The Pennsylvania State University
The Graduate School
Eberly College of Science

DATA COMPRESSION AND FRACTAL DIMENSION FOR MEASURES

A Thesis in
Mathematics
by
Ryan D. Wasson

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Arts

December 2015

The thesis of Ryan D. Wasson was reviewed and approved* by the following:

Jan Reimann
Assistant Professor of Mathematics
Thesis Advisor

Stephen Simpson
Professor of Mathematics

Jason Rute
Postdoctoral Mathematics Research Associate

Svetlana Katok
Professor of Mathematics
Associate Head for Graduate Studies

*Signatures are on file in the Graduate School.

Abstract

The ability to distinguish between data generated by random versus deterministic processes is necessary for scientific discovery. Various tools for achieving this goal exist in several different branches of mathematics, namely geometry and mathematical logic. The amount of irregularity in a data set can be measured using tools from the field of fractal geometry, such as fractal dimension in all its forms. Likewise, the field of algorithmic randomness, built on the foundation of mathematical logic, measures randomness using notions of complexity such as Kolmogorov complexity, a non-computable theoretical limit on the amount of information contained in an object. In the last fifteen years, it has been shown that the ideas from both these fields are strongly related. In this thesis, we summarize the results from the literature detailing this connection, and we demonstrate new numerical approaches for approximating Hausdorff dimension and information dimension using data compression. The validity of using various compression techniques, such as Lempel-Ziv and I-complexity, to approximate non-computable Kolmogorov complexity is also explored.

Table of Contents

Acknowledgments

I would like to thank my adviser, Dr. Jan Reimann, for all of his help and support during this past year while working on this thesis. Thank you for your willingness to make time to meet with me, even when we were on opposite sides of the country via Skype, and thank you for your patience. Your enthusiasm for the subject was contagious!

I would also like to thank Dr. Stephen Simpson and Dr. Jason Rute for the logic courses they gave this past year which introduced me to mathematical logic and gave me a good background on the subject.

Finally, I would also like to thank my family and friends, specifically Anna, Adam and Matt, for their support during my time at Penn State.

Chapter 1
Introduction

Every day we encounter a wealth of information. On a physiological level, our eyes and ears are constantly sending signals to our brains about the light and sound around us. On a social level, we communicate with each other using language in all its forms, be it spoken, written, or body language. On a scientific level, we study patterns and make predictions and conjectures about how the world works.

The difficulty of interpreting all of this information depends on how structured it is. If it is highly organized and contains obvious patterns, then it is easy to draw conclusions about its meaning. But if it lacks structure and appears more or less "random," then it can be very hard to interpret. For the scientist, a fundamental question might be to determine whether there even are any meaningful patterns present in a particular object of study, and if so to discover their origin. This is made particularly difficult when working with chaotic systems which can often appear to exhibit random behavior. Such is the case, for instance, in scientific fields that deal with chaotic data sets such as meteorology.

Although information can sometimes appear random, it is often the case that there is still some underlying mechanism responsible for its behavior. In science, how do we distinguish between such "deterministic chaos" and true randomness? This requires a careful definition of randomness as well as geometric tools for detecting deterministic data.

Two fields have emerged over the last century to help answer this question. The first is the field of fractal geometry, which studies the properties of fractals and deterministic chaos from the perspective of mathematical analysis. The second is the field of algorithmic randomness, which seeks to answer questions about the nature of randomness from the perspective of mathematical logic.

Not surprisingly, these two disciplines are intimately related. The connection between them is both deep and elegant, and it is the subject of this thesis. A central concept bridging these two fields is the connection between fractal dimension, which provides a way to quantify the amount of irregularity in a data set or measure, and Kolmogorov complexity, a measure of complexity based on data compression.

In this thesis, we investigate new methods to estimate fractal dimension using data compression. Before giving the details of our results, we present the necessary background material in the subsequent sections.

Chapter 2
Fractal Geometry

There is no formal definition of the term "fractal" [1]. Attempts to define the term are too constraining because given a particular definition, you can always find sets that do not meet the description but still appear "fractal-like." The simplest explanation of a fractal is that it is a set that is self-similar at all scales.

2.1 The Cantor Set

One of the standard examples of a fractal set is the Cantor set, after the famous 19th century mathematician Georg Cantor. The Cantor set C has the peculiar property that it is an uncountable subset of the unit interval with Lebesgue measure 0. It is defined inductively in the following manner. Let C_1 be the unit interval, and let C_2 be equal to the part of C_1 that remains after deleting the middle third open interval $(\frac{1}{3}, \frac{2}{3})$. Note that C_2 is the union

$$C_2 = \left[0, \frac{1}{3}\right] \cup \left[\frac{2}{3}, 1\right],$$

consisting of 2 closed intervals.

Next, having defined $C_1, C_2, \ldots, C_{k-1}$, let C_k be equal to the part of C_{k-1} that remains after deleting the middle third open intervals inside each closed interval of C_{k-1}. Note that C_k is the union of 2^{k-1} closed intervals. As an example, C_3 is the union

$$C_3 = \left[0, \frac{1}{9}\right] \cup \left[\frac{2}{9}, \frac{3}{9}\right] \cup \left[\frac{6}{9}, \frac{7}{9}\right] \cup \left[\frac{8}{9}, 1\right],$$

Figure 2.1. The first five levels of the construction of the Cantor set C. At each stage of the construction process, the leftmost intervals are shaded black and the rightmost intervals are shaded gray.

consisting of 2^2 closed intervals. After performing this process infinitely many times, you are left with the Cantor set, i.e., the Cantor set C is defined to be the intersection,

$$C = \bigcap_{k=1}^{\infty} C_k.$$

What makes the Cantor set a fractal? It is a fractal because it is self-similar on all scales. If you take any particular closed interval I contained in the kth stage C_k, for any k, then you will notice that the construction process of C is mimicked inside the interval I as k increases. In other words, the part of the Cantor set that is contained in I looks just like the full Cantor set, just on a smaller scale. Topologically, this is expressed by observing that $C \cap I$ is homeomorphic to C.

As an example, consider the second closed interval $[\frac{2}{9}, \frac{3}{9}]$ in the 3rd stage C_3 in Figure 2.1. It is easy to see that the construction process of C is mimicked inside the interval $[\frac{2}{9}, \frac{3}{9}]$ at the levels C_4 and C_5.

2.1.1 Variations of the Cantor Set

There are lots of variations of the Cantor set. The version that we have just described is known as the Cantor ternary set, since the middle third interval is removed at each stage of the construction. The Smith-Volterra-Cantor set, also known as the "fat" Cantor set, is constructed in a similar fashion, except that a subinterval of length $1/2^{2n}$ is removed at each stage. The resulting fractal is actually a set of positive Lebesgue measure.

A two-dimensional variant of the Cantor set is known as "Cantor dust." The Cantor dust is defined to be the product $C \times C$.

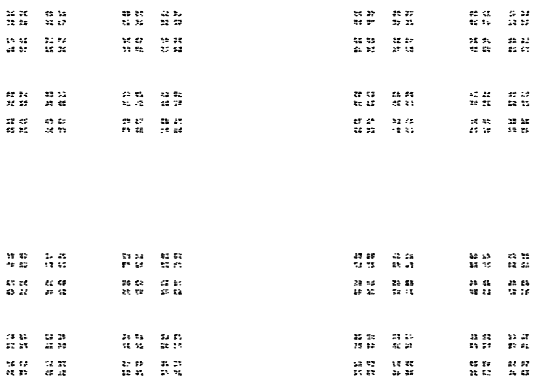

Figure 2.2. A computer generated image of a random sample of points from the first 20 levels of the Cantor dust ($C_{20} \times C_{20}$).

2.2 The Sierpinski Triangle

Another famous fractal is the Sierpinski triangle T. Like the Cantor set, the construction of the Sierpinski triangle is a geometric process involving a sequence of sets $(T_k)_{k\geq 1}$. Each stage of the construction is a set T_k equal to the preceding stage T_{k-1} minus a subset that has been removed. The Sierpinski triangle T is equal to the intersection of all the sets $(T_k)_{k\geq 1}$.

The first set T_1 in the construction is defined to be an equilateral triangle in the plane of side length 1. The next set T_2 is equal to T_1 with an inverted equilateral triangle of side length $\frac{1}{2}$ removed from the center (see Figure 2.3). Continuing inductively, each set T_k is constructed by removing the middle inverted equilateral triangle of side length 2^{1-k} from each triangle remaining in T_{k-1}.

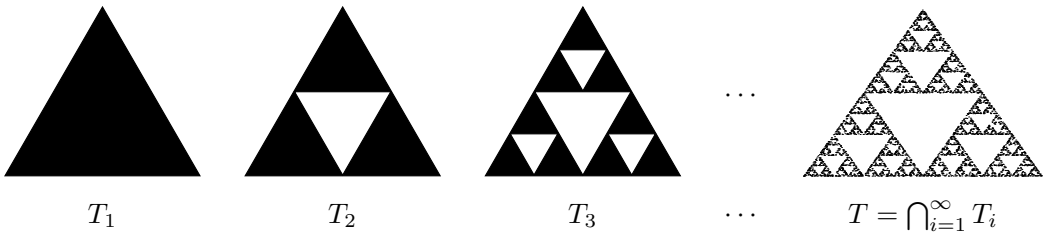

Figure 2.3. The construction of the Sierpinski triangle.

2.3 Fractal Dimension

Just as there is no formal definition of the term "fractal," there is no single definition of the term "fractal dimension" [1]. Instead, there is a whole range of different dimension concepts, each with their own strengths and weaknesses. In this section we summarize two of the most commonly used notions of dimension, namely box dimension and Hausdorff dimension. These two dimension concepts provide a way to express the amount of irregularity in a set of Lebesgue measure zero. In the following section, we extend these notions to dimensions for probability measures.

2.3.1 Box-Counting Dimension

A central question to the notion of dimension is how does the volume of a set scale as a function of its diameter? For example, the length of a line L in $\mathbb{R}$ scales exactly as its diameter, i.e., $\lambda(L) = \text{diam}(L)$, where λ is Lebesgue measure. The area of a circle C in $\mathbb{R}^2$ scales like its diameter squared: $\lambda(C) \sim \text{diam}(C)^2$. The volume of a sphere S in $\mathbb{R}^3$ scales like its diameter cubed: $\lambda(S) \sim \text{diam}(S)^3$. In each case, the value that we would like to assign to each of these sets as a measure of their "dimension" is just equal to the exponent in this relation between volume and diameter. A line should have dimension 1, a circle dimension 2, and a sphere dimension 3. Note that each of these "regular" sets have positive Lebesgue measure in their respective spaces.

To extend this idea to an "irregular" set E of measure zero, we will consider coverings of E by closed balls of fixed diameter, and watch what happens when the diameter of the balls in the covering approaches zero. Specifically, if $N_\epsilon(E)$ is the minimum number of closed balls of diameter ϵ required to cover the set E, we want to know how $N_\epsilon(E)$ scales as a function of the diameter ϵ in the limit as ϵ approaches zero. The box-counting dimension or box dimension of E will be defined to be the limiting value d (if it exists) such that $N_\epsilon(E) \sim (1/\epsilon)^d$. The formal definition of box dimension is given below.

Definition 2.1 (Box-counting dimension, cf. [2])**.** Let E be a nonempty, bounded subset of $\mathbb{R}^n$, and let $N_\epsilon(E)$ be the minimum number of closed balls of diameter ϵ that cover E. The **lower** and **upper box counting dimensions** are defined to

be

$$\underline{\dim}_B(E) = \liminf_{\epsilon \to 0} \frac{\log N_\epsilon(E)}{-\log \epsilon},$$
$$\overline{\dim}_B(E) = \limsup_{\epsilon \to 0} \frac{\log N_\epsilon(E)}{-\log \epsilon},$$

respectively. If $\underline{\dim}_B(E) = \overline{\dim}_B(E)$, then we call the common value the **box-counting dimension** or **box dimension** of E, written $\dim_B(E)$.

The reason this concept is referred to as "box-counting" dimension is because it is often easier to estimate its value using a grid of boxes (i.e., cubes) covering E instead of a collection of closed balls. A grid of cubes, also called an ϵ-coordinate mesh, is just the collection of all products of closed intervals $G_{j_1} \times G_{j_2} \times \cdots \times G_{j_n}$ where each G_{j_k} is a closed interval of the form $[s \cdot \epsilon, (s+1) \cdot \epsilon]$ for some integer s which depends on the size of the grid. The equivalence of these two methods is given in the following theorem.

Theorem 2.2 (cf. [1])**.** *Let E be a nonempty, bounded subset of $\mathbb{R}^n$, and let $\mathcal{G}_\epsilon$ be a grid of cubes covering E of side length ϵ. If $N'_\epsilon(E)$ is defined to be the minimum number of cubes from $\mathcal{G}_\epsilon$ covering E, then*

$$\underline{\dim}_B(E) = \liminf_{\epsilon \to 0} \frac{\log N'_\epsilon(E)}{-\log \epsilon},$$
$$\overline{\dim}_B(E) = \limsup_{\epsilon \to 0} \frac{\log N'_\epsilon(E)}{-\log \epsilon}.$$

Proof. It is easy to construct a cover of E by $N'_\epsilon(E)$ closed balls of diameter $\epsilon\sqrt{n}$. To do so, just replace each of the cubes in $\mathcal{G}$ that intersect E by closed balls with diameter equal to the length of the diagonals of each cube. Since $N_{\epsilon\sqrt{n}}(E)$ represents the minimum number of closed balls of diameter $\epsilon\sqrt{n}$ that cover E, we have $N_{\epsilon\sqrt{n}}(E) \leq N'_\epsilon(E)$. Taking the logarithm of both sides and observing that $-\log(\epsilon\sqrt{n}) > 0$ for small enough ϵ, we have

$$\frac{\log(N_{\epsilon\sqrt{n}}(E)}{-\log(\epsilon\sqrt{n})} \leq \frac{\log(N'_\epsilon(E))}{-\log \epsilon\sqrt{n}} \leq \frac{\log(N'_\epsilon(E))}{-\log \epsilon}.$$

Taking the limit inferior and limit superior of both sides as ϵ goes to zero gives,

$$\underline{\dim}_B(E) \leq \liminf_{\epsilon \to 0} \frac{\log(N'_\epsilon(E))}{-\log \epsilon}, \tag{2.1}$$

and

$$\overline{\dim}_B(E) \leq \limsup_{\epsilon \to 0} \frac{\log(N'_\epsilon(E))}{-\log \epsilon}. \tag{2.2}$$

To get the reverse inequalities in (2.1) and (2.2), note that if you cover E with closed balls of diameter ϵ, then you can construct a cover of E by cubes of side length ϵ by replacing each closed ball by its 3^n neighboring cubes. Hence, $N'_\epsilon(E) \leq 3^n N_\epsilon(E)$ since $N'_\epsilon(E)$ represents the minimum number of closed balls of side length ϵ covering E. Taking the logarithm of both sides as before, we have

$$\frac{\log(N'_\epsilon(E))}{-\log \epsilon} \leq \frac{n \log 3 + \log(N_\epsilon(E))}{-\log \epsilon},$$

which upon taking the limit inferior and limit superior of both sides gives the reverse inequalities.

□

As an application of Theorem 2.2, we compute the box dimension of the Cantor set C.

Example 2.3 (Box dimension of the Cantor Set, cf. [1])**.** For each n, define the grid $\mathcal{G}_n$ to be the collection of all closed intervals of the form $[s \cdot 3^{-n}, (s+1) \cdot 3^{-n}]$ with $s \in \{0, 1, \ldots, 3^n - 1\}$. The number of cubes in $\mathcal{G}_n$ that intersect C is 2^n (as in the construction of the nth stage C_n). Hence, letting $\epsilon = 3^{-n}$ we see that $N'_\epsilon(C) = 2^n$. Dividing by $-\log \epsilon$ and taking the limit superior of both sides gives $\overline{\dim}_B(C) = \log 2 / \log 3$.

2.3.2 Properties of Box-Counting Dimension

Box dimension has a number of nice properties that make it useful as a measure of fractal dimension. Despite its strengths, however, it also has a few weaknesses. We summarize the strengths and weaknesses of box dimension below. We will see

in the following section that the more versatile Hausdoff dimension retains the benefits of box dimension without suffering from the same defects.

Box dimension is useful because it is both a monotonic and stable measure of fractal dimension. We prove these and other nice properties in the theorems below. But first, we prove a lemma.

Lemma 2.4. *If F is a cube in $\mathbb{R}^n$, then $\dim_B(F) = n$.*

Proof. Let m be the side length of the cube F. Partition F into a grid of cubes of side length $\epsilon := 2^{-k}$ each, so that the total number of cubes contained in F is $(m/2^k)^n = m^n 2^{nk}$. Then clearly $N'_\epsilon(F) = m^n 2^{nk}$, so that

$$\dim_B(F) = \lim_{k\to\infty} \frac{n\log m + nk\log 2}{k\log 2} = \lim_{k\to\infty} \left(\frac{n\log m}{k\log 2} + n\right) = n.$$

□

Next, we show that box dimension is monotonic.

Theorem 2.5 (Monotonicity)**.** *If E is contained in a nonempty, bounded subset F in $\mathbb{R}^n$, then $\underline{\dim}_B(E) \le \underline{\dim}_B(F)$ and $\overline{\dim}_B(E) \le \overline{\dim}_B(F)$.*

Proof. This is an immediate consequence of the fact that $N'_\epsilon(E) \le N'_\epsilon(F)$ for a grid of ϵ-length cubes covering $F \supseteq E$. □

Lemma 2.4 and Theorem 2.5 imply the following useful corollary.

Corollary 2.6. *Let E be a nonempty, bounded subset of $\mathbb{R}^n$. Then,*

(a) $\overline{\dim}_B(E) \le n$,

(b) $\dim_B(E) = n$ if E contains an open set $\mathcal{O}$.

Proof. Since E is bounded, it is contained in a cube F. Part (a) then follows from Lemma 2.4 and Theorem 2.5.

If in addition, E contains an open set $\mathcal{O}$, then it also contains a cube G whose box dimension is n. By monotonicity and part (a),

$$n = \underline{\dim}_B(G) \le \underline{\dim}_B(E) \le \overline{\dim}_B(E) \le n$$

which proves the desired result. □

Finally, we show that box dimension is finitely stable.

Theorem 2.7 (Finite Stability)**.** *Let E, F be two nonempty, bounded subsets of $\mathbb{R}^n$. Then,*

$$\overline{\dim}_B(E \cup F) = \max\{\overline{\dim}_B(E), \overline{\dim}_B(F)\}. \tag{2.3}$$

Proof. Let $\mathcal{G}$ be a grid of cubes of side length $\epsilon = 2^{-k}$ each. Note that for any k, the following inequality clearly holds:

$$\max\{N'_\epsilon(E), N'_\epsilon(F)\} \leq N'_\epsilon(E \cup F) \leq 2\max\{N'_\epsilon(E), N'_\epsilon(F)\}.$$

Taking the logarithm and dividing by $-\log\epsilon = k\log 2$ as usual gives,

$$\frac{\log(\max\{N'_\epsilon(E), N'_\epsilon(F)\})}{k\log 2} \leq \frac{\log(N'_\epsilon(E \cup F))}{k\log 2} \leq \frac{\log 2 + \log(\max\{N'_\epsilon(E), N'_\epsilon(F)\})}{k\log 2}.$$

Taking the limit superior of the above inequality as k goes to infinity gives the desired result.

□

Note that stability holds for upper box dimension but not necessarily lower box dimension. The reason is because the limit superior is subadditive ($\limsup(x_k + y_k) \leq \limsup x_k + \limsup y_k$) while the limit inferior is not. In contrast, the limit inferior is actually superadditive ($\liminf(x_k + y_k) \geq \liminf x_k + \liminf y_k$), and so the analog of (2.3) for lower box dimension is,

$$\underline{\dim}_B(E \cup F) \geq \max\{\underline{\dim}_B(E), \underline{\dim}_B(F)\}.$$

This is one example of a weakness of box counting dimension.

A more distressing issue with box counting dimension is that it is not, in general, countably stable [1]. That is, if $(E_i)_{i=1}^\infty$ is a sequence of nonempty, bounded subsets in $\mathbb{R}^n$, then it is not necessarily the case that $\dim_B\left(\bigcup_{i=1}^\infty E_i\right) = \sup_i\{\dim_B(E_i)\}$. The next example demonstrates this.

Example 2.8. Let $(q_i)_{i=1}^\infty$ be an enumeration of the rational numbers in the unit interval, and define $E_i = \{q_i\}$ for each i. Clearly each singleton set has box dimension zero, so $\sup_i\{\dim_B(E_i)\} = 0$. But the box dimension of $\mathbb{Q} \cap [0,1] =$

$\bigcup_{i=1}^{\infty} E_i$ is equal to 1. To see why, note that since this set is dense in $[0,1]$, any grid of cubes covering this set also covers all of $[0,1]$ (in general, the box dimension of a set and its closure are the same). Hence, the two sets have the same box dimension, so $\dim_B(\mathbb{Q} \cap [0,1]) = \dim_B([0,1]) = 1$ by Lemma 2.4. So we have produced a countable sequence of sets with the property that $\dim_B\left(\bigcup_{i=1}^{\infty} E_i\right) > \sup_i\{\dim_B(E_i)\}$.

The failure of box dimension to be countably stable is a problem because it means that box dimension does not do a very good job of distinguishing between "regular" sets and "irregular" sets. The previous example shows, in fact, that it is possible for the box dimension of a regular, countable set in the unit interval to be larger than the irregular, uncountable Cantor set (recall that $\dim_B(C) = \log 2/\log 3 < 1$).

2.3.3 Hausdorff Measure and Dimension

Computing the box-counting dimension of a set E involves counting the minimum number of closed balls of fixed diameter ϵ (or cubes of fixed side length ϵ) that cover E, and then letting ϵ tend to zero. Obviously, the covers used in the construction of box-counting dimension must contain only finitely many sets. While useful in practice, this restriction ignores more precise covers containing infinitely many sets. The problems of box dimension as a measure of fractal dimension seen in the last section are a direct result of this restriction.

Hausdorff dimension resolves this problem by considering a broader range of possible covers. Although the definition of Hausdorff dimension is more abstract than the definition of box dimension, as it is defined in terms of measure theoretic concepts, the underlying idea is similar. The Hausdorff dimension of a set E in $\mathbb{R}^n$ is essentially just the exponent in a relation relating the volume of E to diameter. But before giving the full definition, we must construct a class of Lebesgue-like measures known as s-dimensional Hausdorff measure. This construction is based on the concept of an ϵ-cover.

Definition 2.9 (ϵ-cover, cf. [2])**.** Let E be a nonempty, bounded subset of $\mathbb{R}^n$. Given $\epsilon > 0$, an ϵ-cover of E is a countable collection of sets $(E_i)_{i=1}^{\infty}$ such that $E \subseteq \bigcup_{i=1}^{\infty} E_i$ and $\operatorname{diam}(E_i) \leq \epsilon$ for each $i \in \mathbb{N}$.

Next, we define the class of Hausdorff measures.

Definition 2.10 (Hausdorff measure, cf. [1])**.** Let $s \in \mathbb{R}$ with $s \geq 0$ be given, and let E be a subset of $\mathbb{R}^n$. The s-dimensional Hausdorff measure of E is the limit

$$\mathcal{H}^s(E) = \lim_{\epsilon \to 0} \mathcal{H}^s_\epsilon(E), \tag{2.4}$$

where,

$$\mathcal{H}^s_\epsilon(E) = \inf \left\{ \sum_{i=1}^{\infty} \operatorname{diam}(E_i)^s \mid (E_i)_{i=1}^{\infty} \text{ is an } \epsilon\text{-cover of } E \right\}. \tag{2.5}$$

We need to check that this definition is well defined, i.e., that the limit in equation (2.4) exists or is infinite. If $\epsilon' < \epsilon$, then clearly the set of ϵ'-covers is strictly smaller than the set of ϵ-covers. So when computing $\mathcal{H}^s_{\epsilon'}(E)$, the infimum in (2.5) is being taken over a smaller set. This means that $\mathcal{H}^s_\epsilon(E)$ is non-decreasing as a function of ϵ, and so it approaches either a finite limit or is infinite.

Note that s-dimensional Hausdorff measure is defined similarly to Lebesgue outer measure. Recall that the Lebesgue outer measure of a set $E \subseteq \mathbb{R}^n$ is defined

$$\lambda^*(E) = \inf \left\{ \sum_{i=1}^{\infty} |E_i| \mid (E_i)_{i=1}^{\infty} \text{ is a collection of closed cubes covering } E \right\},$$

where $|E_i|$ denotes the volume of the closed cube E_i. In fact, it is easy to show that 1-dimensional Lebesgue measure λ is just 1-dimensional Hausdorff measure. In general, for any integer $s \geq 1$, s-dimensional Lebesgue measure and s-dimensional Hausdorff measure are equal up to a scaling constant [1].

An interesting question to ask is how s-dimensional Hausdorff measure behaves as a function of s for a fixed set $E \subseteq \mathbb{R}^n$. The answer to this question will motivate the definition of Hausdorff dimension. But before giving its definition, note the following fact.

Theorem 2.11 (cf. [1])**.** *Let $E \subseteq \mathbb{R}^n$ be given. Then $\mathcal{H}^s(E)$ is non-increasing as a function of s. Furthermore, if $\mathcal{H}^s(E) < \infty$ for some s, then $\mathcal{H}^t(E) = 0$ for all $t > s$.*

Proof. Given $\epsilon < 1$ and $t > s$, it is clear that $\operatorname{diam}(E_i)^t \leq \operatorname{diam}(E_i)^s$ for any set E_i belonging to an ϵ-cover of E. The first statement follows.

To see why the second statement holds, let $t > s$ with $\mathcal{H}^s(E) < \infty$. Let $(E_i)_{i=1}^{\infty}$

be an ϵ-cover of E. Observe that,

$$\sum_{i=1}^{\infty} \operatorname{diam}(E_i)^t = \sum_{i=1}^{\infty} \operatorname{diam}(E_i)^{t-s} \operatorname{diam}(E_i)^s \leq \epsilon^{t-s} \sum_{i=1}^{\infty} \operatorname{diam}(E_i)^s.$$

Taking the infima of the above inequality over all ϵ-covers of E gives $\mathcal{H}^t_\epsilon(E) \leq \epsilon^{t-s}\mathcal{H}^s_\epsilon(E)$. Since $t > s$, and since $\mathcal{H}^s(E) < \infty$, taking the limit as ϵ goes to 0 gives $\mathcal{H}^t(E) = 0$.

□

Theorem 2.11 shows that for a given set E in $\mathbb{R}^n$, the value of the s-dimensional Hausdorff measure of E for most values of s is either infinity or zero, depending on the value of s. The Hausdorff dimension of E is defined to be the (possibly nonzero) point at which the graph of the Hausdorff measure as a function of s jumps from infinity to a finite value.

Definition 2.12 (Hausdorff dimension)**.** Let E be a nonempty, bounded subset of $\mathbb{R}^n$. The **Hausdorff dimension** of E is defined to be,

$$\dim_H(E) = \inf\{s \geq 0 \mid \mathcal{H}^s(E) = 0\},$$

or equivalently,

$$\dim_H(E) = \sup\{s \geq 0 \mid \mathcal{H}^s(E) = \infty\}.$$

We will prove below some of the properties of Hausdorff dimension, such as countable stability, which make it a better measure of fractal dimension than box-counting dimension. The main disadvantage of using Hausdorff dimension over box-counting dimension is that Hausdorff dimension can be more difficult to calculate in practice. For some sets, however, such as the Cantor set, we can compute Hausdorff dimension without too much difficulty (although the calculation is arguably still more work than Example 2.3).

Example 2.13 (Hausdorff dimension of the Cantor set, cf. [1])**.** We show that $\frac{1}{2} \leq \mathcal{H}^s(C) \leq 1$ for $s = \log 2/\log 3$. Hence, $\log 2/\log 3$ is the Hausdorff dimension of C since $\mathcal{H}^t(C) = 0$ for any $t > \log 2/\log 3$ by Theorem 2.11.

First, we prove the upper bound. For each k, the kth stage C_k contains 2^k closed intervals each of length 3^{-k}. Hence, C_k is a 3^{-k}-cover of C. The sum of

the sth powers of the lengths of these intervals is thus $2^k 3^{-ks}$. Since $\mathcal{H}^s_{3^{-k}}(C)$ is the infimum over all such covers of C, $\mathcal{H}^s_{3^{-k}}(C) \leq 2^k 3^{-ks}$. If we let $s = \log 2/\log 3$, then

$$2^k 3^{-ks} = 1, \tag{2.6}$$

so $\mathcal{H}^s_{3^{-k}}(C) \leq 1$. This implies $\mathcal{H}^s \leq 1$ by taking the limit as k goes to infinity.

To prove the lower bound, let $(E_i)_{i=1}^N$ be an arbitrary ϵ-cover of C (we can consider an arbitrary finite cover and not an infinite cover since C is compact). There is a unique integer k such that

$$3^{-(k+1)} \leq \operatorname{diam}(E_i) < 3^{-k}. \tag{2.7}$$

Note that each set E_i intersects at most one closed interval in the kth stage C_k since $\operatorname{diam}(E_i) < 3^{-k}$ and the width of each interval in C_k is 3^{-k}.

Next, observe that each of the 2^k closed intervals of C_k contains $2^j/2^k = 2^{j-k}$ closed intervals from the jth stage C_j, if $j \geq k$. So each E_i intersects at most 2^{j-k} closed intervals belonging to C_j. But this upper bound can be made even more flexible. Equations (2.6) and (2.7) together imply that

$$2^{j-k} = 2^j 3^{-ks} \leq 2^j 3^s \operatorname{diam}(E_i)^s,$$

and so each E_i intersects at most $2^j 3^s \operatorname{diam}(E_i)^s$ intervals belonging to C_j. Since $\bigcup_{i=1}^N E_i$ and C_j are both covers of C, the cover $\bigcup_{i=1}^N E_i$ intersects each of the intervals of C_j. Hence, the total number of closed intervals in C_j must be bounded by the maximum number of possible intersections that occur, i.e.,

$$2^j \leq \sum_{i=1}^N 2^j 3^s \operatorname{diam}(E_i)^s.$$

Equivalently, this can be written,

$$\sum_{i=1}^N \operatorname{diam}(E_i)^s \geq 3^{-s} = \frac{1}{2}.$$

Taking the infima of the last inequality over all ϵ-covers of C gives $\mathcal{H}^s_\epsilon(C) \geq 1/2$, or $\mathcal{H}^s(C) \geq 1/2$ in the limit as ϵ goes to zero.

2.3.4 Properties of Hausdorff Dimension

The main reason that Hausdorff dimension is superior to box-counting dimension is because Hausdorff dimension is countably stable. We prove this property along with other nice properties of Hausdorff dimension below, such as monotonicity, and we demonstrate that box-counting dimension provides an upper bound to Hausdorff dimension.

Theorem 2.14 (Monotonicity)**.** *If a nonempty set E is contained in a bounded subset F in $\mathbb{R}^n$, then* $\dim_H(E) \leq \dim_H(F)$.

Proof. Since $E \subseteq F$, we clearly have $\mathcal{H}^s(E) \leq \mathcal{H}^s(F)$. Thus, $\infty > \mathcal{H}^{\dim_H(F)}(F) \geq \mathcal{H}^{\dim_H(F)}(E)$, and so $\dim_H(E) \leq \dim_H(F)$.

□

Theorem 2.15 (Countable Stability, cf. [1])**.** *Let $(E_i)_{i=1}^{\infty}$ be a sequence of nonempty, bounded subsets in $\mathbb{R}^n$. Then* $\dim_H(\bigcup_{i=1}^{\infty} E_i) = \sup_i\{\dim_H(E_i)\}$.

Proof. Clearly $\dim_H(E_i) \leq \dim_H(\bigcup_{i=1}^{\infty} E_i)$ for each i by monotonicity, so $\dim_H(\bigcup_{i=1}^{\infty} E_i)$ is an upper bound of the set $\{\dim_H(E_i)\}$.

To see that it is the least upper bound, let s be a strict upper bound of the set $\{\dim_H(E_i)\}$. Since $s > \dim_H(E_i)$ for each i, it follows that $\mathcal{H}^s(E_i) = 0$ by Theorem 2.11. Hence, $\mathcal{H}^s(\bigcup_{i=1}^{\infty} E_i) = 0$ by the countable subadditivity of the Hausdorff measure. This implies that $\dim_H(\bigcup_{i=1}^{\infty} E_i) \leq s$, so it is indeed the least upper bound.

□

The countable stability of Hausdorff dimension solves one of the problems of box dimension. In Example 2.8, we showed that a "regular" countable subset of the unit interval (the rationals) can have positive box dimension. This is clearly not the case with Hausdorff dimension; countable stability implies that countable sets have dimension zero since individual points have dimension zero.

Finally, Hausdorff dimension satisfies properties similar to those given in Corollary 2.6 for box-counting dimension. Namely, $\dim_H(E) \leq n$ for every bounded set in $E \subseteq \mathbb{R}^n$, and if E contains an open ball, then $\dim_H(E) = n$.

2.4 Dimension for measures

The idea of assigning a fractal dimension to a set can be extended to include objects with more structure, such as measure spaces and probability spaces. But unlike fractal dimension for sets, a single number is not sufficient for describing the rich fractal structure that can be present in a measure. Instead, the fractal dimension of a measure μ is described using a collection of numbers, called the dimension spectrum or dimension distribution of the measure, with each value in the distribution having its own special meaning.

The reason an entire dimension spectrum is used to describe the fractal nature of a measure as opposed to a single dimension number is because a measure may actually generate a collection of fractal sets. Such a measure is called a multifractal and its dimension spectrum is called its multifractal spectrum. This multifractal spectrum can be defined in two different ways by considering both the local and global scaling behavior of the measure.

In this section, we give some examples of multifractal measures, and we summarize both the local and global definitions of the multifractal spectrum. For convenience, in this section all measures μ defined on a bounded set X in $\mathbb{R}^n$ are assumed to be probability measures, and so $\mu(X) = 1$.

2.4.1 Local Definition of Multifractal Spectra

One way to define the multifractal spectrum of a measure is in terms of its local scaling behavior. Much like the question put forth at the beginning of Section 2.3.1, a central question underlying the local scaling behavior of a measure is how does the measure of a set scale as a function of its diameter? Of course, in order to make sense of this question, we need to know what set we are measuring; after all, a measure μ takes values on the entire σ-algebra Σ over which the measure space (μ, X, Σ) is defined.

It is possible that different subsets of X may scale as a function of diameter in different ways. If this relation obeys a power law of the form $\mu(E) \sim \epsilon^\alpha$, where E is a subset of X with diameter ϵ, then the exponent α might be different for each subset of X. It is this difference that is used to construct the multifractal spectrum. For each $\alpha \geq 0$, let F_α denote the set of points x such that $\mu(B(x, \epsilon)) \sim \epsilon^\alpha$ in

the limit as ϵ goes to zero. The multifractal spectrum of μ is defined to be the set $\{\dim_H(F_\alpha) \mid \alpha \geq 0\}$, i.e., the set of Hausdorff dimensions of each set F_α. We formalize this concept in the two definitions below.

Definition 2.16 (Local Pointwise Dimension, cf. [1])**.** Let (μ, X, Σ) be a probability space, and let $x \in X$. Then the **local pointwise dimension** of μ at x is defined as,

$$\dim_{\text{loc}} \mu(x) = \lim_{\epsilon \to 0} \frac{\log \mu(B(x, \epsilon))}{\log \epsilon},$$

provided this limit exists.

In other words, $\dim_{\text{loc}} \mu(x)$ is the exponent in the scaling relation $\mu(B(x, \epsilon)) \sim \epsilon^\alpha$. Next, we define the local multifractal spectrum of a measure.

Definition 2.17 (Local Multifractal Spectrum, cf. [1])**.** Let (μ, X, Σ) be a probability space, and for each $\alpha \geq 0$ define $F_\alpha = \{x \in X \mid \dim_{\text{loc}} \mu(x) = \alpha\}$. Then the **local multifractal spectrum** of μ is the set, $\{\dim_H(F_\alpha) \mid \alpha \geq 0\}$.

A measure μ is considered to be a multifractal since it may generate a different fractal set for each value of $\alpha \geq 0$.

An alternate and yet equivalent method used to investigate the local fractal nature of a measure involves the concept of a dimension distribution. Given a random subset $E \subseteq X \subseteq \mathbb{R}^n$, a dimension distribution of a probability measure μ will tell you the probability that E has Hausdorff dimension α. In other words, it tells you how dimension is distributed over the interval $[0, n]$.

Definition 2.18 (Dimension distribution, cf. [2])**.** Let (μ, X, Σ) be a probability space. Given $\alpha \geq 0$, define

$$\mu_{\dim}([0, \alpha]) = \sup\{\mu(D) \mid \dim_H(D) \leq \alpha \text{ where } D \text{ is a Borel set in } \mathbb{R}^n\}.$$

The relationship between these two methods is demonstrated in the following theorem.

Theorem 2.19 (cf. [2])**.** *Let (μ, X, Σ) be a probability space. For each $\alpha \geq 0$,*

$$\mu(F_\alpha) = \mu_{dim}([0, \alpha]).$$

For an outline of the proof of this theorem, see [2].

2.4.2 Global Definition of Multifractal Spectra

Instead of focusing on the local scaling behavior of a measure, it is also possible to define the multifractal spectrum of a measure in terms of its global scaling behavior. This approach is commonly used to compute the fractal dimension of real world data, and is the popular method of choice among physicists.

The idea is similar as in the previous section, but instead of focusing on the sets of points where the measure of open balls containing those points scales like a power law, the focus is on the average measure of such balls across the entire set. If this average scales as a power law, then the multifractal spectrum is defined to be the set of exponents.

We begin with a definition.

Definition 2.20 (cf. [2])**.** Given a probability space (μ, X, Σ) and $\epsilon > 0$, let $\mu_\epsilon : X \to \mathbb{R}$ be the function defined by $\mu_\epsilon(x) = \mu(B(x, \epsilon))$.

To examine the global scaling behavior of a measure μ, we are interested in how the L^q norm of the function μ_ϵ scales as a function of ϵ. If it scales according to a power law, i.e., if there is an exponent d such that $\|\mu_\epsilon\|_q \sim \epsilon^d$, then we say that d is the qth Rényi dimension of μ.

Definition 2.21 (Rényi Dimensions, cf. [2])**.** Let (μ, X, Σ) be a probability space, and let $q \neq 0$ be given. The qth generalized Rényi dimension of μ, written $d_\mu(q)$ is defined as,

$$d_\mu(q) = \lim_{\epsilon \to 0} \frac{\log \|\mu_\epsilon\|_q}{\log \epsilon},$$

provided that the limit exists.

2.4.3 Correlation Dimension

To measure the fractal dimension of real world data, scientists assume that the data reflects the existence of some underlying measure. The data points are assumed to be elements of subsets of the space which have positive measure. The most commonly used Rényi dimension to measure the fractal dimension of real world

data is the case $q = 1$, called **correlation dimension**. When $q = 1$, the L^q norm of μ_ϵ is just the expected value of the measure of a random ball of radius ϵ, i.e.,

$$\|\mu_\epsilon\|_1 = \int \mu(B(x,\epsilon))\mu(dx) = E(B(X,\epsilon)), \tag{2.8}$$

where X is a random variable. If in the limit as ϵ goes to zero, the expected value scales according to a power law, i.e., if $E(B(X,\epsilon)) \sim \epsilon^d$, then the exponent $d = d_\mu(1)$ is the correlation dimension of μ.

Two physicists, Peter Grassberger and Itamar Procaccia [3], developed the most commonly used numerical algorithm for estimating the correlation dimension of real world discrete data. The Grassberger-Procaccia algorithm consists of estimating (2.8) for various values of ϵ, plotting the result in a log-log plot, and estimating the slope of the log-log plot using linear regression (assuming the data is linear).

To estimate (2.8), the Grassberger-Procaccia algorithm does not attempt to compute the expected value of a random ball of radius ϵ directly, but rather it approximates an equivalent form of this expression called the *spatial correlation integral*, which we define below.

Definition 2.22 (Spatial Correlation Integral, cf. [2])**.** Given a probability space (μ, X, Σ) and $\epsilon > 0$, let $C_\mu(\epsilon)$ (called the **spatial correlation integral**) be the probability that two random, independent points are separated by a distance less than or equal to ϵ. In other words, $C_\mu(\epsilon) = \mu \times \mu(\{(x,y) \mid d(x,y) \leq \epsilon\})$.

The spatial correlation integral is clearly equal to the L^1 norm of μ_ϵ. Indeed, if we let $E = \{(x,y) \mid d(x,y) \leq \epsilon\}$, then

$$\mu \times \mu(E) = \int I_E(x,y)\,\mu(dy)\,\mu(dx) = \int \mu(E_x)\,\mu(dx), \tag{2.9}$$

where $E_x = \{y \in X \mid\mid d(x,y) \leq \epsilon\}$ and $I_E(x,y)$ is the indicator function of E. But for each $x \in X$, the set E_x is exatly equal to the ball or radius ϵ centered at x. Hence, (2.8) and (2.9) are equivalent.

Given a discrete set of data, the Grassberger-Procaccia algorithm approximates the spatial correlation integral by counting the number of points in the data set which are separated by a distance less than or equal to ϵ and dividing by the total number of pairs of points in the data set. This is known as the *sample correlation integral*.

Definition 2.23 (Sample Correlation Integral, cf. [2])**.** Let $(x_i)_{i=1}^N$ be a sequence of N points in $\mathbb{R}^n$. The **sample correlation integral** of this sequence, denoted $C(N, \epsilon)$, is defined to be

$$C(N, \epsilon) = \binom{N}{2}^{-1} \sum_{i=1}^{N} \sum_{j>i}^{N} I_E(x_i, x_j),$$

where $E = \{(x_i, x_j) \mid d(x_i, x_j) \leq \epsilon\}$.

If $(x_i)_{i=1}^N$ is an i.i.d. sequence of points, then the sample correlation integral converges in probability to the spatial correlation integral as N approaches infinity (see [4] and [5]).

2.4.4 Information Dimension

Another popular way to assign a fractal dimension to a measure related is with **information dimension**. Information dimension is very similar to correlation dimension, and as we will show in the next chapter, information dimension is closely related to information-theoretic entropy.[1] The difference in definition between information and correlation dimension is just the placement of the logarithm in the numerator. In correlation dimension, the logarithm is outside of expected value; in information dimension, the logarithm is inside the expected value.

Definition 2.24 (Information dimension, cf. [2])**.** Let (μ, X, Σ) be a probability space, and let $q \neq 0$ be given. The information dimension of μ, denoted σ_μ, is the limit (provided it exists)

$$\sigma_\mu = \lim_{\epsilon \to 0} \frac{E(\log \mu(B(X, \epsilon))}{\log \epsilon}.$$

[1] See equation (3.3).

Chapter 3
Algorithmic Information Theory

We all possess some intuitive ideas about the concept of "randomness." Things that are "random" have less structure and are more disorganized, while things that are "not random" are intrinsically ordered, containing recognizable patterns and repetition. The field of algorithmic randomness seeks to answer questions about the nature of randomness using the tools of mathematical logic and information theory. Algorithmic randomness is concerned with measuring the randomness of data in the form of strings and sequences of characters contained in some alphabet. In this chapter, we concentrate on one of the most popular ways to measure the randomness of information, namely Kolmogorov complexity. We conclude the chapter with a discussion on how Kolmogorov complexity is related to information-theoretic entropy. In the following chapter, we connect the dots and show how these concepts are related to the field of fractal geometry discussed in Chapter 2. In Chapter 4, we will also introduce an alternate and yet equivalent method of defining randomness based on ideas from measure theory called Martin-Löf randomness.

3.1 Kolmogorov Complexity

One way to measure the relative randomness of a string of characters is by measuring how complex the string is relative to its length. There are many different measures of string complexity, but Kolmogorov complexity is based on the idea that complex strings are harder to describe and reproduce using an algorithm or computer program.

Definition 3.1 (Kolmogorov Complexity, informal definition)**.** The Kolmogorov

complexity $C(x)$ of a string or integer x is the length in bits of the shortest computer program P written in any universal programming language that prints x and halts.

By a universal programming language, we mean one that can implement a Turing or Register Machine.

At its heart, Kolmogorov complexity is about **data compression**. Given a string x, a computer program P that prints x and halts can be viewed as a "compressed version" of x. Presumably, if x has some structure and is easy to describe, then the length of a computer program that prints x should in principle have a smaller size than the length of x. The Kolmogorov complexity of a string x is essentially just the length of the absolute smallest program prints x, i.e., the size of the "most compressed version" of x possible (see Definition 3.2 below where these statements are made more rigorous).

An example of a relatively simple string with with some structure is given by $x = 111\cdots11$ consisting of the character 1 repeated, say, 10^6 times. This string has relatively low Kolmogorov complexity; for example, it is very easy to implement a program that can print x and halt. All that is required is a simple for-loop that repeats printing the character "1" 100 times. The number of bits necessary to store such a program would (in principle) be less than a program that stored the entire string x. Hence, x does not seem to be very random.

On the other hand, the string $y = aks92hgf9nei0n3jgiurbhueh$ contains many more characters and does not appear to have any kind of structure or discernible pattern. The simplest possible description of y might just be y itself (see Theorem 3.3 below). Hence, y would have high Kolmogorov complexity. According to intuition, y seems to be very random, at least much more so than the string x.

In the theorems that follow, it will be easier to work with binary strings. Let 2^N be the set of binary strings of length N, $2^{<\omega}$ be the set of finite binary strings, and 2^ω the set of infinite binary strings. Komlogorov complexity can thus be viewed as a function from $2^{<\omega}$ to the natural numbers $\mathbb{N}$. In some instances, however, it is more convenient to view it as a function from $\mathbb{N}$ to $\mathbb{N}$. This can be done without any problem since there is a 1-1 correspondence between the set of finite binary strings and the set of natural numbers (just apply the lexicographical ordering to $2^{<\omega}$ and number the strings one by one). The next definition redefines Kolmogorov complexity within the context of binary strings and highlights the data compression aspect of Kolmogorov complexity [6].

Definition 3.2 (Kolmogorov complexity, binary string definition)**.** The Kolmogorov complexity of a binary string $x \in 2^{<\omega}$ is given by

$$C(x) = \min\{|y| \mid y \in 2^{<\omega} \text{ and } U(y) = x\},$$

where U is a universal Turing machine, and where $|y|$ denotes the length of y in bits. The binary string y corresponds to the program P in Definition 3.1, and it is the most compressed string containing all of the information needed to reproduce x.

We now prove some of the fundamental properties of Kolmogorov complexity. We first show that $C(x)$ is not computable. This will unfortunately limit the practicality of Kolmogorov complexity, demonstrating that it is more useful as a theoretical tool.

Theorem 3.3 (Noncomputability, cf. [7])**.** *C(x) is not a computable function.*

Proof. Assume that C is a function on the natural numbers $\mathbb{N}$. If C were computable, then the function

$$\psi(m) := \min_{x \in \mathbb{N}} \{x : C(x) \geq m\}$$

would also be computable, for then it would be possible to write a program that computes $C(x)$ for each positive integer beginning with 1 and ending when $C(x) \geq m$, which will eventually terminate.

If ψ is computable, then there is a program P of some fixed size c that prints $\psi(m)$ on input m and halts. Since $C(\psi(m))$ is the length in bits of the shortest program that prints $\psi(m)$, it is bounded by the size of P, together with its input m. The integer m can be stored using $\log m$ bits (log base 2); hence, we have $C(\psi(m)) \leq c + \log m$.

Conversely, by definition of ψ we have $C(\psi(m)) \geq m$. Combining these inequalities gives the desired contradiction since $m \leq c + \log m$ is false for large enough m.

□

Next, we demonstrate that Kolmogorov complexity is well defined, in the sense that it is invariant with respect to the universal programming language used to

define it. No matter which universal programming language is used, Kolmogorov complexity is invariant up to an additive constant.

Theorem 3.4 (Invariance, cf. [7])**.** *For any two universal programming languages L_1 and L_2 there exists a constant c such that*

$$|C_1(x) - C_2(x)| \leq c$$

for all integers x, where $C_1(x)$ and $C_2(x)$ denote the Kolmogorov complexities of x with respect to L_1 and L_2, respectively.

Proof. Let (p, y) be the smallest pair consisting of an L_1 program and input y that outputs x. Hence, $C_1(x) = l(p) + l(y)$, where $l(\cdot)$ denotes length in bits. Next, let Λ be a program that translates programs written in L_1 into L_2. Hence, Λ outputs x on input (p, y). So,

$$\begin{aligned} C_2(x) &\leq l(\Lambda) + l(p) + l(y) \\ &= l(\Lambda) + C_1(x). \end{aligned}$$

In other words, $C_2(x)$ is bounded by $C_1(x)$ and a constant number $l(\Lambda)$ that depends only on the languages L_1 and L_2. A similar argument proves the reverse inequality. □

Finally, we prove the following obvious fact about Kolmogorov complexity that will be useful later.

Theorem 3.5 (cf. [6])**.** *Let x be a finite binary string, and let $|x|$ denote its length. Then $C(x) \leq |x| + \mathcal{O}(1)$.*

Proof. Let P be a program that outputs x on input x. In other words, P is the identity function. If P can be represented using a binary string c bits long, then the minimum program that outputs x is at least the size of P, together with the size of its input. That is, $C(x) \leq c + |x|$ as required.

□

3.1.1 Prefix-free Kolmogorov complexity

There are a few problems with Kolmogorov complexity as we have so far defined it. For instance, it is not **subadditive**, i.e., it is not the case that $C(xy) \leq$

$C(x)+C(y)+\mathcal{O}(1)$ for all binary strings x, y, where xy denotes the concatenation of x and y. This is unfortunate because we would like to be able to compare Kolmogorov complexity as a measure of information content to entropy, a subadditive measure of information based on probability.

Furthermore, we would also like to be able to use Kolmogorov complexity to assign a probability to each finite binary string. A natural way to do this would be to assign the probability of each finite string x equal to $2^{-C(x)}$, but as we show below the sum $\sum_{x\in 2^{<\omega}} 2^{-C(x)}$ diverges, so this will not work.

Both of these problems can be fixed by restricting our attention to prefix-free sets of binary strings. The resulting measure of complexity is called **prefix-free Kolmogorov complexity**. Before demonstrating how prefix-free Kolmogorov complexity resolves the above problems, we give a few definitions.

Definition 3.6 (Prefixes and Prefix-free Sets)**.** A finite binary string x of length N is called a **prefix** of a binary string y, written $x \preceq y$, if the first N bits of y are equal to x. A **prefix-free** set of binary strings is such that no element is a prefix of another element in the set.

The original definition of Kolmogorov complexity given in Definition 3.1 and 3.2, called plain Kolmogorov complexity, was defined in terms of a universal Turing machine whose domain was $2^{<\omega}$, i.e., the entire set of finite binary strings. Prefix-free Kolmogorov complexity, in contrast, is defined in terms of a universal Turing machine whose domain is a prefix-free subset of $2^{<\omega}$.

Definition 3.7 (Prefix-free Kolmogorov complexity)**.** The **prefix-free Kolmogorov complexity** of a binary string $x \in 2^{<\omega}$ is given by

$$K(x) = \min\{|y| \mid y \in 2^{<\omega} \text{ and } U(y) = x\},$$

where U is a universal **prefix-free** Turing machine, i.e., the domain of U is a prefix-free subset of $2^{<\omega}$.

3.1.2 Subadditivity

The reason that plain Kolmogorov complexity fails to satisfy subadditivity is because string concatenation results in a loss of structure. Given two finite binary strings a and b, it is impossible to tell what a and b are given the concatenation

ab. This loss of structure increases complexity, making it difficult to prove that plain Kolmogorov complexity satisfies subadditivity. In fact, it can be shown (see Theorem 3.1.4 in [6]) that for any positive integer k, you can always find binary strings x and y such that

$$C(xy) > C(x) + C(y) + k. \tag{3.1}$$

In order to bound the complexity of a concatenation of two strings, it is necessary to know the length of the first string in the concatenation. This fact is expressed in the following theorem giving an upper bound on the plain Kolmogorov complexity of a concatenation.

Theorem 3.8 (cf. [6])**.** *Let x, y be two finite binary strings. Then,*

$$C(xy) \leq C(x) + C(y) + 2\log C(x) + \mathcal{O}(1).$$

Proof. Let z be the string equal to the concatenation of two strings xy. If U is a universal Turing machine, let x^* denote the binary string of length $C(x)$ such that $U(x^*) = x$, and let y^* denote the binary string of length $C(y)$ such that $U(y^*) = y$. Clearly the strings x^* and y^* hold enough information to generate the string $z = xy$ (just concatenate $U(x^*)$ and $U(y^*)$). Does this mean that the concatenation x^*y^* holds enough information to generate the string z? The problem is that the string x^*y^* does not contain any information about where x^* ends and y^* begins. To generate the string $z = xy$, you can use x^* and y^*, but you also need to know the length of x^*, which is $C(x)$. The string z^* of minimal length such that $U(z^*) = z$ therefore must be bounded by the size of x^*y^* plus the number of bits necessary to encode the number $C(x)$. In this case, the number of bits necessary to encode $C(x)$ is $2\log C(x)$[1].

□

In contrast, prefix-free Kolmogorov complexity does satisfy subadditivity. By restricting the domain of the universal Turing machine U in the definition of $K(x)$ to a prefix-free set, the start and end points of a string are always known, and it is not

[1]Ordinarily it only takes $\log n$ bits to store an integer n, but for technical reasons it is necessary to double the number of bits so that it is clear where $C(x)$ ends in the encoding of $C(x)x^*y^*$. See Proposition 3.1.5 in [6]

necessary to encode extra information. Hence, we have $K(xy) \leq K(x)+K(y)+\mathcal{O}(1)$ for every pair of finite binary strings.

3.1.3 Algorithmic Probability

In the previous few sections, we have presented Kolmogorov complexity as a method for measuring the relative randomness of a string of characters. Strings that have high complexity are considered to be more random than strings with low complexity. The use of the word "random" suggests that we can express these ideas in terms of probability. In this section we demonstrate another advantage of prefix-free Kolmogorov complexity over plain complexity; namely, the former can be used to define a probability distribution on the set of finite binary strings.

We want to define a probability distribution that assigns high probability to strings with low complexity (less random strings), and low probability to strings with high complexity (random strings). In terms of plain Kolmogorov complexity, a natural way to do this would be to assign the probability $2^{-C(x)}$ to each finite binary string x. The motivation for this construction is illustrated by the following experiment. Given a finite binary string x, imagine flipping a coin $C(x)$ times. If heads is represented by a 1 and tails by a 0, the probability of having generated the binary form of a program that outputs x is $2^{-C(x)}$. We extend this idea to the entire space by assigning the probability $2^{-C(z)}$ to each finite binary string z. This construction gives us exactly what we want; indeed, high complexity strings have comparatively lower probability than low complexity. But as the following theorem demonstrates, it suffers from a major problem.

Theorem 3.9. *The series* $\sum_{x\in 2^{<\omega}} 2^{-C(x)}$ *diverges.*

Proof. The proof follows directly from Theorem 3.5. To see this, for each natural number n, let x_n be any string of length $\log n$. Theorem 3.5 implies that $C(x_n) \leq \log n$ up to a constant term. Then,

$$\sum_{x\in 2^{<\omega}} 2^{-C(x)} \geq \sum_{k=1}^{\infty} 2^{-C(x_k)} \geq \sum_{k=1}^{\infty} 2^{-\log k} = \sum_{k=1}^{\infty} \frac{1}{k}.$$

Since the harmonic series $\sum_{k=1}^{\infty}$ diverges, the series $\sum_{x\in 2^{<\omega}} 2^{-C(x)}$ must also diverge. □

So we see that our prescribed method of constructing a probability distribution using plain Kolmogorov complexity fails. But what if we use prefix-free Kolmogorov complexity instead? Certainly if prefix-free complexity satisfies the same bound in Theorem 3.5, then the series $\sum_{x\in 2^{<\omega}} 2^{-K(x)}$ must also diverge. But fortunately this is not the case. The fact that this method works for prefix-free Kolmogorov complexity is a consequence of the Kraft inequality, which we present below without proof (for a complete proof, see [8]).

Theorem 3.10 (The Kraft Inequality)**.** *Let $(n_k)_{k\geq 1}$ be a sequence of natural numbers. Then there is a prefix-free set of finite binary strings with this sequence of lengths if and only if $\sum_{k\geq 1} 2^{-n_k} \leq 1$.*

Since the set $\{K(x) \mid x \in 2^{<\omega}\}$ is the length set of a prefix-free set (namely, the domain of a prefix-free universal Turing machine), the Kraft inequality implies that $\sum_{x\in 2^{<\omega}} 2^{-K(x)} \leq 1$, and so it is possible to define a probability distribution on the set of finite binary strings as outlined above using prefix-free complexity.

To see just how badly prefix-free Kolmogorov complexity fails to satisfy the same bound as plain Kolmogorov complexity given in Theorem 3.5, note the following theorem which follows from the Kraft inequality.

Theorem 3.11 (cf. [6])**.** *For any positive integer k, there exists a finite binary string x such that $K(x) > |x| + \log|x| + k$.*

Proof. If this were not true, then there exists a k with $K(x) \leq |x| + \log|x| + k$ for all finite binary strings x. If we combine this inequality with the Kraft inequality, we get that

$$1 \geq \sum_{x\in 2^{<\omega}} 2^{-K(x)} \geq \sum_{x\in 2^{<\omega}} 2^{-|x|}2^{-\log|x|}2^{-k} = \sum_{x\in 2^{<\omega}} \frac{2^{-|x|}2^{-k}}{|x|}. \tag{3.2}$$

Since there are 2^n strings of length n for each n, we can rewrite (3.2) as,

$$1 \geq \sum_{x\in 2^{<\omega}} 2^{-K(x)} \geq \sum_{n=1}^{\infty} 2^n \frac{2^{-n}2^{-k}}{n} = 2^{-k}\sum_{n=1}^{\infty}\frac{1}{n} = \infty,$$

by the divergence of the harmonic series, which gives a clear contradiction. □

One particular theorem regarding prefix-free Kolmogorov complexity that will be useful later is known as the coding theorem. The theorem shows how prefix-free

complexity is related to the concept of maximal discrete semimeasures. A discrete semimeasure is essentially just a function that satisfies a property similar to the Kraft inequality.

Definition 3.12 (Discrete (maximal) semimeasure, cf. [6])**.** A function $m : 2^{<\omega} \to \mathbb{R}$ is called a **discrete semimeasure** if $\sum_{x \in 2^{<\omega}} m(x) \leq 1$. If M is a discrete semimeasure that is also computably enumerable (approximable from below) and $M(x) \geq \mathcal{O}(m(x))$ for every computably enumerable discrete semimeasure, then we say that M is **maximal**.

The Coding theorem basically just says that the function $M : 2^{<\omega} \to \mathbb{R}$ defined by $M(x) = 2^{-K(x)}$ is a maximal discrete semimeasure. The exact form is given below.

Theorem 3.13 (The Coding Theorem, cf. [6])**.** *Let M be a maximal discrete semimeasure. For each finite binary string x, $K(x) = -\log M(x) \pm \mathcal{O}(1)$.*

Proof. See [6] or [8] for a detailed proof. □

3.2 Entropy

One way to view Kolmogorov complexity is as a measure of information content. Strings with high complexity and low algorithmic probability can be thought of as containing more information than strings with low complexity and high algorithmic probability. In this section, we consider another measure of information content called entropy, and we show how it is related to Kolmogorov complexity. Entropy has a long and diverse history in science, particularly in the study of thermodynamics. Long before Claude Shannon introduced entropy in 1948 as a tool for studying information theory (see [9]), physicists were quite familiar with the concept of thermodynamic entropy, which measures the amount of "disorder" present in a system. The definition of entropy from an information theory perspective is given below.

Definition 3.14 (Entropy, cf. [10])**.** Given a probability distribution (μ, S, Σ) on a discrete sample space $S = (x_i)_{i \geq 1}$ and σ-algebra Σ, the **entropy** of μ, written $H(\mu)$, is defined to be the average amount of information contained in a single

observation of a discrete random variable X. The formula for computing $H(\mu)$ is given by,

$$H(\mu) = \sum_i \mu(x_i) \log \frac{1}{\mu(x_i)} = -\sum_i \mu(x_i) \log \mu(x_i) = -E(\log \mu(X)).$$

Entropy determines the information dimension σ_μ of the measure since σ_μ can be written,

$$\sigma_\mu = \lim_{\epsilon \to 0} \frac{H(\mu_\epsilon)}{\log \epsilon}, \tag{3.3}$$

where μ_ϵ is the measure with respect to a mesh of size ϵ. As we will show in Theorem 3.17 below, entropy and Kolmogorov complexity are related; hence, we should expect a similar relationship between information dimension and Kolmogorov complexity. We will explore this topic in Chapter 5.

In the meantime, let us consider some more facts about entropy. An equivalent way to view entropy is in terms of codes. A code is just a function E (the encoding function) from a set of source words S to a set of code words. Given an alphabet of characters, the set of code words is defined to be some subset of the set of finite sequences formed from these characters. For example, Morse code is a function from $S =$ "the set of words in the English language" to the set of finite sequences formed from the characters dash, dot, and two space characters. A binary code is a code from a set of source words S to the set of finite binary strings.

Most codes are not useful unless there is a way to uniquely decode them. A uniquely decodable code is a code that is one-to-one. Prefix codes, i.e., codes that have the property that no codeword is a prefix of another codeword, are uniquely decodable. We will show below that the entropy of a probability distribution is the minimum expected code word length over all possible prefix codes. Since prefix-free Kolmogorov complexity can be used to construct a prefix code on the set of finite binary strings, it is not surprising that entropy and Kolmogorov complexity are related.

Before we demonstrate how entropy and Kolmogorov complexity are related, we prove that entropy is just the minimum expected code word length over all possible prefix codes. This will require first proving the following lemma relating two probability distributions, called Gibb's inequality. Gibb's inequality directly

follows from Jensen's inequality.

Lemma 3.15 (Gibb's Inequality)**.** *If μ, τ are two probability measures on a discrete sample space $S = (x_i)_{i \geq 1}$, then*

$$H(\mu) \leq \sum_i \mu(x_i) \log \frac{1}{\tau(x_i)}. \tag{3.4}$$

Proof. Since the logarithm function is concave, the discrete form of Jensen's inequality states that,

$$\log \left(\frac{\sum_i a_i b_i}{\sum_i a_i} \right) \geq \frac{\sum_i a_i \log(b_i)}{\sum_i a_i}, \tag{3.5}$$

for positive numbers a_i and b_i. If we take $a_i = \mu(x_i)$ and $b_i = \tau(x_i)/\mu(x_i)$, then (3.5) becomes

$$0 \geq \sum_i \mu(x_i) \log \left(\frac{\tau(x_i)}{\mu(x_i)} \right),$$

or equivalently,

$$\sum_i \mu(x_i) \log(\tau(x_i)) \leq \sum_i \mu(x_i) \log(\mu(x_i)) = -H(\mu),$$

which is just (3.4) upon rearranging some terms.

□

Using Gibb's inequality, we can now prove that entropy is the minimum expected code word length over all possible prefix codes. This is part of the statement of the Noiseless Coding Theorem found in [8]. For simplicity, we work exclusively with binary codes.

Theorem 3.16 (cf. [11])**.** *For any binary prefix code $E : S \to 2^{<\omega}$ and probability distribution (μ, S, Σ), the entropy of μ satisfies,*

$$H(\mu) \leq \sum_i \mu(x_i) |E(x_i)|, \tag{3.6}$$

where $|E(x_i)|$ is the length in bits of the code word for x_i.

Proof. Define a probability measure τ on the set of source words S by,

$$\tau(x_i) := \frac{2^{-|E(x_i)|}}{\sum_j 2^{-|E(x_j)|}},$$

for each $x_i \in S$. Then Gibb's inequality (3.4) implies,

$$\begin{aligned} H(\mu) &\leq \sum_i \mu(x_i) \log \left(\frac{\sum_j 2^{-|E(x_j)|}}{2^{-|E(x_i)|}} \right) \\ &= \sum_i \mu(x_i) \log \left(\sum_j 2^{-|E(x_j)|} \right) - \sum_i \mu(x_i) \log \left(2^{-|E(x_i)|} \right) \\ &= \sum_i \mu(x_i) \log \left(\sum_j 2^{-|E(x_j)|} \right) + \sum_i \mu(x_i) |E(x_i)|. \end{aligned} \tag{3.7}$$

Since we are working with a prefix-free set of code words, $\sum_j 2^{-|E(x_j)|} \leq 1$ by the Kraft inequality, and hence $\log \left(\sum_j 2^{-|E(x_j)|} \right) \leq 0$. Equation (3.7) can thus be rewritten as

$$H(\mu) \leq \sum_i \mu(x_i) |E(x_i)|.$$

□

It is possible to achieve equality in (3.6) by constructing the encoding function $E(x)$ in as efficient a manner as possible. A necessary and sufficient condition for equality is to define a code with the property[2] that each code word $E(x_i)$ has length equal to $\log 1/\mu(x_i)$. If $\log 1/\mu(x_i)$ are not all integers, then the ceiling of this value is used instead. This is known as the Shannon-Fano code (see [8]). It has the property that each code word is within one bit of the optimal code word length.

As mentioned above, the relationship between prefix-free Kolmogorov complexity and entropy as measures of information content follows from the fact that prefix-free Kolmogorov complexity can be used to construct a prefix code on the set of finite binary strings. To see this, observe that we can define an encoding function $E : 2^{<\omega} \to 2^{<\omega}$ by $E(x) = y$, where $y \in 2^{<\omega}$ is the string with the shortest length such that $U(y) = x$, with U equal to a universal prefix-free Turing machine. Then

[2] In general, the base of the logarithm used must be equal to the size of the alphabet from which the code words are constructed.

$|E(x_i)| = K(x_i)$ for each finite binary string x_i, and so Theorem 3.16 implies that

$$H(\mu) \leq \sum_i \mu(x_i)K(x_i) = E(K(X)), \tag{3.8}$$

for any probability measure μ, where $E(K(X))$ denotes the expected value of prefix-free Kolmogorov complexity. Of course, this code is not necessarily optimal like the Shannon-Fano code. Although $K(x_i)$ is the length of the shortest binary string that determines x_i, shorter codes are possible by using the probability distribution to explicitly construct the code.

If the probability distribution is effective, then Kolmogorov compexity gives more or less the optimal code (up to a constant).

Theorem 3.17 (cf. [8])**.** *Let μ be a recursive probability measure, with finite Shannon entropy $H(\mu)$. Then,*

$$0 \leq \sum_i \mu(x_i)K(x_i) - H(\mu) \leq K(\mu) + O(1).$$

Proof of Theorem 3.17. The lower bound follows from (3.8) above. To prove the upper bound, it remains to show that

$$\sum_i \mu(x_i)K(x_i) \leq H(\mu) + K(\mu) + \mathcal{O}(1).$$

Let m be the universal semi-computable distribution. By definition, this means

$$\mu(x_i) \leq Cm(x_i),$$

for each string x_i. It is possible to assume WLOG that $C = 2^{K(\mu)}$, with $K(\mu) = \min\{K(j) \mid \mu = \mu_j\}$, where (μ_j) is an enumeration of the set of effective probability measures.[3]

Taking the logarithm of both sides of this inequality and multiplying by -1 gives,

$$\log \frac{1}{\mu(x_i)} \geq \log \frac{1}{m(x)} - K(\mu).$$

[3]see Example 4.3.3 in [8] for details.

Hence,

$$H(\mu) = \sum_i \mu(x_i) \log \frac{1}{\mu(x_i)} \geq \sum_i \mu(x_i) \log \frac{1}{m(x)} - K(\mu).$$

But by the Coding theorem,

$$\log \frac{1}{m(x)} = K(x) + \mathcal{O}(1),$$

and so

$$H(\mu) \geq \sum_i \mu(x_i) K(x) - K(\mu) + \mathcal{O}(1).$$

This completes the proof.

□

Another theorem relating Kolmogorov complexity and Shannon entropy is known as the effective Shannon-McMillan-Breiman theorem. We conclude this chapter by stating the theorem without proof. The proof can be found in the paper by Mathieu Hoyrup [12]. For the definition of μ-randomness, see Definition 4.6.

Theorem 3.18 (The effective Shannon-McMillan-Breiman theorem)**.** *Let μ be a computable shift-invariant probability measure on 2^ω. For each μ-random $x \in 2^\omega$,*

$$H(\mu) = \lim_{n \to \infty} \frac{K(x \upharpoonright n)}{n}.$$

Chapter 4
Dimension and Complexity

In this chapter, we explore the relationship between fractal geometry and algorithmic randomness. The central idea underlying these two fields is the connection between fractal dimension and Kolmogorov complexity of infinite strings. We will demonstrate that for special, well behaved sets of infinite sequences, the Hausdorff dimension of a set E is exactly equal to the complexity of the most complex string in E. As in the last chapter, we limit our discussion to the context of binary strings to keep the concepts as simple as possible. The results presented in this chapter are very recent, having been discovered within the last 15 years by a number of researchers in this field.

Although the connection between fractal dimension and Kolmogorov complexity might sound surprising and mysterious, upon inspection it is actually quite natural and expected. The set of infinite binary strings, denoted 2^ω and suggestively called Cantor space, is analogous to the Cantor set (see Figure 4.1 below). And just as it is possible to construct measures such as the Lebesgue measure and Hausdorff measure on sets of real numbers, there exist analagous constructions on sets of infinite sequences. This makes it possible to formulate a relationship between the two concepts. We begin this chapter by defining the Lebesgue and Hausdorff measures on Cantor space. We then introduce the notion of Martin-Löf randomness, an alternate way of defining algorithmic randomness based on ideas from measure theory, and use it to motivate the definition of effective Hausdorff measure zero and effective Hausdorff dimension. We then proceed to discuss how the Hausdorff dimension of certain sets of infinite strings can be expressed in terms of the Kolmogorov complexity of the individual strings they contain.

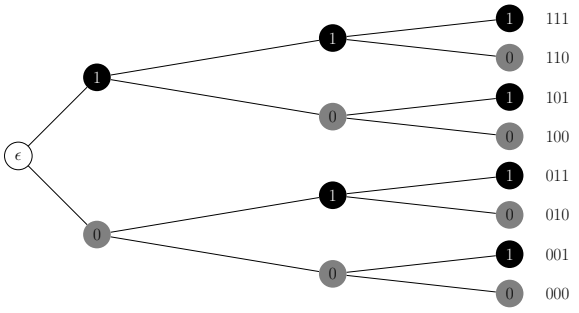

Figure 4.1. Building an infinite binary string in Cantor Space mimics the construction process of selecting a point in the Cantor set. The first three levels are shown here. Compare to Figure 2.1.

4.1 Measure Theory in Cantor Space

To define the Lebesgue and Hausdorff measures on Cantor space, it is necessary to introduce a topology on 2^ω. A countable basis for this topology is given by the set of extensions of finite binary strings. That is, given a finite binary string $x \in 2^{<\omega}$, a basic open set in Cantor space is defined to be the set of all possible infinite extensions of x. This set is called a cylinder set and is denoted $[\![x]\!]$. It is formally defined by $[\![x]\!] = \{xy \mid y \in 2^\omega\}$. A set is called effectively open if it can be written as a union of cylinder sets $\bigcup_{w \in F} [\![w]\!]$ with F recursively enumerable.

The Lebesgue measure of a basic open set $[\![x]\!]$ in Cantor space is defined by $\mu([\![x]\!]) = 2^{-|x|}$, where $|x|$ denotes the length of x in bits. The definition of s-dimensional Hausdorff measure of a subset $E \subseteq 2^{<\omega}$ is based on the concept of an n-cover, which we define below. We then proceed to give the definition of s-dimensional Hausdorff measure. Note the similarity between these two definitions and the corresponding definitions in $\mathbb{R}^n$ (Definitions 2.9 and 2.10).

Definition 4.1 (n-cover, cf. [6])**.** Let E be a subset of 2^ω. An n-cover of E is a subset $F \subseteq 2^{<\omega}$ with the property that $|x| \geq n$ for each $x \in F$, and $E \subseteq \bigcup_{x \in F} [\![x]\!]$.

Definition 4.2 (Hausdorff measure, cf. [6])**.** Let $s \geq 0$ be given, and let E be a subset of 2^ω. The s-dimensional Hausdorff measure of E is the limit

$$\mathcal{H}^s(E) = \lim_{n \to \infty} \mathcal{H}^s_n(E), \tag{4.1}$$

where,

$$\mathcal{H}^s_n(E) = \inf\left\{\sum_{x\in F}\mu([\![x]\!])^s \mid F \text{ is an } n\text{-cover of } E\right\}. \tag{4.2}$$

Here, $\mu([\![x]\!]) = 2^{-|x|}$ denotes the Lebesgue measure of the cylinder set $[\![x]\!]$.

With these definitions in place, the Hausdorff dimension of a subset E in Cantor space is defined as before.

Definition 4.3 (Hausdorff dimension)**.** Let E be a subset of 2^ω. The **Hausdorff dimension** of E is defined to be $\dim_H(E) = \inf\{s \geq 0 \mid \mathcal{H}^s(E) = 0\}$.

Hausdorff dimension over Cantor space satisfies many of the same properties as Hausdorff dimension over Euclidean space (see Section 2.3.3). Recall that Hausdorff dimension is just the point at which graph of $\mathcal{H}^s(E)$ as a function of s jumps from infinity to a finite value. Hence, we could have equivalently defined Hausdorff dimension by setting $\dim_H(E) = \sup\{s \geq 0 \mid \mathcal{H}^s(E) = \infty\}$.

In order to obtain information about the Hausdorff dimension of a set E, it is useful to check whether E has Hausdorff measure zero for a particular value of s. The following two theorems provide us with equivalent conditions that we can use for this purpose.

Theorem 4.4 (cf. [13])**.** *Let E be a subset of 2^ω, and fix a positive real number s. Then $\mathcal{H}^s(E) = 0$ if and only if for every $n \in \mathbb{N}$ there exists a subset $C_n \subseteq 2^{<\omega}$ such that C_n is an m-cover of E for some $m \in \mathbb{N}$ and $\sum_{x\in C_n} 2^{-s|x|} < 2^{-n}$.*

Proof. ($\Rightarrow$) Assume that $\mathcal{H}^s(E) = 0$. Then (4.1) implies that for any n, there is an m such that $\mathcal{H}^s_m(E) < 2^{-n}$. By (4.2), there exists an m-cover C_n with the property that $\sum_{x\in C_n} 2^{-s|x|} < 2^{-n}$ (otherwise $\mathcal{H}^s_m$ would not be the greatest lower bound).

($\Leftarrow$) Assume that for every n there exists $C_n \subseteq 2^{<\omega}$ such that C_n is an m-cover of E for some m and $\sum_{x\in C_n} 2^{-s|x|} < 2^{-n}$. Let $\epsilon > 0$ be arbitary. To show that $\mathcal{H}^s(E) = 0$, we must find an M such that $\mathcal{H}^s_M(E) < \epsilon$.

To this end, pick an integer N such that $\epsilon > 2^{-N}$. By assumption, there is a subset of finite binary strings C_N such that C_N is an M-cover of E for some M and $\sum_{x\in C_N} 2^{-s|x|} < 2^{-N} < \epsilon$. Since $\mathcal{H}^s_M(E)$ is a lower bound of the set $\{\sum_{x\in F}\mu([\![x]\!])^s \mid F \text{ is an } n\text{-cover of } E\}$, we conclude that $\mathcal{H}^s_M(E) < \epsilon$ as required. □

Theorem 4.5 (cf. [13])**.** *Let E be a subset of 2^ω, and fix a positive real number s. Then $\mathcal{H}^s(E) = 0$ if and only if there exists a subset $C \subseteq 2^{<\omega}$ such that $\sum_{x\in C} 2^{-s|x|} < \infty$ and for every $y \in E$, there exist infinitely many $x \in C$ such that x is a prefix of y.*

Proof. ($\Rightarrow$) Assume $\mathcal{H}^s(E) = 0$. By Theorem 4.4, there is a sequence of subsets $(C_n)_{n=1}^\infty$ contained in $2^{<\omega}$ with the property that each C_n is an m_n-cover of E for some $m_n \in \mathbb{N}$ and $\sum_{x\in C_n} 2^{-s|x|} < 2^{-n}$. If we set $C = \bigcup_{n=1}^\infty C_n$, then,

$$\sum_{x\in C} 2^{-s|x|} = \sum_{n=1}^{\infty} \sum_{x\in C_n} 2^{-s|x|} < \sum_{n=1}^{\infty} 2^{-n} = 1 < \infty.$$

Furthermore, since each C_n is an m_n-cover of E, given $y \in E$ there are clearly infinitely many $x \in C$ such that x is a prefix of y.

($\Leftarrow$) Assume that there is a subset $C \subseteq 2^{<\omega}$ with $\sum_{x\in C} 2^{-s|x|} < \infty$ and for each $y \in E$ there are infinitely many $x \in C$ such that x is a prefix of y. To show that $\mathcal{H}^s(E) = 0$, for any $n \in \mathbb{N}$ it is enough to show by Theorem 4.4 that there is an m-cover $C_n \subseteq 2^{<\omega}$ of E such that $\sum_{x\in C_n} 2^{-s|x|} < 2^{-n}$.

To this end, let $m \in \mathbb{N}$ be arbitrary and enumerate the elements of $C = (x_k)_{k=1}^\infty$ by positive integers. Since $\sum_{k=1}^\infty 2^{-s|x_k|}$ converges to a finite value, its tail must converge to zero. Hence, for any integer $n \in \mathbb{N}$, there is an integer N with the property that $\sum_{k\geq N} 2^{-s|x_n|} < 2^{-n}$. Thus, the set $C_n := \{x_k \in C \mid k \geq N\}$ gives us what we need, although we must check that C_n is actually an m-cover of E. While it is possible that C_n contains strings of length smaller than m, we can throw these strings away without loss of generality since the inequality $\sum_{k\geq N} 2^{-s|x_n|} < 2^{-n}$ would still be satisified. Hence, we conclude that C_n is an m-cover of E since each $y \in E$ extends infinitely many strings $x_k \in C_n$.

□

4.2 Martin-Löf Randomness

We have seen that Kolmogorov complexity gives us a way of measuring the inherent randomness of a string or sequence. Objects that are random are difficult to describe using an algorithmic procedure and hence have higher Kolmogorov complexity,

whereas objects that are produced in a recursive manner have low Kolmogorov complexity and are not considered random.

An alternate way of defining randomness uses the ideas of measure theory. Intuitively, a sequence that is random should have no "rare properties." A rare property can be thought of as a sequence of computably enumerable covers whose measure goes to zero. A sequence with no rare properties is one that is not contained in any such cover. This is the underlying idea of Martin-Löf randomness which we define below.

Definition 4.6 (Martin-Löf Randomness, cf. [6])**.** Given a measure μ on 2^ω, a sequence $x \in 2^\omega$ is called **Martin-Löf random** or **μ-random** if for any family $\{U_n\}_{n=1}^\infty$ of effectively open sets with each $U_n \subseteq 2^\omega$ and $\mu(U_n) \leq 2^{-n}$ (called a **Martin-Löf test**), we have $x \notin \bigcap_{n=1}^\infty U_n$. Otherwise, x is called **Martin-Löf null**.

Similarly, a subset $E \subseteq 2^\omega$ is called **Martin-Löf random** if there is no a Martin-Löf test covering E.

It turns out that Martin-Löf randomness is very much related to Kolmogorov complexity. This is the content of the famous Schnorr theorem, which we give next. For a nice proof of Schnorr's theorem, see the text on algorithmic randomness by Downey and Hirschfeldt.

Theorem 4.7 (Schnorr's theorem)**.** *A sequence $x \in 2^\omega$ is Martin-Löf random if and only if for every $n \in \mathbb{N}$, $K(x \upharpoonright n) \geq n - \mathcal{O}(1)$.*

A sequence x with the property that $K(x \upharpoonright n) \geq n - \mathcal{O}(1)$ is called 1-random, so Schnorr's theorem could be expressed as: a sequence is Martin-Löf random if and only if it is 1-random.

The class of Martin-Löf null sets is just the class of effective Lebesgue (1-dimensional Hausdorff) measure zero sets. This class can be generalized to s-dimensional Hausdorff measure for any nonnegative real number s. We say such sets have effective s-dimensional Hausdorff measure zero.

Definition 4.8 (Effective Hausdorff Measure Zero, cf. [13])**.** A set $E \subseteq 2^\omega$ is said to have **effective s-dimensional Hausdorff measure zero**, written Σ_1^0-$\mathcal{H}^s(E) = 0$, if there exists a uniformly recursively enumerable family (C_n), where each $C_n \subseteq 2^{<\omega}$, such that $E \subseteq \bigcap_n \bigcup_{\sigma \in C_n} [\sigma]$ and for each n, $\sum_{\sigma \in C_n} 2^{-|\sigma|s} \leq 2^{-n}$.

The definition of effective Hausdorff measure zero natural leads to a definition of effective Hausdorff dimension.

Definition 4.9 (Effective Hausdorff Dimension, cf. [13])**.** Let E be a subset of 2^ω. The **effective Hausdorff dimension** of E is defined to be $\dim_H(E) = \inf\{s \geq 0 \mid \Sigma^0_1\text{-}\mathcal{H}^s(E) = 0\}$.

4.3 Hausdorff Dimension and Kolmogorov Complexity

There are several different strategies that one can take to prove that Hausdorff dimension and Kolmogorov complexity are related. One popular approach involves introducing the notion of a supergale, a type of function that is also used to model betting strategies of games of chance. This approach was used by Ludwig Staiger [14], Elvira Mayordomo [15], and John Hitchcock [16] in their groundbreaking papers on the subject and is also outlined in the popular textbook on algorithmic randomness by Downey and Hirschfeldt [6]. The approach taken in this thesis follows [13] and uses the concept of a maximal discrete semimeasure defined in the previous chapter (see Definition 3.12). The purpose of proving the relationship in this manner is to avoid having to introduce new notions while keeping the exposition concise.

Kolmogorov complexity is actually related to effective Hausdorff dimension; the reason why essentially follows from the Coding Theorem. As we will see below, if the maximal discrete semimeasure of Definition 3.12 satisifies a special limit for each element of a set E of binary sequences, then E has effective Hausdorff measure zero. Since the maximal discrete semimeasure is related to Kolmogorov complexity via the Coding Theorem (Theorem 3.13), this gives us a way to express effective Hausdorff dimension in terms of Kolmogorov complexity.

The following theorem relates sets of effective Hausdorff measure zero to maximal discrete semimeasures. See [13] for details of the proof.

Theorem 4.10 (cf. [13])**.** *Let E be a subset of 2^ω. Then E has effective Hausdorff measure zero ($\Sigma^0_1\text{-}\mathcal{H}^s(E) = 0$) if and only if, for any $x \in E$ and $s \in \mathbb{R}$ with $s > 0$, we have*

$$\limsup_{n\to\infty} \frac{M(x \upharpoonright n)}{2^{-sn}} = \infty, \tag{4.3}$$

where M denotes the maximal, discrete semimeasure defined in Definition 3.12, and where $x \upharpoonright n$ denotes the first n bits of the infinite binary string x.

As discussed above, the equivalence of effective Hausdorff dimension and Kolmogorov complexity follows from the Coding theorem. We demonstrate this in the proof of the following theorem relating the Hausdorff dimension of a single point in Cantor space to Kolmogorov complexity below.

Theorem 4.11 (cf. [13])**.** *For any infinite binary string x, the effective Hausdorff dimension of $\{x\}$ is related to the Kolmogorov complexity of x via,*

$$\dim_H^1(\{x\}) = \liminf_{n\to\infty} \frac{K(x \upharpoonright n)}{n} \tag{4.4}$$

Proof. First, we prove that the quantity $\alpha := \liminf_{n\to\infty} \frac{K(x\upharpoonright n)}{n}$ is a lower bound of the set $A = \{s \geq 0 \mid \Sigma_1^0\text{-}\mathcal{H}^s(\{x\}) = 0\}$. Since $\dim_H^1(\{x\})$ is by definition the greatest lower bound of A, this would prove that $\alpha \leq \dim_H^1(\{x\})$.

To this end, let $s \in A$. To show that α is a lower bound of A, we must show that $\alpha \leq s$. By the preceding theorem, equation (4.3) holds, so for any integer N there are infinitely many positive integers n such that

$$M(x \upharpoonright n) = N2^{-sn}. \tag{4.5}$$

Taking the base 2 logarithm of both sides of (4.5) and rearranging gives,

$$s = \frac{\log N}{n} - \frac{\log M(x \upharpoonright n)}{n},$$

which can be rewritten as (up to a constant),

$$s = \frac{\log N}{n} + \frac{K(x \upharpoonright n)}{n},$$

by the Coding Theorem (Theorem 3.13). Hence, for infinitely many n we have that

$$\frac{K(x \upharpoonright n)}{n} < s.$$

Taking the limit inferior of both sides as n approaches infinity gives $\alpha \leq s$, and so α is a lower bound of the set A. Thus, $\alpha \leq \dim_H^1(\{x\})$.

To prove the reverse inequality, suppose for the sake of contradiction that $\alpha < \dim_H^1(\{x\})$. Let s be such that $\alpha < s < \dim_H^1(\{x\})$. We show that $\Sigma_1^0\text{-}\mathcal{H}^s(\{x\}) = 0$, contradicting the definition of $\dim_H^1(\{x\})$. To show that $\Sigma_1^0\text{-}\mathcal{H}^s(\{x\}) = 0$, we will use the equivalent conditions given in Theorem 4.5.

Since $\alpha < s$, there are infinitely many n such that $K(x \upharpoonright n) < ns$. Let C be the set of finite binary strings defined by $C = \{y \in 2^{<\omega} \mid K(y) < |y|s\}$. Then clearly the infinite binary string x extends infinitely many finite binary strings $y \in C$. In addition, the series $\sum_{y \in C} 2^{-s|y|}$ converges since,

$$\sum_{y \in C} 2^{-s|y|} < \sum_{y \in C} 2^{-K(y)},$$

by construction. The Coding Theorem allows us to conclude that,

$$\sum_{y \in C} 2^{-K(y)} = \sum_{y \in C} 2^{\log M(y)} = \sum_{y \in C} M(y) \leq 1$$

where M denotes the maximal, discrete semimeasure of Definition 3.12. The last inequality follows from the definition of a semimeasure. Hence, Theorem 4.5 implies that $\{x\}$ has effective Hausdorff measure zero, giving the desired contradiction.[1]

□

In a similar manner, the effective Hausdorff dimension of a subset E of Cantor space is equal to the effective Hausdorff dimension of the most complex element x contained in E. This is known as the pointwise stability property of effective Hausdorff dimension. Unlike Hausdorff dimension, which is only countably stable, effective Hausdorff dimension satisfies a stronger stability property that allows for uncountable unions (absolute stability) [18]. The proof of the pointwise stability property of effective Hausdorff dimension fact can be found in papers by Hitchcock [16] and Staiger [14].

Theorem 4.12 (Pointwise stability of effective Hausdorff dimension)**.** *Let E be a subset of 2^ω. Then the effective Hausdorff dimension of E is given by,*

$$\dim_H^1(E) = \sup_{x \in E} \dim_H^1(\{x\}) = \sup_{x \in E} \left(\liminf_{n \to \infty} \frac{K(x \upharpoonright n)}{n} \right).$$

[1] As shown by Reimann and Stephan [17], the equivalences of Theorem 4.5 are not quite effective. But they are sufficient for the purpose of this proof. For details see [13] and [17].

For special, well behaved sets of infinite sequences, Kolmogorov complexity is related to Hausdorff dimension (not just effective Hausdorff dimension). If working with a subset of Cantor space that is an arbitrary union of Π^0_1 sets, then Hausdorff dimension and effective Hausdorff dimension are equivalent. For the proof, see the paper by Hitchcock [16].

Theorem 4.13. *If E is an arbitrary union of Π^0_1 subsets of 2^ω, then* $\dim_H(E) = \dim^1_H(E)$.

Chapter 5
Theoretical Results

In the last chapter, we saw that the mathematical distinct areas of fractal geometry and algorithmic randomness were united by the bridge of Theorems 4.12 and 4.13. What makes these theorems powerful is the fact that they provide an alternate way to compute the Hausdorff dimension of a given set. It can be rather difficult to directly measure the dimension of a given set using traditional methods, particularly if the set in question contains experimental data. These two theorems change the focus from an analytical perspective to a computational perspective involving the use of data compression techniques.

Ultimately, in this thesis we are interested in whether we can easily apply the results of Theorems 4.12 and 4.13 to measure the fractal dimension of experimental data. The issue with this is that Theorems 4.12 and 4.13 were given in terms of infinite sequences, not points in Euclidean space. Furthermore, any set of experimental data will only be accurate up to a certain precision, so we will not be able to work with infinite sequences. Another potential issue that we must address is the fact that Theorems 4.12 and 4.13 tell us only that the Hausdorff dimension of a set of sequences is equal to a supremum over the set, and so we must be careful to make sure that when we estimate the fractal dimension of experimental data we are actually measuring the supremum or maximum.

Before detailing our results in this area, the answer to some of these questions for special types of fractals generated by iterated function systems can be found in recent work by Jack Lutz and Elvira Mayordomo. An iterated function system (IFS) is just a finite family of contractions $f_1, f_2, \ldots, f_n$ from a closed set X into itself. To generate a fractal F on X using an IFS, construct an infinite sequence using an alphabet Σ equal to the set of contraction indices $\{1, 2, \ldots, n\}$. Then

(cf. [1]) each such sequence $S = \{s_1, s_2, \ldots\}$ in Σ^ω corresponds to a point $x \in F$ by,

$$x = \bigcap_{k=1}^{\infty} f_{s_1} \circ f_{s_2} \circ \cdots \circ f_{s_k}(X). \tag{5.1}$$

The Cantor set and the Sierpinski triangle are examples of fractals generated by iterated function systems. The Cantor set is generated by the IFS consisting of two contractions; namely, $f_1(x) = x/3$ and $f_2(x) = x/3 + 2/3$. The Sierpinski triangle consists of three contractions; f_1 maps a point into the lower left corner of the equilateral triangle, f_2 into the upper corner, and f_3 into the lower right corner.

Lutz and Mayordomo in [18] proved that for a computably self-similar fractal $F \subseteq \mathbb{R}^n$ generated by an IFS, the dimension of an individual point $x \in F$ satisfies,

$$\dim_H(x) = \dim_H(F) \dim_H^1(S), \tag{5.2}$$

where S is any sequence such that (5.1) holds. In this case, the dimension of the point x in Euclidean space is defined by the equation $\dim_H(x) = n \dim_H^1(T)$, where T is any expansion of x as an infinite sequence in any base k (most commonly base 10). The reason this is helpful for us is that if x is a randomly chosen point on the fractal, then by (a Σ-alphabet generalization of) Theorem 4.11, the dimension of its corresponding sequence will be equal to 1. Therefore, for a randomly chosen point x on the fractal, we have $\dim_H(F) = \dim_H(x)$. Hence, to measure the fractal dimension of a set of data, we can just measure the fractal dimension of any random point in the set.

Our results show that we can still compute an estimate of the fractal dimension in the case of imprecise experimental data. Given a sample set of points $\{x_i\}_{i=1}^{\infty}$ on some IFS-generated fractal F, if the $\{x_i\}$ are distributed randomly on F, then for any j, the Hausdorff dimension of F can be measured by compressing the sequence formed by concatening the first j digits of each x_i. In the case that F is not IFS-generated or the distribution of points on the fractal is not uniform, we show that this procedure still measures the information dimension of F.

Before proving these results in Section 5.2 and 5.3, we introduce some new notation and terminology in the next section as well as prove some lemmas and theorems that we will need for the main results.

5.1 Product Measures and Sequence Spaces

Given a finite alphabet $\Sigma = \{1, 2, \ldots, k\}$ and a probability p on Σ, we can construct a probability measure μ on the space of infinite sequences Σ^ω in the following way. First, for each finite string $\sigma \in \Sigma^{<\omega}$, define μ on the cylinder set induced by σ according to the product,

$$\mu([\![\sigma]\!]) = \prod_{i=1}^{|\sigma|} p(\sigma(i)).$$

Then by the Kolmogorov extension theorem (see [19]), μ extends uniquely to a measure on all of Σ^ω.

In a similar manner, we can define a probability measure μ^∞ on the space of infinite Σ^ω-sequences which will be denoted $(\Sigma^\omega)^\infty$. First, we will define μ^∞ on each ∞-cylinder $[\![\overline{\sigma}]\!]^\infty$, the $(\Sigma^\omega)^\infty$ analog of a cylinder.

Definition 5.1 (∞-cylinder)**.** Let $\overline{\sigma}$ be a finite list $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ with each $\sigma_i \in \Sigma^{<\omega}$. Then the ∞-cylinder $[\![\overline{\sigma}]\!]^\infty$ is defined to be the set,

$$[\![\overline{\sigma}]\!]^\infty = \{\overline{x} = (x_1, x_2, \ldots) \in (\Sigma^\omega)^\infty \mid x_j \in [\![\sigma_j]\!] \text{ for all } 1 \leq j \leq n\}.$$

To define μ^∞ on each ∞-cylinder, let

$$\mu^\infty([\![\overline{\sigma}]\!]^\infty) = \prod_{i=1}^{|\overline{\sigma}|} \mu([\![\sigma_i]\!]) = \prod_{i=1}^{|\overline{\sigma}|} \prod_{j=1}^{|\sigma_i|} p(\sigma_i(j)).$$

Then μ^∞ extends uniquely to a measure on all of $(\Sigma^\omega)^\infty$ again by the Kolmogorov extension theorem.

It will be helpful to visualize finite lists of strings $\overline{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ as tables, with each string σ_i corresponding to the ith column of the table. For example, if $\Sigma = \{0, 1\}$, given $\sigma_1 = 100101, \sigma_2 = 001, \sigma_3 = 01011$, then the finite list

$\overline{\sigma} = (\sigma_1, \sigma_2, \sigma_3)$ can be visualized as the table $M = (m_{ij})$,

$$\overline{\sigma} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & - & 1 \\ 0 & - & 1 \\ 1 & - & - \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \\ m_{41} & - & m_{43} \\ m_{51} & - & m_{53} \\ m_{61} & - & - \end{pmatrix}.$$

Think of M is a 6×3 matrix with 4 blank entries, with each $m_{ij} = \sigma_i(j)$.

In a similar manner, sequences in $(\Sigma^\omega)^\infty$ can be viewed as infinite matrices. That is, given a sequence $\overline{x} = (x_1, x_2, \ldots) \in (\Sigma^\omega)^\infty$, the sequence $\overline{x}$ can be viewed as the infinite matrix $M = (m_{ij})$ where each sequence x_j corresponds to the jth column of M.

Next, we outline the details of a famous bijection between Σ^ω and $(\Sigma^\omega)^\infty$ called the Cantor pairing function. This bijection can be used to show that the set of rational numbers $\mathbb{Q}$ and natural numbers $\mathbb{N}$ are in 1-1 correspondence.

Theorem 5.2. *There is a bijection between Σ^ω and $(\Sigma^\omega)^\infty$.*

Proof. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be defined by $f(x, y) = \frac{1}{2}(x + y - 2)(x + y - 1) + x$. It is well known that this function is bijective.

Next, let $\overline{x} = (x_1, x_2, \ldots)$ be a sequence in $(\Sigma^\omega)^\infty$, and let $M = (m_{ij})$ be an infinite matrix corresponding to $\overline{x}$. A bijection F between Σ^ω and $(\Sigma^\omega)^\infty$ is given by $F(\overline{x}) = (y_1, y_2, \ldots) = y \in \Sigma^\omega$, where each $y_k \in \Sigma$ is defined by,

$$y_k = x_i(j),$$

with $f^{-1}(k) = (i, j)$. The first few characters in the sequence y are,

$$(x_1(1), x_1(2), x_2(1), x_3(1), x_2(2), x_3(2), \ldots),$$

etc. Essentially, F just rearranges the elements of the infinite matrix M into a single sequence by tracing out a reverse diagonal pattern thoughout the matrix.

□

In the following lemmas and theorems, we will use the notation $\bigoplus_{i \in \mathbb{N}} x_i$ to

denote the image of $\overline{x} \in (\Sigma^\omega)^\infty$ under the bijection F described above.

Next, we use this well known 1-1 correspondence between Σ-sequences and Σ^ω sequences to prove a relationship between cylinders and ∞-cylinders. Although there is a bijection between Σ^ω and $(\Sigma^\omega)^\infty$, the relationship between cylinders and ∞-cylinders is not as strong. This is the content of the following lemma.

Lemma 5.3. *The following statements hold:*

(a) There is an injection Φ from the set of cylinders in Σ^ω to the set of ∞-cylinders in $(\Sigma^\omega)^\infty$. Furthermore, if A is a cylinder in Σ^ω, then $\mu(A) = \mu^\infty(\Phi(A))$.

(b) There is an injection Ψ from the the set of ∞-cylinders in $(\Sigma^\omega)^\infty$ to the set of all finite unions of cylinders in Σ^ω. Furthermore, if A^∞ is an ∞-cylinder in $(\Sigma^\omega)^\infty$, then $\mu^\infty(A^\infty) = \mu(\Psi(A^\infty))$.

Proof. (a) The injection Φ is built using the natural bijective correspondence between Σ-sequences and Σ^ω-sequences. Given a cylinder $A = [\![\sigma]\!]$ for some finite length-k string $\sigma \in \Sigma^{<\omega}$, arrange the k characters of σ into rows and columns along the reverse diagonals of a table $M = (m_{ij})$ as in the proof of Theorem 5.2 (let $m_{11} = \sigma(1)$, $m_{12} = \sigma(2)$, $m_{21} = \sigma(3)$, $m_{13} = \sigma(4)$, etc.). Let M_i denote the finite string which represents the ith column of the table M, and let k denote the total number of columns of M. Then Φ can be defined by setting $\Phi(A) = [\![\overline{\sigma}]\!]^\infty \subseteq (\Sigma^\omega)^\infty$, where $\overline{\sigma} = (M_1, M_2, \ldots, M_k)$. Furthermore, if $B \neq A$ is a different cylinder in Σ^ω, then clearly $\Phi(B) \neq \Phi(A)$ since the corresponding table used to define Φ cannot be the same.

Next, given a cylinder $A = [\![\sigma]\!]$ in Σ^ω, we show that $\mu(A) = \mu^\infty(\Phi(A))$. By definition of the product measure on Σ^ω, we have $\mu(A) = \prod_{i=1}^{|\sigma|} \tau(\sigma(i))$. Let M be the table as constructed above using the finite string σ. By definition of the product measure on $(\Sigma^\omega)^\infty$, we have

$$\mu^\infty(\Phi(A)) = \prod_{i=1}^{k} \mu(M_i) = \prod_{i=1}^{k} \prod_{j=1}^{n_i} \tau(m_{ij}), \tag{5.3}$$

where k is the number of columns of M and n_i is the number of entries in the ith column of M. Since each of the characters of σ appear only once in the table M (i.e., since $\sum_{i=1}^{k} n_i = |\sigma|$ by construction), we conclude that $\mu(A) = \mu^\infty(\Phi(A))$.

(b) Let $A^\infty = [\![\overline{\sigma}]\!]$ be an ∞-cylinder in $(\Sigma^\omega)^\infty$ defined by $\overline{\sigma} = (M_1, M_2, \ldots, M_k)$, with each M_i being a finite string of length n_i. Arrange the strings $M_1, \ldots, M_k$ as columns of a table M. Unlike the table constructed in part (a) of the proof above (where $n_1 > n_2 > \cdots > n_k$), the columns of M are not necessarily ordered by size. Adjacent columns of M may or may not have the same size. This makes it difficult to trace out a reverse diagonal pattern (as in part (a)) to define a cylinder $[\![\sigma]\!]$ in Σ^ω. To make it work, we must expand M into a larger table M^* with the property that $n_1^* > n_2^* > \cdots > n_k^*$, where each n_i^* in this case refers to the length of the ith column of M^*. Then we could construct a cylinder $[\![\sigma]\!]$ by tracing out a reverse diagonal pattern.

Unfortunately, we cannot simply define $\Psi(A^\infty) = [\![\sigma]\!]$. The problem with this suggestion is that Ψ would not be injective. There are many possible tables M^* that expand M since the additional entries of M^* can freely range over the alphabet Σ. In other words, each expansion M^* of M depends on which characters we use to fill in the gaps in the original table. The good news is that we only need to fill in finitely many such gaps. This is why the set of ∞-cylinders in $(\Sigma^\omega)^\infty$ injects into the set of finite unions of cylinders in Σ^ω and not strictly the set of cylinders in Σ^ω.

To see that $\mu^\infty(A^\infty) = \mu(\Psi(A^\infty))$, let $[\![\sigma_1]\!], \ldots, [\![\sigma_N]\!]$ be cylinder sets whose union is the output of $\Psi(A^\infty)$. Let $s = |\sigma_1| = \cdots = |\sigma_N|$. Observe that $\mu^\infty(A^\infty) = \prod_{i=1}^k \mu(M_i)$ by definition of the product measure on $(\Sigma^\omega)^\infty$. Similarly, by definition of the product measure on Σ^ω,

$$\mu(\Psi(A^\infty)) = \sum_{i=1}^{N} \prod_{j=1}^{|\sigma_i|} \tau(\sigma_i(j)).$$

In the sum above, the terms corresponding to entries shared by all the finite strings $\sigma_1, \ldots, \sigma_N$ (i.e., the entries corresponding to the original table M), can be factored out as the product $\prod_{i=1}^k \mu(M_i)$. What is left is the sum over the products of every possible outcome of the remaining $s - k$ entries, which must equal 1. Hence, this proves that $\mu^\infty(A^\infty) = \mu(\Psi(A^\infty))$.

□

We can now prove that Martin-Löf randomness of a point $\overline{x} \in (\Sigma^\omega)^\infty$ ensures that $\bigoplus_{i\in\mathbb{N}} x_i$ is Martin-Löf random in Σ^ω, and vice versa.

Theorem 5.4. *A Σ^ω-sequence $\overline{x} = (x_1, x_2, \ldots)$ is μ^∞-random if and only if the Σ-sequence $\bigoplus_{i\in\mathbb{N}} x_i$ is μ-random.*

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{i\in\mathbb{N}} x_i =: x^\oplus$ is not μ-random. We must show that $\overline{x}$ is not μ^∞-random. Since $x^\oplus \in \Sigma^\omega$ is not μ-random, it is Martin-Löf null, i.e., there is a Martin-Löf test $(U_n)_{n\in\mathbb{N}} \subseteq \Sigma^\omega$ that covers $x^\oplus$, with each subset U_n effectively open. We will use the test $(U_n)_{n\in\mathbb{N}}$ to construct a test $(U_n^\infty)_{n\in\mathbb{N}} \subseteq (\Sigma^\omega)^\infty$ that covers $\overline{x}$.

To this end, note that the fact that $x^\oplus$ is Martin-Löf null means that $x^\oplus \in \bigcap_{n\in\mathbb{N}} U_n$ and $\mu(U_n) \leq 2^{-n}$ for each $n \in \mathbb{N}$. Furthermore, since each subset U_n is effectively open, for each $n \in \mathbb{N}$ there exists an at most countable set of finite strings $\{\sigma_1^{(n)}, \sigma_2^{(n)}, \ldots\}$ with the property that

$$U_n = \bigcup_{i\geq 1} [\![\sigma_i^{(n)}]\!]. \tag{5.4}$$

By Lemma 5.3, each cylinder set $[\![\sigma_i^{(n)}]\!]$ in Σ^ω corresponds to exactly one ∞-cylinder set $[\![\overline{\sigma}_i^{(n)}]\!]^\infty$ in $(\Sigma^\omega)^\infty$. It follows that $(U_n^\infty)_{n\in\mathbb{N}}$ defined by

$$U_n^\infty = \bigcup_{i\geq 1} [\![\overline{\sigma}_i^{(n)}]\!]^\infty \tag{5.5}$$

is a Martin-Löf test in $(\Sigma^\omega)^\infty$ that covers $\overline{x}$.

Indeed, since each U_n covers $x^\oplus$, for each n there is a cylinder set $[\![\sigma_j^{(n)}]\!]$ containing $x^\oplus$ by (5.4). The corresponding ∞-cylinder $[\![\overline{\sigma}_j^{(n)}]\!]$ must contain $\overline{x}$, and so $(U_n^\infty)_{n\in\mathbb{N}}$ covers $\overline{x}$. Furthermore, Lemma 5.3 implies that $\mu^\infty([\![\overline{\sigma}_i^{(n)}]\!]) = \mu([\![\sigma_i^{(n)}]\!])$ for each cylinder set, and so $\mu^\infty(U_n^\infty) = \mu(U_n) \leq 2^{-n}$ for each $n \in \mathbb{N}$. Therefore, $\overline{x}$ is Martin-Löf null and hence not μ^∞-random.

($\Leftarrow$) Suppose that $\overline{x}$ is not μ^∞-random. We must show that $\bigoplus_{i\in\mathbb{N}} x_i =: x^\oplus$ is not μ-random. The strategy is essentially the same as the reverse direction. Since $\overline{x}$ is not μ^∞-random, there exists a Martin-Löf test of effectively open sets $(U_n^\infty)_{n\in\mathbb{N}}$ that covers $\overline{x}$. Using this test, it is possible to construct another test $(U_n)_{n\in\mathbb{N}}$ that covers $x^\oplus$.

Indeed, each subset U_n^∞ can be written as a union of ∞-cylinders as in (5.5). By Lemma 5.3, each ∞-cylinder set $[\![\overline{\sigma}_i^{(n)}]\!]^\infty$ corresponds to a finite union of cylinder

sets, $\bigcup_{j=1}^{k}[\![\sigma_{i,j}^{(n)}]\!]$. It follows that the test $(U_n)_{n\in\mathbb{N}}$ defined by,

$$U_n = \bigcup_{i\geq 1}\bigcup_{j=1}^{k}[\![\sigma_{i,j}^{(n)}]\!],$$

is a Martin-Löf test in Σ^ω covering $x^\oplus$, where the total number of cylinders k depends on each index i. The fact that $\mu(\bigcup_{j=1}^{k}[\![\sigma_{i,j}^{(n)}]\!]) = \mu^\infty([\![\overline{\sigma}_i^{(n)}]\!]^\infty)$ follows again from Lemma 5.3, and so $\mu(U_n) = \mu^\infty(U_n^\infty) =\leq 2^{-n}$ for each $n \in \mathbb{N}$. Thus, $x^\oplus$ is Martin-Löf null and hence not μ-random.

□

A useful corollary of Theorem 5.4 holds for subsequences of $\bigoplus_{i\in\mathbb{N}} x_i$.

Corollary 5.5. *If $\overline{x} = (x_1, x_2, \ldots) \in \Sigma^\omega$ is μ^∞-random, then any computable subsequence of $\bigoplus_{i\in\mathbb{N}} x_i$ is μ-random.*

Proof. Since $\bigoplus_{i\in\mathbb{N}} x_i$ is μ-random, it is enough to show that any computable subsequence of a μ-random sequence is also μ-random.

Let y be a computable subsequence of a μ-random sequence $z \in \Sigma^\omega$. Assume that y is not μ-random. Since y is a subsequence of z, there exist indices $n_1, n_2, \ldots$ such that $y = (z(n_1), z(n_2), \ldots)$. Then a Martin-Löf test $(U_i)_{i\in\mathbb{N}}$ that covers z can be constructed in the following manner. Let each U_i be defined by,

$$U_i = \{w \in \Sigma^\omega \mid w(n_1) = z(n_1), w(n_2) = z(n_2), \ldots, w(n_i) = z(n_i)\}.$$

Since for each i there are finitely many strings $\tau_j^{(i)} \in \Sigma^{<\omega}$ of length n_i such that

$$\tau_j^{(i)}(n_1) = z(n_1),\ \tau_j^{(i)}(n_2) = z(n_2),\ \ldots,\ \tau_j^{(i)}(n_i) = z(n_i),$$

each U_i can be written as the finite union,

$$U_i = \bigcup_j [\![\tau_j^{(i)}]\!],$$

which has measure 2^{-i}. This contradicts the fact that z is μ-random.

□

5.2 Estimating Hausdorff Dimension

Our first main result follows immediately from the preceding discussion. By a *uniform* iterated function system in the statement of the theorem below, we mean one in which each of the images $f_1(X), f_2(X), \ldots, f_k(X)$ have the same contraction ratio, as in the case of the Cantor set and Sierpinski triangle.

Theorem 5.6. *Let $F \subseteq X \subseteq \mathbb{R}^n$ be a computably self-similar fractal generated by a uniform iterated function system $f_1, f_2, \ldots, f_k$, and let $\Sigma = \{1, 2, \ldots, k\}$. Let μ be the uniform, recursive measure on Σ^ω. Then if $\overline{S} = (S_i)_{i=1}^\infty \in (\Sigma^\omega)^\infty$ is a μ^∞-random sequence, with each $S_i \in \Sigma^\omega$,then given any computable subsequence $\tilde{S}$ of $\bigoplus_{i=1}^\infty S_i$, we have,*

$$\dim_H(F) = \dim_H(x), \tag{5.6}$$

where $x \in F$ corresponds to the address string $\tilde{S}$.

Proof. The preceding corollary implies that $\tilde{S}$ is μ-random. The comments following equation (5.2) show that $\dim_H(F) = \dim_H(x)$. Hence, (5.6) follows from Theorems 4.12 and 4.13.

□

5.3 Estimating Information Dimension

It is important to note that Theorem 5.6 only holds when the fractal F is generated by an IFS. Otherwise, equation (5.2) does not necessary hold. In the event that F is not generated by an IFS, it is still possible to measure the information dimension of the fractal using Kolmogorov complexity. In fact, it is even possible in an ideal setting to construct such an estimator by concatenating a sequence of truncated data together. We show how information dimension and Kolmogorov complexity are related below.

Theorem 5.7. *Let d, m, n be positive integers. Let μ be a recursive measure on $F = [0,1]^d \subseteq \mathbb{R}^d$, and let $(x_i)_{i=1}^\infty$ be a μ^∞-random sequence of points in F. Define*

a sequence of intervals by

$$E_j := \begin{cases} [j \cdot 2^{-m}, (j+1) \cdot 2^{-m}), & \text{for } 0 \leq j \leq 2^m - 2, \\ [j \cdot 2^{-m}, (j+1) \cdot 2^{-m}], & \text{for } j = 2^m - 1. \end{cases}$$

Let $\mathcal{G}$ be the grid of cubes consisting of every possible product of intervals $E_{r_1} \times E_{r_2} \times \cdots \times E_{r_{2^m}}$, with each $r_j \in \{0, 1, \ldots, 2^m - 1\}$. To each interval E_j, assign an index $I_j^{(m)}$ equal to the m-bit binary representation of j. To each cube $E_{r_1} \times E_{r_2} \times \cdots \times E_{r_{2^m}}$ in $\mathcal{G}$, assign the md-bit index $I_{r_1}^{(m)} + I_{r_2}^{(m)} + \cdots + I_{r_{2^m}}^{(m)}$. To each point x_i, assign an index $X_i^{(m)}$ equal to the md-bit index corresponding to the cube $E_{r_1} \times E_{r_2} \times \cdots \times E_{r_{2^m}}$ containing x_i. Then,

$$T_n^m := \frac{K(X_1^{(m)} + X_2^{(m)} + \cdots + X_n^{(m)})}{nm} \to \sigma_\mu$$

as n and m approach infinity.

Proof. Fix a positive integer n. Let $\mathcal{S} := (x_i)_{i=1}^n$. For each positive integer m and cube $E = E_{r_1} \times E_{r_2} \times \cdots \times E_{r_{2^m}} \in \mathcal{G}$, let μ_m be the probability distribution defined by

$$\mu_m(E) := \frac{|\mathcal{S} \cap E|}{n}.$$

Next, since $(X_i^{(m)})_{i=1}^n$ is a μ^∞-random sequence, prefix-free symmetry of information (see [20]) allows us to write

$$T_n^m \stackrel{+}{=} \frac{K(X_1^{(m)}) + K(X_2^{(m)}) + \cdots + K(X_n^{(m)})}{nm}. \tag{5.7}$$

But each of the index strings $X_i^{(m)}$ corresponds to an index I of a particular cube $E = E_{r_1} \times E_{r_2} \times \cdots \times E_{r_{2^m}}$ in $\mathcal{G}$. By counting the number of duplicate indices in the sum $K(X_1^{(m)}) + K(X_2^{(m)}) + \cdots + K(X_n^{(m)})$, we can rewrite (5.7) as,

$$\begin{aligned} T_n^m &\stackrel{+}{=} \frac{\sum_{E \in \mathcal{G}} |\mathcal{S} \cap E| \cdot K(I)}{nm} \\ &= \frac{\sum_{E \in \mathcal{G}} \mu_m(E) \cdot K(I)}{m} \end{aligned}$$

$$= \frac{E(K(X))}{-\log \epsilon},$$

where I is the index of the cube E and $\epsilon = 2^{-m}$. By $E(K(X))$ we mean the expected value of $K(X)$ with respect to the measure μ_m.

As m and n go to infinity, ϵ goes to zero, μ_m goes to μ, and hence T_n^m goes to σ_μ by Theorem 3.17.

□

Chapter 6
Numerical Experiments

The theoretical results of the previous chapter are only useful for computing the fractal dimension of real world data if we have a way to approximate Kolmogorov complexity. As we have seen, Kolmogorov complexity is non-computable, so we must use other data compression techniques to approximate it. In this section we consider two alternate measures of complexity: Lempel-Ziv complexity, which is the foundation of many compression programs in use today (e.g., gzip in Linux), as well as a more recent measure of complexity developed by Verónica Becher and Pablo Heiber called I-complexity.

6.1 Normal Compressor Tests

To check that Lempel-Ziv complexity and I-complexity have desirable properties that mimic Kolmogorov complexity, we follow a strategy outlined by Rudi Cilibrasi and Paul Vitányi in their 2005 paper "Clustering by Compression" [21]. Previous results of Vitányi et al. (see [22]) established the existence of a universal metric, called the normalized information distance, based on Kolmogorov complexity for detecting similarities in data sets. Cilibrasi and Vitányi were interested in using real world data compressors to approximate the normalized information distance. They defined the notion of a normal compressor for doing so, identifying four desirable properties that real world compressors should possess to mimic Kolmogorov complexity. The four properties are idempotency, monotonicity, symmetry, and distributivity.

Definition 6.1 (Normal compressor, cf. [21])**.** Let $x, y, z \in \Sigma^{<\omega}$, where Σ is a finite alphabet, and let λ denote the empty string. A compressor C is normal if it

satisfies, up to an additive $\mathcal{O}(\log n)$ term, the following axioms:

(1) Idempotency: $C(xx) = C(x)$ and $C(\lambda) = 0$.

(2) Monotonicity: $C(xy) \geq C(x)$.

(3) Symmetry: $C(xy) = C(yx)$.

(4) Distributivity: $C(xy) + C(z) \leq C(xz) + C(yz)$.

In this section, we present the results of several numerical experiments designed to check whether Lempel-Ziv complexity and I-complexity satisfy idempotency and symmetry. We also briefly describe how each complexity measure is defined.

6.1.1 Lempel-Ziv Complexity

In 1976, Abraham Lempel and Jacob Ziv introduced their measure of complexity based on the concept of the "production complexity" of a sequence [23]. The idea is based on the fact that many sequences can be built using a recursive procedure out of their initial substrings. In other words, given the first n bits of a sequence x, it may be possible to write a program that computes the next bit of x. If $x = 10010111001$ but we were only given the first seven bits of x (1001011), for example, then we could still generate all of x by writing a program that computes the last four bits of x (just write a program that copies the first four bits of x and concatenates the result). The last four bits of x contain no new information given the first seven.

The Lempel-Ziv (LZ) complexity of a string S is defined to be the minimum number of components that S can be decomposed into such that each component is not a substring of the previous components (with the possible exception of the last component). Each component in such a parsing of S contains some new information that cannot be obtained by just looking at the substrings of the previous components. The LZ complexity of the string x given above, for example, is equal to five since x can be decomposed into five such components: $1 \cdot 0 \cdot 01 \cdot 011 \cdot 1001$. Each component is not a substring of any of the previous components (except for the last one in this case).

To check that LZ-complexity satisfies idempotency and symmetry, we fixed an alphabet $\Sigma = \{0, 1\}$ and randomly generated between 1,000 to 10,0000 strings x

and y of a specified length. We then created a histogram of the percent difference between $LZ(x)$ and $LZ(xx)$ as well as $LZ(xy)$ and $LZ(yx)$. The graphs were noisy for short words (less than 100 characters) but seemed to converge as the word length grew larger (at the sacrifice of computational time). LZ-complexity can be seen to clearly satisfy idempotency as the length of the word increases, while symmetry has exponentially decaying error.

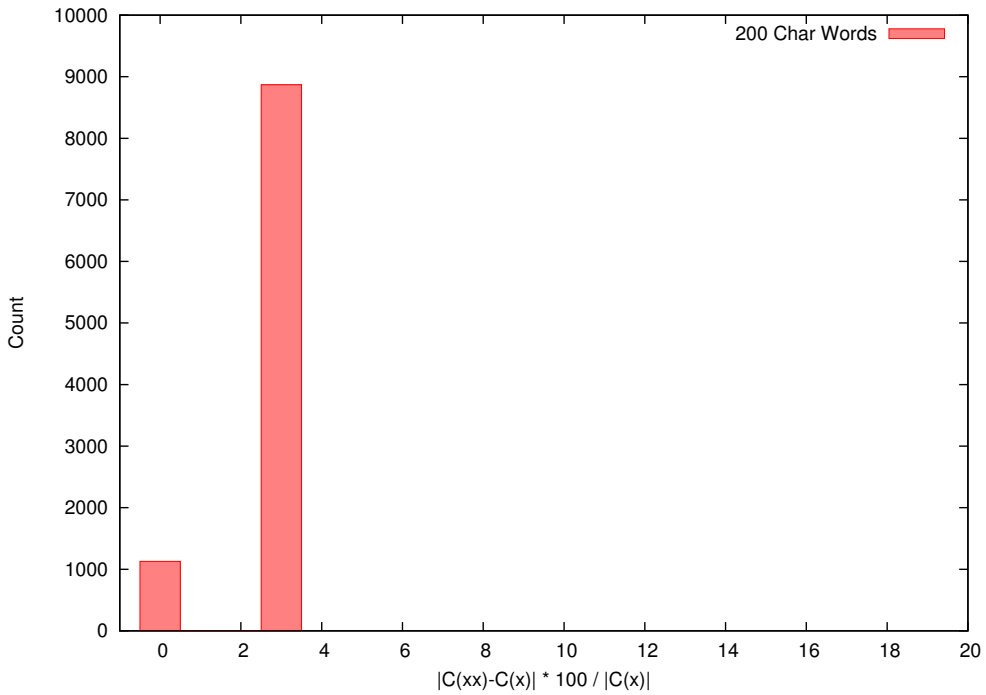

Figure 6.1. Histogram of 10,000 words each 200 characters long binned by percent difference from idempotency using LZ-complexity.

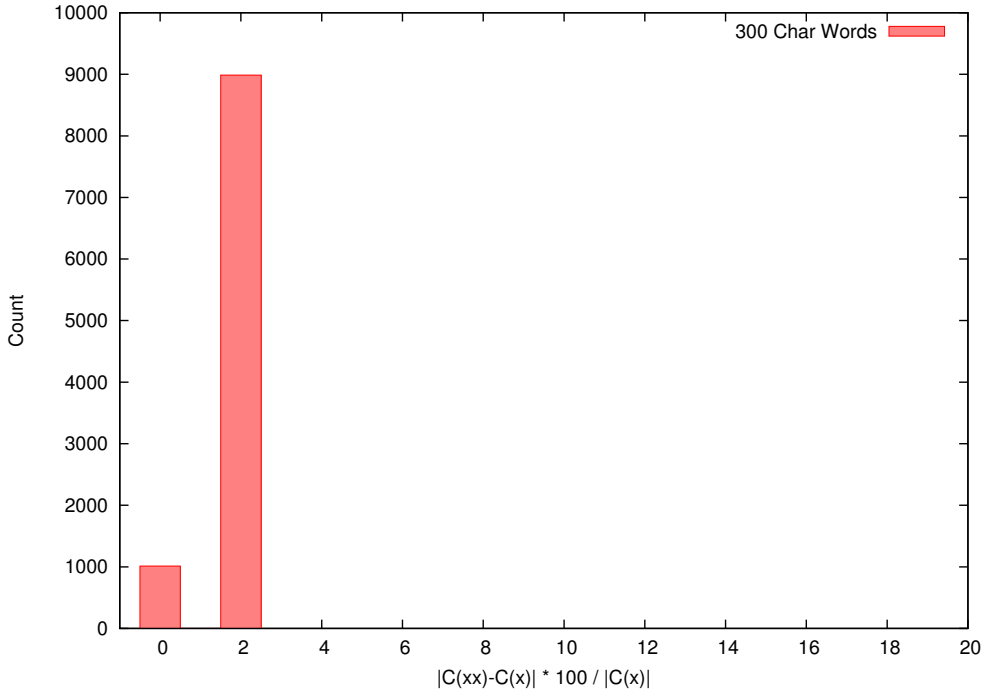

Figure 6.2. Same experiment as in Figure 6.1 but using words 300 characters long each.

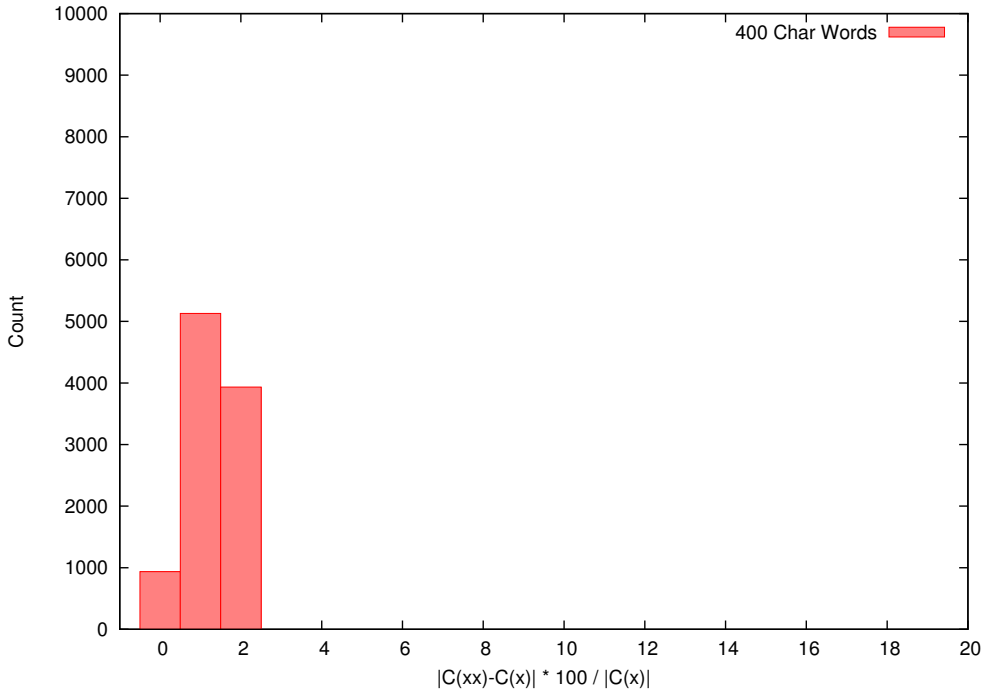

Figure 6.3. Same experiment as in Figure 6.1 but using words 400 characters long each.

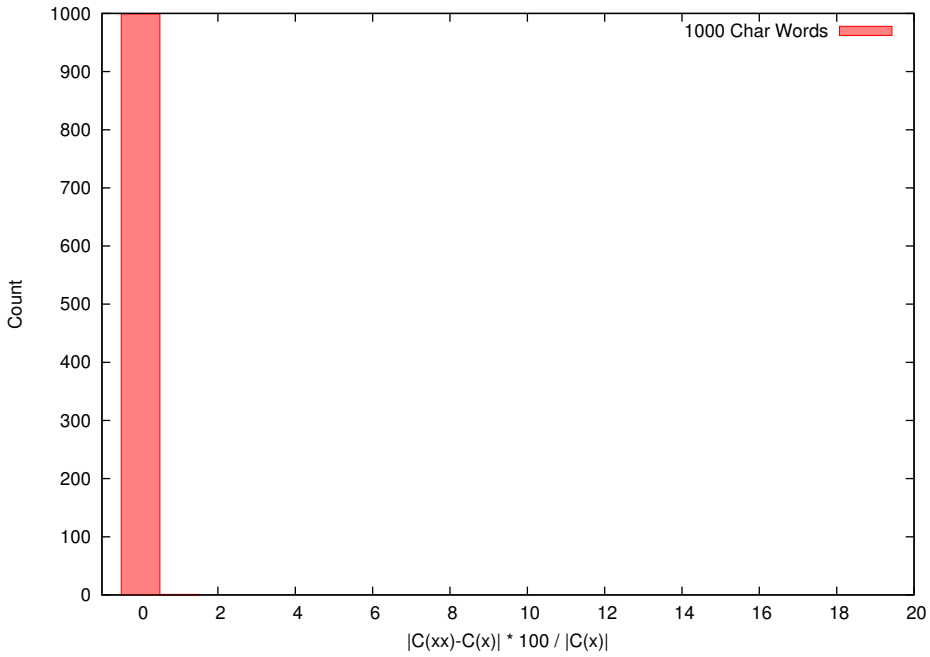

Figure 6.4. Same experiment as in Figure 6.1 but using words 1,000 characters long each.

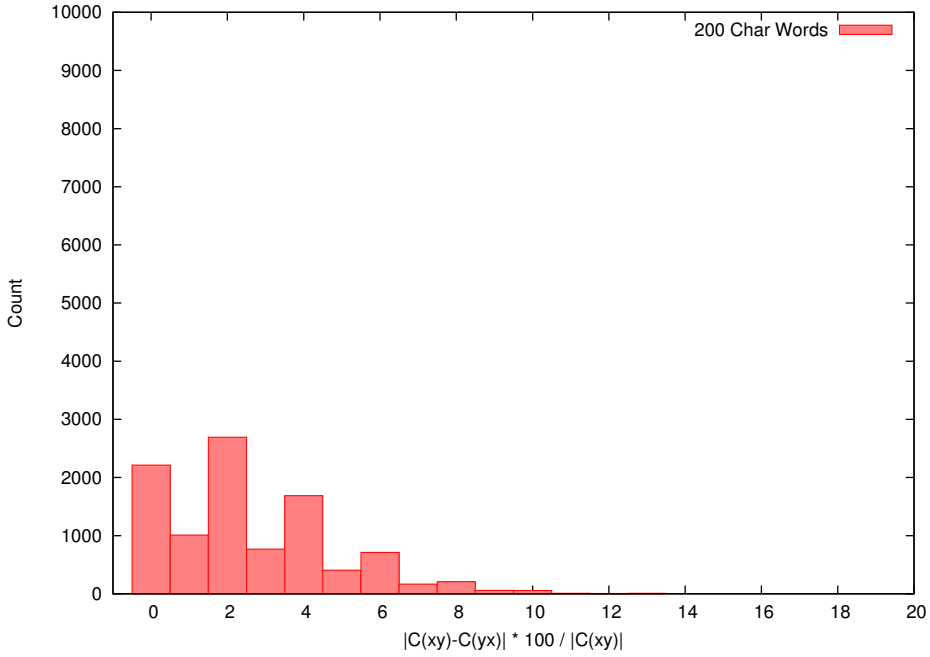

Figure 6.5. Histogram of 10,000 words each 200 characters long binned by percent difference from symmetry using LZ-complexity.

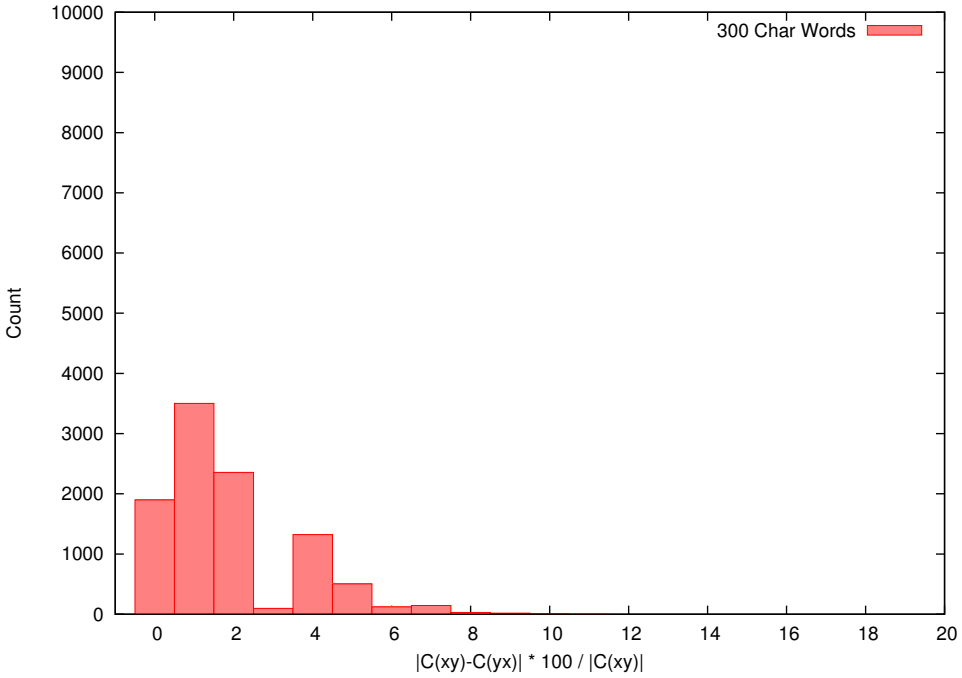

Figure 6.6. Same experiment as in Figure 6.5 but using words 300 characters long each.

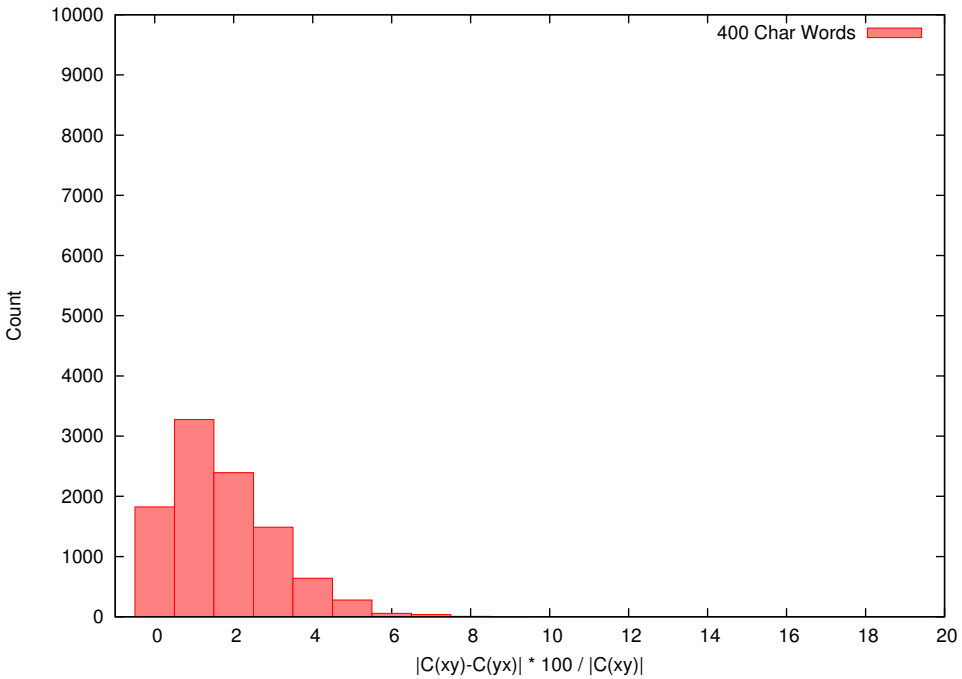

Figure 6.7. Same experiment as in Figure 6.5 but using words 400 characters long each.

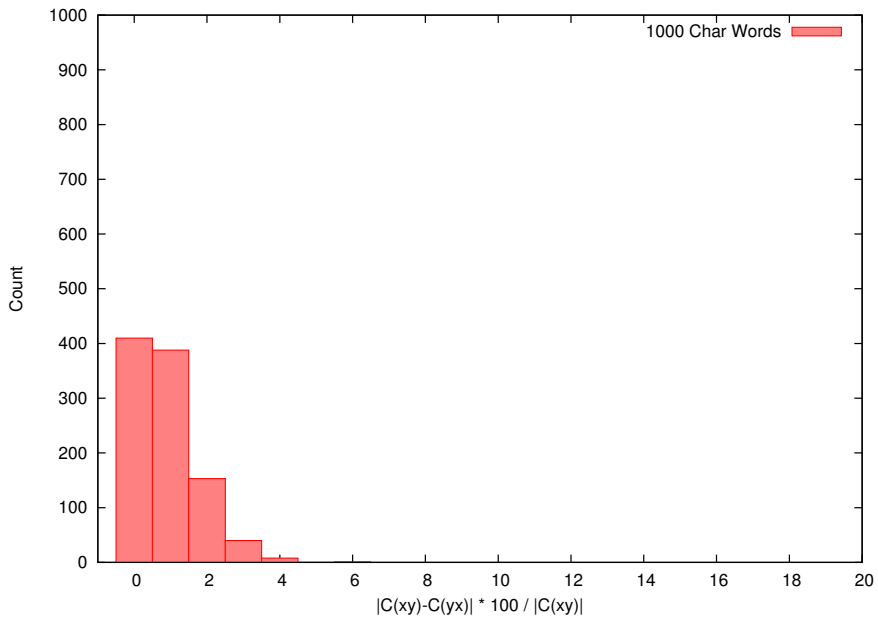

Figure 6.8. Same experiment as in Figure 6.5 but using 1,000 total words each 1,000 characters long.

6.1.2 I-Complexity

Another measure of complexity similar to Lempel-Ziv was recently introduced by Becher and Heiber in 2012 called I-complexity [24]. A shortcoming of Lempel-Ziv is that for each positive integer n, there are infinitely many strings with Lempel-Ziv complexity equal to n. This is not the case with I-complexity; the number of strings with I-complexity equal to a given value is bounded. I-complexity is also easier to work with theoretically as it is based on a more combinatorial approach. Its definition, however, is less intuitive. The I-complexity of a string $x \in \Sigma^{<\omega}$ is defined to be the sum,

$$I(x) = \sum_{i=1}^{|x|} \left(log_\alpha(B_x(i) + 2) - \log_\alpha(B_x(i) + 1) \right),$$

where α is the size of the finite alphabet $|\Sigma|$. The string B_x is called the Backwards Repetition Array and stores the length of the largest repeated substring at the ith position of x.

We repeated similar experiments for I-complexity as was discussed above for Lempel-Ziv complexity.

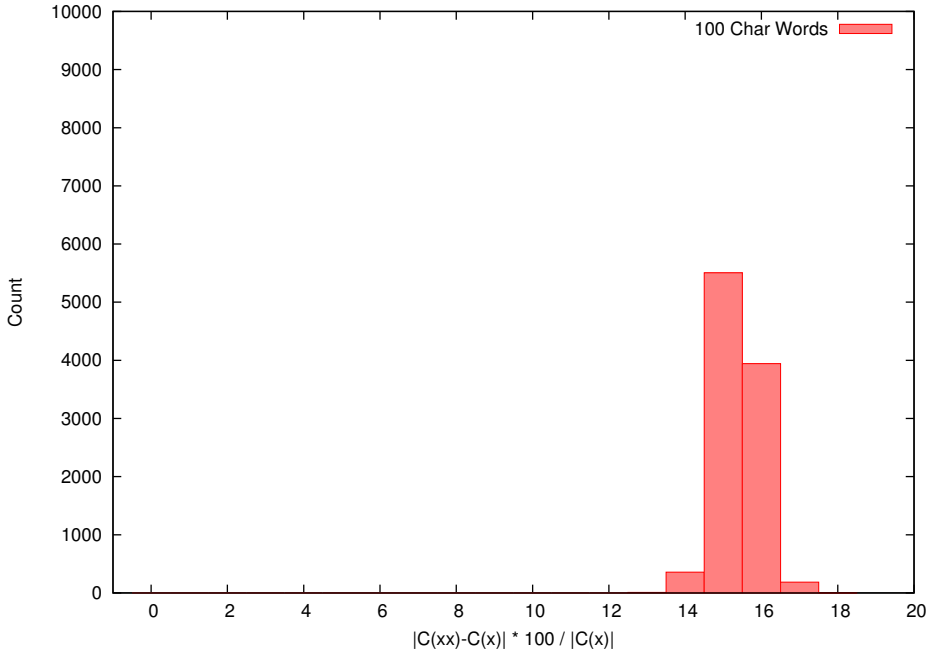

Figure 6.9. Histogram of 10,000 words each 100 characters long binned by percent difference from idempotency using I-complexity.

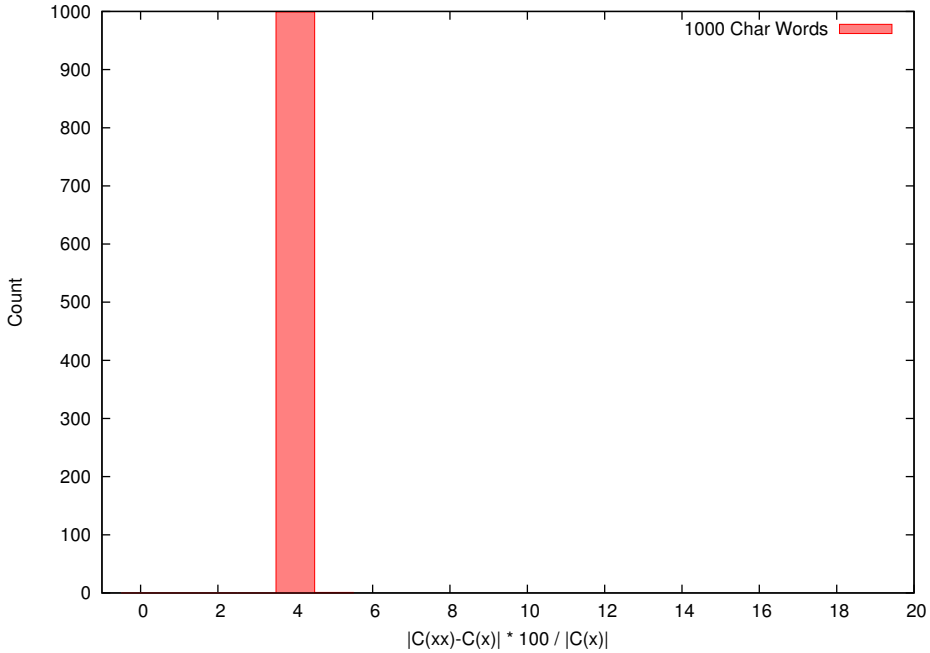

Figure 6.10. Same experiment as in Figure 6.9 but using 1,000 total words each 1,000 characters long.

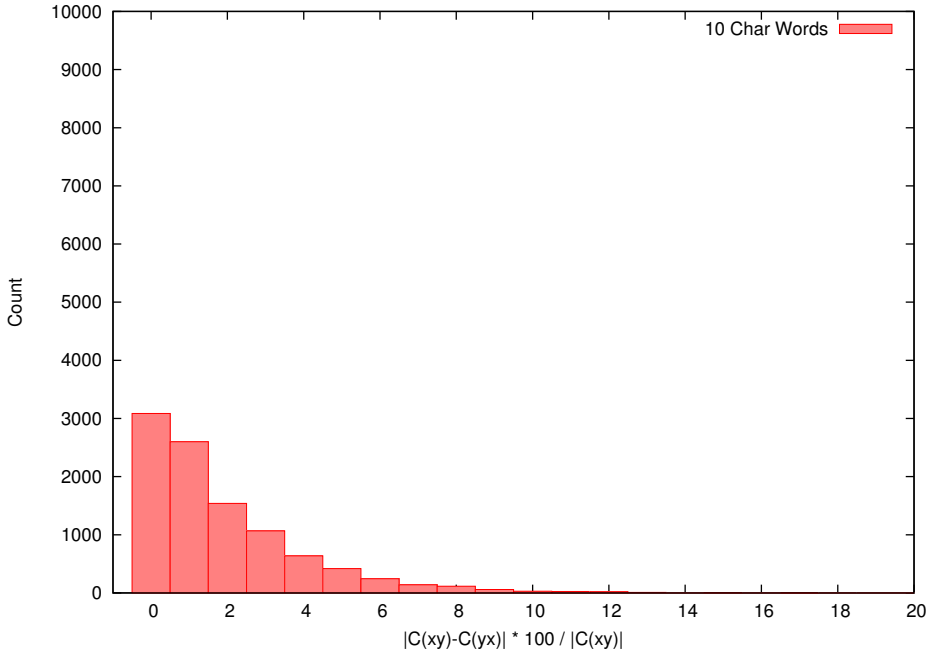

Figure 6.11. Histogram of 10,000 words each 10 characters long binned by percent difference from symmetry using I-complexity.

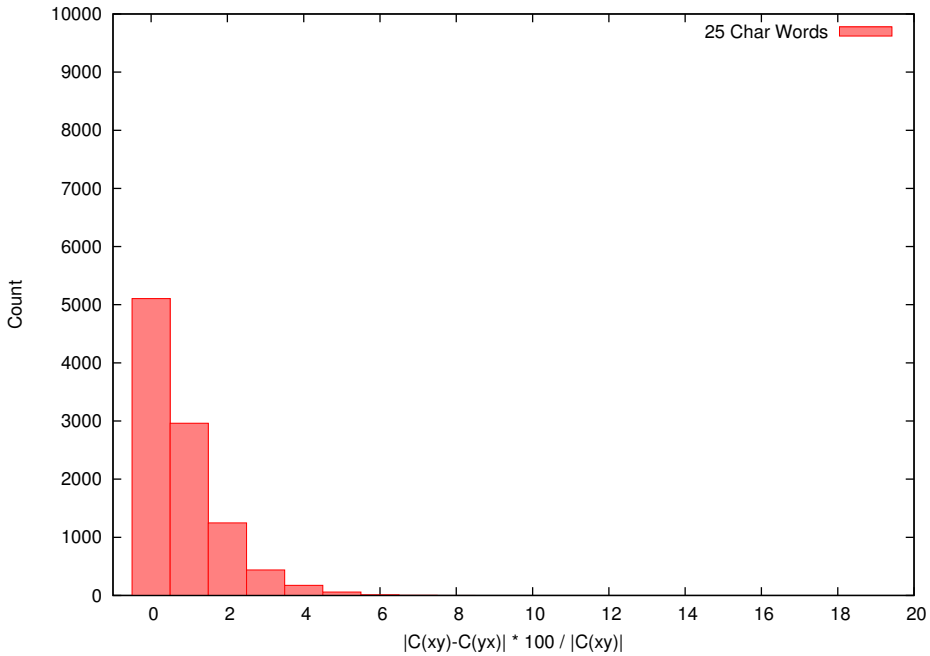

Figure 6.12. Same experiment as in Figure 6.11 but using words 25 characters long each.

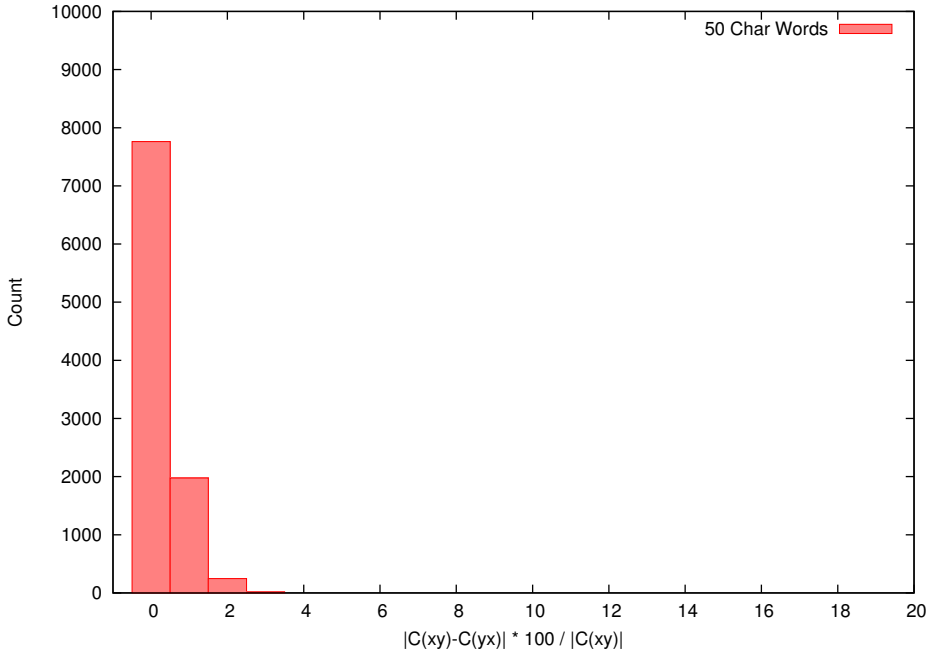

Figure 6.13. Same experiment as in Figure 6.11 but using words 50 characters long each.

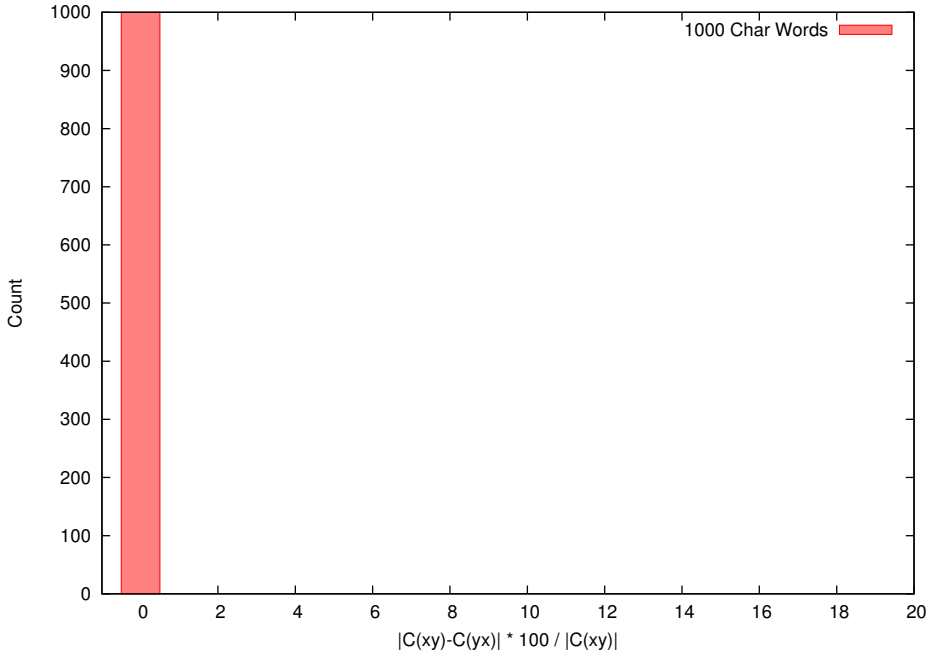

Figure 6.14. Same experiment as in Figure 6.11 but using 1,000 total words each 1,000 characters long.

6.2 Dimension Estimates of Common Fractals

As we have seen in Theorem 4.12 and equation (5.2), the Hausdorff dimension of a computable fractal F generated by an IFS can be measured by compressing (using Kolmogorov complexity) any base-k expansion of the Euclidean representation of a random point $x \in F$. Furthermore, we showed in Theorem 5.6 and 5.7 that this procedure still works if you compress the string formed by concatenating the first M bits of the Euclidean representations of a random sample of points from the fractal F.

Unfortunately, replacing Kolmogorov complexity with computable complexity measures such as Lempel-Ziv and I-Complexity does not appear to work very well for the purposes of measuring fractal dimension, at least for measuring the Hausdorff dimension of the Cantor set and Sierpinski triangle. We outline the details of an experiment that attempted to measure the dimension of the middle-third and middle-half[1] Cantor set below, as well as the Sierpinski triangle. The experiment demonstrates that Lempel-Ziv and I-Complexity only measure the dimension of these fractals accurately when the points being compressed are expanded in specific bases.

6.2.1 The Cantor Set

6.2.1.1 Experimental Setup

We used `C++` and `Pari/GP` to conduct the experiment. The compression algorithms (Lempel-Ziv and I-complexity) were coded using `C++`, and the process of generating a random point in the Cantor set and representing it in some base k was done using `Pari/GP`. These languages were chosen because `Pari/GP`, as a computer algebra system, is better than `C++` for handling numbers to high degrees of precision, while `C++` is the author's programming language of choice.

A random point x in the Cantor set was generated in the following way. First, fix a base k and length m. Divide the closed interval $[0, 1]$ into k^m subintervals of equal length k^{-m}. Each subinterval can be represented as a k-ary string of length m

[1] The middle-half Cantor set is similar to the middle-third Cantor set, except that the middle half interval is removed at each stage of the construction instead of the middle third.

using characters from the alphabet $\Sigma = \{0, 1, \ldots, k-1\}$[2]. This essentially overlays the unit interval with a grid of equal boxes of length k^{-m}.

Next, form a random string S of 0's and 2's representing a particular subinterval in the Cantor construction process, as in Figure 4.1, where at each stage of the construction, 0 represents the left third interval and 2 represents the right third interval. Let $n = \lceil m \log_3 k \rceil + 1$ be the length of S. The reason for this is to ensure that the size of the subinterval is smaller than the size of any of the boxes in the grid, i.e., that $3^{-n-1} < k^{-m}$. The point x is then chosen to be either the left or right endpoint of this interval, depending on the last bit of S (hence, the reason for the extra bit $(+1)$ in the definition of n), and its decimal representation in the unit interval is stored.

We used Lempel-Ziv and I-complexity to compress the k-ary string T corresponding to the subinterval in the grid containing the point x. To account for the coding constant of the Lempel-Ziv and I-complexity algorithms, we also computed the complexity of a random k-ary string R of length m. The Hausdorff dimension of the Cantor set C was then estimated by dividing,

$$\dim_H(C) \approx \frac{LZ(T)}{LZ(R)} \approx \frac{I(T)}{I(R)}. \tag{6.1}$$

This approach to normalization of Lempel-Ziv and I-complexity is not perfect. In [25], it was shown that random strings do not have maximum Lempel-Ziv complexity. An alternative method of normalization suggested in [25] is to use strings of maximal Lempel-Ziv complexity. Such maximum Lempel-Ziv strings are inherently not random in the sense of Kolmogorov and Martin-Löf since there is a very straightforward algorithm for constructing them. We compare both methods of normalization in the results section below.

Similarly, randomly generated strings do not have maximal I-complexity (in [24] it was shown that strings of maximal I-complexity are precisely the deBruijn strings). However, deBruijn strings of arbitrary length do not exist; they only exist for specific fixed lengths. So we did not try normalizing by them and instead only used randomly generated strings and strings of maximal Lempel-Ziv complexity.

We also attempted to estimate the dimension by dividing the complexity of T by its length, although with little success; for details, see the Appendix on page 94.

[2]In practice, we took characters from the set of printable ASCII characters.

6.2.1.2 Experimental Results

We ran the experiment for several different values of k and m, with mixed results. We present the results for the middle-third Cantor set first (Tables 6.1 and 6.2), followed by the middle-half Cantor set (Tables 6.3 and 6.4), below. In the case of the middle-third Cantor set, the experiment measured the Hausdorff dimension with reasonable accuracy only when k was a power of 3. For other values of k, the quotient in (6.1) returned a value very close to 1, suggesting that the string T was random, in the sense that the characters of Σ were uniformly distributed throughout T. Note that the true Hausdorff dimension of C is $\log_3(2) \approx 0.6309$.

One word about notation: in the following tables, let $\dim_{LZ,R}(C)$ and $\dim_{I,R}(C)$ denote the estimates of the Hausdorff dimension of C using equation (6.1) (i.e., where the expression is normalized by dividing by the complexity of a randomly generated string R with the same length as T), and let $\dim_{LZ,M}(C)$, $\dim_{I,M}(C)$ denote the estimates of Hausdorff dimension where the expression is normalized using a string M of maximal Lempel-Ziv complexity with the same length as T.

Table 6.1 below gives the results of compressing the first $m = 10,000$ digits of

k	m	$\dim_{LZ,R}(C)$	$\dim_{LZ,M}(C)$	$\dim_{I,R}(C)$	$\dim_{I,M}(C)$
2	10000	0.998968	0.850769	1.00043	0.987049
3	10000	0.646745	0.563901	0.66049	0.644845
4	10000	0.996579	0.897283	0.998214	0.986499
5	10000	1.00223	0.911752	0.999396	0.981673
6	10000	0.999416	0.892268	1.00073	0.961153
7	10000	1.00672	0.921868	1.00042	0.975485
8	10000	0.99814	0.92984	1.00019	0.98666
9	10000	0.659522	0.603159	0.682313	0.663045
10	10000	1.00106	0.898205	1.00005	0.946638
11	10000	1.00165	0.918175	0.999264	0.962282
12	10000	0.996664	0.93259	1.00013	0.973858
13	10000	0.998991	0.939883	0.99999	0.979534
14	10000	0.994044	0.944894	1.00032	0.985329
15	10000	0.999778	0.947195	0.999881	0.987144
16	10000	0.999239	0.944829	1.00004	0.986168
27	10000	0.667642	0.617141	0.698622	0.669176
32	10000	1.00057	0.94418	0.999286	0.974153
64	10000	1.0001	0.945294	0.999833	0.985735
81	10000	0.676577	0.613195	0.708583	0.67564

Table 6.1. The results of compressing **one** random point on the middle-third Cantor set, averaged over 10 separate experiments with standard deviation < 0.01 for each cell. The rows corresponding to a base k equal to a power of 3 have been highlighted.

only one random point in the middle-third Cantor set. Similar results are obtained by concatening the first $m = 100$ digits of $N = 100$ points and compressing the resulting T string. This is summarized in Table 6.2 below.

k	m	N	$m \times N$	$\dim_{LZ,R}(C)$	$\dim_{LZ,M}(C)$	$\dim_{I,R}(C)$	$\dim_{I,M}(C)$
2	100	100	10000	0.993548	0.846154	0.999692	0.986316
3	100	100	10000	0.661102	0.576419	0.688997	0.672678
4	100	100	10000	0.993293	0.894324	0.998342	0.986628
5	100	100	10000	1.00587	0.915064	1.00109	0.983338
6	100	100	10000	0.998406	0.891366	0.999752	0.960214
7	100	100	10000	1.00746	0.922551	1.00044	0.975512
8	100	100	10000	0.997675	0.929407	1.00062	0.98708
9	100	100	10000	0.667848	0.610774	0.69793	0.678222
10	100	100	10000	0.998723	0.896104	0.998919	0.945571
11	100	100	10000	1.0074	0.923454	1.00292	0.965804
12	100	100	10000	0.996823	0.932739	1.00298	0.976639
13	100	100	10000	1.00194	0.942659	1.00424	0.983693
14	100	100	10000	0.993969	0.944823	1.00139	0.986384
15	100	100	10000	0.997779	0.945302	1.00167	0.988914
16	100	100	10000	0.999276	0.944863	1.0007	0.986817
27	100	100	10000	0.67766	0.626402	0.70965	0.679739
64	100	100	10000	0.99818	0.943475	0.999797	0.9857
81	100	100	10000	0.678861	0.615265	0.714345	0.681134

Table 6.2. The results of compressing $N = 100$ random points on the middle-third Cantor set.

The results for the middle-half Cantor set, which has Hausdorff dimension equal to 0.5, are included on the following page. As was done above, the first table listed (Table 6.3) contains the results of compressing the first $10,000$ digits of only one random point in the middle-half Cantor set, while Table 6.4 contains the results of concatening the first $m = 100$ digits of $N = 200$ points and compressing the resulting T string.

6.2.1.3 Discussion

The results indicate that the only values of k for which the experiment approximates the dimension of the Cantor set are when k is a power of 3 (in the case of the middle-third Cantor set) or 2 (in the case of the middle-half Cantor set), with powers of 4 actually producing slightly better results. For other values of k, the string T appears to look the same as the random string R.

One explanation for this is that Lempel-Ziv and I-complexity perform poorly when given the digits of a normal number. A normal number to a base k is a real

k	m	$\dim_{LZ,R}(C)$	$\dim_{LZ,M}(C)$	$\dim_{I,R}(C)$	$\dim_{I,M}(C)$
2	10000	0.596903	0.508352	0.598055	0.590056
3	10000	0.999249	0.871252	0.999921	0.976237
4	10000	0.518176	0.466546	0.538565	0.532244
5	10000	1.00206	0.911592	0.999519	0.981794
6	10000	0.997343	0.890417	0.999847	0.960309
7	10000	1.00622	0.921412	1.0003	0.975373
8	10000	0.602975	0.561715	0.62641	0.617935
9	10000	0.999291	0.913892	1.00135	0.973071
10	10000	1.00132	0.898434	0.999896	0.946495
11	10000	1.0016	0.918137	0.998872	0.961904
12	10000	0.994837	0.930881	1.00011	0.973843
13	10000	1.00012	0.940942	1.00002	0.979561
14	10000	0.995703	0.946471	1.00052	0.985528
15	10000	0.998298	0.945792	1.00026	0.987518
16	10000	0.538501	0.509178	0.568831	0.560939
27	10000	0.999969	0.924331	0.999122	0.957011
32	10000	0.619988	0.585047	0.65287	0.636449
64	10000	0.55806	0.527476	0.594711	0.586325
81	10000	0.999362	0.905741	1.00093	0.954399

Table 6.3. The results of compressing **one** random point on the middle-half Cantor set, averaged over 10 separate experiments with standard deviation < 0.01 for each cell. The rows corresponding to a base k equal to a power of 2 have been highlighted.

k	m	N	$m \times N$	$\dim_{LZ,R}(C)$	$\dim_{LZ,M}(C)$	$\dim_{I,R}(C)$	$\dim_{I,M}(C)$
2	100	200	20000	0.608787	0.523381	0.630468	0.622978
3	100	200	20000	0.995957	0.8868	0.997838	0.982666
4	100	200	20000	0.535598	0.480545	0.561629	0.544864
5	100	200	20000	1.00284	0.897117	0.998754	0.962377
6	100	200	20000	0.99572	0.924503	0.999065	0.986304
7	100	200	20000	1.00587	0.898451	1.00017	0.953716
8	100	200	20000	0.604448	0.561383	0.63258	0.614642
9	100	200	20000	0.998575	0.940703	0.998864	0.985239
10	100	200	20000	1.00252	0.934913	1.00056	0.984472
11	100	200	20000	1.00352	0.916717	1.00279	0.967702
12	100	200	20000	1.00426	0.908495	1.00183	0.947757
13	100	200	20000	0.998134	0.918862	1.00228	0.960125
14	100	200	20000	0.99798	0.93385	1.00261	0.970649
15	100	200	20000	0.999207	0.942265	1.00192	0.978838
16	100	200	20000	0.544518	0.51679	0.577973	0.56821
27	100	200	20000	0.998506	0.88313	0.999928	0.922199
32	100	200	20000	0.613073	0.561626	0.646517	0.613261
64	100	200	20000	0.566695	0.547481	0.605327	0.601287
81	100	200	20000	1.00134	0.957716	1.00067	0.991914

Table 6.4. The results of compressing $N = 200$ random points on the middle-half Cantor set.

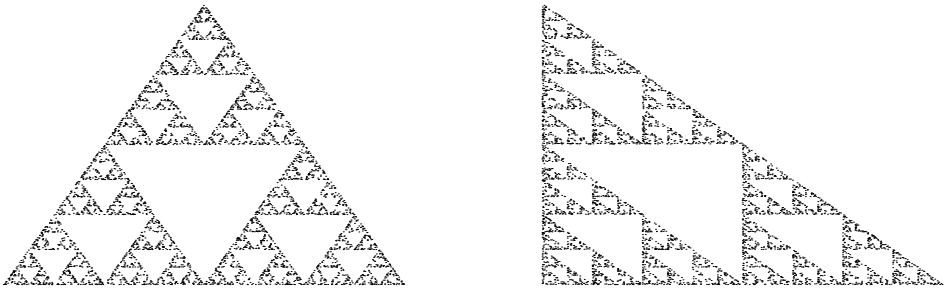

Figure 6.15. The equilateral Sierpinski triangle compared to a right triangle version. Both sets have the same Hausdorff dimension of $\log_2(3) \approx 1.58496$.

number whose expansion in base k is evenly distributed in the limit. Although it is not known if the number π is normal to base 10, it is widely suspected that it is. We measured the LZ-complexity and I-complexity of the first 10,000 digits of a base 10 expansion of π and found it to be about the same as that of a random string. This is relevant to the Cantor set because J. W. S. Cassels showed that almost every point in the middle-third Cantor set is normal to every base k which is not a power of 3 [26]. It is reasonable to believe that a similar fact holds for the middle-half Cantor set.

6.2.2 The Sierpinski Triangle

As was the case for the Cantor set, we used `C++` and `Pari/GP` to conduct the experiment. We considered two different constructions of the Sierpinski triangle: an equilateral Sierpinski triangle and a right triangle version of the Sierpinski triangle (see Figure 6.15 above). Both these sets have the same Hausdorff dimension ($\log_2(3) \approx 1.58496$) since the iterated function systems generating them have the same contraction ratios. Specifically, their similarity dimension s is the same since in both cases $s = \log_2(3)$ is the solution to the equation $\sum_{k=1}^{3} \left(\frac{1}{2}\right)^s = 1$, which equals the Hausdorff dimension (see [18]). The reason we considered the right triangle version was because it provided better estimates of the Hausdorff dimension than the equilateral version.

6.2.2.1 Experimental Setup

A random point (x, y) on the Sierpinski triangle (both versions) was generated in the following way. First, fix a base k and length m. Divide the closed unit square $[0, 1] \times [0, 1] \subseteq \mathbb{R}^2$ into k^m columns and k^m rows to form a grid of k^{2m} boxes of sidelength k^{-m}. Each column and row in this grid can be represented as a k-ary string of length m using characters from the alphabet $\Sigma = \{0, 1, \ldots, k-1\}$, which we will call an address string[3]. Given address strings $U = u_1 u_2 \cdots u_m$ and $V = v_1 v_2 \cdots v_n$ for the ith column and jth row of the grid, respectively, let $T = u_1 v_1 u_2 v_2 \cdots u_m v_m$ denote the address string of length $2m$ for the box in column i and row j. This gives us a way to label each box in the grid.

Next, let S be a random string of 0's, 1's and 2's which represents a particular triangle in the interior of the nth stage of the Sierpinski construction process. Set $n = \lceil m \log_2 k \rceil + 1$ to ensure that the interior triangle represented by S has a side length smaller than the side length of the grid boxes, i.e., so that $2^{-n-1} < k^{-m}$. The point (x, y) is chosen to be one of the vertices of this interior triangle, depending on the last bit of S. The decimal representation of (x, y) in the unit square is then stored to make it easy to compute the address string T of length $2m$ of the box containing (x, y).

We used Lempel-Ziv and I-complexity to compress the k-ary address string T. As with the Cantor set, to account for the coding constant of the Lempel-Ziv and I-complexity algorithms, we also computed the complexity of a random k-ary string R of with the same length as T. The Hausdorff dimension of the Sierpinski triangle $\mathcal{T}$ was then estimated by,

$$\dim_H(\mathcal{T}) \approx 2 \times \frac{LZ(T)}{LZ(R)} \approx 2 \times \frac{I(T)}{I(R)}, \tag{6.2}$$

where the factor of 2 is to account for the fact that $\mathcal{T}$ is a subset of $\mathbb{R}^2$. We also compared this to normalizing by the complexity of a maximum Lempel-Ziv string of the same length as T.

[3]As was the case for the Cantor set, we formed Σ from the set of printable ASCII characters.

k	m	$\dim_{LZ,R}(\mathcal{T})$	$\dim_{LZ,M}(\mathcal{T})$	$\dim_{I,R}(\mathcal{T})$	$\dim_{I,M}(\mathcal{T})$
2	10000	1.98605	1.70743	1.99514	1.97144
3	10000	1.99191	1.7736	2.00038	1.96997
4	10000	1.98482	1.7808	1.99433	1.9348
5	10000	2.01201	1.79989	2.00087	1.928
6	10000	1.99829	1.85536	2.00173	1.97617
7	10000	2.01228	1.79738	2.00278	1.90974
8	10000	1.998	1.85565	1.99773	1.94108
9	10000	1.9886	1.87335	1.99898	1.97171
10	10000	2.00686	1.87153	2.00212	1.96994
11	10000	2.00573	1.83223	2.00124	1.93121
12	10000	1.99957	1.8089	1.99715	1.88936
13	10000	1.99295	1.83467	1.99614	1.91218
14	10000	1.99192	1.86392	2.00048	1.93672
15	10000	2.00991	1.89537	2.00349	1.95734
16	10000	1.99417	1.89262	1.99781	1.96406
27	10000	2.001	1.76978	2.00223	1.84659
32	10000	1.99777	1.83013	1.99532	1.89269
64	10000	1.99692	1.92922	1.99965	1.98631
81	10000	2.01069	1.9231	2.00187	1.98436

Table 6.5. The results of compressing **one** random point on the equilateral Sierpinski triangle.

6.2.2.2 Experimental Results

As with the Cantor set, we ran the experiment for several values of k and m. The results are contained in Tables 6.5 through 6.8. The experiment accurately measured the dimension of the Sierpinski triangle only when we used the right triangle version with k equal to a power of 2. If we used the equilateral version of the Sierpinski triangle, regardless of the value of k, or if we used the right triangle version with k not equal to a power of 2, then the quotient in (6.2) returned a value close to 2 suggesting that the string T was random.

6.2.2.3 Discussion

The results for the Sierpinski triangle $\mathcal{T}$ follow the same pattern as the results for the Cantor set C. At each stage of the Sierpinski construction process, the side length of the triangles are reduced by a factor of 2. The only bases that worked well for measuring the dimension of $\mathcal{T}$ were powers of 2. For the Cantor set, at each stage of the construction process, the length of each subinterval was reduced by a factor of 3 (in the case of the middle-third) or 4 (in the case of the middle-half). And we saw that the only bases that worked well for measuring the dimension

of C were 3 and 2. Thus, the Lempel-Ziv and I-complexity measures seem to only provide accurate approximations to Kolmogorov complexity when the point being compressed is expressed in a base that matches with the underlying iterated function system.

The reason why the right Sierpinski triangle was easier to compress than the equilateral Sierpinski triangle is because the right Sierpinski triangle is naturally encoded using a grid of base 2. Indeed, an easy way to describe the right Sierpinski triangle in base 2 is as the set of all binary pairs $(x = 0.x_1x_2\cdots, y = 0.y_1y_2\cdots)$ such that $x_ky_k = 0$ for all indices k [27]. That is, for each index k, the kth index of the binary expansions of x and y are not both equal to 1 (see Figure 6.16 below). This is what makes the right Sierpinski triangle easy to compress with Lempel-Ziv and I-complexity.

Since we blamed the inability of Lempel-Ziv and I-complexity to accurately measure the dimension of the Cantor set in specific bases on the fact that almost every point in C is normal to these respective bases, it is natural to suspect that a similar phenomenon is happening with the Sierpinski triangle. This leads us to make the following conjecture.

k	m	N	$m \times N$	$\dim_{LZ,R}(\mathcal{T})$	$\dim_{LZ,M}(\mathcal{T})$	$\dim_{I,R}(\mathcal{T})$	$\dim_{I,M}(\mathcal{T})$
2	100	100	10000	1.98745	1.70863	1.9957	1.97199
3	100	100	10000	1.9973	1.7784	2.0014	1.97097
4	100	100	10000	1.99928	1.79377	1.99566	1.93609
5	100	100	10000	2.00253	1.79141	1.99724	1.9245
6	100	100	10000	1.99715	1.8543	2.00037	1.97482
7	100	100	10000	2.01441	1.79928	2.00105	1.9081
8	100	100	10000	1.9935	1.85147	1.99946	1.94276
9	100	100	10000	1.9924	1.87693	1.99787	1.97062
10	100	100	10000	2.00641	1.87111	2.00054	1.96838
11	100	100	10000	2.00925	1.83545	2.00583	1.93564
12	100	100	10000	2.00213	1.81121	2.00489	1.89669
13	100	100	10000	2.00664	1.84727	2.00693	1.92253
14	100	100	10000	1.99798	1.86959	2.00667	1.9427
15	100	100	10000	2.00832	1.89387	2.00634	1.96012
16	100	100	10000	2.00311	1.90111	2.00016	1.96637
27	100	100	10000	2.00299	1.77155	2.00089	1.84535
32	100	100	10000	1.99746	1.82984	1.99539	1.89275
64	100	100	10000	2.00000	1.93219	1.99897	1.98562
81	100	100	10000	2.00454	1.91722	2.00171	1.9842

Table 6.6. The results of compressing $N = 100$ random points on the equilateral Sierpinski triangle.

k	m	$\dim_{LZ,R}(\mathcal{T})$	$\dim_{LZ,M}(\mathcal{T})$	$\dim_{I,R}(\mathcal{T})$	$\dim_{I,M}(\mathcal{T})$
2	10000	1.68675	1.45012	1.7093	1.68899
3	10000	1.99488	1.77624	2.00035	1.96994
4	10000	1.70372	1.5286	1.73972	1.68778
5	10000	2.00272	1.79158	1.99964	1.9268
6	10000	1.99601	1.85325	2.00088	1.97532
7	10000	2.01356	1.79852	2.00114	1.90819
8	10000	1.73028	1.60701	1.77199	1.72175
9	10000	1.99615	1.88047	1.99924	1.97197
10	10000	2.00531	1.87008	2.00121	1.96904
11	10000	2.00542	1.83195	2.00015	1.93016
12	10000	1.99906	1.80844	1.99974	1.89181
13	10000	2.00315	1.84406	2.00175	1.91756
14	10000	1.99725	1.86891	2.00158	1.93778
15	10000	2.00079	1.88677	2.00137	1.95527
16	10000	1.75587	1.66646	1.7961	1.76576
27	10000	1.99741	1.76661	1.99966	1.84422
32	10000	1.7584	1.61084	1.80797	1.71497
64	10000	1.82383	1.76199	1.87136	1.85887
81	10000	2.00329	1.91602	2.00112	1.98362

Table 6.7. The results of compressing one random point on the right Sierpinski triangle, averaged over 10 separate experiments with standard deviation < 0.01 for each cell. The rows corresponding to a base k equal to a power of 2 have been highlighted.

k	m	N	$m \times N$	$\dim_{LZ,R}(\mathcal{T})$	$\dim_{LZ,M}(\mathcal{T})$	$\dim_{I,R}(\mathcal{T})$	$\dim_{I,M}(\mathcal{T})$
2	100	100	10000	1.69456	1.45683	1.72556	1.70506
3	100	100	10000	1.99012	1.772	1.99781	1.96743
4	100	100	10000	1.71955	1.5428	1.75242	1.70011
5	100	100	10000	2.00569	1.79423	1.99983	1.92699
6	100	100	10000	1.99486	1.85219	1.99894	1.97341
7	100	100	10000	2.01014	1.79547	2.00056	1.90763
8	100	100	10000	1.72914	1.60594	1.77586	1.7255
9	100	100	10000	1.99477	1.87917	1.99923	1.97196
10	100	100	10000	2.00778	1.87239	2.00168	1.9695
11	100	100	10000	2.01277	1.83866	2.00809	1.93782
12	100	100	10000	2.00809	1.81661	2.00707	1.89875
13	100	100	10000	2.01327	1.85338	2.00743	1.923
14	100	100	10000	2.00121	1.87261	2.00753	1.94353
15	100	100	10000	1.99881	1.8849	2.00717	1.96093
16	100	100	10000	1.75855	1.669	1.8014	1.77097
27	100	100	10000	1.99768	1.76685	1.99572	1.84059
32	100	100	10000	1.76399	1.61597	1.8128	1.71955
64	100	100	10000	1.8269	1.76496	1.87409	1.86158
81	100	100	10000	2.00427	1.91696	2.00103	1.98353

Table 6.8. The results of compressing $N = 100$ random points on the right Sierpinski triangle.

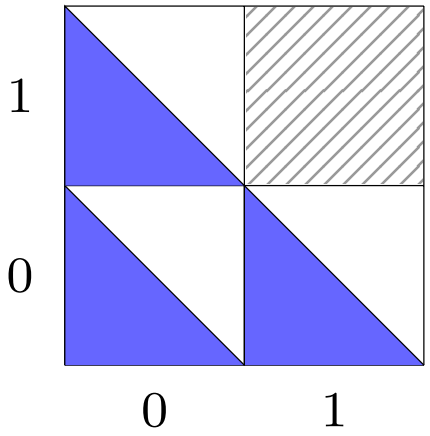

Figure 6.16. The right Sierpinski triangle consists of all pairs (x, y) of binary sequences such that for every k, the kth index of the binary expansions of x and y are not both equal to 1, as can be seen for the case $k = 1$ in this figure.

Conjecture 1. *Almost every point in the equilateral Sierpinski triangle expressed in any base is normal, and almost every point in the right Sierpinski triangle is normal when expressed in any base not equal to a power of 2.*

Appendix A
Source Code

We include the Pari/GP and C++ source code used to generate the Cantor set, Sierpinski triangle, and to estimate their fractal dimension.

A.1 The Cantor Set

The following Pari/GP function was used to output the base-10 expansion of a point in the middle-third Cantor set.

```
cantorSetOneThird(k,base) = {

   local(n, i, x) ;

   n = ceil(k * log(base) / log(3)) ;

   x = 0. ;

   for (i = 1, n,
      j = random(2) ;
      x = x + 2 * j * 3^(-i) ;
   ) ;

   j = random(2) ;
   x = x + j * 3^(-n) ;
```

```
   return(x) ;

}
```

The code for output the base-10 expansion of a point in the middle-half Cantor set is similar.

```
cantorSetOneHalf(k,base) = {

   local(n, i, x) ;

   n = ceil(k * log(base) / log(4)) ;

   x = 0. ;

   for (i = 1, n,
      j = random(2) ;
      x = x + 3 * j * 4^(-i) ;
   ) ;

   j = random(2) ;
   x = x + j * 4^(-n) ;

   return(x) ;

}
```

To construct the address string T corresponding to one or more points on the Cantor set in some grid, the following was used (we only show the code for the middle-third Cantor set; the middle-half is similar).

```
addressCantorSetOneThird(bits,base,n) = {

   local(gridWidth, x, i, addressInt, address) ;

   gridWidth = base^bits ;
```

```
   address = "" ;

   for (i = 1, n,
      x = cantorSetOneThird(bits,base) ;
      addressInt = floor( gridWidth * x ) ;

      /* fixes a bug that occurs when x equals 1*/
      if (addressInt == gridWidth,
        addressInt = gridWidth-1 ;
      ) ;

      address = concat(address,intToBaseRep(addressInt,bits,base)) ;
   ) ;

   file = concat("./data/addresses_",bits) ;
   file = concat(file,"_bits_") ;
   file = concat(file,base) ;
   file = concat(file,"_base_") ;
   file = concat(file,n) ;
   file = concat(file,"_pts_CSOneThird.txt") ;

   write(file,address) ;
}
```

The function `addressCantorSetOneThird` above uses the function `intToBaseRep` to get the address string of the point `x` corresponding to the box in the grid containing `x`. We include that below for reference.

```
intToBaseRep(x,bits,base) = {

   local(sequence, quotient, remainder, asciiOffset) ;

   quotient = real(x) ;
   remainder = 0. ;
```

```
   sequence = "" ;

   if (quotient == 0, concat(sequence,0) ; ) ;

   /* offsets ascii codes to use certain characters depending on base size */
   if (base > 10, asciiOffset = 33 ;, asciiOffset = 48 ; ) ;

   while (quotient > 0,
      quotient = x \ base ;
      remainder = x % base ;
      sequence = concat(Strchr(remainder+asciiOffset),sequence) ;

      x = quotient ;
   ) ;

   while (length(sequence) < bits,
      sequence = concat(0,sequence) ;
   ) ;

   return(sequence) ;

}
```

A.2 The Sierpinski Triangle

A similar set of functions was used to construct points on the equilateral and right Sierpinski triangles. We provide the code used to output the base-10 expansions of points on these triangles below.

```
sierpinskiTriangle(k,base) = {

   local(n, i, x) ;

   n = ceil(k * log(base) / log(2)) ;
```

```
   x = 0. ;
   y = 0. ;

   for (i = 1, n,
      j = random(3) ;
      if (j == 1, x = x + 2^(-i) ,
          if (j == 2, x = x + 2^(-i-1) ; y = y + sqrt(3)/(2^(i+1)) ; )
      ) ;
   ) ;

   j = random(3) ;
   if (j == 1, x = x + 2^(-n) ,
       if (j == 2, x = x + 2^(-n-1) ; y = y + sqrt(3)/(2^(n+1)) ; )
   ) ;

   return([x,y]) ;

}

sierpinskiRightTriangle(k,base) = {

   local(n, i, x) ;

   n = ceil(k * log(base) / log(2)) ;

   x = 0. ;
   y = 0. ;

   for (i = 1, n,
      j = random(3) ;
      if (j == 1, x = x + 2^(-i) ,
          if (j == 2, y = y + 2^(-i) ; )
      ) ;
   ) ;
```

```
    j = random(3) ;
    if (j == 1, x = x + 2^(-n) ,
        if (j == 2, y = y + 2^(-n) )
    ) ;

    return([x,y]) ;

}
```

The address strings corresponding to point(s) on the Sierpinski triangle were constructed in a very similar manner to `addressCantorSetOneThird` above, and so we omit these functions.

A.3 Estimating Fractal Dimension

After we created the address strings corresponding to point(s) on a fractal and stored them in text files, we used the following `C++` code to read the strings from these files and then compress them to estimate fractal dimension. Information about such as the number of bits, base, location of maximum Lempel-Ziv strings, etc. were provided as command line input, making it easy to run this program multiple times with different input values in a BASH script.

```
#include<iostream>
#include<fstream>
#include<vector>
#include<sstream>
#include<ctime>
#include<cmath>
#include<random>
#include"common_functions.h"
#include"globals.cpp"
#include"lzcomplexity.cpp"
#include"icomplexity.cpp"
using namespace std ;
```

```
extern int alphabetSize ;  // declared in "globals.cpp"

int main(int argc, char* argv[]) {

   alphabetSize = 2 ;
   int n = 1 ;            // dimension of euclidean space
   int bits = 10000 ;
   int pts = 1 ;

   string fileName, fileMaxLZ ;

   // command line input

   for (int i = 1; i < argc; i++)
      if (argv[i][0] == '-')
         switch (argv[i][1]) {
         case 'f': fileName = argv[++i] ;
            break;
         case 'm': fileMaxLZ = argv[++i] ;
            break;
         case 's': alphabetSize = atoi(argv[++i]) ;
            break;
          case 'b': bits = atoi(argv[++i]) ;
            break;
         case 'n': n = atoi(argv[++i]) ;
            break;
         case 'p': pts = atoi(argv[++i]) ;
            break;
         }

   if (fileName == "" || fileMaxLZ == "") {
      cerr << "Usage: ./hausdorffDimension -f <filename> -m <fileMaxLZ>
      [-s <alphabetSize>] [-n <euclideanDim>]
```

```
    [-b <bits>] [-p <points>]" << endl ;
   return 1 ;
}

ifstream fin( fileName ) ;

// Read data from an input file that user specifies on the command line

cerr << "\nReading data files..." << endl ;

string x, temp ;
while(fin >> temp) {
   x += temp ;
}

string control ;
for (int i = 0; i < x.size() ; ++i) {
   // generate random control string
   int randomBit = randomInteger(0,alphabetSize-1) ;
   control += char(33+randomBit) ;
}

fin.close() ;
fin.open(fileMaxLZ) ;

string controlMaxLZ ;
double compressedLZControlMaxLZ, compressedIControlMaxLZ ;

fin >> temp >> compressedLZControlMaxLZ >> compressedIControlMaxLZ ;

fin.close() ;

// compute and output interesting values
```

```
    cerr << "Beginning LZ compression..." << endl ;
    double compressedLZ = LZComplexity(x) ;
    cerr << "Beginning LZ compression (random control)..." << endl ;
    double compressedLZControl = LZComplexity(control) ;
    double dimMaxLZ = double(compressedLZ)/compressedLZControlMaxLZ ;
    double dimControlLZ = double(compressedLZ)/compressedLZControl ;
    cerr << "LZ(X)/LZ(control) = " << dimControlLZ << endl ;
    cerr << "LZ(X)/LZ(MaxLZ) = " << dimMaxLZ << endl << endl ;

    cerr << "Beginning I compression..." << endl ;
    double compressedI = IComplexity(x) ;
    cerr << "Beginning I compression (random control)..." << endl ;
    double compressedIControl = IComplexity(control) ;
    double dimMaxI = double(compressedI)/compressedIControlMaxLZ ;
    double dimControlI = double(compressedI)/compressedIControl ;
    cerr << "I(X)/I(control) = " << dimControlI << endl ;
    cerr << "I(X)/I(MaxLZ) = " << dimMaxI << endl << endl ;

    // output data for LaTeX tables
    cout << alphabetSize << " & "
<< bits << " & "
<< pts << " & "
<< bits*pts << " & "
<< n * dimControlLZ << " & "
<< n * dimMaxLZ << " & "
<< n * dimControlI << " & "
<< n * dimMaxI << "\\\\ \\hline" << endl ;

    return 0 ;

}
```

A.4 Lempel-Ziv Complexity

The Lempel-Ziv complexity of a string was computed using the following function.

```
double LZComplexity(const string &s) {

   vector<string> components ;

   string total, temp ;

   for (int i = 0; i < s.size(); ++i) {
      temp += s[i] ;

      if (!isASubString(temp,total)) {
          components.push_back(temp) ;
          temp = "" ;
      }

      total += s[i] ;

   }

   // store last component
   if (temp != "") components.push_back(temp) ;

   return components.size() ;

}
```

A.5 I-Complexity

Similarly, the following was used to measure the I-complexity of a string.

```
extern int alphabetSize ;  // declared in "globals.cpp"
```

```
void computeBRepArray(const string &s, vector<int> & B) {

   for (int i = 0; i < B.size(); ++i) {

      int max = 0 ;

      for (int j = 1; j <= i; ++j) {

         int k = i - j + 1 ;

         string s1 = s.substr(k,j) ;
         string s2 = s.substr(0,i) ;

         if ( !isASubString(s1,s2) ) {
            max = j - 1 ;
            break ;
          }

          else max = j ;

      }

      B[i] = max ;

   }

}

double IComplexity(const string &s) {

   vector<int> B (s.size(),0) ;

   computeBRepArray(s, B) ;
```

```
   double I_B = 0 ;

   for (int i = 0; i < B.size(); ++i)
      I_B += dlog(B[i]+1, alphabetSize) ;

   return I_B ;

}
```

A.6 Maximum Lempel-Ziv Strings

We used the following program to generate maximum Lempel-Ziv strings of length N and alphabet size σ.

```
#include<iostream>
#include<fstream>
#include<vector>
#include<cstdlib>
#include<sstream>
#include"common_functions.h"
#include"globals.cpp"
#include"lzcomplexity.cpp"
#include"icomplexity.cpp"
using namespace std ;

extern int alphabetSize ;  // declared in "globals.cpp"

// nextLexicographic returns the next lexicographically ordered word
// if input word is last in the ordering, then nextLexicographic returns
// the first string that is one character larger than input word
string nextLexicographic(string word, int sigma, int asciiStart) {

   for (int i = word.size()-1; i >= 0; --i) {
      if (word[i] != char(asciiStart+sigma-1)) {
```

```
 word[i] = char(int(word[i])+1) ;
 return word ;
      }
      else
 if (i != 0) word[i] = char(asciiStart) ;
   }

   string bigger ;
   for (int i = 0; i < word.size() + 1; ++i)
        bigger += string(1,char(asciiStart)) ;

   return bigger ;

}

int main(int argc, char* argv[]) {

   // command line input

   string file ;
   int sigma = 2 ;
   int N = -1 ;

   for (int i = 1; i < argc; i++)
      if (argv[i][0] == '-')
         switch (argv[i][1]) {
         case 's':  sigma = atoi(argv[++i]) ;
             break;
         case 'N':  N = atoi(argv[++i]) ;
             break;
 }

   if (N < 0) {
      cerr << "Usage: ./generateMLZ -N <size> [-s <alphabet size>]" << endl ;
```

```
      return 1 ;
   }

   int asciiStart = 33 ; // the character '0'
   string MLZ ;
   int k = 0 ;

   while(MLZ.size() < N) {
      ++k ;
      string lambda ;
      //for (int i = 0; i < k; ++i) lambda += string(1,char(asciiStart)) ;
      for (int i = 0; i < pow(sigma,k); ++i) {
          string candidate = MLZ + lambda ;
          if (!isASubString(lambda,candidate.substr(0,candidate.size()-1)))
              MLZ = candidate ;
          if (MLZ.size() >= N) break ;
          lambda = nextLexicographic(lambda,sigma,asciiStart) ;
      }
   }

   alphabetSize = sigma ;
   double compressedLZ = LZComplexity(MLZ) ;
   double compressedI = IComplexity(MLZ) ;

   cout << MLZ << " " << compressedLZ << " " << compressedI << endl ;

   return 0 ;

}
```

A.7 common_functions.cpp

Some of the functions used in the C++ code above are stored in `common_functions.cpp`, which we provide below.

```
double log_n(double x, double n) {
   return log(x) / log(n) ;
}

double dlog(double x, double n) {
   return log_n(x+1,n) - log_n(x,n) ;
}

// is A a substring of B?
bool isASubString(string A, string B) {

   for (int i = 0; i < B.size(); ++i) {

      int j = 0 ;

      for (j; j < A.size(); ++j) if (A[j] != B[i+j]) break ;

      if (j == A.size()) return true ;

   }

   return false ;


}

// randomInteger returns a random integer in the interval [min,max]
int randomInteger(int min, int max) {
   return (double)(max-min+1)*rand() / (RAND_MAX+1.0) + min ;
}
```

A.8 Creating Table 6.2

As an example of how the tables in Chapter 6 were created, and to see all the above code in action, we provide the scripts that were used to create Table 6.2.

First, the address strings for each value of k were generated and stored in text files using the following `Pari/GP` script:

```
\r addressCantorSetOneThird.gp
\r cantorSetOneThird.gp
\r intToBaseRep.gp

bits=100 ;
n=100 ;

/* set precision so that precision > log_10(81^bits) */
\p 200

for (i = 2, 16, addressCantorSetOneThird(bits,i,n) ; ) ;

addressCantorSetOneThird(bits,27,n) ;
addressCantorSetOneThird(bits,64,n) ;
addressCantorSetOneThird(bits,81,n) ;

quit
```

This created 18 text files which followed the naming format,

```
addresses_<# of bits>_bits_<base>_base_<# of points>_pts_CSOneThird.txt
```

The fractal dimension of each of these 18 address strings (corresponding to each of the rows in Table 6.2) was measured using the following `BASH` script:

```
#!/bin/bash

# check for correct number of arguments
if [ "$#" -ne 4 ] ; then
    echo "Must supply command line arguments"
```

```
    echo "Usage: "$0" <bits> <points> <name_of_fractal> <euclidean_dimension>"
    exit
fi

# parse command line
bits=$1
pts=$2
fractal=$3
n=$4
MLZSize=‘echo "$bits*$pts*$n" | bc -l‘

# set up file names
maxLZ="./data/maxLZ_"$MLZSize""
outputFile="./data/compression_results_"$pts"_pts_"$fractal".txt"

# create empty output file
echo "creating output file..."
echo "" > $outputFile

# compile
g++ estimateDimension.cpp common_functions.cpp -o estimateDimension
g++ generateMLZ.cpp common_functions.cpp -o generateMLZ

# loop over different bases
for i in 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 27 32 64 81
do
    # create maxLZ string if necessary
    if [ ! -f "$maxLZ"_"$i".txt ] ; then
        ./generateMLZ -N $MLZSize -s $i > "$maxLZ"_"$i".txt
    fi

    # estimateDimension and output result to file
    ./estimateDimension
            -f ./data/addresses_"$bits"_bits_"$i"_base_"$pts"_pts_"$fractal".txt
```

```
            -m "$maxLZ"_"$i".txt
            -s $i -b $bits -n $n -p $pts >> $outputFile

done

exit
```

Appendix B
Additional tables

An alternative to the estimate of the Hausdorff dimension of the Cantor set in (6.1) is to divide the complexity of the k-ary string T of length m by its length. To account for the fact that the alphabet used may not be binary (i.e., $k \neq 2$), an additional multiplicative factor of $\log_2 k$ is required in the calculation. Unfortunately this method did not produce good results; we include the table below as an example.

k	m	$\dim_{LZ}(C)$	$\dim_I(C)$
2	20000	0.07175	0.101466
3	20000	0.11388	0.101561
4	20000	0.2763	0.189422
5	20000	0.36907	0.215413
6	20000	0.451722	0.235783
7	20000	0.529186	0.252811
8	20000	0.5997	0.266821
9	20000	0.437925	0.189495
10	20000	0.726173	0.289871
11	20000	0.78581	0.299881
12	20000	0.843004	0.308897
13	20000	0.891436	0.316205
14	20000	0.944224	0.3231
15	20000	0.987076	0.32913
16	20000	1.0268	0.334887
27	20000	0.95383	0.266854
64	20000	2.1435	0.447361
81	20000	1.63346	0.335015

Table B.1. Estimating Hausdorff dimension by dividing by length. In this case, $\dim_{LZ}(C) := (LZ(T) \times \log_2 k)/m$ and $\dim_I(C) := (I(T) \times \log_2 k)/m$.

Bibliography

[1] Falconer, K. (2004) *Fractal geometry: mathematical foundations and applications*, John Wiley & Sons.

[2] Cutler, C. D. (1993) "A review of the theory and estimation of fractal dimension," *Nonlinear time series and chaos*, **1**, pp. 1–107.

[3] Grassberger, P. and I. Procaccia (2004) "Measuring the strangeness of strange attractors," in *The Theory of Chaotic Attractors*, Springer, pp. 170–189.

[4] Denker, M. and G. Keller (1986) "Rigorous statistical procedures for data from dynamical systems," *Journal of Statistical Physics*, **44**(1-2), pp. 67–93.

[5] Aaronson, J., R. Burton, H. Dehling, D. Gilat, T. Hill, and B. Weiss (1996) "Strong laws for L- and U-statistics," *Transactions of the American Mathematical Society*, **348**(7), pp. 2845–2866.

[6] Downey, R. G. and D. R. Hirschfeldt (2010) *Algorithmic randomness and complexity*, Springer Science & Business Media.

[7] Grünwald, P. D. and P. M. B. Vitányi (2008) "Algorithmic information theory," *CoRR*.

[8] Li, M. and P. M. Vitányi (2009) *An introduction to Kolmogorov complexity and its applications*, Springer Science & Business Media.

[9] Shannon, C. E. (2001) "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, **5**(1), pp. 3–55.

[10] Rényi, A. (1965) "On the foundations of information theory," *Revue de l'Institut International de Statistique*, pp. 1–14.

[11] Cover, T. M. and J. A. Thomas (2012) *Elements of information theory*, John Wiley & Sons.

[12] Hoyrup, M. (2011) "The dimension of ergodic random sequences," *arXiv preprint arXiv:1107.1149*.

[13] Reimann, J. (2004) "Computability and fractal dimension," .

[14] Staiger, L. (2005) "Constructive dimension equals Kolmogorov complexity," *Information Processing Letters*, **93**(3), pp. 149–153.

[15] Mayordomo, E. (2002) "A Kolmogorov complexity characterization of constructive Hausdorff dimension," *Information Processing Letters*, **84**(1), pp. 1–3.

[16] Hitchcock, J. M. (2005) "Correspondence principles for effective dimensions," *Theory of Computing Systems*, **38**(5), pp. 559–571.

[17] Reimann, J. and F. Stephan (2005) "On hierarchies of randomness tests," in *Proceedings of the 9th Asian Logic Conference, World Scientific Publishing*, World Scientific, pp. 16–19.

[18] Lutz, J. H. and E. Mayordomo (2008) "Dimensions of points in self-similar fractals," *SIAM Journal on Computing*, **38**(3), pp. 1080–1112.

[19] Breiman, L. (1968) *Probability*, Addison-Wesley series in statistics, Addison-Wesley Publishing Company.

[20] Reimann, J. (2015) "Effective multifractal analysis," *In preparation.*

[21] Cilibrasi, R. and P. Vitanyi (2005) "Clustering by compression," *Information Theory, IEEE Transactions on*, **51**(4), pp. 1523–1545.

[22] Li, M., X. Chen, X. Li, B. Ma, and P. Vitányi (2004) "The similarity metric," *Information Theory, IEEE Transactions on*, **50**(12), pp. 3250–3264.

[23] Lempel, A. and J. Ziv (1976) "On the complexity of finite sequences," *Information Theory, IEEE Transactions on*, **22**(1), pp. 75–81.

[24] Becher, V. and P. A. Heiber (2012) "A linearly computable measure of string complexity," *Theoretical Computer Science*, **438**, pp. 62–73.

[25] Estevez-Rams, E., R. L. Serrano, B. A. Fernández, and I. B. Reyes (2013) "On the non-randomness of maximum Lempel Ziv complexity sequences of finite size," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, **23**(2), p. 023118.

[26] Cassels, J. (1959) "On a problem of Steinhaus about normal numbers," in *Colloquium Mathematicae*, vol. 1, pp. 95–101.

[27] Edgar, G. (2007) *Measure, topology, and fractal geometry*, Springer Science & Business Media.