible strings, the resulting strings still fail to communicate graph complexity intuitively. Consider the graph G_3 in FIGURE 2. There is a clear regularity to the graph The subgraph on vertices 1, 2, 3, and 4 is isomorphic to the subgraph on 5, 6, 7, and 8. While G_1 has some regularities and the same number of vertices as G_3, it has 4 more edges, is non-planar and is not visually broken easily

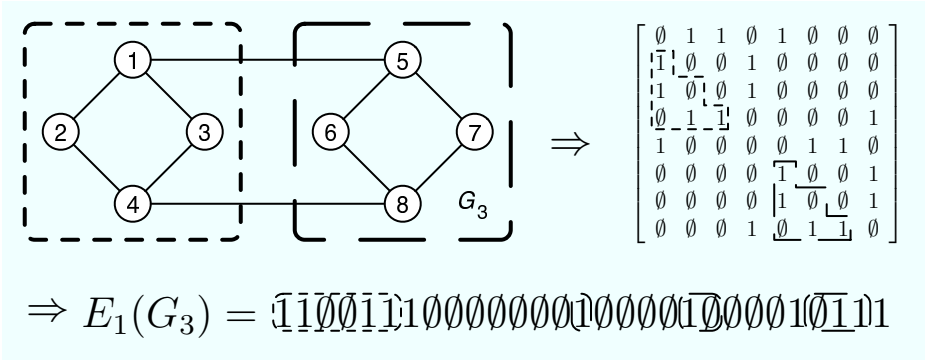

Figure 2.2: *Graph regularity does not necessarily correlate to string regularity.*

into a set of isomorphic subgraphs. As far as our intuition is concerned, G_1 is more 'complex' than G_3, yet (presumably) G_1 has the lower Kolmogorov complexity. Apparently a graph with 'high' complexity can be encoded by a string with 'low' complexity. We also see in the figure that regularities we see in the graph do not necessarily produce regularities in the encoded string, even though the pattern is still visible in the adjacency matrix from which the string was taken. Really, this is only a specific instance of a more general problem: how to represent spatial relationships in a single dimension. In fact, our problem is not that the regularities in the graph are not present in the string. They are present, but are not easily recognized in a one dimensional string. The substring that produces the first subgraph is repeated, but the second instance is not contiguous, so actually our problem is that the information necessary to describe where and how to look for the regularity costs more than we would save by using that information. We would like, then, an encoding method that creates some compromise between simplicity of implementation and preservation of some topological properties. One possibility, which also draws inspiration from the adjacency matrix, is to take the positions 'between' the matrix's diagonals. As shown in FIGURE 3, concatenate the string of bits in the west quadrant, the southwest half-diagonal, and the bits of the east quadrant. It is not demonstrated here, but we could alternately encode a graph by taking the north quadrant, northeast half-diagonal, and south quadrant. We will refer to these encodings as $E_3(G)$ and $E_2(G)$, respectively. We would also append some constant number of bits to the resulting string to indicate the encoding method we are using, but for the time being we can ignore them. Graph G_3 from the previous figure lends itself unusually well to the E_3 en-

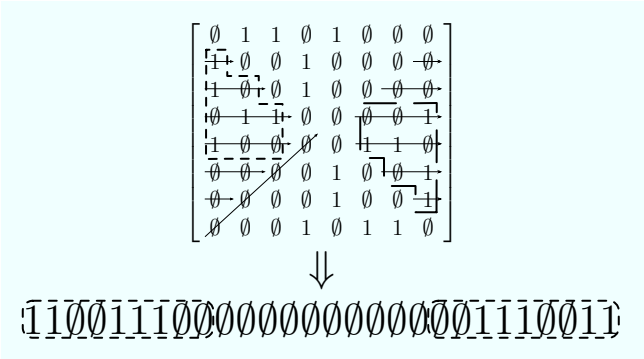

Figure 2.3: *Encoding method for end-cap preservation*.

coding scheme. The resulting string is a palindrome and contains a string of 14 zeros (half the length of the full string)—properties that suggest a low Kolmogorov Complexity. This is coincidental and most graphs will not produce such a clean result. However, we can alter our labeling strategy somewhat to produce an encoding mechanism that should both guarantee some compressibility and give strings which present, in an intuitive way, some significant topological properties of the graph.

As before, we identify the maximal sized cliques and independent sets, and without loss of generality, assume the cliques to be larger. (If there are no cliques or independent sets large relative to the graph, we will take an alternate approach. This is explained below.) If maximal clique has at least $\log n$ vertices, we do as follows.

(1) We select and label a clique by the same precedences as in the previous labeling strategy.

(2) We select a maximal sized independent set from the unlabeled vertices. We give first precedence in our selection to sets which share the fewest edges with labeled vertices. From the remaining sets, we select those with the lowest sum of degrees. From the remaining independent sets, $V_1, V_2, \ldots, V_k$, we let $U = \cup_{i=1}^{k} V_i$. From U, select a vertex v with maximal degree D which is not shared by all of the V_i, giving precedence to whichever vertex is shared by the fewest V_i. Delete v from U and eliminate any of the independent sets containing v. Repeat this step on the vertices in U until no more vertices can be deleted without eliminating all remaining independent sets. Do this for vertices with degree $D-1$, $D-2$, et cetera, until only one inde-

pendent set V_i remains. We reserve for these $|V_i| = m$ vertices labels $n-m, n-m+1, \ldots, n-1$. Repeat (1) on all unlabeled vertices not in V_i. Then sort V_i by degree in descending order and assign their labels, lowest to highest, with precedence given to vertices which share the most edges with labeled vertices.

(3) Repeat (1) and (2) until all vertices are labeled.

(4) Find the E_3 encoding of the graph.

Alternately, if there are no large cliques or independent sets, we use a simpler strategy. With the vertices sorted by degree in descending order, do as follows.

(1) Of the vertices with maximal degree, select those which share the most edges with labeled vertices. From those, select the vertices for which the sum of its neighbors' degrees are maximal. Then select based on the sum of degrees of vertices 2 edges away, 3 edges away, et cetera, until only one vertex remains. Label it with the lowest unused label.

(2) Using the same selection precedences, label the neighbors of the vertex selected in (1).

(3) Repeat (1) and (2) until all vertices are labeled.

(4) Find the E_2 encoding of the graph.

Both strategies, put simply, increase the ratio of ones to zeros near the beginning of the string and reduce the ratio towards the end. Presumably, there should be compressible substrings at the beginning and end of the graph encoding, allowing some compression. Depending on the graph and the labeling strategy used, our string should also tell us something about the graph. We know immediately that an E_2 encoded graph has no large cliques or independent sets, and by counting the ones in the first n bits, we know the maximum of the vertex degrees. Qualitatively, a high density of ones near the beginning of the string indicates multiple vertices of maximal or near maximal degree. A high density of zeros near the end of the encoding suggests the graph may not be connected, and can be used to quickly determine the minimum of the vertex degrees. An E_3 encoding gives different properties. For a leading string of k ones, we know that there is a maximal clique of size $m = \max\{x | x(x-1) \leq k, x \in \mathbb{N}\}$. Similarly, a trailing substring of zeros can be used to quickly find a lower bound on the size of the largest independent sets.

It seems likely that further investigation could reveal easy ways to determine some other properties of the the graph by visual inspection of the string generated by this encoding strategy. Some algorithmic analysis, possibly using Kolmogorov complexity, should be used to determine the running time of the algorithm. It should also be possible to determine an average case lower bound on the compressibility of such strings. It seems clear that the algorithm should produce strings by some constant amount, but it remains to be seen how much compressibility is sacrificed for a (hopefully) reasonable running time and a visually useful string encoding. We have seen examples demonstrating that our encoding methods will often produce strings far less compressible than ideal, but we also know those cases are quite rare. We would like to have an average case analysis showing that the strings are logarithmically compressible, as that would put us within a constant of the Kolmogorov complexity, though that may be difficult or impossible to prove. If nothing else, though, we have found a string encoding that reflects, to some extent, what we intuitively understand of graph complexity.

Chapter 3

Current Research: Measure Theory

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 is that now our standard enumeration of Turing machines enumerates *only* the machines whose encodings do not have as an initial proper substring the encoding of another halting machine. That is, if we have a Turing machine T_i that halts on some input, and can encode a Turing machine T_j (which also halts on some input) by appending bits to the encoding of T_i, then T_j would not be enumerated by standard enumeration. For a string x, where we denote the plain complexity by $C(x)$, we use $K(x)$ for the prefix complexity. For most theorems developed for plain complexity, analogous theorems hold for prefix complexity, the Invariance and Incompressibility theorems included. Some of the advantages of prefix complexity are that it is subadditive (where plain complexity $C(x,y) \leq C(x) + C(y)$ holds only to within an additive term logarithmic in x and y), all initial substrings of infinite random sequences are random (no infinite sequence satisfies this property under plain complexity), and classical Shannon information-theoretic identities have prefix complexity analogues satisfied up to an additive constant (where they can only be satisfied up to a logarithmic term otherwise).

Remark. Prefix complexity deserves a more thorough treatment than I have given here, but that will have to wait until I have a deeper and clearer understanding of the differences. Subsequent theory was developed using prefix complexity, but I am not sure it was necessary to have done so. Appearances of prefix complexity should generally be thought of as analogous to Kolmogorov complexity we have seen, but it was necessary here to introduce the notational difference.

3.2 Probability and Continuous Sample Spaces

The goal of this chapter is to lay the groundwork for proving the Coding theorem, a 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).

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.

3.2.1 Definition. Using the binary expansion of real numbers ω on the half open interval $[0, 1)$, *cylinder* Γ_x is the set of all real numbers starting with $0.x$

where x is a finite binary string. 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.

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 $\lambda(\Gamma_y) = 2^{-l(y)}$.
Remark. 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 our 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.
3.2.2 Definition. A function $\mu : \mathcal{G} \to \mathbb{R}$ defines a *probability measure* if

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

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

3.3 Real-valued Functions and Semi-measures

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. The provides us a means of extending our definitions of recursive and (recursively) enumerable to real-valued functions.
3.3.1 Definition. A real-valued function f is *enumerable* if there exists a 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* iff there is a recursive function $g(x, k)$ such that $|f(x) - g(x, k)| < 1/k$.

The value of this definition 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 approximated from one side, but without knowing to what precision. A recursive real-valued function can be approximated by any degree of precision. (Consequently, real-valued functions that are enumerable and co-enumerable are recursive.)

3.3.2 Definition. An enumerable function$f$ is *universal* if there is an effective enumeration $f_1, f_2, \ldots$ of enumerable functions such that $f(i,x) = f_i(x)$, for all $i, x \in \mathbb{N}$. (We define $f(i,x) = f(\langle i,x\rangle)$.)

Remark. Proofs of the following lemmas and theorem 3.3.1 will be included in future drafts, as will some examples and illustrations.

3.3.1 Lemma. *There is a universal enumerable function.*

3.3.2 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.*

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

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

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

3.3.4 Definition. A semimeasure μ is *enumerable* (*recursive*) if the function μ is enumerable (recursive).

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

3.3.6 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.

3.3.1 Theorem. *There exists a universal enumerable discrete semi-measure. We denote it by* **m**.

3.4 A Priori Probability and Algorithmic Probability

Let $P_1, P_2, \ldots$ be the effective enumeration of all enumerable semimeasures constructed in the proof of theorem 3.3.1. 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, the probability that T computes x on input provided by successive coin flips is

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

Note that $\sum_{x\in\mathbb{N}} Q_T(x) \leq 1$ where equality holds for T such that 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.

3.4.1 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.

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

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

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

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

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

3.5 The Coding Theorem

Remark. The theorem below, besides being remarkable in its own right, is fundamental to the results (Standish 2005, 2) that will be the starting point of next semester's research. Future drafts will include exposition on the proof of the theorem and some of the important consequences.

3.5.1 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.

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.

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 and digesting it has been virtually all the work I have done thus far.

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.

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.

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.