ENCODING ARGUMENTS

by

Tommy Reddad

A thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

at

Carleton University
Ottawa, Ontario
November 2015

Abstract

We study encoding arguments using partial prefix-free codes, which allow us to prove probabilistic statements using the basic fact that uniformly chosen elements from a set of size n cannot be encoded with fewer than $\log_2 n$ bits on average. This technique, in the end, is a more direct version of the incompressibility method from the study of Kolmogorov complexity. We use our technique to produce short original proofs for several results. We also explore extensions of the technique to non-uniform input distributions. Finally, we offer a new manner of encoding which allows for real-valued codeword lengths, thereby refining every other result obtained through encoding.

Acknowledgements

Firstly, I would would like to express my gratitude to my supervisors, Prof. Pat Morin, Prof. Vida Dujmović, and Prof. "Jit" Bose, for welcoming me to Ottawa and Carleton University, and helping me work on and study other topics before settling on the study of encoding arguments as the subject of my thesis. I am especially grateful to Prof. Morin, whose insight, knowledge, and personal supervision have been absolutely invaluable.

I am also indebted to the examiners of my thesis, Prof. Jean-Lou De Carufel and Prof. Daniel Panario, for their meticulous and careful reading of my work, and for their helpful comments.

Thanks to Prof. Luc Devroye at McGill University for teaching a wonderful class in the study of algorithms and data structures, which sparked my interest in information theory.

Of course, I would like to thank my fellow labmates and researchers working in the Computational Geometry Lab at Carleton University, whose company I found to be appropriately motivating whenever I felt inspired, and appropriately distracting whenever I felt overwhelmed. I am convinced that the rare atmosphere sustained in our lab is precious.

Finally, I thank my family and friends, since without them I would be nowhere. I am most grateful to my parents who have supported me throughout all of my studies and all of my life. A thousand thanks go out to my girlfriend Emily, for her support and encouragement. Special thanks go out to my friend Costin Bădescu, who drew me into research and the study of mathematics and computer science. I would also like to thank my friends Ash Chapman, Troy Hildebrandt, and Darryl Hill, whose solidarity as fellow students has been a great comfort. I am glad that we could confront the looming presence of graduate research together. And lastly, thanks to Bavaria Brewery, for offering the most drinkable cheap beer in Ottawa.

Table of Contents

List of Figures

Chapter 1

Introduction

Techniques from the study of information theory have long been used to produce original or elementary proofs of results in discrete mathematics and computer science. Encoding arguments, now commonly framed under the so-called incompressibility method, sprouted from the study of information complexity as introduced by Kolmogorov, Solomonoff, and Chaitin. This technique has been fruitful in recent history, perhaps most famously being used to give the first nontrivial lower bound for the average running time of Shellsort, and for providing a constructive version of the Lovász local lemma [5, 6, 20, 28, 39].

An encoding argument transforms the problem of upper-bounding the probability of some event into the problem of devising short encodings for elements of this event, typically through the Incompressibility Theorem of Kolmogorov complexity. In many cases, it is advantageous to use an encoding argument to solve this kind of problem, rather than the traditional probabilistic analysis. Indeed, encoding arguments are generally intuitive for computer scientists, and their strength relies only on the algorithmic construction of a code, rather than on a careful understanding of probability theory. Moreover, encoding arguments give strong results, and the probability in question often decreases exponentially in the parameter of interest. Such results can otherwise be difficult to obtain without relying on concentration inequalities in probability.

We give a simple example illustrating the basic technique of encoding. Since we are overwhelmingly concerned with binary encoding, we will agree now that the base of logarithms in $\log x$ is 2, except when explicitly stated otherwise.

Define a *run of t ones* in a binary string $x_1 \cdots x_n$ to be a sequence of t bits $x_i = \cdots = x_{i+t-1} = 1$ for some $i \in \{1, \ldots, n-t+1\}$. Note that $1 \leq i \leq n$. We will show that a uniformly random bit string is unlikely to contain a run of length significantly greater than $\log n$.

Proposition 1.1. *A uniformly random bit string $x_1 \cdots x_n$ contains a run of at least $\lceil \log n \rceil + s$ ones with probability at most 2^{-s}.*

Proof. Suppose that $x_1 \cdots x_n$ contains a run of $t \geq \lceil \log n \rceil + s$ ones. By definition, there exists some $i \in \{1, \ldots, n - t + 1\}$ such that $x_i = \cdots = x_{i+t-1} = 1$. Therefore, we can represent any such string by writing the value $i - 1$ in binary, followed by the bits $x_1, \ldots, x_{i-1}, x_{i+t}, \ldots, x_n$. Since $0 \leq i \leq n - 1$, then the binary encoding of the value of $i - 1$ uses $\lfloor \log(n-1) \rfloor + 1 = \lceil \log n \rceil$ bits.

In total, the given representation uses

$$\lceil \log n \rceil + n - t \leq n - s$$

bits, by the choice of t, and allows us to represent any n-bit binary string having a run of t or more ones. Therefore, the number of n-bit binary strings with a run of t ones is at most 2^{n-s}, so the probability that a uniformly random n-bit binary string contains a run of t ones is at most $2^{n-s}/2^n = 2^{-s}$, as required. □

The proof of Proposition 1.1 is basic, yet still it serves as a typical example of an encoding argument. The general approach is as follows: we give an encoding of the elementary events which we care about; usually, the objects we encode are forced to have a relatively large substructure which becomes easy to describe. Accordingly, we devise a code which begins with a concise description of this substructure, and then a straightforward encoding of the remaining information which cannot be deduced from this substructure. The fact that this encoding is short proves that there are not very many such events, which upper bounds their probability.

The result of Proposition 1.1 is loose; we can show, appealing directly to probability, that a uniformly random bit string contains a run of length at least $\log n + s$ with probability at most 2^{-s}. In Chapter 5, we will show how our encoding argument can be refined to recover exactly the same result.

1.1 Contributions

Though the study of Kolmogorov complexity is in itself interesting, it carries a certain amount of excess baggage since it makes reference to a description language, such as a Turing machine, and an input for this language. Neither of these is necessary to develop an encoding argument. Accordingly, in this thesis, we present a uniform encoding lemma as a replacement for the Incompressibility Theorem, and basic view of the theory which simplifies the development of encoding arguments, completely ignoring Kolmogorov complexity.

Moreover, the Incompressibility Theorem used in Kolmogorov complexity encoding arguments only immediately serves as an encoding lemma for uniform input distributions. We present more general versions of our encoding lemma, serving for non-uniform input distributions as well.

We present several encoding arguments developed using these tools, with an emphasis on the study of random graphs and hashing algorithms. Though no new results are obtained, we give several original proofs of known results through encoding.

1.2 Overview

Chapter 2 first presents the preliminary theory required to understand encoding arguments. We define prefix-free codes and give some useful specific prefix-free codes. We introduce the concept of information entropy, how it relates to information complexity, and briefly describe the Kolmogorov complexity approach to incompressibility in encoding arguments.

In Chapter 3, we introduce the uniform encoding lemma, which is the main tool used in our encoding arguments, and we describe several applications. We first show how encoding can be used to study permutations and the structure of random binary search trees. We then give arguments to study the performance of different hashing algorithms: the simple balls-in-urns model, linear probing, cuckoo hashing, and 2-choice hashing. We also prove the existence of expanders.

Chapter 4 begins by presenting an encoding lemma for non-uniform input distributions. Using this, we give several properties of the Erdős-Rényi random graph. We also discuss percolation on the torus grid graph.

Finally, in Chapter 5, we show that the results of Chapter 3 and Chapter 4 can be refined by ignoring the integral constraint on codeword lengths and simply relying on a condition of Kraft's inequality to develop tighter encoding arguments. This allows us to develop encoding arguments by thinking in terms of codes, but without sacrificing any precision; for example, it allows us to remove the ceiling from the statement and proof of Proposition 1.1. We also use this approach to prove a result typically known as Chernoff's bound.

In accordance with the elementary nature of encoding arguments, we aim to make all proofs in this thesis easily accessible to someone with only a basic understanding of probability theory.

Chapter 2

Background

As before, we will agree that the base of logarithms in $\log x$ is 2.

Due to the elementary nature of our work, we avoid the presentation of probability theory. For a rigorous introduction to the basics of probability theory, including Kolmogorov's axioms for probability measures, and the study of random variables and their distributions, see the book by Rohatgi and Saleh [35]. We similarly avoid a discussion of basic graph theory; the book by West serves as a suitable introduction [41].

For a random variable $X : \Omega \to \mathbb{R}$ with probability density $p : \Omega \to [0,1]$, we will denote the probability of $x \in \Omega$ as p_x. Finally, we will often refer to the following result as the *union bound*.

Lemma 2.1 (Boole's Inequality)**.** *For any sequence of events $A_1, A_2, \ldots$,*

$$\Pr\left[\bigcup_{i\geq 1} A_i\right] \leq \sum_{i\geq 1} \Pr[A_i].$$

2.1 Prefix-Free Codes

Definition 2.2. For a set Σ, the set Σ^* is defined to be the set of all finite length strings with characters in Σ, including the empty string ϵ.

For example, $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$, the set of finite length binary bit strings.

Definition 2.3. A *code* for the set Ω is an injective function $C : \Omega \to \{0,1\}^*$. We say that Ω is *encoded*, and that the values of C are *codewords*.

The length of the codeword $x \in \Omega$ will be denoted by $|C(x)|$.

Definition 2.4. A string x is a *prefix* of the string y if there exists some string z such that $xz = y$.

x	$C(x)$
a	0
b	100
c	1010
d	1011
e	110
f	1111

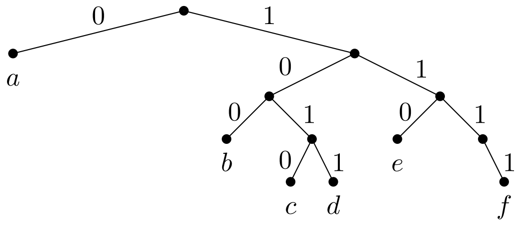


Figure 2.1: A prefix-free code C for the set $\{a, b, c, d, e, f\}$ with its corresponding binary tree.

Definition 2.5. A code $C : \Omega \to \{0, 1\}^*$ is *prefix-free* if, for any $x, y \in \Omega$, $C(x)$ is not a prefix of $C(y)$.

Note that ϵ can never be a codeword in a prefix-free code.

Definition 2.6. A *partial prefix-free code* is a function $C : \Omega \to \{0, 1\}^* \cup \{\bot\}$ which is a prefix-free code on the set $\{x : C(x) \neq \bot\}$, called its *domain.* We will say that $C(x) = \bot$ is undefined, and formally $|\bot| = \infty$.

A partial prefix-free code C is vacuously bijective on its domain, and for all intents and purposes is bijective on the set Ω, in the sense that we are not concerned with undefined codewords.

We can also consider *uniquely decipherable codes*: a code $C : \Omega \to \{0, 1\}^*$ is uniquely decipherable if for any $x_1, \dots, x_n \in \Omega$, the string $C(x_1) \cdots C(x_n)$ can only be written one way as a concatenation of codewords. These are the codes for which there exists an effective decoding procedure. Prefix-free codes are sometimes called instantaneous codes, since any string of codewords from a prefix-free code can be read from left to right and decoded as soon as each codeword is recognized. In fact, uniquely decipherable codes are a strict generalization of prefix-free codes. However, perhaps surprisingly, we will see that prefix-free codes suffice for our purposes.

It is helpful to think of a partial prefix-free code C as a (rooted ordered) binary tree whose leaf set is the domain of C, as in Figure 2.1. In fact, there is a correspondence between rooted ordered binary trees and prefix-free codes. The code for an element $x \in \Omega$ in such a tree is described by the sequence of turns in its root-to-leaf path in this tree, where a left turn (respectively, a right turn) corresponds to a zero bit (respectively, a one bit). Conversely, a prefix-free code describes a binary

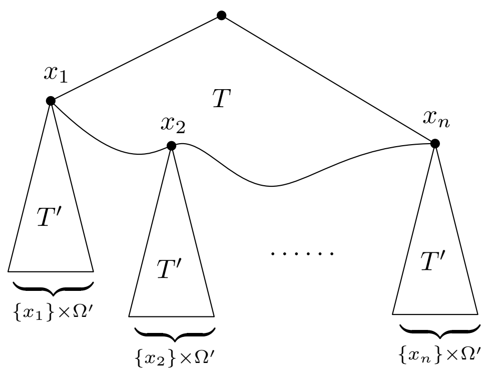


Figure 2.2: The composition of prefix-free codes is prefix-free.

tree recursively, where each codeword beginning with a zero bit (respectively, a one bit) is given to the root's left subtree (respectively, right subtree).

Let $C : \Omega \to \{0,1\}^*$ and $C' : \Omega' \to \{0,1\}^*$ be prefix-free codes corresponding to trees T and T'. Consider the code $C \cdot C' : \Omega \times \Omega' \to \{0,1\}^*$, where

$$(C \cdot C')(x, y) = C(x)C'(y).$$

It is easily seen that the tree of Figure 2.2, in which each leaf of T is replaced by a copy of T', is the tree corresponding to $C \cdot C'$. In other words, the composition of prefix-free codes is prefix-free. We will make use of this fact.

Let h be such that $2^{h-1} < |\Omega| \le 2^h$, and consider the complete binary tree of height h. If we assign the leftmost leaves of this tree to the elements of Ω and discard the remaining leaves, truncating the tree, we effectively obtain a prefix-free code C for Ω for which

$$|C(x)| = h = \lceil \log |\Omega| \rceil, \tag{2.1}$$

i.e. any finite set Ω can be given a prefix-free code for which every codeword has length $\lceil \log |\Omega| \rceil$. We will call this the *fixed-length encoding* of Ω. It is important to note that such a code requires the decoder to know the size of Ω to determine the string's delimitations.

Example 2.7. In some cases, we are interested in encoding a set of size $n!$. Recall Stirling's approximation,

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{r(n)},$$

where $r(n)$ satisfies $\frac{1}{12n+1} < r(n) < \frac{1}{12n}$ [34]. We can use this approximation to bound the length of fixed-length codes of these sets, so

$$\begin{aligned} \log n! &= n \log n - n \log e + \frac{1}{2} \log n + \log \sqrt{2\pi} + \Theta(1/n) \\ &\le n \log n - n \log e + O(\log n). \end{aligned} \tag{2.2}$$

Note that such a fixed-length code would have length $\lceil \log n! \rceil$, but we have chosen to ignore the ceiling in this analysis for reasons to be made clear later.

The correspondence between prefix-free codes and binary trees is useful in proving the next result, which is central to the function of prefix-free encoding arguments.

Definition 2.8. A function $\ell : \Omega \to \mathbb{R}$ is said to *satisfy Kraft's condition* if

$$\sum_{x \in \Omega} 2^{-\ell(x)} \le 1.$$

Lemma 2.9 (Kraft's Inequality [22])**.** *If $C : \Omega \to \{0,1\}^*$ is a prefix-free code, then $|C(\cdot)| : \Omega \to \mathbb{N}$ satisfies Kraft's condition. Conversely, given some function $\ell : \Omega \to \mathbb{N}$ satisfying Kraft's condition, there exists a prefix-free code $C : \Omega \to \{0,1\}^*$ for which $|C(x)| = \ell(x)$ for all $x \in \Omega$.*

Proof. First, consider the tree corresponding to the prefix-free code C. Let each degree 2 node represent a Bernoulli(1/2) random variable, where each one of the left or right children is chosen with probability 1/2; degree 1 nodes automatically choose their only child. Let X denote the leaf node obtained by following this random process from the root. Then

$$1 = \Pr\left[\bigcup_{x \in \Omega} (X = x)\right] = \sum_{x \in \Omega} \Pr[X = x] \ge \sum_{x \in \Omega} 2^{-|C(x)|},$$

with equality if and only if each internal node has degree 2.

Conversely, write $\ell(x_1) \le \cdots \le \ell(x_n)$. Consider the complete binary tree T of height $\ell(x_n)$. We transform T into a tree which corresponds to a prefix-free code satisfying Kraft's condition.

Choose a node u_1 at depth $\ell(x_1)$ (say, the first encountered in a depth first traversal) and remove its subtree from T. The code $C(x_1)$ will correspond to the root-to-leaf path to u_1. Continue this process for depths $\ell(x_2)$ through $\ell(x_n)$. The only way this process can fail is if the tree has no leaves at depth $\ell(x_n)$ at any point. But the number of leaves at depth $\ell(x_n)$ before step j in which the code $C(x_j)$ is assigned is

$$2^{\ell(x_n)} - \sum_{i=1}^{j-1} 2^{\ell(x_n)-\ell(x_i)} = 2^{\ell(x_n)}\left(1 - \sum_{i=1}^{j-1} 2^{-\ell(x_i)}\right) > 2^{\ell(x_n)}\left(1 - \sum_{i=1}^{n} 2^{-\ell(x_i)}\right) \geq 0,$$

so the process completes. If T has any extra leaves which do not correspond to elements of Ω, then prune them from the tree. The remaining tree T corresponds to a prefix-free code C for which $|C(x_i)| = \ell(x_i)$ for each i. □

It is a fact that Lemma 2.9 also holds for uniquely decipherable codes. Therefore, in some sense, we are justified in restricting our study strictly to prefix-free codes, since for any uniquely decipherable encoding of a set, there exists a prefix-free code for the same set with exactly the same codeword lengths [16, 26].

Suppose now that we are interested in encoding an incremental sequence of choices for which the number of options at any point depends upon previous choices. Indeed, we define sets $\Omega_1, \ldots, \Omega_t$ such that Ω_1 is fixed and Ω_{i+1} depends on some sequence of selections $(x_1, \ldots, x_i) \in \Omega_1 \times \cdots \times \Omega_i$. The set $\Omega_1 \times \cdots \times \Omega_t$ is called the *decision space* for the sequence of choices $x = (x_1, \ldots, x_t)$. Suppose that the decision space for x has size d. We can encode x by presenting a sequence of fixed-length encodings for $x_i \in \Omega_i$, and the code for x then has length

$$\sum_{i=1}^{t} \lceil \log |\Omega_i| \rceil \leq \log \prod_{i=1}^{t} |\Omega_i| + t = \log d + t,$$

and indeed, each choice may incur an extra bit in the code, *e.g.* if $|\Omega_i| = 2^n + 1$, then this code has length $nt + t$, but

$$\log d = \log(2^n + 1)^t = t \log(2^n + 1) \leq nt + O(t 2^{-n}).$$

The next result, attributed to Jiang, allows us to encode x without requiring extra bits.

Lemma 2.10 (Lucier *et al.* [24]). *If x is an incremental choice of values with a decision space of size d, then x can be encoded with $\lceil \log d \rceil$ bits.*

Proof. Suppose that $x = (x_1, \ldots, x_t)$ has the decision space $\Omega_1 \times \cdots \times \Omega_t$, and let $k_i \in \{0, \ldots, |\Omega_i| - 1\}$ be the index of x_i in Ω_i, for each $1 \leq i \leq t$. Consider the integer

$$k = k_1 + k_2|\Omega_1| + k_3|\Omega_1||\Omega_2| + \cdots + k_t \prod_{i=1}^{t-1} |\Omega_i|.$$

We know that

$$0 \leq k \leq (|\Omega_1| - 1) + (|\Omega_2| - 1)|\Omega_1| + \cdots + (|\Omega_t| - 1) \prod_{i=1}^{t-1} |\Omega_i| = d - 1,$$

since telescoping kills each $\prod |\Omega_i|$ except for the last one. So our encoding for x is simply a fixed-length encoding of k.

To decode $x = (x_1, \ldots, x_t)$ from the value of k, we can compute and retrieve $k_1 = k \bmod |\Omega_1|$, since we know $|\Omega_1|$. From the value k_1, we determine x_1, which in turn determines $|\Omega_2|$. We can then retrieve k_2 from $k_2|\Omega_1| + k_1 = k \bmod |\Omega_2|$, so we determine x_2. Continuing in this way, we can determine all of $x = (x_1, \ldots, x_t)$. From (2.1), this code has length $\lceil \log d \rceil$. $\square$

This result becomes useful when analyzing constant factors or lower order terms, or if the number of choices is significant (*e.g.* in Section 3.2.1). In many cases, only a constant number of choices are made, which causes only at most an insignificant constant number of wasted bits. Moreover, other approximations in codeword lengths sometimes involve a super-constant lower order term, in which case the result is likely useless (*e.g.* whenever (2.2) is invoked).

2.2 Entropy

Each code $C : \Omega \to \{0, 1\}^*$ has an associated random variable of codeword lengths $|C(\cdot)| : \Omega \to \mathbb{N}$, which is our main object of study. The following definition will help to understand and quantify codeword length optimality.

Definition 2.11. The *entropy* $\mathrm{H}(X)$ of a random variable $X : \Omega \to \mathbb{R}$ with probability density p is

$$\mathrm{H}(X) = \sum_{x \in \Omega} p_x \log \frac{1}{p_x},$$

sometimes denoted by $\mathrm{H}(p)$. The *binary entropy function* $H : [0, 1] \to \mathbb{R}$ is defined by

$$H(\alpha) = \mathrm{H}(\mathrm{Bernoulli}(\alpha)) = \alpha \log \frac{1}{\alpha} + (1 - \alpha) \log \frac{1}{1 - \alpha},$$

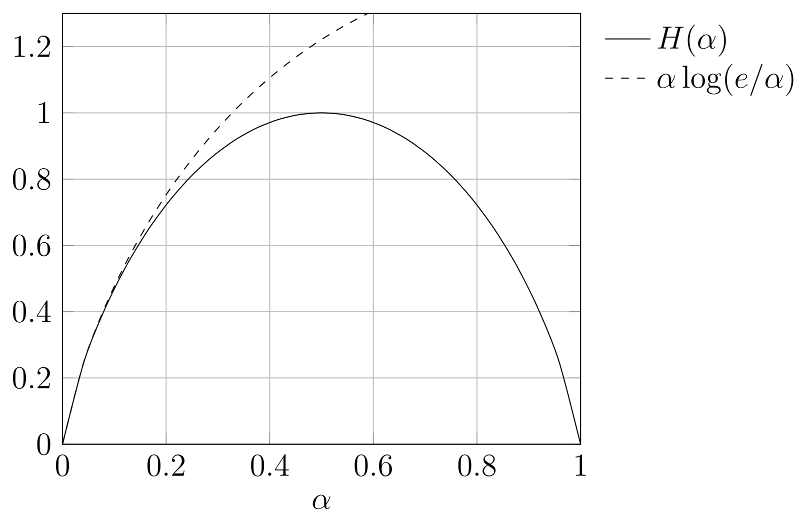


Figure 2.3: Comparison between $H(\alpha)$ and its approximation from (2.3).

where $H(0) = H(1) = 0$.

In this thesis, we sometimes use the following approximation of the binary entropy function:

$$\begin{aligned} H(\alpha) &= \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha} \\ &= \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \left(1 + \frac{\alpha}{1-\alpha}\right) \\ &\leq \alpha \log \frac{1}{\alpha} + \alpha \log e = \alpha \log(e/\alpha), \end{aligned} \tag{2.3}$$

since $1 + x \leq e^x$ for every $x \in \mathbb{R}$. See Figure 2.3 for a comparison between $H(\alpha)$ and its approximation.

We will see how the entropy of X describes the expected amount of information in bits required to represent X. Consider the following examples:

Example 2.12. Let $X : \{1, \ldots, n\} \to \mathbb{R}$ be uniformly distributed. From (2.1), we can encode each element of $\{1, \ldots, n\}$ with a fixed-length code of length $\lceil \log n \rceil$. Moreover, we easily see that $\mathrm{H}(X) = \log n$.

Example 2.13. Let $X : \{1, \ldots, n\} \to \mathbb{R}$ have probability density $p_i = 1/2^i$ for $1 \leq i < n$, and $p_n = 1/2^{n-1}$. Then

$$\lim_{n\to\infty} \mathrm{H}(X) = 2.$$

This is a dramatic difference in entropy from the previous example, demonstrating that a fixed-length code for X would be terribly inefficient, since several elements of X appear with exponentially low probability. In this case, a more judicious coding strategy is required. Consider the code C such that

$$C(i) = \begin{cases} 0^{i-1}\,1 & \text{if } 1 \leq i < n; \\ 0^{n-1} & \text{if } i = n. \end{cases}$$

Notice that this code is prefix-free with expected codeword length exactly $\mathrm{H}(X)$.

We can prove that $\log(1/p_x)$ is the optimal codeword length for any element $x \in \Omega$. We rely on the following classic result.

Theorem 2.14 (Jensen's Inequality). *For any concave function f and any random variable X,*

$$\mathrm{E}[f(X)] \leq f(\mathrm{E}[X]).$$

This inequality is reversed if f is convex.

Theorem 2.15. *If $\ell : \Omega \to \mathbb{R}$ satisfies Kraft's condition, where Ω is endowed with the probability density p, then*

$$\sum_{x\in\Omega} p_x \log(1/p_x) = \mathrm{H}(p) \leq \mathrm{E}[\ell(x)] = \sum_{x\in\Omega} p_x \ell(x).$$

Proof. We examine the value $\mathrm{H}(p) - \mathrm{E}[\ell(x)]$, where

$$\begin{aligned} \mathrm{H}(p) - \mathrm{E}[\ell(x)] &= \sum_{x\in\Omega} p_x \log(1/p_x) - \sum_{x\in\Omega} p_x \ell(x) \\ &= \sum_{x\in\Omega} p_x \log\left(\frac{2^{-\ell(x)}}{p_x}\right) \\ &= \mathrm{E}\left[\log\left(\frac{2^{-\ell(x)}}{p_x}\right)\right] \\ &\leq \log\left(\mathrm{E}\left[\frac{2^{-\ell(x)}}{p_x}\right]\right), \end{aligned} \tag{2.4}$$

by Jensen's inequality, since log is a concave function. Then, (2.4) becomes

$$\log\left(\mathrm{E}\left[\frac{2^{-\ell(x)}}{p_x}\right]\right) = \log\left(\sum_{x\in\Omega} 2^{-\ell(x)}\right) \leq \log 1 = 0$$

by Lemma 2.9. □

x	$p(x)$	$\log(1/p_x)$	$C(x)$	$\lvert C(x)\rvert$
a	0.45	$1.152\dots$	00	2
b	0.21	$2.251\dots$	010	3
c	0.12	$3.058\dots$	0110	4
d	0.12	$3.058\dots$	0111	4
e	0.06	$4.058\dots$	10000	5
f	0.04	$4.643\dots$	10001	5

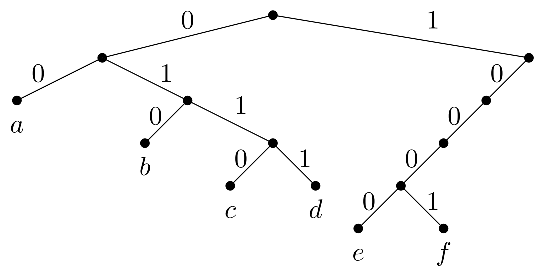


Figure 2.4: The Shannon-Fano code and corresponding tree for a set and probability distribution as given by the construction in Lemma 2.9. Notice the values $\log(1/p_x)$ and $|C(x)|$.

Shannon proved the following result, which describes how entropy relates to codeword lengths; in addition to satisfying the conditions of the previous theorem, codeword lengths must also be integers.

Theorem 2.16 (Noiseless Coding Theorem [36]). *If $C : \Omega \to \{0,1\}^*$ is a prefix-free code with probability density p minimizing its expected codeword length, then*

$$\mathrm{H}(p) \leq \mathrm{E}[\,|C(x)|\,] < \mathrm{H}(p) + 1.$$

2.3 Shannon-Fano Codes

Earlier, we showed how to encode the set Ω using a fixed-length code. Suppose now that Ω is augmented with a probability distribution. Our goal is to provide a better code, in the sense that more probable elements are given appropriately shorter codes.

The *Shannon-Fano code* for Ω is a prefix-free code constructed from a probability distribution over the elements of Ω [15, 36].

Let Ω be endowed with the probability density p. Since

$$\sum_{x\in\Omega} 2^{-\lceil \log(1/p_x)\rceil} \leq \sum_{x\in\Omega} 2^{-\log(1/p_x)} = \sum_{x\in\Omega} p_x = 1,$$

then the proof of Lemma 2.9 gives a deterministic construction for a prefix-free code $C : \Omega \to \{0,1\}^*$, where

$$|C(x)| = \left\lceil \log \frac{1}{p_x} \right\rceil.$$

This is the Shannon-Fano code: see Figure 2.4 for an example construction.

A useful fact about the codes presented in this section is that they are locally optimal in the sense of Theorem 2.16, since the optimal codeword length for $x \in \Omega$ is $\log(1/p_x)$. Local optimality will prove to be more useful to us than global optimality.

Note that if we choose p to be the uniform distribution, then $p_x = 1/|\Omega|$, and the Shannon-Fano code C for Ω is such that

$$|C(x)| = \lceil \log |\Omega| \rceil,$$

as seen previously with (2.1).

We will mostly apply Shannon-Fano codes in the encoding of random bit strings $x = x_1 \cdots x_n$, where all x_i are independent and identically distributed as Bernoulli(p) random variables. We will say that p is the *parameter* for this Shannon-Fano code. Let $n_1(x)$ denote the number of one bits in x, and $n_0(x) = n - n_1(x)$ denote the number of zero bits in x. Then, a particular string x appears with probability exactly $p^{n_1(x)}(1-p)^{n_0(x)}$. In this case, the codeword for x has length

$$|C(x)| = \left\lceil n_1(x) \log \frac{1}{p} + n_0(x) \log \frac{1}{1-p} \right\rceil.$$

Example 2.17. We are often concerned with producing fixed-length encodings for sets of size $\binom{n}{k}$. Since

$$\left(\frac{k}{n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k} = 2^{-nH(k/n)}$$

is the probability that a particular n-bit binary string with exactly k ones appears if each bit is chosen with probability k/n, then the Shannon-Fano code with parameter k/n for this bit string has length $\lceil nH(k/n) \rceil$. Since the number of bit strings of length n containing exactly k ones is $\binom{n}{k}$, then

$$\log \binom{n}{k} \leq \lceil nH(k/n) \rceil,$$

establishing a bound on the length of fixed-length encodings for such sets—in fact, using Theorem 2.15 we can show that this ceiling can be omitted, and

$$\log \binom{n}{k} \leq nH(k/n). \tag{2.5}$$

Applying the estimate from (2.3), we obtain that

$$\log \binom{n}{k} \leq k \log n - k \log k + k \log e. \tag{2.6}$$

This estimate is only useful when k/n is small. We can instead obtain a similar and slightly worse bound than (2.5) by relying on Stirling's approximation and the fact that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

We give an example of when it might be more useful to apply (2.5) over (2.6): suppose we wish to encode a sparse bit string, *i.e.* a bit string in which $n_1(x)$ is relatively small.

Lemma 2.18. *Let $x \in \{0,1\}^n$, where $n_1(x) \leq \alpha n$ for some $0 \leq \alpha \leq 1/2$. Then, we can give a prefix-free code for x of length at most $\log n + nH(\alpha) + O(1)$.*

Proof. Encode x by giving the value $n_1(x)$, and then the set of positions of the $n_1(x)$ ones in the string. This allows us to deduce the entire bit string. Since $n_1(x) \leq \alpha n$ and $0 \leq \alpha \leq 1/2$, then $\binom{n}{n_1(x)} \leq \binom{n}{\alpha n}$, so our code has length at most

$$\log n + nH(\alpha) + O(1). \qquad \square$$

In this case, (2.5) grants us immediate higher order savings for any $0 \leq \alpha < 1/2$ over the trivial encoding of x, since H is strictly increasing on $[0, 1/2]$, and since $H(1/2) = 1$. If instead (2.6) were used, savings would only be obtained for $\alpha \geq 0$ satisfying $\alpha \log \frac{1}{\alpha} + \alpha \log e < 1$, *i.e.* $\alpha < 0.32756\ldots$.

2.4 Elias Codes

We have so far only explicitly discussed encodings for finite sets. In this section, we present a preliminary set of prefix-free codes developed by Elias [13] and used to encode the set of natural numbers. Indeed, we are only ever concerned with encoding positive integer values: once an ordering for a set $\Omega = \{x_1, x_2, \ldots\}$ is established, we can encode the element x_i by giving the value $i \in \mathbb{N}$.

Let $i \geq 1$ be the subject of our encoding. Let s_i be the binary representation of i, so $|s_i| = \lfloor \log i \rfloor + 1$. The *Elias γ-code* for i is then

$$E_\gamma(i) = 0^{|s_i|-1}\, s_i,$$

of length $|E_\gamma(i)| = 2|s_i| - 1 = 2\lfloor \log i \rfloor + 1$. Note that s_i begins with a one. To decode $E_\gamma(i)$ and recover the value i, we read the code from left to right. If n zeroes are observed before the first one, then the next $n+1$ bits represent s_i, from which we obtain i.

The *Elias δ-code* will prove to be more useful to us. As before, let s_i be the binary representation of i, and $|s_i| = \lfloor \log i \rfloor + 1$. Note that $|s_i| \geq 1$. The code is

$$E_\delta(i) = E_\gamma(|s_i|)\ s'_i,$$

where s'_i is the string s_i minus its leading bit, which is always a one. Thus

$$\begin{aligned}|E_\delta(i)| = |E_\gamma(|s_i|)| + |s_i| - 1 &= \lfloor \log i \rfloor + 2\lfloor \log(\lfloor \log i \rfloor + 1)\rfloor + 1 \\ &\leq \log i + O(\log \log i).\end{aligned}$$

To decode $E_\delta(i)$, we read it from left to right, and encounter the self-delimiting code $E_\gamma(|s_i|)$ which tells us the value of $|s_i|$. From this, we then know that the remaining $|s_i| - 1$ bits form the truncated binary representation of i.

Finally, in some cases we may wish to use the *Elias ω-code*, which is the recursive variant of the previous scheme. As before, let s_i be the binary representation of i. The ω-code is

$$E_\omega(i) = \begin{cases} \widetilde{E}_\omega(|s_i| - 1)\ s_i\ 0 & \text{if } i \neq 1; \\ 0 & \text{if } i = 1, \end{cases}$$

where

$$\widetilde{E}_\omega(i) = \begin{cases} \widetilde{E}_\omega(|s_i| - 1)\ s_i & \text{if } i \neq 1; \\ \epsilon & \text{if } i = 1. \end{cases}$$

To decode the string $E_\omega(i)$, we read it from left to right according to the following procedure:

1. If $E_\omega(i) = 0$, then return 1;

2. Let $k \leftarrow 1$, and point to the leftmost bit of $E_\omega(i)$;

3. The next $k+1$ bits represent some integer n;

4. If the next bit is a zero, then return n. If not, go to step 3 with $k \leftarrow n$.

i	s_i	$E_\gamma(i)$	$E_\delta(i)$	$E_\omega(i)$
1	1	1	1	0
2	10	010	0100	100
3	11	011	0101	110
4	100	00100	01100	101000
5	101	00101	01101	101010
66	1000010	0000001000010	00111000010	1011010000100
67	1000011	0000001000011	00111000011	1011010000110
68	1000100	0000001000100	00111000100	1011010001000
69	1000101	0000001000101	00111000101	1011010001010
70	1000110	0000001000110	00111000110	1011010001100

Figure 2.5: Some Elias codes for small integers.

The Elias ω-code has length

$$\begin{aligned}|E_\omega(i)| &= 2 + \lfloor \log i \rfloor + |\widetilde{E}_\omega(\lfloor \log i \rfloor)| \\ &= 3 + \lfloor \log i \rfloor + \lfloor \log \lfloor \log i \rfloor \rfloor + |\widetilde{E}_\omega(\lfloor \log \lfloor \log i \rfloor \rfloor)| \\ &\leq \log i + \log\log i + \log\log\log i + \cdots + \underbrace{\log\cdots\log}_{\log^* i \text{ times}} i + O(\log^* i),\end{aligned}$$

where $\log^* i$ is the slowly growing iterated logarithm function, defined by

$$\log^* i = \begin{cases} 1 + \log^*(\log i) & \text{if } i > 1; \\ 0 & \text{if } i \leq 1. \end{cases}$$

Elias also proved that his δ-codes and ω-codes are asymptotically optimal in the sense of Theorem 2.16.

Proposition 2.19. *If $i \in \{1, \ldots, n\}$ is chosen according to the probability density p, where $p_1 \geq \cdots \geq p_n$, then*

$$\lim_{\mathrm{H}(p)\to\infty} \frac{\mathrm{E}[\,|E_\gamma(i)|\,]}{\mathrm{H}(p)} = 2$$

and

$$\lim_{\mathrm{H}(p)\to\infty} \frac{\mathrm{E}[\,|E_\delta(i)|\,]}{\mathrm{H}(p)} = \lim_{\mathrm{H}(p)\to\infty} \frac{\mathrm{E}[\,|E_\omega(i)|\,]}{\mathrm{H}(p)} = 1.$$

See Figure 2.5 for a comparison between the codes developed in this section.

2.5 Kolmogorov Complexity

Encoding arguments have largely been studied through the lens of Kolmogorov complexity. We present a brief overview of the theory of Kolmogorov complexity and its central results. See the books by Li and Vitányi [23], or Cover and Thomas for a more in-depth introduction [8].

Informally, the Kolmogorov complexity of a bit string x is the length of a "shortest effective binary description" for x, first proposed independently by Kolmogorov, Solomonoff, and Chaitin [7, 21, 37]. This notion of effectiveness, in the end, subsumes a model of computation.

Consider for example the following bit strings:

$$111001000101100101011000; \tag{2.7}$$

$$101010101010101010101010. \tag{2.8}$$

There does not appear to be a concise way of describing (2.7), rather than simply giving the string itself. In other words, this string appears to be highly random. The string (2.8), however, exhibits clear regularity, and can be described by a program which is instructed to print the string '10' twelve times. Such a program has length $O(\log n)$, if n is the length of the given bit string—exponentially less than the length of the description for (2.7), a "random" string.

The Kolmogorov complexity $K(x)$ of a string x is the length of the shortest program in binary which outputs x and halts, in some fixed language. Formally, we fix a universal Turing machine U.

Definition 2.20. The *Kolmogorov complexity* of $x \in \{0,1\}^n$ is

$$K_U(x) = \min\{|p| : U(p) = x\}.$$

Conveniently, the following well-known result indicates that our choice of universal Turing machine does not matter.

Theorem 2.21 (Universality of Kolmogorov Complexity)**.** *For any $x \in \{0,1\}^n$ and for any universal Turing machines U, A, we have that*

$$K_U(x) \leq K_A(x) + O(1).$$

Thus, we disregard the specification of U and always speak of $K(x)$ with some loose constant term in mind.

Unsurprisingly, Kolmogorov complexity is related to entropy in a kind of law of large numbers.

Theorem 2.22. *If $x_1, \ldots, x_n$ are independent and identically distributed Bernoulli(p),*

$$\lim_{n\to\infty} \frac{\mathrm{E}[K(x_1 \cdots x_n)]}{n} = H(p).$$

The discussion of Kolmogorov complexity surrounding (2.7) and (2.8) describes one of the central concepts in the study of encoding arguments through Kolmogorov complexity. By saying that no concise description of (2.7) exists, we essentially said that it is *incompressible.* The following result says that the overwhelming majority of strings are incompressible:

Theorem 2.23 (Incompressibility Theorem)**.** *If $x \in \{0,1\}^n$ is chosen uniformly at random, then for any $s \geq 0$,*

$$\Pr[K(x) \leq n - s] \leq 2^{-s}.$$

These results establish most of the background necessary to develop encoding arguments via Kolmogorov complexity. In such an argument, we are interested in bounding the probability of some event happening. We choose an object at random—when the event in question occurs, we can describe the object concisely, and it becomes compressible. From the previous result, the random object should be largely incompressible with high probability, so we obtain a bound on the probability of the given event.

Chapter 3

Uniform Encoding and Applications

The partial prefix-free codes designed and studied in this chapter will all be denoted $C : \Omega \to \{0,1\}^* \cup \{\perp\}$, where Ω is to be understood from context.

3.1 The Uniform Encoding Lemma

Lemma 3.1. *Let $C : \Omega \to \{0,1\}^* \cup \{\perp\}$ be a partial prefix-free code. Let $x \in \Omega$ be chosen uniformly at random. Then, for any $s \geq 0$,*

$$\Pr[\,|C(x)| \leq \log|\Omega| - s\,] \leq 2^{-s}.$$

Proof. Recall that we chose $|\perp| = \infty$, so if $C(x) \leq \log|\Omega| - s$, then $C(x)$ is defined. Let $k = \log|\Omega| - s$. Since C has $|\Omega|$ codewords, and at most 2^k of those have length at most k, then the probability that $C(x)$ has length at most k is at most

$$\frac{2^k}{|\Omega|} = \frac{2^{\log|\Omega|-s}}{|\Omega|} = \frac{1}{2^s}. \qquad \square$$

This lower tail inequality is analogous to Theorem 2.23 and is similarly crucial in proving results through encoding. The approach is virtually the same as that described in Section 2.5—only now, the length of an effective description of an object is the length of its corresponding prefix-free code.

For example usage of Lemma 3.1, see the proof of Proposition 1.1.

3.2 Permutations and Binary Search Trees

Definition 3.2. A *permutation* σ of size n is a sequence of n distinct integers, sometimes denoted

$$\sigma = (\sigma(1), \ldots, \sigma(n)).$$

Except when explicitly stated, we will assume that the set of integers involved in a permutation of size n is precisely $\{1, \ldots, n\}$. The number of such permutations is $n!$.

3.2.1 Records

Definition 3.3. A *(max) record* in a permutation σ of size n is some value $\sigma(i)$ with $1 \le i \le n$ such that $\sigma(i) = \max\{\sigma(1), \dots, \sigma(i)\}$.

It is easy to see that the probability that $\sigma(i)$ is a record is exactly $1/i$, so the expected number of records in a uniformly random permutation is

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1),$$

the n^{th} harmonic number. It is harder to establish concentration with non-negligible probability. To do this, one first needs to show the independence of certain random variables, which quickly becomes tedious. We instead give an encoding argument to show concentration of the number of records inspired by a technique used by Lucier *et al.* to study the height of random binary search trees [24].

First, we describe an incremental and recursive manner of encoding a permutation σ: begin by providing the first value of the permutation $\sigma(1)$; followed by the indices of the permutation for which σ takes on a value strictly smaller than $\sigma(1)$, including an explicit encoding of the induced permutation on the elements at those indices; and a recursive encoding of the permutation induced on the elements strictly larger than $\sigma(1)$.

If σ contains k elements strictly smaller than $\sigma(1)$, then the size S of the decision space above satisfies the recurrence

$$S(n) = n\binom{n-1}{k} k! \cdot S(n-k-1),$$

with $S(0) = 1$ and $S(1) = 1$. This solves to $S(n) = n!$, so the encoding described above is no better than a fixed-length encoding for σ.

We can readily modify the encoding above to obtain a result about the concentration of records in a random permutation.

Theorem 3.4. *A uniformly random permutation of size n has at least $c \log n$ records with probability at most*

$$2^{-c(1-H(1/c))\log n + O(\log\log n)},$$

for $c > 2$.

Proof. We encode a permutation σ chosen uniformly at random from a set of size $n!$.

Suppose that σ has $t \geq c \log n$ records $r_1 < r_2 < \cdots < r_t$, for some fixed $c > 2$. First, give a bit string $x = x_1 \cdots x_t \in \{0,1\}^t$, where $x_i = 0$ if and only if r_i lies in the first half of the interval $[r_{i-1}, n]$. It follows that $n_1(x) \leq \log n$. In other words, x is a sparse bit string, and $n_1(x)/t \leq 1/c$.

To begin our encoding of σ, we encode the bit string x by giving the set of $n_1(x)$ ones in x; followed by the incremental recursive encoding of σ from earlier. In this case, our knowledge of the value of x_i halves the size of the space of options for encoding the record r_i. In other words, our knowledge of x allows us to encode each record using roughly one less bit per record. More precisely, since the encoding from above recurses t times on decreasing values $n_1 > \cdots > n_t$, then the number of bits spent encoding records is at most

$$\begin{aligned}\sum_{i=1}^{t} \log\lceil n_i/2\rceil &\leq \sum_{i=1}^{t} \log(n_i/2+1) \leq \sum_{i=1}^{t} \log(n_i/2) + \sum_{i=1}^{t} O(1/n_i) \\ &\leq \sum_{i=1}^{t} \log(n_i/2) + H_t = \sum_{i=1}^{t} \log n_i - t + O(\log\log n).\end{aligned}$$

Thus, by Lemma 2.10, the total length of the code is

$$\begin{aligned}|C(\sigma)| &\leq \binom{t}{n_1(x)} + \log n! - t + O(\log\log n) \\ &\leq \log n! - t(1 - H(n_1(x)/t)) + O(\log\log n) \qquad \text{(by (2.5))} \\ &\leq \log n! - c(1 - H(1/c))\log n + O(\log\log n).\end{aligned}$$

This last inequality holds since the function $xH(a/x)$ has derivative

$$\frac{d}{dx} xH(a/x) = \log \frac{1}{1 - a/x},$$

which is positive as long as $0 < a/x < 1$, and so increases as a function of x. Applying Lemma 3.1, we obtain the desired result. □

3.2.2 Tree Height

Every permutation σ determines a binary search tree $\mathrm{BST}(\sigma)$ created through the sequential insertion of the keys $\sigma(1), \ldots, \sigma(n)$. Specifically, if σ_L (respectively, σ_R) denotes the permutation of elements strictly smaller (respectively, strictly larger) than

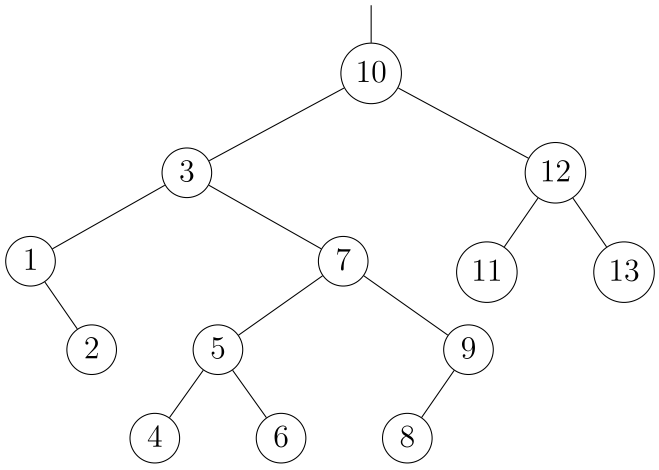


Figure 3.1: The tree BST(σ), where $\sigma = (10, 12, 3, 7, 1, 2, 5, 11, 9, 6, 13, 4, 8)$. σ has records $10, 12, 13$. The nodes $5, 7, 12$ are balanced.

$\sigma(1)$, then BST(σ) has $\sigma(1)$ as its root, with BST(σ_L) and BST(σ_R) as left and right subtrees. For any σ, the value $\sigma(1)$ is called its pivot.

Lucier *et al.* [24] use an encoding argument via Kolmogorov complexity to study the height of BST(σ). They show that for a uniformly chosen permutation σ, the tree BST(σ) has height at most $c \log n$ with probability $1 - O(1/n)$ for $c = 15.498...$; we can extend our result on records to obtain a tighter result.

For a node u, let $s(u)$ denote the number of nodes in the tree rooted at u. Then, u is called balanced if $s(u_L), s(u_R) > s(u)/4$, where u_L and u_R are the left and right subtrees of u respectively. In other words, a node is said to be balanced if it occurs in the middle half of its subrange. See Figure 3.1.

Theorem 3.5. *Let σ be a uniformly random permutation of size n. Then, BST(σ) has height $O(\log n)$ with high probability.*

Proof. Suppose that the tree BST(σ) has a rooted path $Y = y_1 \cdots y_t$ of length $t \geq c \log n$ for some fixed $c > 2/\log(4/3)$.

Our encoding for σ has two parts. The first part includes an encoding of the value y_t, and a bit string $x = x_1 \cdots x_t$, where $x_i = 1$ if and only if y_i is balanced. From our definition, if y_i is balanced, then

$$s(y_{i+1}) \leq (3/4)s(y_i),$$

and since $n_1(x)$ counts the number of balanced nodes along Y, then

$$1 \leq (3/4)^{n_1(x)} n \iff n_1(x) \leq \log_{4/3} n.$$

The second part of our encoding is recursive: first, encode the value of the root r using $\log\lceil n/2 \rceil$ bits. If the path Y goes to the left of r, then specify the values in the right subtree of r, including an explicit encoding of the permutation induced by these values; and recursively encode the permutation of values in the left subtree of r. If, instead, Y goes to the right of r, we proceed symmetrically.

The first part of our encoding uses at most

$$\log n + tH\left(\frac{n_1(x)}{t}\right)$$

bits. The same analysis as performed in Theorem 3.4 shows that the second part of our encoding has length at most

$$\log n! - t + O(\log\log n).$$

Applying Lemma 3.1, we see that $\mathrm{BST}(\sigma)$ has height at most $c \log n$ with probability $1 - O(1/n)$ for $c > 2/\log(4/3)$ satisfying

$$c\left(1 - H\left(\frac{1}{c\log(4/3)}\right)\right) > 1,$$

and a computer-aided calculation shows that $c = 8.12669...$ is the minimal solution to this inequality. $\square$

Remark 3.6. Devroye [9] shows how the length of the path to the key i in $\mathrm{BST}(\sigma)$ relates to the number of records in σ. Specifically, he notes that the number of records in σ is the number of nodes along the rightmost path in $\mathrm{BST}(\sigma)$. Since the height of a tree is the length of its longest root-to-leaf path, we obtain as a corollary that the number of records in a uniformly random permutation is $O(\log n)$ with high probability; the result from Theorem 3.4 only improves upon the implied constant.

3.3 Hashing

In this section, we tackle the problem of implementing the hash table data type. For an introduction to the study of data structures in general and including hash tables, see the book by Morin [27]. A hash table supports the following operations:

- Insert(x): inserts x into the hash table;
- Delete(x): deletes x from the hash table;
- Search(x): determines whether or not x is currently held in the hash table.

Many hash tables are implemented by storing elements in an array. As such, to store n elements from a set Ω, we typically require an array A of size $m \geq n$, and some method of assigning array positions to elements of Ω. In each case, we will determine these array indices using Uniform($\{1, \ldots, m\}$) random variables which we call *hash functions*.

Any good hash table performs all operations in $O(1)$ expected time. It can be shown, however, that it is impossible for a hash table to achieve $O(1)$ worst-case time for all operations [12]. We are thus concerned with obtaining $O(1)$ expected time for all operations, and minimizing the worst-case time of operations in hash table implementations.

3.3.1 Balls in Urns

Our first look at hashing employs a simple solution. We are concerned with hashing the set $\Omega = \{x_1, \ldots, x_n\}$ using a hash function $h : \Omega \to \{1, \ldots, n\}$. In practice, each value of h points to one of n lists, to which an element $x \in \Omega$ is appended during insertion. This is an instance of *hashing with chaining.* The worst case time of any operation is the maximum number of potential hashing collisions, or the length of the longest list.

It may be helpful to think of this scenario as modeled by balls and urns: effectively, we throw n balls into n urns independently and uniformly at random, and we are concerned with the maximum number of balls in any urn. We call this the *balls-in-urns* model hash table.

Theorem 3.7. *The cost of any operation in the balls-in-urns model hash table is* $O(\log n/\log\log n)$ *with probability* $1 - O(1/n)$.

Proof. We identify the values of h with lists. Since each $h(x_i)$ is independent and uniformly distributed in $\{1, \ldots, n\}$, then the sequence $h(\Omega) = (h(x_1), \ldots, h(x_n))$ is uniformly distributed in a set of size n^n, and so will be the subject of our encoding.

#	$h(x)$	0	1	2	3	4	5	6	7	8	9
1	5						x_1				
2	3				x_2		–				
3	9				–		–				x_3
4	8				–		–			x_4	–
5	8	x_5			–		–			–	–
6	2	–		x_6	–		–			–	–

Figure 3.2: A linear probing hash table undergoing the sequential insertion of elements $x_1, \ldots, x_6$. In the end, we see that x_2 and x_6 are in a block of size 2; x_1 in a block of its own; and x_3, x_4, x_5 are in a block of size 3.

Suppose that the list corresponding to $h(x_i)$ has t elements. Our encoding begins by specifying the value of i; then the set of t elements contained in this list; and finally the positions of the remaining $n - t$ hashed elements. Our code then has length

$$\begin{aligned}
|C(h(\Omega))| &\leq \log n + \log \binom{n}{t} + (n-t)\log n + O(1) \\
&\leq \log n + t\log n - t\log t + t\log e + (n-t)\log n + O(1) \qquad \text{(by (2.6))} \\
&= \log n^n + \log n - t\log(t/e) + O(1) \\
&\leq \log n^n - s,
\end{aligned}$$

as long as t satisfies $t\log(t/e) \geq \log n + s + O(1)$. Particularly, if we let $s = \log n$, then we can choose

$$t = \left\lceil \frac{(2+\varepsilon)\log n}{\log\log n} \right\rceil$$

for any $\varepsilon > 0$ and sufficiently large n. We finish by applying Lemma 3.1. $\square$

3.3.2 Linear Probing

In a *linear probing* hash table, we require that $m = cn$, for some $c > 1$. We again hash the set Ω using a hash function $h : \Omega \to \{1, \ldots, m\}$. In this case, h points to array indices in some array A. To insert x into the hash table, we insert it into the first empty spot among $A[h(x)], A[h(x)+1], A[h(x)+2], \ldots$, where each array index is taken modulo m. Since $m > n$, this is always possible.

A maximal consecutive sequence of occupied indices in A is called a *block*. The running time of any operation in a linear probing hash table is bounded by the size of the largest block. See Figure 3.2 for an example execution of the above algorithm

Lemma 3.8. *Fix some $x \in \Omega$. In a linear probing hash table with $c > e$, the probability that x belongs to a block of size at least t satisfying*

$$t \log(c/e) - \log t \geq s + O(1)$$

is at most 2^{-s}.

Proof. Again, we encode the sequence $h(\Omega) = (h(x_1), \dots, h(x_n))$ which, for the same reason, is uniformly chosen from a set of size m^n.

Suppose that x belongs to a block of size t. Our encoding begins by providing the first index i of the block containing x; then the set of the remaining $t-1$ values in the block, which we call $y_1, \dots, y_{t-1}$; followed by enough information to decode the hash values $h(x), h(y_1), \dots, h(y_{t-1})$; and finally, the hash values of the $n-t$ remaining elements of Ω.

Note that each of the values $h(x) - i, h(y_1) - i, \dots, h(y_{t-1}) - i$ are within the range $0, 1, \dots, t-1$ modulo m, and so the hash values $h(x), h(y_1), \dots, h(y_{t-1})$ can be encoded using $\lceil t \log t \rceil$ bits. From this discussion, we see that our code uses at most

$$\begin{aligned} |C(h(\Omega))| &\leq \log n + \log \binom{n}{t-1} + t \log t + (n-t) \log m + O(1) \\ &\leq \log m^n + t \log n + (t-1) \log e + \log t - t \log m + O(1) \qquad \text{(by (2.6))} \\ &\leq \log m^n + t \log(e/c) + \log t + O(1) \\ &\leq \log m^n - s \end{aligned}$$

bits, as long as t is such that

$$t \log(c/e) - \log t \geq s + O(1).$$

The result follows from Lemma 3.1. □

Theorem 3.9. *A linear probing hash table with $c > e$ achieves $O(1)$ expected time for all operations.*

Proof. Let S denote the size of the largest block in such a hash table. From Lemma 3.8, any fixed $x \in \Omega$ is contained in a block of size at least $2s/\log(c/e) + d$ with probability at most 2^{-s}, for some constant d. Then,

$$\begin{aligned}
\mathrm{E}[S] = \sum_{s=1}^{\infty} \Pr[S \geq s] &= \sum_{s=1}^{d} \Pr[S \geq s] + \sum_{s=1}^{\infty} \Pr[S \geq s + d] \\
&\leq d + \sum_{s=1}^{\infty} 2^{-s \log(c/e)/2} = d + \sum_{i=1}^{\infty} \left(\frac{c}{e}\right)^{-s/2} = O(1). \qquad \square
\end{aligned}$$

3.3.3 Cuckoo Hashing

Cuckoo hashing is a simple and efficient hashing solution developed by Pagh and Rodler which achieves $O(1)$ deterministic search time [30]. The encoding arguments in this section, due to Pătraşcu [31], neatly establish some of the nice properties of cuckoo hashing.

The hash table consists of two arrays A and B of size $m = 2n$, and two hash functions $h, g : \Omega \to \{1, \ldots, m\}$. To insert an element x into the hash table, we insert it into $A[h(x)]$; if $A[h(x)]$ already contains an element y, we insert y into $B[g(y)]$; if $B[g(y)]$ already contains some element z, we insert z into $A[h(z)]$, etc. If an empty location is eventually found, the algorithm terminates successfully. If a cycle is detected, then the hash table is rebuilt using different hash functions. Any element x either is held in $A[h(x)]$ or $B[g(x)]$, so we can search for x in $O(1)$ time.

To study the performance of insertion in cuckoo hashing, we study the random bipartite *cuckoo graph* G, where $V(G) = A \cup B$ with bipartition (A, B), with each vertex corresponding either to a location in the array A or B, and with edge multiset $E(G) = \{(h(x), g(x)) : x \in \Omega\}$. Note that in this graph, each edge corresponds to an element of x, and each vertex corresponds to a position in the hash table.

An edge-simple path is a path which uses each edge at most once. Such a path in this graph describes the potential motion of keys in a hash table insertion. Thus, in bounding the length of edge-simple paths in the cuckoo graph, we bound the worst case insertion time.

Lemma 3.10. *The cuckoo graph has an edge-simple path of length greater than* $s + \log n + O(1)$ *with probability at most* 2^{-s}*.*

Proof. We encode G by presenting its set of edges. Since each endpoint of an edge is chosen independently and uniformly at random from a set of n choices, then the set of all edges is chosen uniformly from a set of size m^{2n}.

Suppose that some vertex v is the endpoint of an edge-simple path of length t; such a path has $t+1$ vertices and t edges. In the encoding, we present the t edges of the path in order; then, we indicate whether $v \in A$ or $v \in B$; and we give the $t+1$ vertices in order starting from v; and finally, the remaining $2n-2t$ endpoints of edges of the graph. This code has length

$$\begin{aligned}|C(G)| &\leq t\log n + 1 + (t+1)\log m + (2n-2t)\log m + O(1)\\ &= \log m^{2n} + t\log n - t\log m + \log m + O(1)\\ &= \log m^{2n} - t + \log n + O(1) && \text{(since } m = 2n\text{)}\\ &\leq \log m^{2n} - s\end{aligned}$$

for $t \geq s + \log n + O(1)$. We finish by applying Lemma 3.1. □

Choosing $s = \log n$ and applying the union bound, we see that successful insertion takes time at most $2\log n + O(1)$ with probability $1 - O(1/n)$.

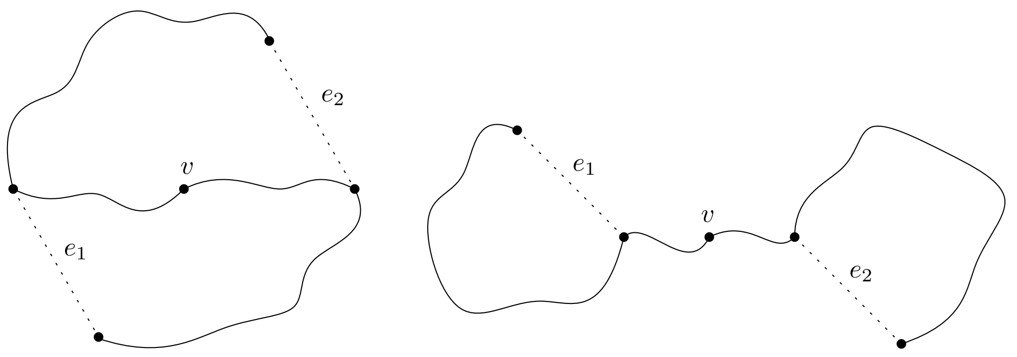
The cuckoo hashing insertion algorithm can fail if some subgraph of the cuckoo graph contains more edges than vertices, since edges correspond to keys, and vertices correspond to array locations.

Lemma 3.11. *Fix some vertex $v \in A$. The probability that v is part of a subgraph with more edges than vertices is $O(1/n^2)$.*

Proof. Suppose that v is part of a subgraph with more edges than vertices, and in particular a minimal such subgraph with $t+1$ edges and t vertices. Such a subgraph appears exactly as in Figure 3.3. For every such subgraph, there are two edges e_1 and e_2 whose removal disconnects the graph into two paths of length t_1 and t_2 starting from v, where $t_1 + t_2 = t - 1$.

We encode G by presenting Elias δ-codes for the values of t_1 and t_2; then the edges of the above paths in order; then the vertices of the paths in order; and the edges e_1 and e_2; and finally the remaining $2n - 2(t+1)$ endpoints of edges in the graph. Such a code has length

$$\begin{aligned}|C(G)| &\leq (t-1)(\log n + \log m) + 2\log n + (2n - 2(t+1))\log m + O(\log t)\\ &= \log m^{2n} - 2\log m - t + O(\log t)\\ &\leq \log m^{2n} - 2\log n + O(1).\end{aligned}$$



(a) A cycle with a chord. **(b)** Two cycles connected by a path.

Figure 3.3: The potential minimal subgraphs of the cuckoo graph.

We finish by applying Lemma 3.1. □

From the union bound, we obtain the following:

Corollary 3.12. *Cuckoo hashing succeeds upon insertion of n elements with probability $1 - O(1/n)$.*

3.3.4 2-Choice Hashing

We showed in Section 3.3.1 that if n balls are thrown independently and uniformly at random into n urns, then the maximum number of balls in any urn is $O(\log n/\log\log n)$ with high probability. In 2-choice hashing, another instance of hashing with chaining, each ball is instead given a choice of two urns, and the urn containing the fewer balls is preferred.

More specifically, we are given two hash functions $h, g : \Omega \to \{1, \dots, m\}$. Each value of h and g points to one of m lists. The element $x \in \Omega$ is appended to the smaller list between $h(x)$ and $g(x)$ during an insertion. The worst case search time is at most the maximum number of hashing collisions, or the length of the longest list.

Perhaps surprisingly, the simple change of having two choices instead of only one results in an exponential improvement over the strategy of Section 3.3.1. Indeed, 2-choice hashing was first studied by Azar *et al.* [2], who showed that the expected maximum size of an urn is $\log\log n + O(1)$. Our encoding argument is based on Vöcking's use of witness trees to analyze 2-choice hashing [40].

As in Section 3.3.3, we use random graphs to study the performance of 2-choice hashing. Let G be the random multigraph with $V(G) = \{1, \dots, m\}$, where $m = cn$

for some constant $c > e$, and $E(G) = \{(h(x), g(x)) : x \in \Omega\}$.

Lemma 3.13. *G has a subgraph with more edges than vertices with probability $O(1/n)$.*

Proof. The proof is similar to that of Lemma 3.11, with an additional application of the union bound. More specifically, we can encode G by giving the same encoding as in Lemma 3.11, with an additional bit for each edge (u, v) in the encoding, indicating whether $u = h(x)$ and $v = g(x)$, or $u = g(x)$ and $v = h(x)$. Our code thus has length

$$\begin{aligned}|C(G)| &\le (t-1)(\log n + \log m) + 2\log n + (2n - 2(t+1))\log m + t + O(\log t)\\ &= \log m^{2n} - 2\log n - t\log c + t + O(\log t)\\ &\le \log m^{2n} - 2\log n + O(1),\end{aligned}$$

since $\log c > 1$. □

Lemma 3.14. *G has a component of size at least $(2/\log(c/e))\log n + O(1)$ with probability $O(1/n)$.*

Proof. Suppose G has a connected component X with $t - 1$ edges and at most t vertices. We encode the edge set of G by providing the set of vertices in a spanning tree of X, and we choose to root this tree at the first vertex described in this set; then the shape of this tree; and the labels of the $t - 1$ edges encountered in an inorder traversal of this tree; and finally the remaining $2(n - t + 1)$ endpoints of edges. By Cayley's formula and since G is a multigraph, there are at most t^{t-2} rooted spanning trees of G. In total, our code has length

$$\begin{aligned}|C(G)| &\le \log\binom{m}{t} + (t-2)\log t + (t-1)\log n + 2(n-t+1)\log m + O(1)\\ &\le \log m^{2n} - t\log c + t\log e - 2\log t + \log n + O(1) && \text{(by (2.6))}\\ &\le \log m^{2n} - s,\end{aligned}$$

as long as t is such that

$$t\log(c/e) + 2\log t \ge s + \log n + O(1).$$

In particular, if we choose $s = \log n$, applying Lemma 3.1 tells us that X has size at least $(2/\log(c/e))\log n + O(1)$ with probability $O(1/n)$. □

Suppose that when x is inserted, it is placed in a list with t other elements. Then, we say that the age of x is $a(x) = t$.

Theorem 3.15. *The cost of any operation in 2-choice hashing is at most* $\log\log n + O(1)$ *with probability* $1 - O(1/n)$.

Proof. Suppose that some element x has $a(x) = t$. This leads to a binary *witness tree* T of height t as follows.

The root of T is the element x. When x was inserted into the hash table, it had to choose between the lists $h(x)$ and $g(x)$, both of which contained at least $t - 1$ elements; in particular, $h(x)$ has an element x_h with $a(x_h) = t - 1$, and $g(x)$ has an element x_g with $a(x_g) = t - 1$. The elements x_h and x_g become left and right children of x in T. The process continues recursively. If some element appears more than once on a level, we only recurse on its leftmost occurence.

Note that each vertex of T corresponds to the edge $(h(x), g(x))$ of G. Moreover, the subgraph of G induced by the edges $\{(h(x), g(x)) : x \in V(T)\}$ is connected.

Suppose that some node appears more than once on a level of T. From above, this means that some component of G has a cycle. If this happens twice, then G has a subgraph with more edges than vertices. By Lemma 3.13, this happens with probability $O(1/n)$. If instead this happens at most once, then T has at most one subtree removed, so T has at least 2^t nodes. If we choose $t = \lceil \log\log n + d \rceil$, then T has at least $2^d \log n$ nodes, which we know from Lemma 3.14 happens with probability $O(1/n)$ for a sufficiently large choice of the constant d. □

3.4 Bipartite Expanders

Expanders are families of graphs which share some isoperimetric quality, *i.e.* where subgraphs “expand” in their neighbourhoods. These graphs have received much research attention, and have been shown to have many applications in computer science. See, for instance, the survey by Hoory, Linial, and Wigderson [19].

Though expanders have proven to have fascinating qualities and be remarkably useful, they have yet been difficult to construct. The first proof of existence of expanders by Pinkser was probabilistic [32]. The first explicit construction, given by

Margulis, relied on advanced concepts in representation theory [25]. Later constructions have become simpler to analyze.

We offer an encoding argument to prove that a random bipartite graph is an expander with high probability. Indeed, expanders are often called "pseudorandom" graphs due to their many shared properties with random graphs; one should expect, just as a random graph is incompressible, so too are expanders.

There are many different notions of expansion. We will consider what is commonly known as vertex expansion in bipartite graphs.

Definition 3.16. Fix some $0 < \alpha \leq 1$. A bipartite graph G with bipartition (A, B) is a (c, α)-*expander* if

$$\min_{\substack{A' \subseteq A \\ |A'| \leq \alpha |A|}} \frac{|N(A')|}{|A'|} \geq c,$$

where $N(A') \subseteq B$ is the set of neighbours of A' in G.

Let G be a random bipartite graph with bipartition (A, B) where $|A| = |B| = n$ and such that each vertex in A is connected to three vertices chosen uniformly at random in B.

Theorem 3.17. *For some constant $\alpha > 0$, G is a $(3/2, \alpha)$-expander with probability at least $1 - O(n^{-1/2})$.*

Proof. Suppose that G is not a $(3/2, \alpha)$-expander, so there exists some $A' \subseteq A$ where $|A'| \leq \alpha |A|$ and

$$\frac{|N(A')|}{|A'|} < 3/2.$$

We encode the graph G by encoding its edge set, which is chosen uniformly at random from a set of size n^{3n}.

Our encoding presents the value $k = |A'|$ using an Elias γ-code; then the sets A' and $N(A')$, and the edges between them; and finally, the remaining edges of the graph. This costs

$$\begin{aligned}
|C(G)| &\leq 2\log k + \log\binom{n}{k} + \log\binom{n}{3k/2} + 3k\log(3k/2) + (3n - 3k)\log n + O(1) \\
&\leq \log n^{3n} - (k/2)\log n + (k/2)\log k + \beta k + 2\log k + O(1) \qquad \text{(by (2.6))} \\
&= \log n^{3n} - s(k)
\end{aligned}$$

bits, where $\beta = (3/2)\log(3/2) + (5/2)\log e$. Note that

$$\frac{d^2}{dk^2}s(k) = \frac{4-k}{2k^2}\log e,$$

so $s(k)$ is concave for all $k \geq 4$. Thus, $s(k)$ is minimized either when $k = 1, 2, 3, 4$, or $k = \alpha n$. We have

$$s(1) = (1/2)\log n + c_1, \qquad s(2) = \log n + c_2,$$
$$s(3) = (3/2)\log n + c_3, \qquad s(4) = 2\log n + c_4,$$

for constants c_1, c_2, c_3, c_4. For $k = \alpha n$ we have

$$s(\alpha n) = (\alpha n/2)\log\left(\frac{1}{2^{2\beta}\alpha}\right) - 2\log \alpha n + O(1)$$

and $2^{-s(\alpha n)} = 2^{-\Omega(n)}$ for $\alpha < (1/2)^{2\beta}$. In each case, apply Lemma 3.1. □

Chapter 4

Non-Uniform Encoding and Applications

The partial prefix-free codes designed and studied in this chapter will all be denoted $C : \Omega \to \{0,1\}^* \cup \{\bot\}$, where Ω is to be understood from context.

4.1 The Non-Uniform Encoding Lemma

We can extend Lemma 3.1 to handle non-uniform input distributions with a result of Barron, and independently rediscovered through a joint effort between Devroye, Lugosi, and Morin [10]. First, recall the following classic result.

Theorem 4.1 (Markov's Inequality)**.** *For a non-negative random variable X with finite expectation, and any $a > 0$,*

$$\Pr[X \geq a] \leq \mathrm{E}[X]/a.$$

Lemma 4.2 (Barron [3])**.** *Let $C : \Omega \to \{0,1\}^* \cup \{\bot\}$ be a partial prefix-free code. Let $x \in \Omega$ be chosen with probability p_x. Then, for any $s \geq 0$,*

$$\Pr[\,|C(x)| \leq \log(1/p_x) - s\,] \leq 2^{-s}.$$

Proof. We see that

$$\begin{aligned}\Pr[\,|C(x)| \leq \log(1/p_x) - s\,] &= \Pr[\,\log(1/p_x) - |C(x)| \geq s\,] \\ &= \Pr\big[2^{\log(1/p_x)-|C(x)|} \geq 2^s\big] \\ &\leq 2^{-s}\,\mathrm{E}\big[2^{\log(1/p_x)-|C(x)|}\big] \qquad (4.1)\end{aligned}$$

by Markov's inequality. Further, by definition of the expected value, (4.1) becomes

$$2^{-s}\,\mathrm{E}\big[\,2^{\log(1/p_x)-|C(x)|}\,\big] = 2^{-s}\left(\sum_{C(x)\neq\bot} p_x 2^{\log(1/p_x)-|C(x)|}\right) = 2^{-s}\left(\sum_{C(x)\neq\bot} 2^{-|C(x)|}\right).$$

By Lemma 2.9, $\sum_{C(x)\neq\bot} 2^{-|C(x)|} \leq 1$, and the result is obtained. □

Indeed, this non-uniform encoding lemma immediately implies Lemma 3.1 by choosing $p_x = 1/|\Omega|$.

Proposition 4.3. *Let $x_1, \ldots, x_n$ be independent Bernoulli(p) random variables. Then, the probability that $x = x_1 \cdots x_n \in \{0,1\}^n$ has a run of ones of length at least*

$$\frac{s + \lceil \log n \rceil + 1}{\log(1/p)}$$

is at most 2^{-s}.

Proof. The proof is almost identical to that of Proposition 1.1. Suppose that x has a run of ones of length $t \geq (s + \lceil \log n \rceil + 1)/\log(1/p)$.

Our code begins with the index where this run begins, which allows us to deduce t one bits of x; and then the Shannon-Fano code with parameter p for the remaining bits. Recall that $n_1(x)$ and $n_0(x)$ denote the number of one and zero bits in x, respectively. In total, this code has length

$$\begin{aligned}
|C(x)| &= \lceil \log n \rceil + \left\lceil (n_1(x) - t) \log \frac{1}{p} + n_0(x) \log \frac{1}{1-p} \right\rceil \\
&\leq \log \frac{1}{p_x} + \lceil \log n \rceil - t \log \frac{1}{p} + 1 \\
&\leq \log \frac{1}{p_x} - s
\end{aligned}$$

by our choice of t, and since $p_x = p^{n_1(x)}(1-p)^{n_0(x)}$. The result follows from Lemma 4.2. □

This result appears even more unnatural than that of Proposition 1.1—again, we can prove that this bit string has a run of length at least $(s + \log n)/\log(1/p)$ by appealing directly to probability. We will show how this can be resolved in Chapter 5.

4.2 The Erdős-Rényi Random Graph

Definition 4.4. The *Erdős-Rényi random graph $G_{n,p}$* is the space of random undirected graphs on n vertices in which each edge is included independently at random with probability p.

The study of the Erdős-Rényi random graph model played an important role in the popularization of the now pervasive probabilistic method [1, 14]. In general, probabilistic proofs appear to be easily adapted into encoding arguments. We show how it is simple to recover some of Erdős and Rényi's classic results about $G_{n,p}$.

Lemma 4.5. *The probability that $G \in G_{n,p}$ has a partition (U, W) of its vertex set such that $|U| = k$ and there is no edge of G with one endpoint in U and the other in W is at most $O\big(\binom{n}{k}(1-p)^{k(n-k)}\big)$.*

Proof. We encode G by giving its adjacency matrix A, which determines the graph. The entire matrix is determined by its portion which lies above its diagonal, which is a random bit string of length $\binom{n}{2}$ in which each bit appears with probability p.

We give a list of the k vertices in U, which determines $k(n-k)$ non-edges of the graph between U and W; and a Shannon-Fano code with parameter p for the remaining $\binom{n}{2} - k(n-k)$ edges of the graph. This code has length

$$\begin{aligned}|C(G)| &\le \log \binom{n}{k} + n_1(A) \log \frac{1}{p} + (n_0(A) - k(n-k)) \log \frac{1}{1-p} + O(1) \\ &= \log \frac{1}{p_G} + \log \binom{n}{k} - k(n-k) \log \frac{1}{1-p} + O(1) \\ &= \log \frac{1}{p_G} + \log\left(O\left(\binom{n}{k}(1-p)^{k(n-k)}\right)\right).\end{aligned}$$

We finish by applying Lemma 4.2. □

The typical proof of the above result, given by the probabilistic method, says that this probability is at most $\binom{n}{k}(1-p)^{k(n-k)}$. Nevertheless, Lemma 4.5 is sufficient to obtain the classic threshold connectivity results for the Erdős-Rényi random graph.

Theorem 4.6. *When $p = (1+\varepsilon)\frac{\ln n}{n}$, $G \in G_{n,p}$ has no isolated vertices with high probability.*

Proof. Choose $k = 1$ in Lemma 4.5. Then, the probability that G has an isolated vertex is at most

$$O\left(n\left(1-(1+\varepsilon)\frac{\ln n}{n}\right)^{n-1}\right) = O\big(ne^{-(1+\varepsilon)\ln n}(1+o(1))\big) = O(n^{-\varepsilon}). \qquad \square$$

Theorem 4.7. *When $p = (1+\varepsilon)\frac{\ln n}{n}$, $G \in G_{n,p}$ is connected with high probability.*

Proof. This follows from Lemma 4.5 and the union bound. □

Theorem 4.8. *When $p = \frac{1}{\alpha n}$ for $\alpha > e$, the graph $G \in G_{n,p}$ has a component of size at least*

$$\frac{\log n + s + O(1)}{\log(\alpha/e)}$$

with probability at most 2^{-s}.

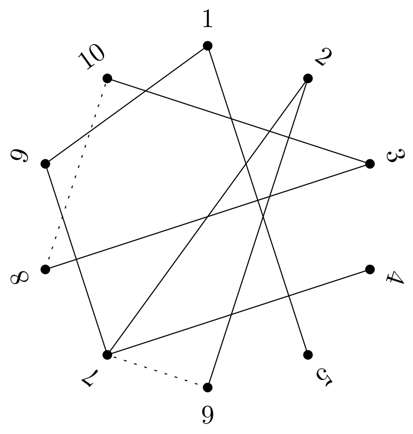
Proof. We again encode the adjacency matrix A of G. Suppose that G has a component of size t. We encode G by giving these t vertices; the shape of a spanning tree for this component; and a Shannon-Fano code with parameter p for the remaining edges of the graph. This code has length

$$\begin{aligned}
|C(G)| &\leq \log\binom{n}{t} + (t-2)\log t + (n_1(A) - (t-1))\log\frac{1}{p} + n_0(A)\log\frac{1}{1-p} + O(1) \\
&\leq \log\frac{1}{p_G} + t\log n - t\log t + t\log e + t\log t - (t-1)\log\frac{1}{p} + O(1) \quad \text{(by (2.6))} \\
&\leq \log\frac{1}{p_G} + \log n + t\log e - t\log\alpha + O(1) \\
&= \log\frac{1}{p_G} + \log n - t\log\frac{\alpha}{e} + O(1) \\
&\leq \log\frac{1}{p_G} - s
\end{aligned}$$

by our choice of t. We finish by applying Lemma 4.2. □

Remark 4.9. This result and its proof are similar to that of Lemma 3.14 from our study of random multigraphs in 2-choice hashing. Indeed, the random multigraph we studied contained exactly n edges for cn vertices, *i.e.* n/c edges for n vertices, and the Erdős-Rényi graph from Theorem 4.8 is expected to contain $(n-1)/(2\alpha)$ edges for n vertices. Since the random graph models are so similar, and since the edge densities we are concerned with are practically identical, this observation should be of no surprise.

A triangle in a graph is a K_3 subgraph. Our nextt encoding argument bounds the probability that $G_{n,c/n}$ has no triangle. It is easy to see that the expected number of triangles in $G_{n,c/n}$ is $\binom{n}{3}(c/n)^3 = c/6 - O(1/n)$, so $G_{c,c/n}$ is expected to have at least one triangle for large enough values of c. However, a typical proof that $G_{n,c/n}$ has a triangle with non-negligible probability usually uses the second moment method which involves computing the variance of the number of triangles [38]. To show that $G_{n,c/n}$ has a triangle with exponential probability is even more complicated, and a



$$\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
2 & & & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
3 & & & & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
4 & & & & & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & & & & & & 0 & 0 & 0 & 0 & 0 \\
6 & & & & & & & \boxed{0} & 0 & 0 & 0 \\
7 & & & & & & & & 0 & 1 & 0 \\
8 & & & & & & & & & 0 & \boxed{0} \\
9 & & & & & & & & & & 0
\end{array}$$

Figure 4.1: The random graph $G_{n,c/n}$ has triangles when c is large enough. The depicted zero bits in the adjacency matrix can be deduced from the first $n/2$ rows.

proof of this result would still typically rely on an advanced probabilistic inequality [1]. Our argument is easy and avoids this trouble.

We will, however, rely on the following classic result, which we later prove in Chapter 5 using an encoding argument.

Theorem 5.7 (Additive Chernoff Bound [29])**.** *Let B be a Binomial(n,p) random variable. Then*

$$\Pr[B \leq (p-t)n] \leq 2^{-nD(p-t\|p)}$$

where

$$D(p\|q) = p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q}$$

is the Kullback-Leibler divergence *or* relative entropy *between Bernoulli random variables with parameters p and q.*

Theorem 4.10. *When $p = c/n$, the graph $G \in G_{n,p}$ contains at least one triangle with probability at least $1 - 2^{-\Omega(c^3)}$.*

Proof. Refer to Figure 4.1. Suppose that G contains no triangles. We encode the adjacency matrix A of G. Let M be the $n/2 \times n/2$ submatrix of A determined by the rows $1, \dots, n/2$ and the columns $n/2+1, \dots, n$. Note that $\mathrm{E}[n_1(M)] = cn/4$.

Suppose that $n_1(M) < cn/8$. Note that $n_1(M)$ is a Binomial$(n^2/4, c/n)$ random variable whose value deviates largely from its mean. Then, by Chernoff's bound, we obtain that

$$\Pr[n_1(M) < cn/8] = \Pr[n_1(M) < (c/2n)(n^2/4)] \leq 2^{-(n^2/4)D(c/2n\|c/n)}.$$

The function $D(x/2\|x) : [0,1] \to \mathbb{R}$ has derivative

$$\frac{d}{dx}D(x/2\|x) = -\frac{1}{2} + \log\frac{1-x}{1-x/2} + \frac{\log e}{2(1-x)}$$

which is strictly increasing, so $D(x/2\|x) \geq \left(\frac{\log e - 1}{2}\right)x$. Thus, $D(c/2n\|c/n) = \Omega(c/n)$, and

$$\Pr[n_1(M) < cn/8] \leq 2^{-\Omega(cn)}.$$

Suppose instead that $n_1(M) \geq cn/8$. Note that if $A_{i,j} = 1$ and $A_{i,k} = 1$ for $i < j < k$, then $A_{j,k} = 0$ since G has no triangles. Let m_i denote the number of ones in the i^{th} row of M. Since the function $x(x-1)/2$ is convex, then by Theorem 2.14, we have that

$$\frac{1}{n/2}\sum_{i=1}^{n/2}\binom{m_i}{2} \geq \binom{2n_1(M)/n}{2}.$$

From these observations, we see that by specifying the rows $1, \ldots, n/2$ of A, we can deduce

$$m = \sum_{i=1}^{n/2}\binom{m_i}{2} \geq \frac{n}{2}\binom{2n_1(M)/n}{2} \geq \frac{n}{2}\binom{c/4}{2} = \Omega(c^2 n)$$

zero bits in the rows $n/2+1, \ldots, n$.

We encode G by giving a Shannon-Fano code with parameter p for the first $n/2$ rows of A; and a Shannon-Fano code with parameter p for the rest of A, excluding the bits which can be deduced from the preceding information. Such a code has length

$$\begin{aligned}|C(G)| &\leq n_1(A)\log\frac{1}{p} + (n_0(A) - m)\log\frac{1}{1-p} + O(1)\\ &= \log\frac{1}{p_G} - m\log\frac{1}{1-p} + O(1)\\ &\leq \log\frac{1}{p_G} - mp + O(1)\\ &\leq \log\frac{1}{p_G} - \Omega(c^3).\end{aligned}$$

Applying Lemma 4.2, we see that G has no triangles with probability at most $2^{-\Omega(c^3)}$. □

Theorem 4.11. *When $p = \frac{1}{\alpha n}$, $G \in G_{n,p}$ has no triangle with probability $1 - O(\alpha^{-3})$.*

Proof. Suppose that G contains a triangle. We encode the adjacency matrix A of G. First, we specify the triangle's vertices; and finish with a Shannon-Fano code with parameter p for the remaining vertices of the graph. This code has length

$$\begin{aligned}|C(G)| &\leq 3\log n + (n_1(A) - 3)\log\frac{1}{p} + n_0(A)\log\frac{1}{1-p} + O(1)\\ &= \log\frac{1}{p_G} + 3\log n - 3\log\frac{1}{p} + O(1)\\ &= \log\frac{1}{p_G} - 3\log\alpha + O(1)\\ &= \log\frac{1}{p_G} - \log O(\alpha^3).\end{aligned}$$

We finish by applying Lemma 4.2. □

Together, Theorem 4.10 and Theorem 4.11 establish that $1/n$ is a threshold function for triangle-freeness.

4.3 Percolation on the Torus

Percolation theory studies the emergence of large components in random graphs. We give an encoding argument proving that percolation occurs on the torus when edge survival rate is greater than 2/3. Our line of reasoning follows what is known as a *Peierls argument*. For more precise results, including a general study of percolation theory, see the book by Grimmett [17].

Definition 4.12. When $\sqrt{n}$ is an integer, the $\sqrt{n} \times \sqrt{n}$ *torus grid graph* is defined to be the graph with vertex set $\{1, \ldots, \sqrt{n}\}^2$, where (i, j) is adjacent to (k, ℓ) if

- $|i - k| \equiv 1 \pmod{\sqrt{n}}$ and $|j - \ell| = 0$, or
- $|i - k| = 0$ and $|j - \ell| \equiv 1 \pmod{\sqrt{n}}$.

Theorem 4.13. *Let G be the subgraph of the $\sqrt{n} \times \sqrt{n}$ torus grid graph in which each edge is chosen with probability $p < 1/3$. Then, the probability that G contains a cycle of length at least*

$$\frac{s + \log n + O(1)}{\log(1/(3p))}$$

is at most 2^{-s}.

Proof. Let A be the bitstring of length $2n$ encoding the existence of edges in G. Suppose that G contains a cycle C' of length $t \geq (s + \log n + O(1))/\log(1/(3p))$. Encode A by giving a single vertex u in C'; the sequence of directions that the cycle moves along from u; and a Shannon-Fano code with parameter p for the remaining edges of G.

Note that there are four possibilities for the direction of the first step taken by C' from u, but only three for each subsequent choice. Thus, this sequence can be specified by $\lceil 2 + (t-1)\log 3 \rceil$ bits. The total length of our code is then

$$\begin{aligned} |C(G)| &\leq \log n + t \log 3 + (n_1(A) - t)\log\frac{1}{p} + n_0(A)\log\frac{1}{1-p} + O(1) \\ &= \log\frac{1}{p_G} + \log n - t\log\frac{1}{3p} + O(1) \\ &\leq \log\frac{1}{p_G} - s \end{aligned}$$

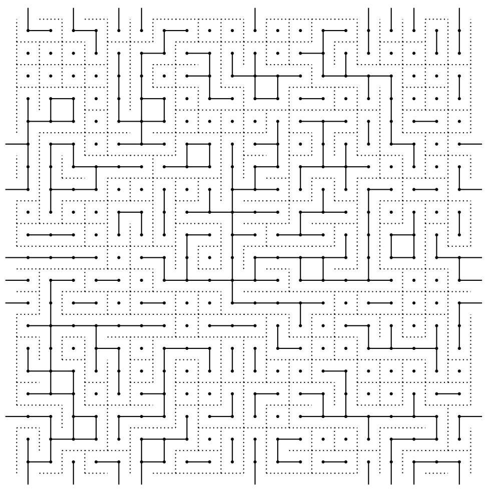
by our choice of t. We finish by applying Lemma 4.2. □

The torus grid graph can be drawn in the obvious way without crossings on the surface of a torus. This graph drawing gives rise to a dual graph, in which each vertex corresponds to a face in the primal drawing, and two vertices are adjacent if and only their primal faces are incident to the same edge. In this sense, the torus grid graph is self-dual.

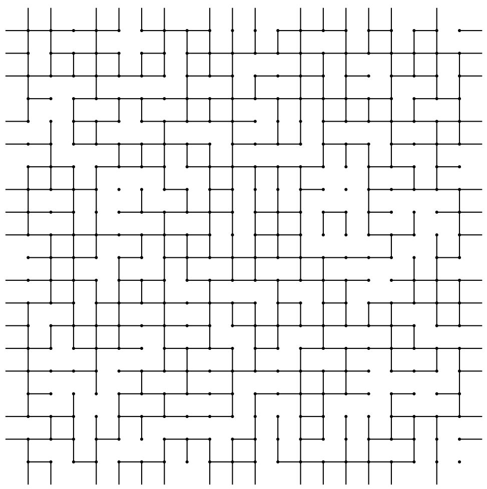
The obvious drawing of the torus grid graph also induces drawings for any of its subgraphs. Such a subgraph also has a dual, where each vertex corresponds to a face in the dual torus grid graph, and two vertices are adjacent if and only if their corresponding faces are incident to the same edge of the original subgraph.

Theorem 4.14. *Let G denote the subgraph of the $\sqrt{n} \times \sqrt{n}$ torus grid graph in which each edge is chosen with probability greater than $2/3$. Then, G has at most one component of size $\omega(\log^2 n)$ with high probability.*

Proof. See Figure 4.2 for a visualization of this phenomenon. Suppose that G has at least two components of size $\omega(\log^2 n)$. Then, there is a cycle of faces separating these components whose length is $\omega(\log n)$. From the discussion above, such a cycle corresponds to a cycle of $\omega(\log n)$ missing edges in the dual graph, as in Figure 4.2a. From the previous result, we know that this does not happen with high probability. □



(a) When $p = 0.33 < 1/3$, long cycles are rare. Dotted lines show missing edges in the dual.



(b) When $p = 0.67 > 2/3$, there is likely only one large component.

Figure 4.2: Random subgraphs of the 20×20 torus grid graph.

Chapter 5

Encoding with Kraft's Condition

We have seen how Lemma 3.1 and Lemma 4.2 can be used to prove well-known bounds in intuitive and unexpected manners, within some constant factor or some lower order term. However, when exactness is required, it may be that neither suffices, even if Lemma 2.10 is used, due to the discrete integral constraint on codeword lengths, as shown in Proposition 1.1. Instead, we will use a continuous analogue of Lemma 4.2 due to Mulzer.

Observe that neither Lemma 3.1, Lemma 4.2, nor any application of either of these results in Chapter 3 and Chapter 4 requires the specification of any prefix-free code; we know, by construction, that every such code is prefix-free, but we could also deduce from Lemma 2.9 that, since our described codes satisfy Kraft's condition, a prefix-free code with the same codeword lengths exists. Instead, codeword lengths satisfying Kraft's condition suffice. Indeed, we will show how such codeword lengths actually need not be integers; this refinement will allow us to tighten Lemma 4.2, thereby tightening all of the results established throughout the previous chapters.

Let $[0, \infty]$ denote the set of extended non-negative real numbers, *i.e.* the usual open interval $[0, \infty)$, including the formal symbol ∞, satisfying all real arithmetic operations and all usual extended operations. In particular, we have that $x + \infty = \infty$ for all $x \in [0, \infty]$ and $2^{-\infty} = 0$. We will also allow that $1/0 = \infty$.

Lemma 5.1 (Mulzer [29]). *Let $\ell : \Omega \to [0, \infty]$ satisfy Kraft's condition, and let $x \in \Omega$ be chosen with probability p_x. Then, for any $s \geq 0$,*

$$\Pr[\, \ell(x) \leq \log(1/p_x) - s \,] \leq 2^{-s}.$$

Proof. The proof is identical to that of Lemma 4.2. □

The sum of functions $\ell : \Omega \to [0, \infty]$ and $\ell' : \Omega' \to [0, \infty]$ is the function $\ell + \ell' : \Omega \times \Omega' \to [0, \infty]$, where $(\ell + \ell')(x, y) = \ell(x) + \ell'(y)$. Note that for any partial codes $C : \Omega \to \{0,1\}^* \cup \{\perp\}, C' : \Omega' \to \{0,1\}^* \cup \{\perp\}$, any $x \in \Omega$, and any $y \in \Omega'$,

$$(|C| + |C'|)(x, y) = |C(x)| + |C'(y)| = |(C \cdot C')|(x, y).$$

In other words, the sum of these functions describes the length of codewords in composed codes.

Lemma 5.2. *If $\ell : \Omega \to [0, \infty]$ and $\ell' : \Omega' \to [0, \infty]$ satisfy Kraft's condition, then so does $\ell + \ell'$.*

Proof. Kraft's condition still holds:

$$\sum_{(x,y)\in\Omega\times\Omega'} 2^{-(\ell+\ell')(x,y)} = \sum_{x\in\Omega}\sum_{y\in\Omega'} 2^{-\ell(x)-\ell'(y)} = \sum_{x\in\Omega} 2^{-\ell(x)} \sum_{y\in\Omega'} 2^{-\ell'(y)} \leq 1. \qquad \square$$

This is analogous to the fact that the composition of prefix-free codes is prefix-free. This result renders Lemma 2.10 obsolete, and now nothing is wasted in the composition of codes.

Lemma 5.3. *For any probability density $p : \Omega \to [0, 1]$, the function $\ell : \Omega \to [0, \infty]$ with $\ell(x) = \log(1/p_x)$ satisfies Kraft's condition.*

Proof.

$$\sum_{x\in\Omega} 2^{-\ell(x)} = \sum_{x\in\Omega} 2^{-\log(1/p_x)} = \sum_{x\in\Omega} p_x = 1. \qquad \square$$

This now allows us to refine every instance of a fixed-length code and every instance of a Shannon-Fano code used while encoding so that, again, nothing is wasted. Note that these lengths now match Shannon's lower bound from Theorem 2.16.

We now give a tight notion corresponding to Elias ω-codes.

Theorem 5.4 (Beigel [4])**.** *Fix some $0 < \varepsilon < e - 1$. Let $\ell : \mathbb{N} \to [0, \infty]$ have*

$$\ell(i) = \log i + \log\log i + \cdots + \underbrace{\log\cdots\log}_{\log^* i \text{ times}} i - (\log\log(e - \varepsilon)) \log^* i + O(1).$$

Then, ℓ satisfies Kraft's condition.

Theorem 5.5 (Beigel [4])**.** *The function $\ell : \mathbb{N} \to [0, \infty]$ with*

$$\ell(i) = \log i + \log\log i + \cdots + \underbrace{\log\cdots\log}_{\log^* i \text{ times}} i - (\log\log e) \log^* i + c$$

does not satisfy Kraft's condition for any choice of the constant c.

Recall how the result of Proposition 1.1 carried an artifact of binary encoding. Using our new tools, we can now refine this and recover the exact result.

Proposition 5.6. *A uniformly random bit string $x_1 \cdots x_n$ contains a run of at least $\log n + s$ ones with probability at most 2^{-s}.*

Proof. Let $\ell : \{0,1\}^n \to [0, \infty]$ be such that if x contains a run of $t \geq \log n + s$ ones, then $\ell(x) = \log n + n - t$, and otherwise $\ell(x) = \infty$. We will show that ℓ satisfies Kraft's condition.

Recall that a binary string $x = x_1 \cdots x_n$ is said to contain a run of t ones if there exists some $i \in \{1, \ldots, n - t + 1\}$ such that $x_i = \cdots = x_{i+t-1} = 1$.

Let the function $f : \{1, \ldots, n - t + 1\} \to [0, \infty]$ have $f(i) = \log n$ for all i, and $g : \{0,1\}^{n-t} \to [0, \infty]$ have $g(x) = n - t$ for all x. Both of these functions satisfy Kraft's condition by Lemma 5.3. By Lemma 5.2, so does the function

$$h = f + g : \{1, \ldots, n - t + 1\} \times \{0,1\}^{n-t} \to [0, \infty],$$

where $h(i, x) = \log n + n - t$ for all i and x. Crucially, each element of the set $\{1, \ldots, n - t + 1\} \times \{0,1\}^{n-t}$ corresponds to an n-bit binary string containing a run of t ones: the element $(i, x) \in \{1, \ldots, n - t + 1\} \times \{0,1\}^{n-t}$, where $x = x_1 \cdots x_{n-t}$, corresponds to the binary string

$$x_1 \cdots x_{i-1} \underbrace{11 \cdots 1}_{t \text{ times}} x_i \cdots x_{n-t}.$$

Therefore, ℓ satisfies Kraft's condition. By our choice of t, we have that $\ell(x) \leq n - s$ if and only if x contains a run of t ones. We finish by applying Lemma 5.1. □

It is not hard to see how Lemma 5.1, Lemma 5.2, Lemma 5.3, and Theorem 5.4 can be used to reframe and refine every encoding argument presented in Chapter 3 and Chapter 4, just as we did in the proof above.

5.1 Chernoff Bound

The following proof is due to Mulzer, and serves as a refinement to an encoding argument given by Devroye, Lugosi, and Morin, which only used Lemma 4.2 instead of the tight framework developed in this chapter, and thus was imprecise.

Theorem 5.7 (Additive Chernoff Bound [29])**.** *Let B be a Binomial(n,p) random variable. Then*

$$\Pr[B \le (p-t)n] \le 2^{-nD(p-t\|p)}$$

where

$$D(p\|q) = p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q}$$

is the Kullback-Leibler divergence *or* relative entropy *between Bernoulli random variables with parameters p and q.*

Proof. By definition, $B = \sum_{i=1}^{n} x_i$ for independent Bernoulli(p) random variables $x_1, \ldots, x_n$, so $B = n_1(x)$ for $x = x_1 \cdots x_n$. We are interested in encoding the string $x \in \{0,1\}^n$.

Informally, we encode x using a Shannon-Fano code with parameter $p-t$. The result of Lemma 5.3 allows us to ignore the ceiling in the lengths of these codewords. In other words, we are interested in the function $\ell : \{0,1\}^n \to [0,\infty]$ with

$$\ell(x) = n_1(x)\log\frac{1}{p-t} + (n - n_1(x))\log\frac{1}{1-p+t}.$$

Now, x occurs with probability exactly $p_x = p^{n_1(x)}(1-p)^{n-n_1(x)}$, so

$$\begin{aligned}\ell(x) &= \log(1/p_x) + n_1(x)\log\frac{p}{p-t} + (n-n_1(x))\log\frac{1-p}{1-p+t}\\ &= \log(1/p_x) + n_1(x)\log\left(1+\frac{t}{p-t}\right) + (n_1(x)-n)\log\left(1+\frac{t}{1-p}\right),\end{aligned}$$

and $\ell(x)$ increases as a function of $n_1(x)$. Therefore, if $n_1(x) \le (p-t)n$, then

$$\begin{aligned}\ell(x) &\le \log(1/p_x) + n(p-t)\log\frac{p}{p-t} + n(1-p+t)\log\frac{1-p}{1-p+t}\\ &= \log(1/p_x) - nD(p-t\|p).\end{aligned}$$

The Chernoff bound is obtained by applying Lemma 5.1. □

Chapter 6

Conclusion

6.1 Contributions

We have described a simplified method for producing encoding arguments. Typically, one would invoke the incompressibility method after developing some of the theory of Kolmogorov complexity. Our technique requires only a basic understanding of prefix-free codes and one simple lemma. We are also the first to suggest a simple and tight manner of encoding using only Kraft's condition with real-valued codeword lengths. In this light, we posit that there is no reason to develop an encoding argument through the incompressibility method: our uniform encoding lemma from Chapter 3 is simpler, the non-uniform encoding lemma from Chapter 4 is more general, and our technique from Chapter 5 is less wasteful. Indeed, though it would be easy to state and prove our non-uniform encoding lemma in the setting of Kolmogorov complexity, it seems as if the general encoding lemma from Chapter 5 only can exist in our simplified framework.

Using our encoding lemmas, we gave original proofs for several previously established results. Specifically, we showed the following:

- the number of records in a uniformly random permutation is $O(\log n)$ with high probability;
- the height of the binary search tree built from the sequential insertion of a uniformly random permutation of n integers is $O(\log n)$ with high probability;
- searching in the balls-in-urns model for hashing takes time $O(\log n / \log\log n)$ with high probability;
- linear probing succeeds in $O(1)$ expected time;
- any operation in 2-choice hashing takes time $O(\log\log n)$ with high probability;

- a connectivity threshold in the Erdős-Rényi random graph;
- a threshold for small components in the Erdős-Rényi random graph;
- a strong threshold for triangle-freeness in the Erdős-Rényi random graph;
- a certain random bipartite graph is an expander with high probability;
- percolation occurs in random dense subgraphs of the torus.

6.2 Future Work and Open Problems

Due to the nature of our research, there are countless avenues for future work. Any result bounding the probability of an event can be trivially rephrased as an encoding argument; the real question is whether or not an intuitive encoding argument exists.

Some of our encoding arguments sacrifice tight constants. Our first group of open problems focuses on these:

- We showed that a uniformly random permutation has at most $c \log n$ records with high probability only for $c > 2$. We know that the number of records in such a permutation is instead concentrated around $\ln n + O(1)$ [9]. Using an extended argument, we then showed that a random binary search tree has height at most $c \log n$ with probability $1 - O(1/n)$ for $c = 8.12669...$; we know, in fact, that the height of such a tree is instead concentrated around $\alpha \log n$ for $\alpha = 2.98820...$ [33]. We are interested in whether or not these gaps can be closed through encoding. Perhaps if the gap for records can be closed, then so can the gap for binary search tree height;
- It is known that linear probing hash tables of size cn achieve constant expected search time for $c > 1$ [27]. Unfortunately, our encoding proof requires that $c > e$; it is yet unclear if the extraneous e factor is retained only as an artifact of encoding, or if some intuitive refinement exists;
- Our analysis of 2-choice hashing also only works for hash tables of size cn for $c > e$, while $c > 0$ suffices to show that all operations achieve $O(\log \log n)$ running time. We also rely on this assumption several times in our proof.

Other open problems arising from this thesis have to do with finding encoding arguments for new problems:

- Robin Hood hashing is another hashing solution which achieves $O(\log \log n)$ worst case running time for all operations [11]. The original analysis is difficult, but might lend itself to a similar kind of analysis as we used to study 2-choice hashing. Indeed, when a Robin Hood hashing operation takes a significant amount of time, a large witness tree is again implied, which suggests an easy encoding argument. Unfortunately, this approach appears to involve unwieldy hypergraph encoding.

- The analysis of random binary search trees as performed in Section 3.2.2 is closely related to the analysis of quicksort, which is in turn closely related to the analysis of Hoare's quickselect algorithm [18]. Therefore, we expect to be able to use encoding naturally to study the performance of quickselect.

- We showed in Section 5.1 how Chernoff's bound can be exactly obtained through a natural encoding argument. Perhaps encoding can be used in a similar way to give information-theoretic proofs of other results in probability.

Finally, our method of encoding with Kraft's condition as in Chapter 5 may refine existing encoding arguments to the extent that they become tight. This should be investigated.

Bibliography

[1] N. Alon and J. H. Spencer. *The Probabilistic Method.* 2nd ed. Wiley Series in Discrete Mathematics and Optimization. Wiley-Interscience, 2004.

[2] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. “Balanced Allocations”. In: *SIAM Journal on Computing* 29.1 (Feb. 2000), pp. 180–200.

[3] A. R. Barron. “Logically Smooth Density Estimation”. PhD thesis. Stanford University, Sept. 1985.

[4] R. Beigel. “Unbounded Searching Algorithms”. In: *SIAM Journal on Computing* 19.3 (1990), pp. 522–537.

[5] H. Buhrman, T. Jiang, M. Li, and P. Vitányi. “New Applications of the Incompressibility Method (Extended Abstract)”. In: *Automata, Languages and Programming.* Ed. by J. Wiedermann, P. van Emde Boas, and M. Nielsen. Vol. 1644. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 1999, pp. 220–229.

[6] H. Buhrman, T. Jiang, M. Li, and P. Vitányi. “New Applications of the Incompressibility Method: Part II”. In: *Theoretical Computer Science* 235.1 (2000), pp. 59–70.

[7] G. J. Chaitin. “On the Length of Programs for Computing Finite Binary Sequences”. In: *Journal of the ACM* 13.4 (Oct. 1966), pp. 547–569.

[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. Wiley-Interscience, 1991.

[9] L. Devroye. “Applications of the Theory of Records in the Study of Random Trees”. In: *Acta Informatica* 26.1 (1988), pp. 123–130.

[10] L. Devroye, G. Lugosi, and P. Morin. Ninth Annual Workshop on Probability, Combinatorics, and Geometry. unpublished. McGill University’s Bellairs Research Institute, Apr. 2014.

[11] L. Devroye, P. Morin, and A. Viola. "On Worst-Case Robin Hood Hashing". In: *SIAM Journal on Computing* 33.4 (2004), pp. 923–936.

[12] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. "Dynamic Perfect Hashing: Upper and Lower Bounds". In: *SIAM Journal on Computing* 23.4 (1994), pp. 738–761.

[13] P. Elias. "Universal Codeword Sets and Representations of the Integers". In: *IEEE Transactions on Information Theory* 21.2 (Mar. 1975), pp. 194–203.

[14] P. Erdős and A. Rényi. "On Random Graphs, I". In: *Publicationes Mathematicae (Debrecen)* 6 (1959), pp. 290–297.

[15] R. M. Fano. *The Transmission of Information*. Tech. rep. 65. Cambridge, Massachusetts: Research Laboratory of Electronics at MIT, 1949.

[16] R. G. Gallager. *Information Theory and Reliable Communication*. Wiley-Interscience, 1968.

[17] G. R. Grimmett. *Percolation*. 2nd ed. 321. Springer-Verlag Berlin Heidelberg, 1999.

[18] C. A. R. Hoare. "Algorithm 65: Find". In: *Communications of the ACM* 4.7 (July 1961), pp. 321–322. ISSN: 0001-0782. DOI: 10.1145/366622.366647. URL: http://doi.acm.org/10.1145/366622.366647.

[19] S. Hoory, N. Linial, and A. Wigderson. "Expander Graphs and Their Applications". In: *Bulletin of the American Mathematical Society* 43.4 (Oct. 2006), pp. 439–561.

[20] T. Jiang, M. Li, and P. Vitányi. "The Incompressibility Method". In: *SOFSEM 2000: Theory and Practice of Informatics*. Ed. by V. Hlaváč, K. G. Jeffery, and J. Wiedermann. Vol. 1963. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2000, pp. 36–53.

[21] A. Kolmogorov. "Three Approaches to the Quantitative Definition of Information". In: *Problems of Information Transmission* 1.1 (1965), pp. 1–11.

[22] L. G. Kraft. "A Device for Quantizing, Grouping, and Coding Amplitude-Modulated Pulses". MA thesis. Cambridge, Massachusetts: Massachusetts Institute of Technology, 1949.

[23] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 3rd ed. Springer-Verlag New York, 2008.

[24] B. Lucier, T. Jiang, and M. Li. "Average-Case Analysis of Quicksort and Binary Insertion Tree Height using Incompressibility". In: *Information Processing Letters* 103.2 (July 2007), pp. 45–51.

[25] G. A. Margulis. "Explicit Constructions of Concentrators". In: *Problems of Information Transmission* 9.4 (1973), pp. 325–332.

[26] B. McMillan. "Two Inequalities Implied by Unique Decipherability". In: *IRE Transactions on Information Theory* 2.4 (Dec. 1956), pp. 115–116.

[27] P. Morin. *Open Data Structures: An Introduction*. Edmonton: Athabasca University Press, 2013.

[28] R. A. Moser and G. Tardos. "A Constructive Proof of the General Lovász Local Lemma". In: *Journal of the ACM* 57.2 (Jan. 2010), 11:1–11:15.

[29] W. Mulzer. *Chernoff Bounds*. `http://page.mi.fu-berlin.de/mulzer/notes/misc/chernoff.pdf`. (last accessed June 12, 2015).

[30] R. Pagh and F. F. Rodler. "Cuckoo Hashing". In: *Journal of Algorithms* 51.2 (2004), pp. 122–144.

[31] M. Pătraşcu. *Cuckoo Hashing*. `http://infoweekly.blogspot.ca/2010/02/cuckoo-hashing.html`. (last accessed April 15, 2015).

[32] M. S. Pinsker. "On the Complexity of a Concentrator". In: *The 7th International Teletraffic Conference*. Vol. 4. 1973, pp. 1–4.

[33] B. Reed. "The Height of a Random Binary Search Tree". In: *Journal of the ACM* 50.3 (May 2003), pp. 306–332.

[34] H. Robbins. "A Remark on Stirling's Formula". In: *The American Mathematical Monthly* 62.1 (Jan. 1955), pp. 26–29.

[35] V. K. Rohatgi and A. K. M. E. Saleh. *An Introduction to Probability and Statistics*. 2nd ed. Wiley Series in Probability and Statistics. Wiley-Interscience, 2001.

[36] C. E. Shannon. "A Mathematical Theory of Communication". In: *Bell Systems Technical Journal* 27 (1948), pp. 379–423.

[37] R. J. Solomonoff. “A Formal Theory of Inductive Inference. Part I”. In: *Information and Control* 7.1 (1964), pp. 1–22.

[38] A. Souza. *Lecture Notes in Randomized Algorithms & Probabilistic Methods*. http://www2.informatik.hu-berlin.de/alcox/lehre/lvss11/rapm/probabilistic_method.pdf. (last accessed July 29, 2015). Apr. 2011.

[39] P. Vitányi. “Analysis of Sorting Algorithms by Kolmogorov Complexity (A Survey)”. In: *Entropy, Search, Complexity*. Vol. 16. Bolyai Society Mathematical Studies. Springer-Verlag New York, 2007, pp. 209–232.

[40] B. Vöcking. “How Asymmetry Helps Load Balancing”. In: *Journal of the ACM* 50.4 (July 2003), pp. 568–589.

[41] D. B. West. *Introduction to Graph Theory*. 2nd ed. Prentice Hall, 2001.