SOURCE CODING FOR CHANNELS WITH FINITE-STATE LETTER COSTS

by

Serap Ayşe Savari

S.B., Massachusetts Institute of Technology
(1990)

Submitted to the
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE

and

MASTER OF SCIENCE IN OPERATIONS RESEARCH

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1991

Signature of Author..
Department of Electrical Engineering and Computer Science
July 22, 1992

Certified by..
Robert G. Gallager
Professor of Electrical Engineering
Thesis Supervisor

Accepted by..
Campbell L. Searle
Chairman, Departmental Committee on Graduate Students

Accepted by..
Thomas L. Magnanti
Co-Director, Operations Research Center

SOURCE CODING FOR CHANNELS WITH FINITE-STATE LETTER COSTS

by

Serap Ayşe Savari

Submitted to the Department of Electrical Engineering and Computer Science
on July 22, 1992, in partial fulfillment of the
requirements for the degrees of
Master of Science
and
Master of Science in Operations Research

Abstract

The goal of source coding is to encode the output of a discrete information source into a sequence of letters from a given channel alphabet with the minimum expected cost required per transmitted source symbol. Although much is known about good coding techniques when all of the channel letters are equally costly, there are few results for more general channels. In this thesis, we propose and examine various coding schemes, primarily for the case of noiseless, memoryless cost channels. We first study Varn coding and demonstrate a duality between Varn coding and Tunstall coding. We also establish new bounds on the performance of Varn coding. We then turn to a heuristic to find good prefix condition codes when the channel is binary with memoryless letter costs. The heuristic is based on Huffman's algorithm; we consider its strengths and weaknesses. Next, we describe a new mathematical formulation to find optimal single letter prefix condition codes when the channel is either memoryless cost, or has memory in the form of constraints on individual code words. The resulting formulation is an integer bilinear programming problem. We mention some additional constraints to get better solutions when we drop the integer constraints. Finally, we generalize arithmetic coding to handle discrete, noiseless finite-state channels. We demonstrate that these codes are efficient, have asymptotically optimal behavior, and can be used to encode sources with memory.

Thesis Supervisor: Robert G. Gallager
Title: Professor of Electrical Engineering

Acknowledgements

I would like to thank my parents, Mr. Aykut Savari and Mrs. Şirin Savari, for their continual love and encouragement. I could have never come this far without them. I am also grateful to my sister Phylis for her support and helpfulness. She assisted in the typing of this document.

I would like to thank my supervisor, Professor Robert G. Gallager, for his on-going guidance and insights. His teachings inspired my interest in source coding and working with him has been a very important and rewarding part of my educational experience.

I am grateful to Emre Telatar and David Tse for their encouragement and for their help with my computer-related difficulties. I also wish to express my appreciation to Murat Azizoğlu, Rick Barry, Daniel Lee, Ying Li, Abhay Parekh, Rajesh Pankaj and Jane Simmons for providing a very friendly environment at LIDS.

This work was supported by an AT&T Bell Laboratories Graduate Research Program for Women fellowship and a Vinten Hayes fellowship.

Contents

Chapter 1

Introduction

Source coding is a topic that is well understood for channels with equal cost symbols, but is rather poorly understood for more general channels. In this thesis, we attempt to gather additional insights about good encoding techniques when we work with one of these more general channels. More specifically, we will be investigating the encoding of the output of a discrete information source into a sequence of letters from a given channel alphabet. We assume that there is a cost associated with transmitting a given channel letter which may depend on the previous channel sequence transmitted. We wish to choose the encoding rules to minimize the expected cost required per transmitted source symbol.

In one family of encoding rules, which we will call *single letter codes*, each source symbol is represented by a code word, which is a sequence of letters from the channel alphabet. In order for decoding to be unambiguous, we insist that no two distinct source sequences of finite length correspond to the same sequence of channel letters. Codes with this property are said to be *uniquely decodable*. A subset of the class of uniquely decodable codes is the set of *prefix condition codes*, in which no code word is the prefix of any other code word. Prefix condition codes have the property that the end of each code word is immediately recognizable once the starting point of a sequence of code words is known. In this thesis, all of the single letter codes that we will study fall into the category of prefix condition codes.

The set of single letter codes is a subset of the family of *fixed to variable length codes* in which single letter encoding techniques are used on the product ensemble of L source letters. There are two problems with using fixed to variable length codes. In the next section, we will discuss the minimum expected cost of transmitting a source letter that can be achieved by any source coding technique and we will demonstrate that it is not computationally attractive to use a fixed to variable length code to get arbitrarily close to this ideal average cost. The other problem is that often we are not given the source probabilities. Hence, we would like to have source coding techniques that are "universal" in the sense that they work well independent of the source probabilities, or "adaptive" in the sense that they adapt to the existing source probabilities. We will classify any encoding rules which do not fall into the category of fixed to variable length codes as *non-block codes.*

1.1 Definitions and Background

1.1.1 Discrete Memoryless Cost Channels

We assume that we are given a discrete memoryless source. This means that at each unit of time, the source emits one of a finite ensemble U of source symbols, say $u_1, u_2, \dots, u_K$, with associated probabilities $p_1, p_2, \dots, p_K$, and successive letters are statistically independent. In this subsection, we also assume that our channel has memoryless costs; i.e., the cost of transmitting any code letter depends only on that letter. Therefore, our channel is a device which accepts input from a specified set X of letters, say $x_1, x_2, \dots, x_N$ with (positive) letter costs $c_1, c_2, \dots, c_N$, respectively.

Shannon (1948) demonstrated that under this assumption, the minimum expected cost per source symbol that can be achieved by any source coding technique is greater than or equal to $\frac{H(U)}{C}$ where $H(U)$, the *entropy of the source ensemble*, is defined by

$$H(U) = -\sum_{i=1}^{K} p_i \log_2 p_i$$

and C, the *capacity of the channel,* is the real root of the equation

$$\sum_{i=1}^{N} 2^{-Cc_i} = 1. \tag{1.1}$$

Definition: We say that a code γ^* is *optimal* within a class of codes Γ if there is no code

$\gamma \in \Gamma$ for which γ has a smaller average cost of transmitting a source symbol than γ^*.

We begin our discussion of known coding techniques with some results on single letter codes. Here we assume that $N \leq K$. McMillan (1956) showed that when all of the letters in the code alphabet have the same transmission cost, any uniquely decodable code can be replaced by a prefix condition code without changing any of the code word lengths. Karp (1961) studied memoryless cost channels where the channel letters all have positive integer transmission costs and conjectured that every uniquely decodable code can be replaced by a prefix code with the same set of code word costs. Carter and Gill (1974) demonstrated that Karp's hypothesis is equivalent to the same statement under the more general premise of an arbitrary assignment of positive channel letter costs. Therefore, in our study of single letter codes for memoryless cost channels, it is likely that we do not lose anything by restricting our attention to prefix condition codes.

Huffman (1952) used an elegant combinatorial-type argument to find the optimal prefix condition code in the special case where the channel letters are of equal cost. Varn (1971) gave an efficient algorithm to produce the optimal prefix condition code when the source symbols are equally probable. Karp suggested an integer linear programming solution to obtain the optimal prefix condition code for arbitrary rational channel costs and source probabilities.

Because most integer linear programming problems can not be solved efficiently, heuristics have been developed to get "good," but not necessarily optimal, prefix condition codes. Shannon, in his aforementioned paper, describes a heuristic that he and Fano independently discovered to get good prefix condition codes when all of the channel letters have the same transmission cost. The Shannon-Fano algorithm can be described as follows: Suppose we label our source ensemble so that $p_1 \geq p_2 \geq \cdots \geq p_K$. Let $P_1 = 0$ and let $P_{k+1} = P_k + p_k$ for $k = 1, \ldots, K-1$. The code word for message k is obtained by expressing P_k in base N and then selecting the first m_k places of the base N representation, where m_k is the integer satisfying

$$-\frac{\log_2 p_k}{\log_2 N} \leq m_k < 1 - \frac{\log_2 p_k}{\log_2 N}.$$

Krause (1962) generalizes the Shannon-Fano algorithm to handle arbitrary memoryless cost channels. The primary difference between the Krause and Shannon-Fano algorithms lies in Krause's use of a "generalized base N" representation of P_k, $k = 1, \ldots, K$. The unit interval and all subsequent intervals are now split into N segments with lengths in the ratio $2^{-Cc_1} : 2^{-Cc_2} : \cdots : 2^{-Cc_N}$. The code word for p_k is chosen to be the shortest initial segment of the generalized base N representation of P_k whose total cost d_k is greater than or equal to $-\frac{\log_2 p_k}{C}$. Hence, for $k = 1, \ldots, K$,

$$-\frac{\log_2 p_k}{C} \leq d_k < \max_{i \in \{1,\ldots,N\}} c_i - \frac{\log_2 p_k}{C}.$$

Mehlhorn (1980) essentially uses many of the same ideas as Krause to develop a different, but similar, heuristic. Krause's paper shows that an optimal prefix condition code satisfies

$$\frac{H(U)}{C} \leq \overline{d} < \frac{H(U)}{C} + \max_{i \in \{1,\ldots,N\}} c_i$$

where $H(U)$ and C are defined as before and $\overline{d}$ is the average cost per source symbol. Mehlhorn demonstrates a slightly weaker result. The heuristics of both Krause and Mehlhorn require the execution of a number of steps on the order of the size of the source ensemble to be encoded.

To use these algorithms to obtain codes with expected cost per source symbol guaranteed to be arbitrarily close to $\frac{H(U)}{C}$, we can encode large blocks of source symbols instead of individual source symbols. More specifically, if we use Krause's encoding technique on the product ensemble of sequences of L source letters, then an argument almost identical to the one in Section 3.3 of Gallager (1968) shows that

$$\frac{H(U)}{C} \leq \overline{d} < \frac{H(U)}{C} + \frac{1}{L} \max_{i \in \{1,\ldots,N\}} c_i$$

where $\overline{d}$ and U refer to the original source ensemble. Note that the size of the product ensemble of sequences of L source symbols is K^L and as L increases, it rapidly becomes computationally unattractive or infeasible to use Krause's algorithm. This motivates a study of non-block codes.

However, before we turn to non-block codes, we will discuss two more results on prefix condition codes that are interesting and useful. For $i = 1, \ldots, N$, we define y_i to be the code word for source symbol u_i, l_i to be the number of code letters in y_i, and d_i to be the cost of transmitting y_i. Then for any prefix condition code, Kraft (1949) showed that $\sum_{i=1}^{K} N^{-l_i} \leq 1$ and Krause demonstrated that $\sum_{i=1}^{K} 2^{-Cd_i} \leq 1$. Both of these inequalities are met with equality when the prefix condition code is *exhaustive*; i.e., any string of channel letters is the concatenation of a string of code words possibly followed by a proper prefix of a code word. Alternatively, a prefix condition code is exhaustive if, for any two channel letters x_j and x_k, αx_j is a prefix of a code word if and only if αx_k is also a prefix of a code word, where α is a string of channel letters.

To begin our discussion of non-block codes, we consider a broader viewpoint of source coding than the one we have been using until this point. We now decompose our encoder into two parts, a *parser* and a *string encoder*. The parser segments the source output sequence into a concatenation of strings from a dictionary. In the fixed to variable length codes we considered earlier, the strings were of fixed length. For non-block codes, we are interested in the case where the parser output is a set of variable length strings. The function of the string encoder is to map the set of such strings into (uniquely decodable) code words.

In order to ensure that any source sequence can be encoded, we insist that every possible source sequence has a prefix that is in the dictionary. Dictionaries with this property are said to be *valid*. To avoid any ambiguity in the parsing process, we restrict our attention to dictionaries in which no entry is the prefix of any other entry.

Non-block codes have been investigated primarily for the case where the channel letters are equally costly. Tunstall (1968) found the dictionary that minimizes the average number of code letters transmitted per source letter under the assumption that the code words all have the same length. Ziv and Lempel (1978) studied the parsing of source strings where the source statistics are unknown. Their coding strategy, like Tunstall's, is essentially a variable-to-fixed length code containing a parsing dictionary of source strings, but their dictionary changes dynamically while the dictionary in Tunstall coding remains fixed. As the length of the encoded source string goes to infinity, the Ziv-Lempel codes have the ideal of $\frac{H(U)}{\log_2 N}$ code letters per source symbol. For the more general memoryless cost channel, Lempel, Even, and Cohn (1973) found the optimal dictionary with $N \geq K$ entries for the special case where the source symbols are equiprobable.

For channels with equal cost letters, Rissanen and Langdon (1979) investigated another class of codes called arithmetic codes. In arithmetic coding, there is no parsing done on the source sequence; instead, an infinite source sequence is mapped into a point x in the unit interval on the real line and then x is represented by a sequence of channel letters. As with the Ziv-Lempel codes, the arithmetic codes asymptotically have an ideal number of code letters per source symbol.

1.1.2 Finite-State Noiseless Channels

We now turn our attention to noiseless discrete finite-state channels, a class of channels which includes the family of noiseless discrete memoryless cost channels as a special case. A finite-state channel with finite alphabet $X = \{x_1, \ldots, x_N\}$ and set of states $S = \{s_1, \ldots, s_r\}$ is defined by specifying

1. for each pair $(s, x_j) \in S \times X$, the cost $c_{s,x_j} \in [0, \infty]$ of transmitting x_j when the state is s
2. the state $s[x_j, \psi]$ after channel letter x_j is transmitted, given that the state of the channel is ψ prior to transmission; when $c_{\psi,x_j} = \infty$ for $x_j \in X$, we assume that $s[x_j, \psi] = \psi$.

The second rule inductively specifies the final state after an arbitrary channel string σ is transmitted from initial state ψ, and we denote this state by $s[\sigma, \psi]$. As before, we assume that we are given a discrete memoryless source.

We let X^* denote the set of all strings of letters of X. We say that $\sigma \in X^*$ is an element of X_i^* if the cost of transmitting σ is finite given that the channel is in (initial) state s_i immediately before the first letter of σ is transmitted.

We let $c_s^* = \min_{x_j \in X} c_{s,x_j}$, $s \in S$. We allow the possibility of $c_{s_i}^* = 0$ for some i, but we assume that for every $1 \leq i \leq r$ and every $\sigma = x_{j_1} \ldots x_{j_n} \in X_i^*$ with $n \geq r$, the cost of transmitting σ is strictly positive.

We say that a finite-state channel is *irreducible* if for each pair of states s_i and s_k, there is a string $\sigma \in X_i^*$ for which σ drives the channel to state s_k given that the channel was in state s_i prior to the transmission of the first letter of σ; i.e., $s[\sigma, s_i] = s_k$. All finite-state channels that we will discuss are assumed to be irreducible.

We let $X_k(s_i) = \{x_j \in X \ : s[x_j, s_i] = s_k\}$ and for $\omega \geq 1$ we let $\mathcal{A}(\omega)$ denote the $r \times r$ matrix $\mathcal{A}(\omega) = [a_{ik}(\omega)]$ where $a_{ik}(\omega) = \sum_{x_j \in X_k(s_i)} \omega^{-c_{s_i,x_j}}$. To include $\omega = 1$, we use the convention that $1^{-\infty} = 0$. Shannon (1948) and Csiszár (1969) showed that there is a unique positive (real) number $\omega_0 > 1$ for which the greatest positive eigenvalue of $\mathcal{A}(\omega_0)$ equals one; furthermore, if $C = \log_2 \omega_0$ and U is our source ensemble, then the minimum expected cost per source symbol that can be achieved by any source coding technique is greater than or equal to $\frac{H(U)}{C}$. For this reason, we again refer to C as the capacity of the channel.

Csiszár generalized Krause's algorithm to handle finite-state channels with the previously mentioned properties. The resulting Csiszár codes are prefix condition codes with average cost per source symbol between $\frac{H(U)}{C}$ and $\frac{H(U)}{C} + V$, where V is a positive constant depending only on the channel. Following the same argument as we used earlier for Krause codes, if we use Csiszár's encoding technique on the product ensemble of L source letters, then the expected cost per source symbol is between $\frac{H(U)}{C}$ and $\frac{H(U)}{C} + \frac{V}{L}$. Unfortunately, as L gets larger, this scheme becomes increasingly unattractive from a computational point of view.

Besides the discrete memoryless cost channel, there are other special subclasses of finite-state channels that are of interest. These are typically binary channels whose letters have the same cost of transmission, but for which there are some constraints on the code words and perhaps on the concatenation of code words. A *runlength constraint* is a constraint on the number of times that the same symbol can be sequentially repeated. If $\{0, 1\}$ is our channel alphabet, one typical constraint may require that there are no strings of length greater than λ consisting of all 0's or all 1's. Another constraint may necessitate a minimum of d and a maximum of k 0's between consecutive 1's in allowable code strings. There has been a considerable effort to find good codes satisfying the (d, k) constraint. Siegal (1985) summarized many of the existing techniques to construct so-called (d, k) codes.

1.2 Goals of this Work

The body of the thesis is comprised of four parts. The next chapter is devoted to Varn coding; a duality between Varn coding and Tunstall coding is demonstrated and bounds on the performance of Varn codes is established.

Chapter 3 develops a heuristic to find good prefix condition codes when we are given a binary, memoryless, noiseless channel and a memoryless source. The heuristic is based on Huffman's algorithm; its strengths and weaknesses are discussed.

The fourth chapter describes a new mathematical formulation to find optimal prefix condition codes when the channel is either

- memoryless cost, or

- with memory, but the memory is limited to individual code words, as opposed to the concatention of code words; e.g., constraints on the maximum code word cost are acceptable, but runlength constraints are not.

The resulting formulation is an integer bilinear programming problem. Additional constraints to provide a better solution if the integer constraints are ignored are mentioned.

The final chapter of the body generalizes arithmetic coding to deal with discrete, noiseless finite-state channels. We demonstrate that these codes have asymptotically optimal behavior and that they can be used to encode sources with memory. Examples of the new algorithm will be provided.

Chapter 2

Some Notes on Varn Coding

Varn (1971) investigated the problem of constructing minimum-redundancy single letter prefix condition codes given a discrete, noiseless, memoryless cost channel and a memoryless source with equiprobable symbols. Each source symbol is to be encoded into a string of channel letters so that the resulting code satisfies the prefix condition and has minimum average transmission cost among all prefix condition codes. More specifically, we are given a source ensemble consisting of equiprobable symbols $u_1, \ldots, u_K$ and a channel which has alphabet $x_1, \ldots, x_N$ with respective letter costs $c_1 \geq c_2 \geq \ldots \geq c_N$ and capacity C, which Shannon (1948) defined to be the real root of equation (1.1); we recall that this equation is

$$\sum_{i=1}^{N} 2^{-Cc_i} = 1.$$

For $1 \leq i \leq K$, we denote the cost of transmitting source symbol u_i by d_i and we let $\overline{d} = \frac{\sum_{i=1}^{K} d_i}{K}$ represent the expected cost of transmitting a source symbol.

Varn first studied the problem of finding an optimal code among the class of exhaustive prefix condition codes. We recall that a prefix condition code is exhaustive if, for any string of channel letters x and any two channel letters x_j and x_k, xx_j is a prefix of a code word if and only if xx_k is also a prefix of a code word. We note that exhaustive prefix codes exist only for those K for which $K = (\alpha + 1)N - \alpha$ for some non-negative integer α; α is then the number of non-root intermediate nodes in our code tree. Using induction on α, Varn demonstrated that any exhaustive prefix condition code with K equiprobable source symbols and α non-root intermediate nodes with costs $Y_1 \leq \ldots \leq Y_\alpha$ has average cost per source symbol equal to

$$\frac{(\alpha+1)\sum_{i=1}^{N} c_i + (N-1)\sum_{j=1}^{\alpha} Y_j}{K}. \tag{2.1}$$

Therefore, an optimal exhaustive prefix condition code will minimize $\sum_{j=1}^{\alpha} Y_j$. Hence, the following algorithm could be used to create an optimal exhaustive prefix condition code:

1. Start with each channel letter as a code word.
2. If the total number of words is less than K, then goto step 3, else stop.
3. Take the least costly code word Z_1 and replace it with the N code words $Z_1 \circ x_1$, $Z_1 \circ x_2$, $\ldots$, $Z_1 \circ x_N$, where $a \circ b$ is defined as the concatenation of b to the right end of a. We do not change the other code words. Goto step 2.

Varn then considered the problem of obtaining optimal prefix condition codes that are not constrained to be exhaustive. He demonstrated that if the code tree of the optimal code has α non-root intermediate nodes, then the exhaustive prefix condition code with those α non-root intermediate nodes is optimal among all exhaustive prefix condition codes with α non-root intermediate nodes; furthermore, the code tree of the optimal code consists of the K least costly words of the exhaustive code tree with the same non-root intermediate nodes. Varn also showed that for any optimal prefix condition code, each intermediate node has at least two immediate descendant nodes. Therefore, any optimal prefix condition code satisfies $\lceil \frac{K-N}{N-1} \rceil \leq \alpha \leq K-2$, where $\lceil x \rceil$ is defined as the least integer greater than or equal to x. The upper bound is met when each intermediate node has exactly two immediate descendant nodes and the lower bound is met when each intermediate node has as many immediate descendant nodes as possible. When $N = 2$, the upper and lower bounds on α are identical and the optimal prefix condition code is exhaustive. When $N > 2$, we do not know a priori the value of α for the optimal prefix condition code. For example, if we have a ternary channel with $(c_1, c_2, c_3) = (10^{100}, 1, 1)$, we expect that the first letter of the alphabet is not used in an optimal code and $\alpha = K-2$; on the other hand, if $(c_1, c_2, c_3) = (1, 1, 1)$, we expect that every intermediate node in an optimal code will have as many immediate descendant nodes as possible and hence $\alpha = \lceil \frac{K-3}{2} \rceil$.

Tunstall (1968) investigated a different problem. For his problem, we are given a discrete, memoryless source emitting symbols $u_1', u_2', \ldots, u_n'$ with respective probabilities $p_1 \geq p_2 \geq \ldots \geq p_n$ and a channel with equally costly letters. We are interested in one-to-one mappings of strings of source symbols, or dictionary entries, into the product ensemble of sequences of $\mathcal{L}$ channel letters. We restrict our attention to valid dictionaries; in other words, every possible source sequence has a prefix that is in the dictionary. Tunstall found a valid dictionary of $k = (\alpha + 1)n - \alpha$ entries that maximizes the expected number $E[L]$ of source symbols per dictionary entry; here α is the number of non-root intermediate nodes in the dictionary tree. It is known that if the probabilities of the α non-root intermediate nodes are $y_1, \ldots, y_\alpha$, then $E[L] = 1 + \sum_{i=1}^{\alpha} y_i$. Therefore, an optimal valid dictionary will maximize $\sum_{i=1}^{\alpha} y_i$. Hence, the following algorithm finds the optimal valid dictionary:

1. Start with each source symbol as a dictionary entry.
2. If the total number of entries is less than k, then goto step 3, else stop.
3. Take the most probable entry z_1 and replace it with the n strings $z_1 \circ u_1', \ldots, z_1 \circ u_n'$. Do not alter the other entries. Goto step 2.

Varn's algorithm for finding optimal exhaustive prefix condition codes and Tunstall's method of obtaining optimal valid dictionaries look very similar. In fact, they are identical because we can always transform an instance of either problem to an instance of the other problem and use the corresponding algorithm to get the right tree. For example, if we have channel letter costs $c_1 \geq c_2 \geq \ldots \geq c_N$ for Varn's problem, the corresponding probabilities for Tunstall's problem are $2^{-Cc_1} \leq 2^{-Cc_2} \leq \ldots \leq 2^{-Cc_N}$; likewise, given source probablilities $p_1 \geq \ldots \geq p_n$, in an instance of Tunstall's problem, the associated costs that we input into Varn's algorithm can be chosen to be $-\log_2 p_1 \leq \ldots \leq -\log_2 p_n$. However, we note that the problem of finding an optimal prefix condition code is generally a more difficult problem than the problem of finding an optimal *exhaustive* prefix condition code, and hence it is a more difficult problem than the problem that Tunstall solved.

We now consider some bounds on the performance of Varn codes. Since the entropy of a source ensemble consisting of K equiprobable symbols is $\log_2 K$, Krause's results indicate that the average cost $\overline{d}$ of the optimal prefix condition code satisfies $\frac{\log_2 K}{C} \leq \overline{d} < \frac{\log_2 K}{C} + c_1$.

We derive an upper bound on the performance of the optimal exhaustive prefix condition code and use it to derive a different upper bound of the average cost of the optimal prefix condition code without exhaustivity constraints. Suppose we have a modified channel whose alphabet consists of the M least costly letters of the original channel's alphabet. For our modified channel, let $F(\alpha, M)$ denote the sum of the costs of the $(\alpha+1)M-\alpha$ code words in an exhaustive prefix condition code with α non-root intermediate nodes. Let $g(M)=\sum_{i=1}^{M} c_{N+1-i}$ be the sum of the letter costs for our modified channel. From equation (2.1), we see that

$$F(\alpha, M)=(\alpha+1)g(M)+(M-1)\sum_{j=1}^{\alpha} Y_j. \tag{2.2}$$

Hence, recursively we have the relation

$$F(\alpha, M)=F(\alpha-1, M)+g(M)+(M-1)Y_\alpha \tag{2.3}$$

where $F(0,M)=g(M)$ and, since $Y_1=c_N$, $F(1,M)=2g(M)+(M-1)c_N$. In general, we do not know the value of Y_α. However, we do know that the last intermediate node chosen by the Varn algorithm was the least costly leaf prior to being converted to an intermediate node. Hence,

$$Y_\alpha \leq \frac{F(\alpha-1, M)}{\alpha M-(\alpha-1)}. \tag{2.4}$$

Substituting (2.4) into equation (2.3) we find that

$$F(\alpha, M) \leq \left(\frac{(\alpha+1)M-\alpha}{\alpha M-(\alpha-1)}\right) F(\alpha-1, M)+g(M) \tag{2.5}$$

Dividing both sides of (2.5) by $(\alpha+1)M-\alpha$ gives

$$\frac{F(\alpha, M)}{(\alpha+1)M-\alpha} \leq \frac{F(\alpha-1, M)}{\alpha M-(\alpha-1)}+\frac{g(M)}{(\alpha+1)M-\alpha} \tag{2.6}$$

By induction on α, we see that for $\alpha=2,3,\ldots$

$$\frac{F(\alpha, M)}{(\alpha+1)M-\alpha} \leq \frac{F(1, M)}{2M-1}+g(M)\sum_{j=2}^{\alpha} \frac{1}{(j+1)M-j} \tag{2.7}$$

We recall that $F(1, M) = (M-1)c_N + 2g(M)$. Substituting this into (2.7), we have that for $\alpha = 2, 3, \ldots$

$$\begin{aligned}
\frac{F(\alpha, M)}{(\alpha+1)M - \alpha} &\le \frac{M-1}{2M-1}c_N + g(M)\left(\frac{2}{2M-1} + \sum_{j=2}^{\alpha}\frac{1}{(j+1)M - j}\right) \\
&= \frac{M-1}{2M-1}c_N + g(M)\left(\frac{2}{2M-1} + \frac{1}{M-1}\sum_{j=2}^{\alpha}\frac{1}{j + \frac{M}{M-1}}\right) \\
&\le \frac{M-1}{2M-1}c_N + g(M)\left(\frac{2}{2M-1} + \frac{1}{M-1}\log_e\left(\frac{\alpha + \frac{M}{M-1}}{1 + \frac{M}{M-1}}\right)\right) \\
&= \frac{M-1}{2M-1}c_N + g(M)\left(\frac{2}{2M-1} + \frac{1}{M-1}\log_e\left(\frac{(\alpha+1)M - \alpha}{2M-1}\right)\right) \qquad (2.8)
\end{aligned}$$

Since this expression grows with α, we see that our best upper bound for the performance of Varn without exhaustivity constraints is

$$\bar{d} \le \min_{M \in \{2,\ldots,N\}}\left(\frac{M-1}{2M-1}c_N + g(M)\left(\frac{2}{2M-1} + \frac{1}{M-1}\log_e\left(\frac{[\alpha(M)+1]M - \alpha(M)}{2M-1}\right)\right)\right) \tag{2.9}$$

where $\alpha(M) = \lceil \frac{K-M}{M-1} \rceil$. Note that this has to be better than Krause's bound when c_1 is large enough. However, for fixed M, as K increases, we need to estimate the costs of a larger number of intermediate nodes and our upper bound for Y_i becomes weaker as i increases.

We now discuss a better way to find the Y_i. We begin by studying the Varn algorithm more carefully for the special case where $N = 2$. Here the optimal prefix condition code is also exhaustive. The algorithm produces $K-2$ non-root intermediate nodes. For $i = 1, \ldots, K$, we let v_i represent a leaf node in the code tree and d_i be its respective cost. For each i, $d_i \ge Y_{K-2}$ since Y_{K-2} was the cost of the least costly leaf immediately before that leaf was turned into an intermediate node. Similarly, each intermediate node has a cost of at most Y_{K-2}. Since any leaf v_i can be reached by its immediate ancestral intermediate node with cost at most c_1, $d_i \le Y_{K-2} + c_1$. Also, there are two leaves, say v_{K-1} and v_K, whose immediate ancestral intermediate node was the last intermediate node chosen by the Varn algorithm. To summarize,

$$\begin{aligned}
Y_{K-2} \le d_i &\le Y_{K-2} + c_1,\ i = 1, \ldots, K-2 &\qquad (2.10)\\
d_{K-1} &= Y_{K-2} + c_1 &\qquad (2.11)
\end{aligned}$$

$$d_K \;=\; Y_{K-2} + c_2 \tag{2.12}$$

Multiplying the above inequalities by $-C$ and using the fact that 2^x is a monotonically increasing function of x, we find that

$$\begin{aligned} 2^{-CY_{K-2}-Cc_1} \le 2^{-Cd_i} &\le 2^{-CY_{K-2}},\ i = 1,\ldots,K-2 && (2.13)\\ 2^{-Cd_{K-1}} &= 2^{-CY_{K-2}-Cc_1} && (2.14)\\ 2^{-Cd_K} &= 2^{-CY_{K-2}-Cc_2} && (2.15) \end{aligned}$$

Krause showed that for any prefix condition code, $\sum_{i=1}^{K} 2^{-Cd_i} \le 1$ with equality holding if and only if the code is also exhaustive. Applying this fact to the sum of (2.13), (2.14), and (2.15), we obtain

$$(K-1)\,2^{-CY_{K-2}-Cc_1} + 2^{-CY_{K-2}-Cc_2} \le 1 \le (K-2)\,2^{-CY_{K-2}} + 2^{-CY_{K-2}-Cc_1} + 2^{-CY_{K-2}-Cc_2} \tag{2.16}$$

Multiplying the inequalities in (2.16) by $2^{CY_{K-2}}$ and using the fact that $2^{-Cc_1} + 2^{-Cc_2} = 1$, we see that

$$(K-2)\,2^{-Cc_1} + 1 \;\le\; 2^{CY_{K-2}} \;\le\; K-1 \tag{2.17}$$

Taking the logarithm of both sides of each inequality in (2.17) and dividing by C, we have

$$\frac{\log_2(K-2+2^{Cc_1})}{C} - c_1 \;\le\; Y_{K-2} \;\le\; \frac{\log_2(K-1)}{C} \quad \text{for } K = 3,4,\ldots \tag{2.18}$$

Since $c_2 \le c_1$, we have that $Y_1 = c_2$. As we have seen before,

$$\bar{d} \;=\; \frac{F(K-2,2)}{K} \;=\; \frac{K-1}{K}(c_1+c_2) + \frac{1}{K}\sum_{i=1}^{K-2} Y_i \tag{2.19}$$

Combining (2.18) and (2.19), we find that

$$\begin{aligned} &\tfrac{K-1}{K}(c_1+c_2) + \tfrac{1}{K}\left(c_2 + \textstyle\sum_{m=2}^{K-2}\left(\tfrac{\log_2(m+2^{Cc_1})}{C} - c_1\right)\right)\\ &\quad \le \bar{d} \le \tfrac{K-1}{K}(c_1+c_2) + \tfrac{1}{K}\left(c_2 + \textstyle\sum_{m=2}^{K-2}\tfrac{\log_2(m+1)}{C}\right),\ K = 3,4,\ldots \end{aligned} \tag{2.20}$$

If we let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x, then $m + \lfloor 2^{Cc_1} \rfloor \leq m + 2^{Cc_1}$ for all m. By observing that the "sum of the logarithms is equal to the logarithm of the product," we find that

$$\sum_{m=2}^{K-2} \log_2(m+1) = \log_2\left(\frac{(K-1)!}{2}\right) \tag{2.21}$$

$$\sum_{m=2}^{K-2} \log_2(m + \lfloor 2^{Cc_1} \rfloor) = \log_2\left(\frac{(K-2+\lfloor 2^{Cc_1} \rfloor)!}{(1+\lfloor 2^{Cc_1} \rfloor)!}\right) \tag{2.22}$$

Since $\int_a^b \log_2 x \, dx = b \log_2 b - a \log_2 a - (b-a)\log_2 e = \log_2\left(\frac{b^b}{a^a e^{b-a}}\right)$, we can also lowerbound $\sum_{m=2}^{K-2} \log_2(m + 2^{Cc_1})$ by

$$\sum_{m=2}^{K-2} \log_2(m + 2^{Cc_1}) \geq \log_2\left(\frac{(K-2+2^{Cc_1})^{(K-2+2^{Cc_1})}}{(1+2^{Cc_1})^{(1+2^{Cc_1})} e^{K-3}}\right) \tag{2.23}$$

Substituting (2.21), (2.22) and (2.23) into (2.20), we see that

$$\frac{2}{K}c_1 + c_2 + \frac{1}{CK}\log_2\left(\max\left(\frac{(K-2+2^{Cc_1})^{(K-2+2^{Cc_1})}}{(1+2^{Cc_1})^{(1+2^{Cc_1})} e^{K-3}}, \frac{(K-2+\lfloor 2^{Cc_1} \rfloor)!}{(1+\lfloor 2^{Cc_1} \rfloor)!}\right)\right)$$
$$\leq \bar{d} \leq \frac{K-1}{K}c_1 + c_2 + \frac{1}{CK}\log_2\left(\frac{(K-1)!}{2}\right), \ K = 3, 4, \ldots \tag{2.24}$$

We next demonstrate that when $N = 2$, our upper bound is always tighter than Krause's bound of $\bar{d} < \frac{\log_2 K}{C} + c_1$:

Theorem 2.1 *For all positive integers $K > 2$,*

$$\frac{K-1}{K}c_1 + c_2 + \frac{1}{CK}\log_2\left(\frac{(K-1)!}{2}\right)$$
$$< \frac{\log_2 K}{C} + c_1 - \frac{1}{C}\left(\log_2\left(\frac{e}{2^{Cc_2}}\right) + \frac{1}{K}\log_2\left(2^{Cc_1}\, e^{\frac{-1}{12K}}\sqrt{\frac{2K}{\pi}}\right)\right)$$

We note that $c_1 \geq c_2$ and $2^{-Cc_1} + 2^{-Cc_2} = 1$ imply that $2^{Cc_1} \geq 2$, $2^{Cc_2} \leq 2 < e$. To prove Theorem 2.1, we need the following lemma which appears in Feller (1950):

Lemma 2.2 *For all positive integers K, $K! < \sqrt{2\pi K}\left(\frac{K}{e}\right)^K e^{\frac{1}{12K}}$.*

Proof of Theorem 2.1: By Lemma 2.2, we have that

$$\frac{2K^K}{(K-1)!} = \frac{2K^{K+1}}{K!} > \frac{2K^{K+1}}{\sqrt{2\pi K}\left(\frac{K}{e}\right)^K e^{\frac{1}{12K}}} = \sqrt{\frac{2K}{\pi}}\, e^K\, e^{\frac{-1}{12K}}. \qquad (2.25)$$

Rewriting (2.25), we have that

$$(2^{Cc_2})^K \left(\sqrt{\frac{2K}{\pi}}\left(\frac{e}{2^{Cc_2}}\right)^K e^{\frac{-1}{12K}}\right) < \frac{2^{Cc_1}K^K}{(K-1)!}\cdot\frac{1}{\frac{2^{Cc_1}}{2}} \qquad (2.26)$$

Dividing both sides of (2.26) by $\sqrt{\frac{2K}{\pi}}\left(\frac{e}{2^{Cc_2}}\right)^K e^{\frac{-1}{12K}}$ and taking the logarithm of both sides of the resulting inequality gives

$$KCc_2 < Cc_1 + K\log_2 K - \log_2(K-1)! - \log_2\left(\frac{2^{Cc_1}}{2}\sqrt{\frac{2K}{\pi}}\left(\frac{e}{2^{Cc_2}}\right)^K e^{\frac{-1}{12K}}\right) \qquad (2.27)$$

Subtracting Cc_1 from both sides of (2.27) and dividing by CK gives

$$c_2 - \frac{c_1}{K} < \frac{1}{C}[\log_2 K - \frac{1}{K}\log_2(K-1)!] - \frac{1}{C}\left(\log_2\left(\frac{e}{2^{Cc_2}}\right) + \frac{1}{K}\log_2\left(\frac{2^{Cc_1}\, e^{\frac{-1}{12K}}}{2}\sqrt{\frac{2K}{\pi}}\right)\right) \qquad (2.28)$$

Adding $c_1 + \frac{1}{CK}\log_2\left(\frac{(K-1)!}{2}\right)$ to both sides of (2.28) completes the proof. □

We proceed to generalize some of the results to the case where $N > 2$. An exhaustive code with α non-root intermediate nodes has $(\alpha+1)N - \alpha$ code words, N of which have the most costly intermediate node as the immediate ancestral intermediate node. Hence we can extend (2.10)-(2.17) so that

$$\alpha\,(N-1)\,2^{-Cc_1} + 1 \;\le\; 2^{CY_\alpha} \;\le\; \alpha\,(N-1)+1 \qquad (2.29)$$

Taking the logarithm of both sides of each inequality in (2.29) and dividing by C, we have

$$\frac{\log_2[\alpha\,(N-1) + 2^{Cc_1}]}{C} - c_1 \;\le\; Y_\alpha \;\le\; \frac{\log_2[\alpha\,(N-1)+1]}{C} \text{ for } \alpha = 1,2,\ldots \qquad (2.30)$$

Since c_N is the cost of the least costly channel letter, $Y_1 = c_N$. If we consider "modified" channels of the kind we have defined before, and we let $C_{(M)}$ be defined by $\sum_{i=1}^{M} 2^{-C_{(M)}c_{N+1-i}} = 1$, then by substituting into (2.2) we see that

$$\begin{aligned} F(\alpha, M) &\leq (\alpha+1)g(M) + (M-1)\left(c_N + \sum_{j=2}^{\alpha} \frac{\log_2[j(M-1)+1]}{C_{(M)}}\right) \\ &= (\alpha+1)g(M) + (M-1)\left(c_N + \sum_{j=2}^{\alpha} \frac{\log_2(M-1) + \log_2(j+\frac{1}{M-1})}{C_{(M)}}\right) \\ &= (\alpha+1)g(M) + (M-1)\left(c_N + \frac{(\alpha-1)\log_2(M-1) + \sum_{j=2}^{\alpha}\log_2(j+\frac{1}{M-1})}{C_{(M)}}\right) \end{aligned} \tag{2.31}$$

By similar arguments to the ones we used earlier,

$$\begin{aligned} \sum_{j=2}^{\alpha} \log_2(j + \frac{1}{M-1}) &\leq \sum_{j=2}^{\alpha} \log_2(j+1) \\ &= \log_2\left(\frac{(\alpha-1)!}{2}\right) \end{aligned} \tag{2.32}$$

$$\begin{aligned} \sum_{j=2}^{\alpha} \log_2(j + \frac{1}{M-1}) &\leq \int_{2+\frac{1}{M-1}}^{\alpha+1+\frac{1}{M-1}} \log_2 x \, dx \\ &\leq \log_2\left(\frac{((\alpha+1)M-\alpha)^{\alpha+1+\frac{1}{M-1}}}{(2M-1)^{2+\frac{1}{M-1}}\, e^{\alpha-1}}\right) - (\alpha-1)\log_2(M-1) \end{aligned} \tag{2.33}$$

Substituting (2.32) and (2.33) into (2.31), we have that

$$F(\alpha, M) \leq (\alpha+1)g(M) + (M-1)\left(c_N + \frac{1}{C_{(M)}}\log_2(h(\alpha, M))\right) \tag{2.34}$$

where

$$h(\alpha, M) = \min\left(\frac{(\alpha-1)!(M-1)^{\alpha-1}}{2}, \frac{((\alpha+1)M-\alpha)^{\alpha+1+\frac{1}{M-1}}}{(2M-1)^{2+\frac{1}{M-1}}\, e^{\alpha-1}}\right). \tag{2.35}$$

Note that $\frac{F(\alpha,M)}{(\alpha+1)M-\alpha}$ increases as α increases. Therefore, our tightest upper bound for $\bar{d}$ is

$$\bar{d} \leq \min_{M\in\{2,\ldots,N\}} \frac{F(\alpha(M), M)}{[\alpha(M)+1]M - \alpha(M)} \tag{2.36}$$

where $\alpha(M) = \lceil \frac{K-M}{M-1} \rceil$.

For each M, $\frac{F(\alpha(M),M)}{[\alpha(M)+1]M-\alpha(M)}$ is an upper bound on the average cost, $d^*_{(M)}$, of the best exhaustive code with at least K code words when we use the M least expensive channel letters. In turn, $d^*_{(M)}$ is an upper bound on the average cost of the code consisting of the K least costly code words of the exhaustive code corresponding to $d^*_{(M)}$. We note that for K sufficiently large, all code letters will be used and so our bound becomes $\overline{d} \leq \frac{F(\alpha(N),N)}{[\alpha(N)+1]N-\alpha(N)}$. Since this upper bound is an upper bound on the cost of the best exhaustive code with at least K code words, we expect that Krause's bound is a better bound for very large K.

Chapter 3

A New Heuristic Based on Huffman's Algorithm

In this section, we consider a heuristic to find good prefix condition codes when our source is memoryless and the channel is binary with memoryless letter costs. Our source ensemble consists of symbols $u_1, u_2, \ldots, u_K$ with probabilities $p_1 \geq p_2 \geq \cdots \geq p_K$, respectively; our channel alphabet has digits $0, 1$ with positive costs of transmission $c_0 \geq c_1$ respectively. As before, $H(U)$, the source entropy, is defined by

$$H(U) = \sum_{k=1}^{K} p_k \log_2 \frac{1}{p_k}$$

and C, the channel capacity, is defined by

$$2^{-Cc_0} + 2^{-Cc_1} = 1. \tag{3.1}$$

For $i = 1, \ldots, K$, the code word corresponding to u_i is x_i, the cost of transmitting x_i is d_i and the average cost of transmitting a symbol is $\overline{d} = \sum_{i=1}^{K} p_i d_i$.

Krause (1962) and Mehlhorn (1980) have developed heuristics to obtain good prefix condition codes for the more general problem where the channel is discrete, memoryless and noiseless, but not necessarily binary. Their algorithms are generalizations of the Shannon-Fano heuristic, which produces relatively good codes when all of the channel letters have the same cost of transmission. The number of steps required to produce codes for all of these algorithms is $o(NK)$, where N is the number of letters in the channel alphabet. Huffman (1952) found an algorithm to produce optimal prefix condition codes when the channel letters are equally costly. The heuristic we describe below was inspired by Huffman's algorithm. The development of our heuristic closely follows Sections 3.3 and 3.4 of Gallager (1968).

Krause demonstrated that

$$\bar{d} \geq \frac{H(U)}{C}. \tag{3.2}$$

Any prefix condition code satisfies

$$\sum_{i=1}^{K} 2^{-Cd_i} \leq 1 \tag{3.3}$$

with equality for the binary codes we consider. We use (3.3) to get an alternate proof of (3.2):

Lemma 3.1 $\bar{d} \geq \frac{H(U)}{C}$.

Proof: We establish (3.2) by showing that $H(U) - \bar{d}C \leq 0$.

$$H(U) - \bar{d}C \;=\; \sum_{k=1}^{K} p_k \log_2\left(\frac{1}{p_k}\right) - \sum_{k=1}^{K} p_k d_k C \;=\; \sum_{k=1}^{K} p_k \log_2\left(\frac{2^{-Cd_k}}{p_k}\right)$$

Since $\log_2 Z \leq (\log_2 e)(Z-1)$ for $Z > 0$, with strict inequality except when $Z = 1$,
$H(U) - \bar{d}C \;\leq\; (\log_2 e)(\sum_{k=1}^{K} 2^{-Cd_k} - \sum_{k=1}^{K} p_k) \;\leq\; 0$ because of (3.3). □
Notice that equality holds in (3.2) if and only if $p_k = 2^{-Cd_k}, k = 1, \ldots, K$ so that the distances satisfy

$$d_k = -\frac{1}{C}\log_2 p_k \;,\; k = 1, \ldots, K \tag{3.4}$$

Lemma 3.2 *For any given source with $K \geq 2$ letters, an optimum binary code exists in which there is some $L \in \{1, \ldots, K-1\}$ for which x_K and x_L differ only in the last digit, x_K ending in a 0 and x_L ending in a 1.*

Proof: For at least one optimum code, d_K is greater than or equal to each of the other code word costs. To see this, suppose there is a code for which $d_K < d_i$ for some i. If the code words x_i and x_K are interchanged, then the change in $\bar{d}$ is
$\triangle \;=\; p_i d_K + p_K d_i - p_i d_i - p_K d_K \;=\; (p_i - p_K)(d_K - d_i) \;\leq\; 0.$

Next observe that any optimal prefix condition code is also exhaustive; in any non-exhaustive code, one or more code letters in one or more code words could be dropped without violating the prefix condition.

We let x denote the prefix of x_K consisting of all but the last digit of x_K. Clearly, there are either one, two, or more than two code words which have x as a prefix. Since any optimal code is exhaustive, x_K can not be the only code word having x as a prefix. Now suppose that there are at least two code words, other than x_K, which have x as a prefix. The exhaustivity of an optimal prefix condition code then implies that there exists at least one code word, other than x_K, which has x as a prefix and the last digit of x_K among its two or more remaining code letters. The cost of transmission of such a code word is greater than d_K, a contradiction. Hence, there are exactly two code words, x_K and, say, x_L, which have x as a prefix. Let d^* be the cost of transmitting x. Then

$d_K = d^* +$ cost of transmitting the last digit of x_K,

$d_L = d^* +$ cost of transmitting the last digit of x_L,

$d_K \geq d_L$, and $c_0 \geq c_1$ imply that the last digit of x_K is 0 and the last digit of x_L is 1.

With this lemma, we have reduced the problem of constructing an optimal code to that of picking L, and constructing x and x_i for $i \neq L, K$. We now define the reduced ensemble U' to be the ensemble U with the symbols u_L and u_K replaced by a supersymbol α; the unchanged source symbols have the same probabilities as before; i.e., $p_i' = p_i$, $i \neq K, L$, and $p_\alpha' = p_L + p_K$.

We can transform any prefix condition code of U' into a corresponding prefix condition code for U by adding a terminal 0 to x_α' to generate x_K and adding a terminal 1 to generate x_L. If we knew which value of L is optimal, we would have the following result.

Lemma 3.3 *If we know the best value of L, then if a prefix condition code is optimal for U', then the corresponding prefix condition code for U is optimal.*

Proof: The costs d_k' of the code words for U' are related to the costs d_k of the corresponding code for U by

$$d_k = \begin{cases} d_k', & k \neq K, L \\ d_\alpha' + c_0, & k = K \\ d_\alpha' + c_1, & k = L \end{cases}$$

Thus the average costs $\overline{d'}$ and $\overline{d}$ are related by

$$\begin{aligned}
\overline{d} &= \sum_{k=1}^{K} p_k d_k \\
&= \sum_{k \neq K,L} p_k d_k + p_K(d^* + c_0) + p_L(d^* + c_1) \\
&= \sum_{k \neq K,L} p_k d_k + (p_K + p_L) d_\alpha' + c_0 p_K + c_1 p_L \\
&= \sum_{k \neq K,L} p_k' d_k' + p_\alpha' d_\alpha' + c_0 p_K + c_1 p_L \\
&= \overline{d'} + c_0 p_K + c_1 p_L
\end{aligned}$$

Since $\overline{d}$ and $\overline{d'}$ differ by a fixed amount independent of the code for U', Lemma 2.2 implies that we can minimize $\overline{d}$ by minimizing $\overline{d'}$. □

The problem of finding an optimal code has now been reduced to the problem of finding an optimal code for an ensemble with one fewer message. Therefore, at each step, we need to find L. The heurisitic we describe below is a procedure to select L at each step. The algorithm does not always produce optimal codes, but it is intuitively appealing.

As we noted before

$$\begin{aligned}
\overline{d} &= \sum_{k \neq K,L} p_k d_k + p_K(d^* + c_0) + p_L(d^* + c_1) \\
&= \sum_{k=1}^{K} p_k d_k + p_K(d^* + c_0 - d_K) + p_L(d^* + c_1 - d_L) \qquad (3.5)
\end{aligned}$$

We would like to pick the value of L that minimizes (3.5). Since we do not know the values of d^* and d_k, $k = 1, \ldots, K$, we need to estimate them. We use (3.4) to obtain approximate values for d_k, $k = 1, \ldots, K$. To estimate d^*, we note that $d^* = d_\alpha'$ and so we will guess that

$$d^* = d_\alpha' \approx -\frac{1}{C} \log_2 p_\alpha = -\frac{1}{C} \log_2 (p_K + p_L) \qquad (3.6)$$

Equations (3.4), (3.5) and (3.6) imply that

$$\begin{aligned}\bar{d} &\approx \frac{1}{C}\left(\sum_{i=1}^{K} -p_K \log_2 p_K\right) + p_K(-\frac{1}{C}\log_2(p_K+p_L) + c_0 + \frac{1}{C}\log_2 p_K) \\ &\quad + p_L(-\frac{1}{C}\log_2(p_K+p_L) + c_1 + \frac{1}{C}\log_2 p_L) \\ &= \frac{H(U)}{C} + (\frac{1}{C}p_K\log_2 p_K + \frac{1}{C}p_L\log_2 p_L) - \frac{1}{C}(p_K+p_L)\log_2(p_K+p_L) + c_0 p_K + c_1 p_L \end{aligned} \tag{3.7}$$

Intuitively, it seems plausible to pick L to minimize (3.7). Equivalently, we wish to pick z to minimize

$$f_K(z) = \frac{1}{C} z\log_2 z - \frac{1}{C}(p_K+z)\log_2(p_K+z) + c_1 z \tag{3.8}$$

where $z \in \{p_1, p_2, \ldots, p_{K-1}\}$.

We have that

$$\frac{df_K(z)}{dz} = \frac{1}{C}\log_2\left(\frac{2^{Cc_1} z}{z+p_K}\right) \tag{3.9}$$

Setting (3.9) to zero, we see that the value of z which minimizes (3.8) over all $z \geq 0$ is

$$z_K^* = \frac{p_K}{2^{Cc_1}-1}. \tag{3.10}$$

Clearly, if there is some $L \in \{1, \ldots, K-1\}$ for which $p_L = z_K^*$, we will pick this value of L. Otherwise, since $f_K(z)$ is a continuous function of z, there are three possibilities:

1. $p_{K-1} > z_K^*$: In this case, we select $L = K-1$.

2. $p_J > z_K^* > p_{J+1}$ for some $J \in \{1, \ldots, K-2\}$: If $f_K(p_J) < f_K(p_{J+1})$, then we choose $L = J$ and otherwise we select $L = J+1$.

3. $p_1 < z_K^*$: Here, $L = 1$ is our best choice.

As with the Mehlhorn and Krause algorithms for the special case of a binary channel, this heuristic requires $o(K)$ steps to produce a code; however, this algorithm is simpler than either of theirs.

We now make some comments on the performance of our heuristic. First, when $c_0 = c_1$, equation (3.1) implies that $C = \frac{1}{c_1}$ and so $2^{Cc_1} - 1 = 1$. Hence, $z_K^* = p_K$ which implies that at every step of the algorithm, we group together the two least probable symbols of the ensemble. This procedure is better known as Huffman's algorithm and it is well-known to produce optimal codes when $c_0 = c_1$.

We can make the following statement about the process of picking L :

Theorem 3.4 *For $J = 2, \ldots, K$, z_J^* is among the set of $J-1$ source probabilities from which we make our selection if and only if $\overline{d} = \frac{H(U)}{C}$.*

The proof of Theorem 3.4 is based on the following definitions and lemmas: We let q_i, $i = 1, \ldots, K-1$, be the probability associated with intermediate node i and $q_{i,j}$, $j \in \{0,1\}$, be the probability of the node which is connected to its immediate ancestral node i, by an arc corresponding to digit j. Clearly,

$$q_{i,0} + q_{i,1} = q_i, \; i = 1, \ldots, K-1 \tag{3.11}$$

In his previously mentioned paper, Mehlhorn proved the following two lemmas:

Lemma 3.5 $H(U) = \sum_{i=1}^{K-1} q_i \left(\frac{q_{i,0}}{q_i} \log_2 \left(\frac{q_i}{q_{i,0}} \right) + \frac{q_{i,1}}{q_i} \log_2 \left(\frac{q_i}{q_{i,1}} \right) \right)$

Lemma 3.6 $\overline{d} = \sum_{i=0}^{K-1} [c_0 q_{i,0} + c_1 q_{i,1}]$

We are now ready to prove Theorem 3.4.

Proof of Theorem 3.4: Because of the previous two lemmas, $\overline{d} = \frac{H(U)}{C}$ is equivalent to the following statement:

$$\sum_{i=1}^{K-1} \left(q_{i,0} \log_2 \left(\frac{q_i 2^{-Cc_0}}{q_{i,0}} \right) + q_{i,1} \log_2 \left(\frac{q_i 2^{-Cc_1}}{q_{i,1}} \right) \right) = 0 \tag{3.12}$$

Since $\log_2 Z \leq (\log_2 e)(Z-1)$ for $Z > 0$, with strict inequality except when $Z = 1$,

$$\begin{aligned} &\textstyle\sum_{i=1}^{K-1} \left(q_{i,0} \log_2 \left(\frac{q_i 2^{-Cc_0}}{q_{i,0}} \right) + q_{i,1} \log_2 \left(\frac{q_i 2^{-Cc_1}}{q_{i,1}} \right) \right) \\ &\leq (\log_2 e) \sum_{i=1}^{K-1} [q_i 2^{-Cc_0} - q_{i,0}] + [q_i 2^{-Cc_1} - q_{i,1}] \\ &= 0 \end{aligned} \tag{3.13}$$

because of equations (3.1) and (3.11). The condition for equality in (3.13) is

$$
\begin{aligned}
q_{i,0} &= q_i 2^{-Cc_0},\ i = 1, \ldots, K-1 && (3.14)\\
\text{and } q_{i,1} &= q_i 2^{-Cc_1},\ i = 1, \ldots, K-1 && (3.15)
\end{aligned}
$$

Hence, by dividing each side of equation (3.15) by the corresponding side of equation (3.14), we see that $\overline{d} = \frac{H(U)}{C}$ if and only if

$$\frac{q_{i,1}}{q_{i,0}} = \frac{1}{2^{Cc_1}2^{-Cc_0}} = \frac{1}{2^{Cc_1}[1-2^{-Cc_1}]} = \frac{1}{2^{Cc_1}-1} \tag{3.16}$$

This is merely another way of stating the theorem. □

Above we have shown that our heuristic produces optimal codes for two important special cases of the general problem. For these cases, neither the Mehlhorn nor the Krause algorithm is guaranteed to produce optimal codes. Unfortunately, as we will demonstrate below, for any $M > 0$, we can construct an instance of the problem for which the average cost of the code produced by the algorithm exceeds the expected cost of the ideal prefix condition code for that instance by at least M.

Suppose we have a source whose ensemble U consists of K equiprobable elements. In the first step of our heuristic, we group together two symbols with the same source probability. The reduced ensemble U' has $K-2$ symbols with probability $\frac{1}{K}$ and one symbol with probability $\frac{2}{K}$. Using the notation we introduced earlier in the section,. we have that we group together two symbols of U' with probability $\frac{1}{K}$ when

$$f_{K-1}\left(\frac{1}{K}\right) \leq f_{K-1}\left(\frac{2}{K}\right) \tag{3.17}$$

or, equivalently, when

$$\frac{1}{CK}\log_2\left(\frac{1}{K}\right) - \frac{2}{CK}\log_2\left(\frac{2}{K}\right) + \frac{c_1}{K} \leq \frac{2}{CK}\log_2\left(\frac{2}{K}\right) - \frac{3}{CK}\log_2\left(\frac{3}{K}\right) + \frac{2c_1}{K} \tag{3.18}$$

It is easy to show that the above inequality is valid when $2^{-Cc_1} \leq \frac{16}{27}$, independent of the value of K. Therefore, if $2^{-Cc_1} \leq \frac{16}{27}$ and $K = 2^k$ for some positive integer k, the above analysis can be extended to subsequent reduced ensembles to demonstrate that the code produced by the algorithm is the set of K channel sequences of length k. When $2^{-Cc_1} = \frac{1}{2}$, the algorithm produces optimal codes for all k. For $2^{-Cc_1} > \frac{1}{2}$, there is some positive integer l for which the heuristic produces codes that are not optimal for all integers $k \geq l$. To see this, we note that the average cost of transmitting a code word for the code produced by the algorithm is

$$\frac{k}{2}(c_0 + c_1)$$

while the expected code word cost of the optimal prefix condition code is upperbounded by

$$\frac{H(U)}{C} + c_0 = \frac{\log_2 K}{C} + c_0 = \frac{k}{C} + c_0.$$

Hence the difference in the expected costs is at least

$$\frac{k}{C}\left(\frac{Cc_0 + Cc_1}{2} - 1\right) - c_0. \tag{3.19}$$

Since $c_0 > c_1$, equation (3.1) implies that $Cc_0 + Cc_1 > 2$. Therefore, we can make (3.19) arbitrarily large by selecting large enough k.

The fact that our heuristic may produce codes with expected code word cost arbitrarily larger than that of the corresponding optimal prefix condition code is a serious weakness indeed. We note that the Krause and the Mehlhorn algorithms do not share this deficiency. However, the example we discussed above may be a worst case because at each iteration of the algorithm our set of source probabilities consists of repetitions of at most two distinct numbers.

More generally, we would suspect that as K shrinks, the smallest value of

$$\frac{1}{C}[p_K \log_2 p_K + p_L \log_2 p_L - (p_K + p_L)\log_2(p_K + p_L)] + c_0 p_K + c_1 p_L,\ L \in \{1, \ldots, K-1\}$$

increases since we have fewer source probabilities from which to make our selection. For small enough K, we can deal with the problem by bypassing the algorithm and finding the optimal code by means of exhaustive enumeration of the possible codes. We can use this idea to improve the performance of our heuristic. By finding optimal codes for larger reduced ensembles, we improve the performance of the code for the original ensemble at the expense of studying a number of codes that is exponential in the size of the reduced ensemble.

Chapter 4

A Multicommodity Network Flows Approach to the Problem of Finding Optimal Prefix Condition Codes

As usual, we assume that we have a memoryless K-ary source ensemble with source probabilities $p_1, p_2, \ldots, p_K$. Our N-ary channel with alphabet $\{x_1, x_2, \ldots, x_N\}$ may have memory, but that memory is limited to individual code words as opposed to the concatenation of code words. For example, we allow constraints on the maximum code word costs and/or lengths, but we can not accept the runlength constraints described in the introductory section. We also assume that each channel letter always has a non-negative cost of transmission.

The last assumption ensures that each code word in an optimal prefix condition code has at most $K-1$ channel letters. Therefore, our optimal code can be visualized as a subtree of the complete tree of $K-1$ levels, where the nodes on these trees represent sequences of channel letters. The root node represents the null sequence which is assumed to have zero cost. The immediate descendants of the node representing channel sequence α are the N nodes representing the channel sequences $\alpha x_1, \ldots, \alpha x_N$. The branch joining the node corresponding to channel sequence x with its immediate ancestral intermediate node has a cost representing the incremental cost of transmitting the last letter of x. Hence, the cost of transmitting x is the sum of the branch costs on the path from the root to the node corresponding to x.

We say that a node is a *top level node* if it is not an intermediate node in the complete tree. The top level nodes correspond to channel sequences of length $K-1$. The branches which have a top level node as an endpoint are called *top level arcs.*

We wish to send various *flows* from the root node to the set of top level nodes. We denote one set of flows by $b^{(k)}$, $k=1,\ldots,K$; $b^{(k)}$ represents the "flow" of probability p_k from the root node to the top level nodes. We let $b^{(k)}_{i,j}$ be the flow of the commodity corresponding to p_k through the arc connecting node i to node j. To formulate the supply of these flows at the root node, we let r and $r_1,\ldots,r_N$ represent the root node and its immediate descendant nodes, respectively. Then

$$\sum_{i=1}^{N} b^{(k)}_{r,r_i} = p_k,\; k=1,\ldots,K. \tag{4.1}$$

Also, for any non-root intermediate node j with immediate ancestral intermediate node $a(j)$ and immediate descendant nodes $j_1,\ldots,j_N$, conservation of flow at node j implies that

$$\sum_{i=1}^{N} b^{(k)}_{j,j_i} = b^{(k)}_{a(j),j},\; k=1,\ldots,K. \tag{4.2}$$

We insist that all arc flows are non-negative. For all branches (i,j) in the tree,

$$b^{(k)}_{i,j} \geq 0,\; k=1,\ldots,K. \tag{4.3}$$

Finally, for each $k\in\{1,\ldots,K\}$, we would like $b^{(k)}_{i,j}$ to be positive for exactly one arc in each level of the tree. In other words, for each $k\in\{1,\ldots,K\}$, we wish to have a unique path from the root node to a top level node on which $b^{(k)}$ is positive. To obtain this result, we introduce the indicator flows $y^{(k)}$, $k=1,\ldots,K$ which are binary variables. For all arcs (i,j) in the tree we impose the restrictions,

$$y^{(k)}_{i,j} \in \{0,1\},\; k=1,\ldots,K \tag{4.4}$$

$$\text{and } b^{(k)}_{i,j} = p_k\, y^{(k)}_{i,j}\;,\; k=1,\ldots,K. \tag{4.5}$$

We note that equations (4.2) and (4.5) force conservation of flow for each $y^{(k)}$ at all non-root intermediate nodes. In order to prevent the possibility of two of the $b^{(k)}$ paths from being exactly identical, we constrain all top level arcs (i,j) in the tree to satisfy

$$\sum_{i=1}^{K} y_{i,j}^{(k)} \leq 1. \tag{4.6}$$

For each arc (i,j) in the tree, we introduce a new variable $w_{i,j} = \min\{1, \sum_{i=1}^{K} y_{i,j}^{(k)}\}$ which indicates whether or not $y_{i,j}^{(k)}$ is positive for any $k \in \{1,\ldots,K\}$. In order to use linear constraints to define the $w_{i,j}$, we have that for all arcs (i,j) in the tree,

$$\begin{aligned} w_{i,j} &\geq y_{i,j}^{(k)},\ k=1,\ldots,K && (4.7) \\ w_{i,j} &\leq 1 && (4.8) \\ w_{i,j} &\leq \sum_{i=1}^{K} y_{i,j}^{(k)} && (4.9) \end{aligned}$$

For any $y^{(k)}$ path, a *leaf node* j is the node on the path which corresponds to the shortest channel sequence and satisfies $\sum_{k=1}^{K} y_{a(j),j}^{(k)} = 1$. For any leaf node j, $(a(j),j)$ is called a *leaf arc*. The expected cost of a code does not depend at all on the nodes or corresponding arcs of any $y^{(k)}$ path which have the leaf node of this path as an ancestor. Therefore, if we know the leaf node corresponding to path $y^{(k)}$, we can arbitrarily select the path from the leaf node to the set of top level nodes.

For all arcs (i,k) in our code tree, we let $c_{i,k}$ be the cost associated with branch (i,k). For all intermediate nodes j, we label j's immediate descendant nodes $j_1,\ldots,j_N$ so that $c_{j,j_1} \geq c_{j,j_2} \geq \cdots \geq c_{j,j_N}$. We observe that if arc (j,j_i) is a branch in an optimal code tree, then for all $k \in \{i+1,\ldots,N\}$, arc (j,j_k) is also a branch in that code tree. Therefore, for all intermediate nodes j in our tree, we insist that

$$w_{j,j_1} \leq w_{j,j_2} \leq \cdots \leq w_{j,j_N}. \tag{4.10}$$

Note that (4.10) imposes a path from the leaf arcs to the top level arcs since for all non-root intermediate nodes j, we have that

$$w_{a(j),j} = w_{j,j_N}. \tag{4.11}$$

For the purposes of calculating the expected cost of a code, we are interested in the arcs on each $y^{(k)}$ path that are also on the subpath from the root node to the leaf node for that path. The set of such arcs is exactly the same as the set of branches in our code tree. For each arc (j,l) in the tree, we define the variable $f_{j,l}$ to be 1 if (j,l) is on a path between the root node and a leaf node and 0 otherwise. In an optimal code, there is no string of channel letters which is a proper prefix of exactly one code word. Therefore, because of equation (4.11), if $l \neq j_N$, then $w_{j,l} = 1$ if and only if $f_{j,l} = 1$. If $l = j_N$, then $f_{j,l} = 1$ if and only if $f_{j,j_i} = 1$ for some $i \neq N$. Hence the previous statement and (4.10) imply that

$$f_{j,l} = \begin{cases} w_{j,l}, & \text{if } l \neq j_N \\ w_{j,j_{N-1}}, & \text{if } l = j_N \end{cases} \tag{4.12}$$

Our optimization problem is to

$$\text{minimize} \sum_{k=1}^{K} \sum_{(i,j)\in tree} c_{i,j} f_{i,j} x_{i,j}^{(k)}$$

subject to constraints (4.1) through (4.12).
Since

- $c_{i,j}$ is a constant for all arcs (i,j) in the tree and
- all of the above constraints, except for (4.4), are linear,

the minimization problem can be classified as an integer bilinear programming problem. Unfortunately, very little is known about obtaining the solution to these problems.

This formulation of our optimization problem allows arbitrary non-negative branch costs $c_{i,j}$. This freedom in selecting branch costs can be used to provide memory within individual code words. For example, it is easy to forbid certain nodes, their descendants and the corresponding branches from appearing in the code tree by making the appropriate arc costs infinite. We can also produce optimal exhaustive prefix condition codes by adding the constraint that for all intermediate nodes j,

$$f_{j,j_1} = f_{j,j_2} = \cdots = f_{j,j_N}. \tag{4.13}$$

In the remainder of this section, we discuss some of the additional constraints we can impose to get approximate "solutions" if we drop the binary requirement on the $y^{(k)}$. We always have that (r, r_N) and (r, r_{N-1}) are branches in our code tree. Therefore, it is obvious that

$$f_{r,r_N} = f_{r,r_{N-1}} = 1. \tag{4.14}$$

The rest of the constraints we describe involve the leaf arcs, which we now discuss in more detail. We let $z_{i,k}$ be the variable indicating whether or not branch (i, k) is a leaf arc. If the indicator flows $y^{(m)}$ are all binary variables and we know $w_{i,k}$ for all arcs (i, k), then we know the K arcs which are leaf arcs. There are four situations to consider:

1. (j, j_l) is not a top level arc and $l \neq N$: Then w_{j,j_l} is a leaf arc if and only if $w_{j,j_l} = 1$ and $w_{j_l,j_l(N-1)} = 0$, where $j_l(M)$, $M = 1, \ldots, N$ are the N descendants of j_l and $c_{j_l,j_l(1)} \geq c_{j_l,j_l(2)} \geq \cdots \geq c_{j_l,j_l(N)}$. Therefore, it is not difficult to see that

$$z_{j,j_l} = w_{j,j_l} - w_{j_l,j_l(N-1)}. \tag{4.15}$$

2. (j, j_N) is not a top level arc: Then (j, j_N) is a leaf arc if and only if $w_{j,j_N} = 1$, $w_{j,j_{N-1}} = 1$ and $w_{j_N,j_N(N-1)} = 0$. Hence,

$$z_{j,j_N} = w_{j,j_{N-1}} - w_{j_N,j_N(N-1)} \tag{4.16}$$

3. (j, j_l) is a top level arc and $l \neq N$: Then

$$z_{j,j_l} = w_{j,j_l}. \tag{4.17}$$

4. (j, j_N) is a top level arc: $z_{j,j_N} = 0$ if and only if $w_{j,j_l} = 0$ for all $l \in \{1, \ldots, N-1\}$. Hence,

$$z_{j,j_N} = w_{j,j_{N-1}}. \tag{4.18}$$

When we drop the integer constraints on the indicator flows, a very simple result that we can take advantage of is that any code tree must have K leaf arcs. Hence,

$$\sum_{(i,j)\in tree} z_{i,j} = K. \tag{4.19}$$

For all arcs (i,j) in the tree, we define $l(i,j)$ to be the number of letters in the channel sequence corresponding to node j. Then by Kraft's inequality, we have the constraint that

$$\sum_{(i,j)\in tree} z_{i,j}\, N^{-l(i,j)} \leq 1. \tag{4.20}$$

For the special case of a binary channel the above inequality is met with equality for all optimal codes. Similarly, if we have a memoryless channel with capacity C and we define $d(i,j)$ to be the cost associated with transmitting the channel sequence corresponding to node j, then

$$\sum_{(i,j)\in tree} z_{i,j}\, 2^{-C\cdot d(i,j)} \leq 1 \tag{4.21}$$

with equality in the case of a binary channel. Constraints (4.20) and (4.21) are also applicable for channels which are memoryless except for restrictions on individual code words, such as an upper bound on acceptable code word costs or code word lengths.

We can use the solution to the minimization problem without the binary constraints to find a code by creating a code tree from those arcs (i,j) for which $f_{i,j}$ is judged to be sufficiently close to 1.

Chapter 5

Arithmetic Coding for Finite-State Noiseless Channels

In this chapter, we generalize the ideas of arithmetic coding to handle finite-state noiseless channels. Our development closely follows the papers of Gallager (1990) and Csiszár (1969).

We begin by assuming that our K-ary source is memoryless; later, we will waive this assumption. We first consider memoryless cost channels and we subsequently generalize our results to finite-state channels.

We denote the random sequence produced by the source as $y = \{y_1, y_2, y_3, \ldots\}$. For all $i \in \{1, \ldots, K\}$ and all positive integers m, we have that $p_i = P(y_m = u_i)$. Let $y^{(m)}$ be the initial string $\{y_1, y_2, \ldots, y_m\}$ of the source output for each positive integer m. Since the source is memoryless, the probability and self-information of $y^{(m)}$ are then $P[y^{(m)}] = \prod_{j=1}^{m} P[y_j]$ and $I[y^{(m)}] = \sum_{j=1}^{m} I(y_j)$, respectively. We note that for all source strings u, we have the relationship $I(u) = \log_2\left(\frac{1}{P[u]}\right) = -\log_2(P[u])$.

The idea in arithmetic coding is to map the source sequence y into a point x in the unit interval on the real line and then to represent x by a channel sequence $z = \{z_1, z_2, \ldots\}$. First, we will discuss the mapping of source strings into subintervals of the unit interval. For any non-negative integer m, we let $\mathcal{I}(y^{(m)})$ denote the subinterval corresponding to source string $y^{(m)}$; our convention at $m = 0$ is that $y^{(0)}$ is the null source string. In earlier work on arithmetic coding, there were two important properties associated with the mapping of source strings into subintervals of the unit interval. The first is that if we are given $y^{(m)}$, then the point x corresponding to sequence y is uniformly distributed on $\mathcal{I}(y^{(m)})$. The other characteristic is that the length of $\mathcal{I}(y^{(m)})$ gives some measure of the self-information of $y^{(m)}$. As the self-information of the initial source string increases, we should have a better idea of where the final point x will lie on the unit interval. Hence, for all source strings u, the length of $\mathcal{I}(u)$ should be a monotonically decreasing function of $I(u)$. More precisely, the (left half-closed) intervals have traditionally been selected to satisfy the following requirements:

- for all source strings u, the width of interval $\mathcal{I}(u)$ is equal to the a priori probability that the string is a prefix of the source sequence; our convention for the null string is that it is considered to be a prefix of every source sequence and hence $P[\emptyset] = 1$.
- for any source string u, we have that $\mathcal{I}(uu_1)$, ... , $\mathcal{I}(uu_K)$ are disjoint intervals whose union is $\mathcal{I}(u)$; clearly, the previous constraint implies that $\mathcal{I}(\emptyset) = [0, 1)$.

One way to implement these requirements is as follows. We define

$$f(u_i) = f_1(u_i) = \begin{cases} 0, & i = 1 \\ \sum_{j=1}^{i-1} p_j, & i \in \{2, \ldots, K\} \end{cases} \tag{5.1}$$

and for $m > 1$,

$$f(y^{(m)}) = f(y^{(m-1)}) + f_1(y_m)P(y^{(m-1)}). \tag{5.2}$$

We then let $\mathcal{I}(y^{(m)})$ be the interval $[f(y^{(m)}), f(y^{(m)}) + P(y^{(m)}))$. Assuming that all of the letter probabilities are strictly less than one, for all source sequences y, $\mathcal{I}(y^{(m)})$ contracts to a point as m goes to infinity. Therefore, we choose x to be the point on the unit interval which is the limit of these intervals. We note that the mapping of source sequences to points has the following lexicographic properties: Given arbitrary source sequences y_1 and y_2, we let $x(y_1)$ and $x(y_2)$ be the points on the unit interval to which y_1 and y_2 are mapped, respectively. If m is the largest integer for which $y_1^{(m)} = y_2^{(m)}$ and the last symbols of strings $y_1^{(m+1)}$ and $y_2^{(m+1)}$ are u_i and u_j, respectively, then

- $x(y_1) > x(y_2)$ if and only if $i > j$,
- for $i \in \{1, 2\}$, $x(y_i) \in \mathcal{I}(y_1^{(m)})$ and

- for $i \in \{1,2\}$, $x(y_i) \in \mathcal{I}(y_i^{(m+1)})$; we note that $\mathcal{I}(y_1^{(m+1)})$ and $\mathcal{I}(y_2^{(m+1)})$ are disjoint.

As an example to illustrate the above procedure, we consider a ternary source with $p_1 = 0.5$, $p_2 = 0.3$ and $p_3 = 0.2$. Then

$$\begin{array}{lll}
\mathcal{I}(\emptyset) = [0,1) & \mathcal{I}(u_1) = [0,0.5) & \mathcal{I}(u_1u_1) = [0,0.25) \\
 & & \mathcal{I}(u_1u_2) = [0.25,0.4) \\
 & & \mathcal{I}(u_1u_3) = [0.4,0.5) \\
 & & \\
 & \mathcal{I}(u_2) = [0.5,0.8) & \mathcal{I}(u_2u_1) = [0.5,0.65) \\
 & & \mathcal{I}(u_2u_2) = [0.65,0.74) \quad \cdots \\
 & & \mathcal{I}(u_2u_3) = [0.74,0.8) \\
 & & \\
 & \mathcal{I}(u_3) = [0.8,1) & \mathcal{I}(u_3u_1) = [0.8,0.9) \\
 & & \mathcal{I}(u_3u_2) = [0.9,0.96) \\
 & & \mathcal{I}(u_3u_3) = [0.96,1)
\end{array}$$

We now turn our attention to mapping strings of channel letters into subintervals of the unit interval. For any non-negative integer n, we let $z^{(n)}$ denote the initial string $z^{(n)} = \{z_1, \ldots, z_n\}$ and $\mathcal{J}(z^{(n)})$ denote the subinterval corresponding to this string; as before, $z^{(0)}$ represents the null channel string. There are analogies that we can make between the important properties of a mapping from source strings into intervals and the desirable characteristics of a map from channel strings into subintervals. If we do not have any knowledge about the source sequence y, and hence the corresponding code sequence z, then x is a uniformly distributed random variable on $[0,1)$. Therefore, if we are given the initial channel string $z^{(n)}$, then the point x corresponding to source sequence y should be uniformly distributed on $\mathcal{J}(z^{(n)})$; earlier, we saw that if we are given $y^{(m)}$, then x is uniformly distributed on $\mathcal{I}(y^{(m)})$. Hence, corresponding to our earlier constraint on the mapping from source strings into intervals, we require that

- for any channel string σ, we have that $\mathcal{J}(\sigma x_1)$, $\ldots$, $\mathcal{J}(\sigma x_N)$ are disjoint intervals whose union is $\mathcal{J}(\sigma)$; our convention for the null symbol is that $\mathcal{J}(\emptyset) = [0,1)$.

For any channel string σ, we let $c(\sigma)$ and $l(\sigma)$ denote the cost of transmitting channel string

σ and the length of $\mathcal{J}(\sigma)$, respectively. Since we are trying to minimize the expected cost transmitted per source symbol, we insist that $l(\sigma)$ be monotonically decreasing with $c(\sigma)$. One consequence of this constraint is that we can make more accurate statements about the location of x, and hence about the corresponding source sequence y, as the cost of the initial code string increases; we note that we previously established that our knowledge about the whereabouts of point x improves as the self-information of the initial source string increases. The key question at this point is how to select $l(\sigma)$. It is appropriate to utilize the same digital expansion process that Krause used in his construction of relatively good prefix condition codes. He split the unit interval and all subsequent intervals into N segments with lengths in the ratio $2^{-Cc_1} : 2^{-Cc_2} : \cdots : 2^{-Cc_N}$; i.e., for any channel string σ, if $l(\sigma)$ is the length of the interval corresponding to σ, then $l(\sigma x_i) = 2^{-Cc_i} \cdot l(\sigma)$ for $i \in \{1, \ldots, N\}$. This suggests that we should impose the requirement

- for all channel strings σ, the width of interval $\mathcal{J}(\sigma)$ and the cost of transmitting σ have the relationship
$$l(\sigma) = \omega_0^{-c(\sigma)}.$$
where $\omega_0 = 2^C$.

This implies that $c(\sigma) = -\log_{\omega_0}(l(\sigma))$. Returning to our analogy with source string mapping, we remember that for all source strings u, the length of $\mathcal{I}(u)$ is $P[u]$ and that $I(u) = -\log_2(P[u])$.

We can create a mapping to satisfy these requirements that is very similar to the mapping we used for source strings. We define

$$g(x_i) = g_1(x_i) = \begin{cases} 0, & i = 1 \\ \sum_{j=1}^{i-1} l(x_j) = \sum_{j=1}^{i-1} 2^{-C \cdot c_j}, & i \in \{2, \ldots, N\} \end{cases} \tag{5.3}$$

and for $n > 1$,

$$g(z^{(n)}) = g(z^{(n-1)}) + g_1(z_n) \cdot l(z^{(n-1)}) = g(z^{(n-1)}) + g_1(z_n) \cdot 2^{-C \cdot c(z^{(n-1)})}. \tag{5.4}$$

We then let

$$\mathcal{J}(z^{(n)}) = [g(z^{(n)}),\ g(z^{(n)}) + l(z^{(n)})) = [g(z^{(n)}),\ g(z^{(n)}) + 2^{-C \cdot c(z^{(n)})}). \tag{5.5}$$

The mapping of channel sequences to points has similar lexicographic properties to the mapping of source sequences to points. More specifically, given arbitrary channel sequences z_1 and z_2, we let $x(z_1)$ and $x(z_2)$ be the points on the unit interval to which z_1 and z_2 are mapped, respectively. If n is the largest integer for which $z_1^{(n)} = z_2^{(n)}$ and the last letters of strings $z_1^{(n+1)}$ and $z_2^{(n+1)}$ are x_i and x_j, respectively, then

- $x(z_1) > x(z_2)$ if and only if $i > j$,
- for $i \in \{1, 2\}$, $x(z_i) \in \mathcal{J}(z_1^{(m)})$ and
- for $i \in \{1, 2\}$, $x(z_i) \in \mathcal{J}(z_i^{(m+1)})$; we note that $\mathcal{J}(z_1^{(m+1)})$ and $\mathcal{J}(z_2^{(m+1)})$ are disjoint.

As an example, suppose we have a channel with alphabet of size four such that $c_1 = c_2 = 3$,

$c_3 = 2$, and $c_4 = 1$. It is easy to verify that $\omega_0 = 2$ and $C = 1$. Our mapping is then

$$\begin{array}{lll}
\mathcal{J}(\emptyset) = [0,1) & \mathcal{J}(x_1) = [0,\frac{1}{8}) & \mathcal{J}(x_1x_1) = [0,\frac{1}{64}) \\
 & & \mathcal{J}(x_1x_2) = [\frac{1}{64},\frac{1}{32}) \\
 & & \mathcal{J}(x_1x_3) = [\frac{1}{32},\frac{1}{16}) \\
 & & \mathcal{J}(x_1x_4) = [\frac{1}{16},\frac{1}{8}) \\
 & & \\
 & \mathcal{J}(x_2) = [\frac{1}{8},\frac{1}{4}) & \mathcal{J}(x_2x_1) = [\frac{1}{8},\frac{9}{64}) \\
 & & \mathcal{J}(x_2x_2) = [\frac{9}{64},\frac{5}{32}) \\
 & & \mathcal{J}(x_2x_3) = [\frac{5}{32},\frac{3}{16}) \\
 & & \mathcal{J}(x_2x_4) = [\frac{3}{16},\frac{1}{4}) \\
 & & \qquad \qquad \ldots \\
 & \mathcal{J}(x_3) = [\frac{1}{4},\frac{1}{2}) & \mathcal{J}(x_3x_1) = [\frac{1}{4},\frac{9}{32}) \\
 & & \mathcal{J}(x_3x_2) = [\frac{9}{32},\frac{5}{16}) \\
 & & \mathcal{J}(x_3x_3) = [\frac{5}{16},\frac{3}{8}) \\
 & & \mathcal{J}(x_3x_4) = [\frac{3}{8},\frac{1}{2}) \\
 & & \\
 & \mathcal{J}(x_4) = [\frac{1}{2},1) & \mathcal{J}(x_4x_1) = [\frac{1}{2},\frac{9}{16}) \\
 & & \mathcal{J}(x_4x_2) = [\frac{9}{16},\frac{5}{8}) \\
 & & \mathcal{J}(x_4x_3) = [\frac{5}{8},\frac{3}{4}) \\
 & & \mathcal{J}(x_4x_4) = [\frac{3}{4},1)
\end{array}$$

We now have the tools to discuss the encoding of source sequence y. On observing $y^{(m)}$, the encoder knows that the limit point x lies in the interval $\mathcal{I}(y^{(m)})$. Thus, if $\mathcal{I}(y^{(m)})$ is contained in $\mathcal{J}(z^{(n)})$ for some channel string $z^{(n)}$, then the encoder can emit $z^{(n)}$ as the first n letters of z. Hence, as the source emits successive letters y_m, the interval $\mathcal{I}(y^{(m)})$ shrinks and more channel letters can be emitted. To illustrate this procedure, we will continue to use the source and channel that we described in our earlier examples. If the source emits u_2 as its first output, then x_4 is the first letter of the corresponding channel sequence since $\mathcal{I}(u_2)$ is contained in $\mathcal{J}(x_4)$. Similarly, if $y_1 = u_3$, then we know that the first *two* letters of z are $z^{(2)} = x_4x_4$. Finally, the information that $y_1 = u_1$ is not sufficient to to deduce *any* letters of the channel sequence since $\mathcal{J}(x_1)$, $\mathcal{J}(x_2)$, and $\mathcal{J}(x_3)$ are all contained in $\mathcal{I}(u_1)$. This simple example brings out the point that the initial source string and the initial channel string do not necessarily grow at the same pace.

To determine the efficiency of the above procedure, we are interested in how rapidly the cost of our initial channel string builds up and how rapidly the source sequence can be reconstructed from the encoded prefixes of the sequence. In particular, we would like to show that when the source has emitted $y^{(m)}$, the encoder will have issued a channel string $z^{(n)}$ with cost of transmission close to $\frac{I(y^{(m)})}{C}$ and that $z^{(n)}$ will be sufficient for the decoder to decode all but the last few letters of $y^{(m)}$. We first consider the number of letters $m(n)$ that the source must emit in order for the encoder to issue the first n channel letters. Since $P(y^{m(n)})$ is the length of $\mathcal{I}(y^{(m(n))})$ and $2^{-C\cdot[\mathit{cost\ of\ } z^{(n)}]}$ is the length of $\mathcal{J}(z^{(n)})$, the fact that $\mathcal{I}(y^{(m(n))})$ is contained in $\mathcal{J}(z^{(n)})$ implies that

$$P(y^{m(n)}) \leq 2^{-C\cdot[\mathit{cost\ of\ } z^{(n)}]} \tag{5.6}$$

Taking the logarithm of both sides of (5.6) and dividing the resulting inequality by $-C$ gives

$$\mathit{cost\ of\ } z^{(n)} \leq \frac{1}{C} I(y^{(m(n))}) \tag{5.7}$$

This says that the source must produce a certain amount of information before the channel produces a sequence of a certain cost. Since this inequality can be arbitrarily loose, we want to show that for each n, $E\left(\frac{I(y^{(m(n))})}{C} - [\mathit{cost\ of\ } z^{(n)}]\right)$ is bounded.

In order to accomplish this, let $z^{(n)}$ be fixed and let x be the final encoded point. The point x, conditional on $z^{(n)}$, is a uniformly distributed random variable in the interval $\mathcal{J}(z^{(n)})$, but we initially regard it as a fixed value. Define $D(x)$ as the distance between x and the nearest endpoint of $\mathcal{J}(z^{(n)})$. We note that the point x must be contained in $\mathcal{I}(y^{(m)})$ for all m. Also, since $m(n)$, by definition, is the smallest m for which $\mathcal{J}(z^{(n)})$ contains $\mathcal{I}(y^{(m)})$, we see that $\mathcal{I}(y^{(m(n)-1)})$ must contain one of the endpoints of $\mathcal{J}(z^{(n)})$ as well as x and thus must have width of at least $D(x)$. Hence $P(y^{(m(n)-1)} \mid x) \geq D(x)$, so

$$I(y^{(m(n)-1)} \mid x) \leq -\log_2(D(x)). \tag{5.8}$$

Now consider x as a random variable uniformly distributed over $\mathcal{J}(z^{(n)})$. $D(x)$ is then uniformly distributed between zero and half the length of $\mathcal{J}(z^{(n)})$. Using (5.8), we see that

$$E[I(y^{(m(n)-1)} \mid z^{(n)})] \leq -E[\log_2(D(x))]. \tag{5.9}$$

Since $D(x)$ is uniformly distributed, we have that

$$\begin{aligned} E[\log_2(D(x))] &= \int_{D=0}^{\frac{1}{2}\cdot 2^{-C\cdot c(z^{(n)})}} 2 \cdot 2^{C\cdot[cost\ of\ z^{(n)}]}(\log_2 D)\, dD \\ &= -C \cdot [cost\ of\ z^{(n)}] - \log_2(2e) \end{aligned} \tag{5.10}$$

Hence,

$$cost\ of\ z^{(n)} \geq \frac{1}{C}E[I(y^{(m(n)-1)} \mid z^{(n)})] - \frac{1}{C}\log_2(2e) \tag{5.11}$$

If p_{min} is the probability of the least likely source symbol, then for all $y^{(m)}$,

$$I(y^{(m)}) = I(y^{(m-1)}) + I(y_m) \leq I(y^{(m-1)}) + \log_2(\frac{1}{p_{min}}) \tag{5.12}$$

Therefore, (5.11) and (5.12) imply that

$$cost\ of\ z^{(n)} \geq \frac{1}{C}E[I(y^{(m(n))} \mid z^{(n)})] - \frac{1}{C}\log_2\left(\frac{2e}{p_{min}}\right) \tag{5.13}$$

We note that the above inequality is uniformly true for all $z^{(n)}$ and all n. (5.13) and (5.7) imply that the encoder generates cost, on the average, with the ideal of $\frac{H(U)}{C}$ per source symbol; however, we note that there is a slight deficit in the cost of each code string that is produced since the encoder is storing the most recent information about the source sequence in order to correctly emit the next few channel letters. This deficiency in cost becomes increasingly insignificant as we average over longer and longer source strings.

Next, we wish to investigate the delay between the generation of channel letters at the decoder and the generation of decoded source symbols. We select an arbitrary source sequence $y^{(m)}$ and observe the number, $n(m)$, of code letters that must be received at the decoder in order for the sequence $y^{(m)}$ to be decoded. When the decoder sees $z^{(n)}$, the decoder knows that x lies inside $\mathcal{J}(z^{(n)})$, and can thus decode $y^{(m)}$ if $\mathcal{I}(y^{(m)})$ completely contains $\mathcal{J}(z^{(n)})$. Note that $n(m)$ depends on y and z. Continuing our earlier example, we see that for $i \in \{1,2,3\}$, $\mathcal{I}(u_1) \supset \mathcal{J}(x_i)$, and so $z_1 \in \{x_1, x_2, x_3\}$ implies that $y_1 = u_1$; if $z_1 = x_4$, then we need to know a larger initial channel string to determine if $y_1 = u_2$ or $y_1 = u_3$. As before, we define $D^*(x)$ as the distance between x and the nearest endpoint of $\mathcal{I}(y^{(m)})$. Since $n(m)$ is by definition the number of channel letters required to decode $y^{(m)}$, we see that the interval $\mathcal{J}(z^{(n(m)-1)})$ cannot be contained inside $\mathcal{I}(y^{(m)})$. Hence $\mathcal{J}(z^{(n(m)-1)})$ contains one of the endpoints of $\mathcal{I}(y^{(m)})$ and also contains x. Thus the length of $\mathcal{J}(z^{(n(m)-1)})$ is at least $D^*(x)$; i.e.,

$$2^{-C \cdot [cost\ of\ z^{(n(m)-1)}]} \geq D^*(x) \tag{5.14}$$

Therefore, conditioned on x, we have

$$C \cdot [cost\ of\ z^{(n(m)-1)}] \leq -\log_2(D^*(x)) \tag{5.15}$$

For a given $y^{(m)}$, we now regard x as a random variable uniformly distributed over the interval $\mathcal{I}(y^{(m)})$. $D^*(x)$ is then uniformly distributed between 0 and $\frac{P(y^{(m)})}{2}$. Therefore,

$$E[\log_2(D^*(x))] = \int_{D^*=0}^{\frac{P(y^{(m)})}{2}} \frac{2}{P(y^{(m)})} \log_2 D^*\, dD^* = \log_2\left(\frac{P(y^{(m)})}{2e}\right) \tag{5.16}$$

Since $c_1 \geq c_2 \geq \cdots \geq c_N$, (5.15) and (5.16) imply that

$$\begin{aligned} E[cost\ of\ z^{(n(m))} \mid y^{(m)}] &\leq c_1 - \frac{1}{C}\log_2\left(\frac{P(y^{(m)})}{2e}\right) \\ &= \frac{1}{C} I(y^{(m)}) + \frac{1}{C}\log_2\left(2^{Cc_1+1}e\right) \end{aligned} \tag{5.17}$$

We now want to combine (5.13) and (5.17). Consider a given sequence $y^{(m)}$ out of the decoder, and suppose that $z^{(n(m))}$ is the required code sequence to decode $y^{(m)}$. Conditional on both $y^{(m)}$ and $z^{(n(m))}$, we see that x is uniformly distributed over $\mathcal{J}(z^{(n(m))})$, and thus the extended source sequence $y^{(m')}$ required to produce $z^{(n(m))}$ satisfies (from (5.13))

$$cost\ of\ z^{(n(m))} \geq \frac{1}{C} E[I(y^{(m')} \mid z^{(n(m))})] - \frac{1}{C}\log_2\left(\frac{2e}{p_{min}}\right) \tag{5.18}$$

Using (5.17) to take the expected value of this over $z^{(n(m))}$, we see that for any given $y^{(m)}$, the expected self-information of the extended source sequence $y^{(m')}$ required from the source to produce the $n(m)$ channel letters needed to decode $y^{(m)}$ satisfies

$$E[I(y^{(m')} \mid y^{(m)})] - I(y^{(m)}) \leq \log_2 \left(\frac{2^{Cc_1+2}e^2}{p_{min}} \right) \tag{5.19}$$

The expectation here is over the source letters $y_{m+1}, y_{m+2}, \ldots$ for the given sequence $y^{(m)}$. It is important to note that the bound does not depend on m or $y^{(m)}$. The upper bound in (5.19) states that on average there is very little delay from the encoder to the decoder. This bound is stated in terms of the additional self-information needed in additional source letters $y_{m+1}, y_{m+2}, \ldots$ until $y^{(m)}$ can be decoded. To convert this bound into a bound on the number of letters $m' - m$, let p_{max} be the maximum source letter probability. Then $\log_2 \left(\frac{1}{p_{max}}\right)$ is the minimum possible self-information per source letter and

$$E[m' - m \mid y^{(m)}] \leq \frac{\log_2 \left(\frac{2^{Cc_1+2}e^2}{p_{min}}\right)}{\log_2 \left(\frac{1}{p_{max}}\right)} \tag{5.20}$$

We now generalize the preceding analysis to handle finite-state channels. We recall the $r \times r$ matrix $\mathcal{A}(\omega) = [a_{ik}(\omega)]$ where $a_{ik}(\omega) = \sum_{x_j \in X_k(s_i)} \omega^{-c_{s_i, x_j}}$. To include $\omega = 1$, we use the convention that $1^{-\infty} = 0$. Shannon and Csiszár independently demonstrated that there exists a unique real number $\omega_0 \geq 1$ for which the greatest positive eigenvalue of $\mathcal{A}(\omega_0)$ is equal to one; furthermore, C, the capacity of our channel, is related to ω_0 by

$$C = \log_2 \omega_0 \tag{5.21}$$

We assume that both the encoder and decoder know the initial state of the channel. For any channel string σ and any $\psi \in S$, we let $c(\sigma, \psi)$, $s[\sigma, \psi]$, $\mathcal{J}_\psi(\sigma)$ and $l(\sigma, \psi)$ denote the cost of transmitting σ, the state of the channel after transmitting σ, the subinterval corresponding to σ, and the length of this subinterval, respectively, given the channel is in initial state ψ before transmission begins.

In our previous analysis for memoryless channels, we demonstrated that if for any channel string σ, we have $l(\sigma) = 2^{-C \cdot c(\sigma)}$, then the resulting arithmetic codes have many pleasing properties. Unfortunately, if $l(\sigma, \psi) = 2^{-C \cdot c(\sigma,\psi)}$ for all channel strings σ and all channel states ψ, then we do not necessarily satisfy the requirement that for any channel string σ_0, $\mathcal{J}(\sigma_0 x_1), \ldots, \mathcal{J}(\sigma_0 x_N)$ are disjoint intervals whose union is $\mathcal{J}(\sigma_0)$. However, as we will see below, we can remedy this problem by introducing a function $\alpha(\cdot,\cdot)$ which is a mapping from the Cartesian product of initial and final states to a finite set of positive real numbers; we then select our interval lengths to satisfy $l(\sigma, s_0) = \alpha(s_0, s[\sigma, s_0]) \cdot 2^{-C \cdot c(\sigma, s_0)}$ for all channel strings σ, and all initial channel states s_0. By choosing $\alpha(\cdot,\cdot)$ appropriately, we will be able to satisfy the requirement concerning the set of subintervals corresponding to the set of channel strings which differ only in the last letter. The resulting arithmetic codes will have the same desirable attributes that we found earlier for the special case of memoryless cost channels.

We let $\mathcal{A} = [a_{ik}] = \mathcal{A}(\omega_0)$. Since $\mathcal{A}$ is a non-negative irreducible matrix with largest real eigenvalue equal to one, the Frobenius theorem implies that there exists a positive vector

$$\mathbf{v} = \begin{pmatrix} v_{s_1} \\ \vdots \\ v_{s_r} \end{pmatrix}$$

for which

$$\mathbf{v} = \mathcal{A}\mathbf{v}. \tag{5.22}$$

In other words, for all $i \in \{1, \ldots, r\}$, we have

$$\sum_{k=1}^{r} a_{i,k} v_{s_k} = \sum_{k=1}^{r} \sum_{x_j \in X_k(s_i)} v_{s_k} \omega_0^{-c_{s_i,x_j}} = v_{s_i} \tag{5.23}$$

The normalization of $\mathbf{v}$ is not important since we will be using the ratios of components of $\mathbf{v}$.

We set up a mapping h from the Cartesian product of channel strings and channel states to subintervals of the unit interval as follows: for any $x_i \in X$ and $\psi \in S$, we let

$$h_1(x_i, \psi) = \begin{cases} 0, & i = 1 \\ \sum_{j=1}^{i-1} \frac{v_{s[x_j,\psi]}}{v_\psi} \omega_0^{-c_{\psi,x_j}}, & i \in \{2, \ldots, N\} \end{cases} \tag{5.24}$$

$$h(x_i) = h_1(x_i, s_0) \tag{5.25}$$

For $m > 1$, given $z^{(m)}$, we define

$$h(z^{(m)}) = h(z^{(m-1)}) + h_1(z_m, s[z^{(m-1)}, s_0])\, \omega_0^{-c(z^{(m-1)}, s_0)}. \tag{5.26}$$

Letting

$$\mathcal{J}_{s_0}(z^{(m)}) = [h(z^{(m)}),\ h(z^{(m)}) + \frac{v_{s[z^{(m)}, s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}, s_0)}\ ), \tag{5.27}$$

we see that these intervals have the nesting property: $\mathcal{J}_{s_0}(z^{(m+1)})$ is contained $\mathcal{J}_{s_0}(z^{(m)})$, where $z^{(m)}$ is the first m letters of $z^{(m+1)}$. Furthermore,

Lemma 5.1 *For any finite (and possibly empty) channel sequence $z^{(m)}$, we have that $\mathcal{J}_{s_0}(z^{(m)}x_1)$, ..., $\mathcal{J}_{s_0}(z^{(m)}x_N)$ are disjoint intervals whose union is $\mathcal{J}_{s_0}(z^{(m)})$; our convention for the null sequence is that $\mathcal{J}_{s_0}(\emptyset) = [0, 1)$.*

Because the proof requires only simple algebraic manipulations of equations (5.23) to (5.27), the details are left to the appendix.

We see that the length of the subinterval corresponding to channel string $z^{(m)}$ is

$$\frac{v_{s[z^{(m)}, s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}, s_0)} = \frac{v_{s[z^{(m)}, s_0]}}{v_{s_0}}\, 2^{-C \cdot c(z^{(m)}, s_0)}$$

using equation (5.21). Hence, the function $\alpha(\cdot, \cdot)$ we mentioned earlier is $\alpha(s_0, s_f) = \frac{v_{s_f}}{v_{s_0}}$.

The encoding of source sequence y follows the same procedure we used earlier for the special case of a memoryless channel; namely, if $\mathcal{I}(y^{(m)})$ is contained in $\mathcal{J}_{s_0}(z^{(n)})$ for some channel string $z^{(n)}$, then the encoder can emit $z^{(n)}$ as the first n letters of z. We extend the same techniques and notation we used previously to analyze the performance of this scheme. Since the length of $\mathcal{J}_{s_0}(z^{(n)})$ is $\frac{v_{s[z^{(m)}, s_0]}}{v_{s_0}}\, 2^{-C \cdot c(z^{(m)}, s_0)}$, we revise (5.6) and (5.7) to:

$$P(y^{m(n)}) \leq \frac{v_{s[z^{(n)}, s_0]}}{v_{s_0}}\, 2^{-C \cdot c(z^{(n)}, s_0)} \tag{5.28}$$

$$c(z^{(n)}, s_0) \leq \frac{1}{C} I(y^{(m(n))}) + \frac{1}{C} \log_2 \left(\frac{v_{s[z^{(n)}, s_0]}}{v_{s_0}} \right) \tag{5.29}$$

(5.8) and (5.9) remain valid. Modifying equation (5.10) to use the length of $\mathcal{J}_{s_0}(z^{(n)})$, we see that

$$E[\log_2(D(x))] = -C \cdot c(z^{(n)}, s_0) - \log_2(2e \frac{v_{s_0}}{v_{s[z^{(n)}, s_0]}}) \tag{5.30}$$

and so

$$c(z^{(n)}, s_0) \geq \frac{1}{C}E[I(y^{(m(n)-1)} \mid z^{(n)})] - \frac{1}{C}\log_2(2e\frac{v_{s_0}}{v_{s[z^{(n)},s_0]}}). \tag{5.31}$$

(5.31) and (5.12) imply that

$$\begin{aligned} c(z^{(n)}, s_0) &\geq \frac{1}{C}E[I(y^{(m(n))} \mid z^{(n)})] - \frac{1}{C}\log_2\left(\frac{2ev_{s_0}}{p_{min}v_{s[z^{(n)},s_0]}}\right) \\ &\geq \frac{1}{C}E[I(y^{(m(n))} \mid z^{(n)})] - \frac{1}{C}\log_2\left(\frac{2ev^*}{p_{min}}\right) \end{aligned} \tag{5.32}$$

where

$$v^* = \max_{i,j\in\{1,\ldots,r\}} \frac{v_{s_i}}{v_{s_j}}. \tag{5.33}$$

From (5.32) and (5.29), we have that the encoder generates cost, on the average, with the ideal of $\frac{H(U)}{C}$ per source symbol; as with the special case of memoryless cost channels, we note that there is a slight deficit in the cost of each code string that is produced and that this deficiency becomes increasingly insignificant as we average over longer and longer source strings.

To analyze the delay between the generation of channel letters at the decoder and the generation of decoded source symbols, we exploit the same ideas and notation that we used earlier in studying memoryless cost channels. Using $\mathcal{J}_{s_0}(z^{(n(m)-1)})$, we change (5.14) and (5.15) to

$$\frac{v_{s[z^{(n(m)-1)},s_0]}}{v_{s_0}} \cdot 2^{-C\cdot c(z^{(n(m)-1)},s_0)} \geq D^*(x) \tag{5.34}$$

$$C\cdot c(z^{(n(m)-1)}, s_0) \leq -\log_2(D^*(x)) + \log_2\left(\frac{v_{s[z^{(n(m)-1)},s_0]}}{v_{s_0}}\right) \tag{5.35}$$

Equation (5.16) requires no revision. Letting $c_{max} = \max_{c_{s,x}<\infty} c_{s,x}$, (5.35) and (5.16) imply that

$$\begin{aligned} E[c(z^{(n(m))}, s_0) \mid y^{(m)}] &\leq c_{max} - \frac{1}{C}\log_2\left(\frac{P(y^{(m)})}{2e}\right) + \frac{1}{C}\log_2\left(\frac{v_{s[z^{(n(m)-1)},s_0]}}{v_{s_0}}\right) \\ &= \frac{1}{C}I(y^{(m)}) + \frac{1}{C}\log_2\left(\frac{v_{s[z^{(n(m)-1)},s_0]}}{v_{s_0}}\,2^{Cc_{max}+1}e\right) \end{aligned} \tag{5.36}$$

Combining (5.36) and (5.32), we find that the extended source sequence $y^{(m')}$ required to produce $z^{(n(m))}$ satisfies (from (5.32))

$$c(z^{(n(m))}, s_0) \geq \frac{1}{C} E[I(y^{(m')} \mid z^{(n(m))})] - \frac{1}{C} \log_2 \left(\frac{2ev^*}{p_{min}} \right) \tag{5.37}$$

Taking the expected value of both sides of (5.37) over $z^{(n(m))}$, we find that

$$E[I(y^{(m')} \mid y^{(m)})] - I(y^{(m)}) \leq \log_2 \left(\frac{2^{Cc_{max}+2} e^2 v^*}{p_{min}} \right) \tag{5.38}$$

Therefore, a bound on the number of letters $m' - m$ is

$$E[m' - m \mid y^{(m)}] \leq \frac{\log_2 \left(\frac{2^{Cc_{max}+2} e^2 v^*}{p_{min}} \right)}{\log_2 \left(\frac{1}{p_{max}} \right)} \tag{5.39}$$

In actual implementation, it is not possible to calculate the intervals used in encoding and decoding exactly. We view the arithmetic as being performed using binary fixed point arithmetic with M binary digits of accuracy. In order to mitigate the effects of round-off, we will use a two-part arithmetic coder. The first arithmetic coder will map source sequences into sequences from a binary channel with alphabet $\{0, 1\}$; this binary channel has memoryless digit costs and each digit is assumed to have a unit cost of transmission. We let x_b represent the point on the unit interval corresponding to the source sequence y and $b = \{b_1, b_2, \ldots\}$ be the corresponding binary sequence. The capacity of this binary channel is easily seen to be equal to one; therefore, our earlier results show that over the long term, the average number of binary digits per source symbol is $H(U)$. Furthermore, since the mapping from source sequences to points on the unit interval is done so that the random variable x_b is uniformly distributed on the real line, each of the digits $b_1, b_2, \ldots$ in the binary expansion of x_b is independent and equiprobably equal to 0 or 1. Our second arithmetic coder will map the binary sequence b into a sequence of letters from the original channel alphabel. Since $b_1, b_2, \ldots$ are independent and equiprobably equal to 0 or 1, the entropy of the incoming binary sequence is 1. As before, the capacity of the channel is C. Hence, our earlier conclusions indicate that the second encoder generates cost, on the average, with the ideal of $\frac{1}{C}$ per binary digit. Combining these averages, we see that over a large source sequence, this double encoding procedure generates cost, on the average, with the ideal of $\frac{H(U)}{C}$ per binary digit. Therefore, in theory, we do not lose anything by splitting the coder into these two parts.

We will first discuss the behavior of the arithmetic coder which maps source sequences into sequences of binary digits. We view the arithmetic as being performed using binary fixed point arithmetic with M binary digits of accuracy. There is some flexibility in how numbers are rounded to M bits, but it is vital that the encoder and decoder use exactly the same rule and the rounding is done at the appropriate time. It is also essential, since $P(y^{(m)})$ is approaching 0 with increasing m, that the intervals be renormalized as binary digits are emitted.

The encoder keeps in its memory a normalized interval starting at $f_{norm}(y^{(m)})$ and of width $p_{norm}(y^{(m)})$. Initially, $m = 0$, $p_{norm}(\emptyset) = 1$, $f_{norm}(\emptyset) = 0$. In order to ensure that the intervals corresponding to different m tuples $y^{(m)}$ are disjoint, the interval end points are calculated directly and $p_{norm}(y^{(m)})$ is taken as the length of the resulting interval. For $y^{(m)} \neq u_K u_K \ldots u_K$, we let $L_m(y^{(m)})$ be the m tuple of source symbols that is lexicographically next larger than $y^{(m)}$. If $y^{(m)} = u_K u_K \ldots u_K$, we define $L_m(y^{(m)}) = \Lambda$. We use the convention that $f_{norm}(\Lambda) = 1$. The encoder employs the following algorithm.

1. Accept y_{m+1} into the encoder

2. Calculate the new interval as follows:

$$f_{norm}(y^{(m+1)}) = f_{norm}(y^{(m)}) + f_1(y_{m+1})p_{norm}(y^{(m)}) \tag{5.40}$$

$$v^{(m+1)} = L_{m+1}(y^{(m+1)}) = \{v_1, \ldots, v_{m+1}\} \tag{5.41}$$

If $v^{(m+1)} \neq \Lambda$, then

$$f_{norm}(v^{(m+1)}) = f_{norm}(v^{(m)}) + f_1(v_{m+1})p_{norm}(v^{(m)}) \tag{5.42}$$

For all $v^{(m+1)}$,

$$p_{norm}(y^{(m+1)}) = f_{norm}(v^{(m+1)}) - f_{norm}(y^{(m+1)}) \tag{5.43}$$

3. Produce binary outputs and renormalize according to the rule:
If $f_{norm}(y^{(m+1)}) \geq \frac{1}{2}$, then

   (a) emit 1 as an output

(b) renormalize by

$$f_{norm}(y^{(m+1)}) = 2 \cdot f_{norm}(y^{(m+1)}) - 1$$
$$p_{norm}(y^{(m+1)}) = 2 \cdot p_{norm}(y^{(m+1)})$$

(c) Goto step 3.

Else if $f_{norm}(y^{(m+1)}) + p_{norm}(y^{(m+1)}) \leq \frac{1}{2}$, then

(a) emit 0 as an output

(b) renormalize by

$$f_{norm}(y^{(m+1)}) = 2 \cdot f_{norm}(y^{(m+1)})$$
$$p_{norm}(y^{(m+1)}) = 2 \cdot p_{norm}(y^{(m+1)})$$

(c) Goto step 3.

4. Increment m and goto step 1.

The purpose of step 3 is to eliminate the more significant binary digits that are no longer needed in the encoding and decoding and adding less significant digits that increase the precision as the intervals shrink. Note that renormalization is achieved with no additional round-off errors.

To obtain insights into the effect of the round-off errors, we consider the example of a ternary equiprobable source. First we examine the behavior of the encoder when the input consists of a long string of repetitions of the symbol u_2. Without roundoff errors, $\mathcal{I}(y^{(1)}) = [\frac{1}{3}, \frac{2}{3})$, $\mathcal{I}(y^{(2)}) = [\frac{4}{9}, \frac{5}{9})$, and in general, $\mathcal{I}(y^{(m)}) = [\frac{1-3^{-m}}{2}, \frac{1+3^{m}}{2})$. Thus, for this string, $\mathcal{I}(y^{(m)})$ continues to straddle the point $\frac{1}{2}$ and no binary digits are emitted by the encoder. Because arithmetic is performed with only M binary digits of accuracy, the left and right ends of these intervals must each be multiples of 2^{-M} and also must get close to $\frac{1}{2}$. For example, if the rounded off version of $\mathcal{I}(y^{(m)})$ is $[\frac{1}{2} - 2^{-M}, \frac{1}{2} + 2^{-M})$, then no binary digit can be emitted, and it is impossible to split the interval into three distinct intervals to account for all possibilities of y_{m+1}. We will avoid this problem by rounding off to avoid such small intervals around the point $\frac{1}{2}$. In particular, we modify (5.43) to

$$p_{norm} = \begin{cases} \frac{1}{2} - f_{norm}(y^{(m+1)}) \,, & \text{if } f_{norm}(y_{m+1}) < \frac{1}{2}, \\ & f_{norm}\{L_{m+1}(y^{(m+1)})\} > \frac{1}{2}, \text{ and} \\ & f_{norm}\{L_{m+1}(y^{(m+1)})\} - f_{norm}(y^{(m+1)}) \\ & < \frac{2^{-M}}{p_{min}} \\ f_{norm}\{L_{m+1}(y^{(m+1)})\} - f_{norm}(y^{(m+1)}) \,, & \text{otherwise.} \end{cases} \tag{5.44}$$

The first part of (5.44) causes the right endpoint of $\mathcal{I}(y^{(m+1)})$ to be at $\frac{1}{2}$, which allows a binary digit to be emitted. We choose $\frac{2^{-M}}{p_{min}}$ to be the largest interval size that can straddle to point $\frac{1}{2}$ to ensure that the next source symbol to enter the encoder will receive a non-zero interval size without any unusual round-off rules.

Besides (5.44), we impose a few other reasonable restrictions pertaining to the way numbers are rounded off. First, we insist on preserving the nesting property; i.e., before renormalization, $f_{norm}(y_{m+1}) \geq f_{norm}(y_m)$ and $f_{norm}\{L_{m+1}(y^{(m+1)})\} \leq f_{norm}\{L_{m+1}(y^{(m)})\}$. Also, before renormalization, the intervals corresponding to different values of y_{m+1} must be disjoint and non-empty. These rules make sure that no two distinct source sequences give rise to the same binary sequence.

We next consider the decoder. The decoder decodes one channel letter at a time and maintains both a queue of incoming channel letters and a replica of the encoder. Initially, $m = 1$ and the queue is empty. The decoder, in attempting to decode y_m, uses (5.40) to (5.42) and (5.44) to calculate $f_{norm}(y^{(m)})$ and $p_{norm}(y^{(m)})$ for all choices of y_m given $y^{(m-1)}$. As new binary digits enter the queue, we can consider the queued letters as a normalized binary fraction of j significant bits, where j is the queue length. When the interval corresponding to this fraction lies within one of the K normalized intervals calculated above, the decoder decodes y_m, renormalizes f_{norm} and p_{norm} by the encoder rules, and deletes the corresponding binary digits from the front of the queue. It then increments m and repeats the above procedure.

We note that when y_m enters the encoder, the interval end points are calculated to M binary digits of accuracy. Therefore, after the encoder emits M binary digits, the resulting interval must have size 2^{-M} and thus y_m is decodable at this point, if not before. Hence, decoding always occurs with at most M binary digits in the queue. Therefore, by increasing M, we trade off smaller delays between encoding and decoding for additional efficiency. Also, for some improbable set of source sequences, the encoder will produce binary digits considerably after the corresponding source string contains enough self-information, although it is always true that $I(y^{(m(n))}) - n \leq M$. This increases the round-off error and causes the binary digits to be not exactly equiprobable. However, we will still model the input into the second arithmetic coder as a sequence of independent binary digits with each bit equiprobably equal to 0 or 1.

We now turn to the arithmetic coder which maps binary digits into channel letters. As before, we assume that the arithmetic is being performed using binary fixed point arithmetic with M binary digits of accuracy. As with the first coder, we are allowed some leeway in how numbers are rounded to M bits, but it is essential that the encoder and decoder use the same rule and rounding is done at the appropriate time. It is also necessary that the intervals be renormalized as channel letters are emitted.

The encoder operates as follows: based on previous channel outputs, the encoder knows that the point x corresponding to source sequence y lies within a certain subinterval of the unit interval. The first step is to partition this subinterval into N sub-subintervals corresponding to the N possibilities for the next channel output. The encoder sees binary digits coming in one at a time and each string of binary digits of length l corresponds to a subinterval of the unit interval of length 2^{-l}. As each binary digit enters the encoder, an interval corresponding to the current binary string is calculated. If the interval is contained in one of the N channel letter subintervals mentioned above, we emit the corresponding channel letter, eliminate the binary digits that will not give us any further information about the point x from the front of the binary string and renormalize the interval accordingly. Otherwise, we read in another binary digit. Note that after M binary digits are held by the encoder, the resulting interval must have size 2^{-M} and thus the next channel letter is emitted at this point, if not earlier.

More specifically, the encoder keeps track of a normalized interval starting at h_{norm} and of width l_{norm}. Initially, $n = 0$, $h_{norm} = 0$, $l_{norm} = 1$ and the channel string $z^{(n)}$ is the null string. We also set the counters l and m to zero and ψ, the state of the channel, to s_0. l and m represent the length of the binary string stored in the encoder and the total number of binary digits that have been read in by the encoder, respectively. We let $\beta^{(l)}$ be the binary string stored by the encoder. Hence, $\beta^{(l)} = \{\beta_1, \ldots, \beta_l\} = \{b_{m-l+1}, \ldots, b_m\}$. In order to ensure that the intervals corresponding to different n tuples $z^{(n)}$ are disjoint, the interval endpoints are calculated directly and l_{norm} is taken as the difference between the beginning and the end of the interval. For $b^{(m)} \neq 1, 1, \ldots, 1$, we let $L_m^*(b^{(m)})$ be the dyadic rational corresponding to the m tuple of binary digits that is lexicographically next larger than $b^{(m)}$. To include $b^{(m)} = 1, 1, \ldots, 1$, we use the convention that $L_m^*(1, 1, \ldots, 1) = 1$. The encoder operates in the following way:

1. Calculate the new intervals by the equations

$$\begin{aligned} h_{norm}(x_i) &= h_{norm} + h_1(x_i, \psi) \cdot l_{norm} \\ l_{norm}(x_i) &= \begin{cases} h_{norm}(x_{i+1}) - h_{norm}(x_i), & i \in \{1, \ldots, N-1\} \\ h_{norm} + l_{norm} - h_{norm}(x_N), & i = N \end{cases} \\ \mathcal{J}_\psi^*(x_i) &= [h_{norm}(x_i), h_{norm}(x_i) + l_{norm}(x_i)) \end{aligned}$$

2. Accept b_{m+1} into the encoder; $\beta_{l+1} = b_{m+1}$

3. Let $\tilde{\beta}^{(l+1)} = \sum_{i=1}^{l+1} \beta_i 2^{-i}$.
   If $[\tilde{\beta}^{(l+1)}, L_{l+1}^*(\tilde{\beta}^{(l+1)})) \subset \mathcal{J}_\psi^*(x_i)$ for some $x_i \in X$, then

   (a) $z_{n+1} = x_i$

   (b) find the largest integer μ such that there exists a dyadic rational q with μ significant bits for which

   $$\mathcal{J}_\psi^*(x_i) \subset [q, q + 2^{-\mu}).$$

   Possibly, $\mu = 0$.

   (c) Multiply $h_{norm}(x_i)$ and $l_{norm}(x_i)$ by 2^μ and keep the fractional part of each. These are the new values of h_{norm} and l_{norm}, respectively.

   (d) For $i = 1, \ldots, l - \mu$, set $\beta_i = \beta_{i+\mu}$.

   (e) $l = l - \mu$

   (f) Increment l, m , and n.

   (g) $\psi = s[x_i, \psi]$

   (h) Goto step 1.

Else goto step 2.

Next consider the decoder. We can visualize the decoder as decoding one binary digit at a time and maintaining both a queue of incoming channel letters and a copy of the encoder. Initially, $m = 1$ and the queue is empty. The decoder, in attempting to decode b_m, bisects the unit interval. As new channel letters enter the queue, we can consider the queued letters as a normalized interval corresponding to the string of channel letters held by the decoder. When this interval lies within one of the bisected portions of the unit interval, the decoder decodes b_m, renormalizes h_{norm} and l_{norm} by the encoder rules, updates the state and deletes the corresponding channel letters from the front of the queue. It then increments m and repeats the above procedure.

As with the encoder for the first arithmetic coder, we may find that the intervals corresponding to the strings of letters held by the decoder are straddling the point $\frac{1}{2}$, so that no binary digits can be emitted. In this case, when the size of the current interval is sufficiently small, we perform a linear transformation on the interval so that we can create all subintervals corresponding to the next possible channel letter. It is important to keep track of where the linear transformation maps the point $\frac{1}{2}$.

There are no new complications in studying sources with memory. The encoder uses $P(y_m \mid y_{m-1} \cdots y_1)$ in place of $p(y_m)$. The replica of the encoder at the decoder makes this substitution as well. The encoder can also be adaptive in that the probability assignment for y_{m+1} is based on the observed sequence $y^{(m)}$; again, the encoder's replica at the decoder would function in the same way.

Chapter 6

Conclusion

We have investigated four source coding techniques in this thesis. We first considered Varn coding and showed that the construction of optimal single letter exhaustive prefix condition codes when the source ensemble consists of equiprobable symbols is equivalent to Tunstall's algorithm for finding optimal valid dictionaries when the channel letters are assumed to have the same cost. We subsequently obtained some upper bounds on the expected cost of codes produced by Varn's algorithm; we demonstrated that our bound is always tighter than Krause's bound whenever we have a binary memoryless cost channel.

We next developed a heuristic to construct reasonably good single letter prefix condition codes when our channel is binary with memoryless letter costs. We proved that the algorithm produces optimal codes in two important special cases. The first of these is when the letters have the same cost of transmission, in which case the heuristic reduces to Huffman's algorithm. The other instance occurs when there exists a prefix condition encoding of the source ensemble satisfying $\overline{d} = \frac{H(U)}{C}$. We later noticed that we can construct examples for which the heuristic produces codes with expected cost per source symbol arbitrarily larger than that of the corresponding optimal prefix condition code.

In the following chapter, we transformed the problem of constructing optimal single letter prefix condition codes into an integer bilinear programming problem. We found that our formulation could be used to produce optimal prefix condition codes when the channel has memoryless letter costs or when there are restrictions on individual code words. We stated some extra linear constraints that could be used to find better solutions when we remove the integer constraints.

Finally, we extended the ideas of arithmetic coding to operate on channels with finite-state letter costs. We extablished that these codes have asymptotically optimal behavior and are efficient. Furthermore, it was noted that this encoding technique could be applied to sources with memory. We concluded the chapter by considering some implementation issues.

A number of open issues remain. The first is to find an efficient algorithm to generate optimal single-letter prefix condition codes given arbitrary memoryless cost channels and source ensembles. To gain further insights about this problem, it may be useful to look into the number of intermediate nodes in an optimal code tree; we saw the importance of this question in the chapter on Varn codes. It seems very likely that the current adaptive coding schemes for sources with unknown statistics can be easily extended to handle channels that are more general than those with equal letter costs. Lastly, we note that in our arithmetic coding scheme, if a single channel letter is incorrectly transmitted, it would almost surely be impossible to correctly decode the bulk of the remaining source sequence. All of the known adaptive coding techniques share this problem. Hence, it would be of interest to find good encoding techniques that are robust in the presence of a noisy channel.

Appendix

Proof of Lemma 5.1: From (5.24) and (5.26), it is apparent that for all $s_0 \in S$, we have that all (non-empty) $\mathcal{J}_{s_0}(x_i)$ are disjoint and that

$$\bigcup_{x_i \in X} \mathcal{J}_{s_0}(x_i) = [0, \sum_{j=1}^{N} \frac{v_{s[x_j,s_0]}}{v_{s_0}} \omega_0^{-c_{s_0,x_j}}).$$

Using (5.23), we have that

$$\sum_{j=1}^{N} \frac{v_{s[x_j,s_0]}}{v_{s_0}} \omega_0^{-c_{s_0,x_j}} = \sum_{k=1}^{r} \sum_{x_j \in X_k(s_0)} \frac{v_{s_k}}{v_{s_0}} \omega_0^{-c_{s_0,x_j}} = 1.$$

Hence, $\bigcup_{x_i \in X} \mathcal{J}_{s_0}(x_i) = [0,1)$.

For $m \geq 1$, if $z_{m+1} = x_1$, then $h(z^{(m+1)}) = h(z^{(m)})$. We would like to show that for $i = 1, \ldots, N-1$,

$$h(z^{(m)}x_{i+1}) = h(z^{(m)}x_i) + \frac{v_{s[z^{(m)}x_i,s_0]}}{v_{s_0}} \omega_0^{-c(z^{(m)}x_i,s_0)} .$$

We have that

$$\begin{aligned}
&h(z^{(m)}x_i) + \frac{v_{s[z^{(m)}x_i,s_0]}}{v_{s_0}} \omega_0^{-c(z^{(m)}x_i,s_0)} \\
&= h(z^{(m)}) + h_1(x_i, s[z^{(m)}, s_0]) \, \omega_0^{-c(z^{(m)},s_0)} + \frac{v_{s[z^{(m)}x_i,s_0]}}{v_{s_0}} \omega_0^{-c(z^{(m)}x_i,s_0)} \\
&= h(z^{(m)}) + \left(h_1(x_{i+1}, s[z^{(m)}, s_0]) - \frac{v_{s[z^{(m)}x_i,s_0]}}{v_{s_0}} \omega_0^{-c_{s[z^{(m)},s_0],x_i}}\right) \omega_0^{-c(z^{(m)},s_0)} \\
&+ \frac{v_{s[z^{(m)}x_i,s_0]}}{v_{s_0}} \omega_0^{-c(z^{(m)}x_i,s_0)} \\
&= h(z^{(m)}x_{i+1})
\end{aligned}$$

Hence, the collection of intervals $\mathcal{J}_{s_0}(z^{(m)}x_i)$ are disjoint and

$$\bigcup_{z_{m+1}} \mathcal{J}_{s_0}(z^{(m+1)}) \;=\; [h(z^{(m)}) \;,\; h(z^{(m)}x_N) + \tfrac{v_{s[z^{(m)}x_N,s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}x_N,s_0)} \;)$$

where

$$\begin{aligned}
& h(z^{(m)}x_N) + \tfrac{v_{s[z^{(m)}x_N,s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}x_N,s_0)} \\
&\quad = h(z^{(m)}) \;+ h_1(x_N, s[z^{(m)}, s_0])\, \omega_0^{-c(z^{(m)},s_0)} \;+ \tfrac{v_{s[z^{(m)}x_N,s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}x_N,s_0)} \\
&\quad = h(z^{(m)}) \;+ \left(\sum_{j=1}^{N-1} \frac{v_{s[z^{(m)}x_j,s_0]}}{v_{s_0}}\, \omega_0^{-c_{s[z^{(m)},s_0],x_j}} \right) \omega_0^{-c(z^{(m)},s_0)} \;+ \tfrac{v_{s[z^{(m)}x_N,s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)}x_N,s_0)} \\
&\quad = h(z^{(m)}) \;+ \left(\sum_{j=1}^{N} \frac{v_{s[z^{(m)}x_j,s_0]}}{v_{s_0}}\, \omega_0^{-c_{s[z^{(m)},s_0],x_j}} \right) \omega_0^{-c(z^{(m)},s_0)} \\
&\quad = h(z^{(m)}) \;+ \left(\frac{\sum_{k=1}^{r} \sum_{x_j \in X_k(s[z^{(m)},s_0])} v_{s_k}\, \omega_0^{-c_{s[z^{(m)},s_0],x_j}}}{v_{s_0}} \right) \omega_0^{-c(z^{(m)},s_0)} \\
&\quad = h(z^{(m)}) \;+ \frac{v_{s[z^{(m)},s_0]}}{v_{s_0}}\, \omega_0^{-c(z^{(m)},s_0)},
\end{aligned}$$

using equation (5.23). Therefore,

$$\bigcup_{z_{m+1}} \mathcal{J}_{s_0}(z^{(m+1)}) \;=\; \mathcal{J}_{s_0}(z^{(m)}) \;. \;\Box$$

Bibliography

Carter, L. and J. Gill (1974) "Conjectures on uniquely decipherable codes," *I.E.E.E. Trans. Inform. Theory* IT-20, 394-396

Csiszár, I. (1969) "Simple proofs of some theorems on noiseless channels," *Inform. Control* 14, 285-298

Feller, W. (1950) *An Introduction to Probability Theory and its Applications*, Vol. 1, Wiley, New York (3rd ed., 1968)

Gallager, R. G. (1968) *Information Theory and Reliable Communication*, John Wiley & Sons, Inc. New York

Gallager, R. G. (1990) Class Notes for 6.441

Huffman, D. A. (1952) "A method for the construction of minimum-redundancy codes," *Proc. I.R.E.* 40, 1098-1101

Karp, R. S. (1961) "Minimum-redundancy coding for the discrete noiseless channel," *I.R.E. Trans. Inform. Theory* IT-7, 27-38

Kraft, L. G. (1949) "A device for quantizing, grouping, and coding amplitude modulated pulses," M.S. thesis, Dept. of E.E., M.I.T., Cambridge, MA

Krause, R. M. (1962) "Channels which transmit letters of unequal duration," *Inform. Control* 5, 13-24

Lempel, A., S. Even and M. Cohn (1973) "An algorithm for optimal prefix parsing of a noiseless and memoryless channel," *I.E.E.E. Trans. Inform. Theory* IT-19, 208-214

McMillan, B. (1956) "Two inequalities implied by unique decipherability," *I.R.E. Trans. Inform. Theory* IT-2, 115-116

Mehlhorn, K. (1980) "An efficient algorithm for constructing nearly optimal prefix codes," *I.E.E.E. Trans. Inform. Theory* IT-26, 513-517

Rissanen, J. and G. G. Langdon, Jr. (1979) "Arithmetic coding," *I.B.M. J. Res. Develop.* 23, 149-162

Shannon, C. E. (1948) "A mathematical theory of communication," *Bell System Tech. J.* 27, 379-423, 623-656

Siegal, P. H. (1985) "Recording codes for magnetic storage," *I.E.E.E. Trans. Magnetics* MAG-21, 1344-1349

Tunstall, B. P. (1968) "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA

Varn, B. F. (1971) "Optimal variable length codes (arbitrary symbol cost and equal code word probabilities)," *Inform. Control* 19, 289-301

Ziv, J. and A. Lempel (1978) "Compression of individual sequences via variable-rate coding," *I.E.E.E. Trans. Inform. Theory* IT-24, 530-536