Fall 12-2009

Entropy and Certainty in Lossless Data Compression

James Jay Jacobs
University of Southern Mississippi

The University of Southern Mississippi

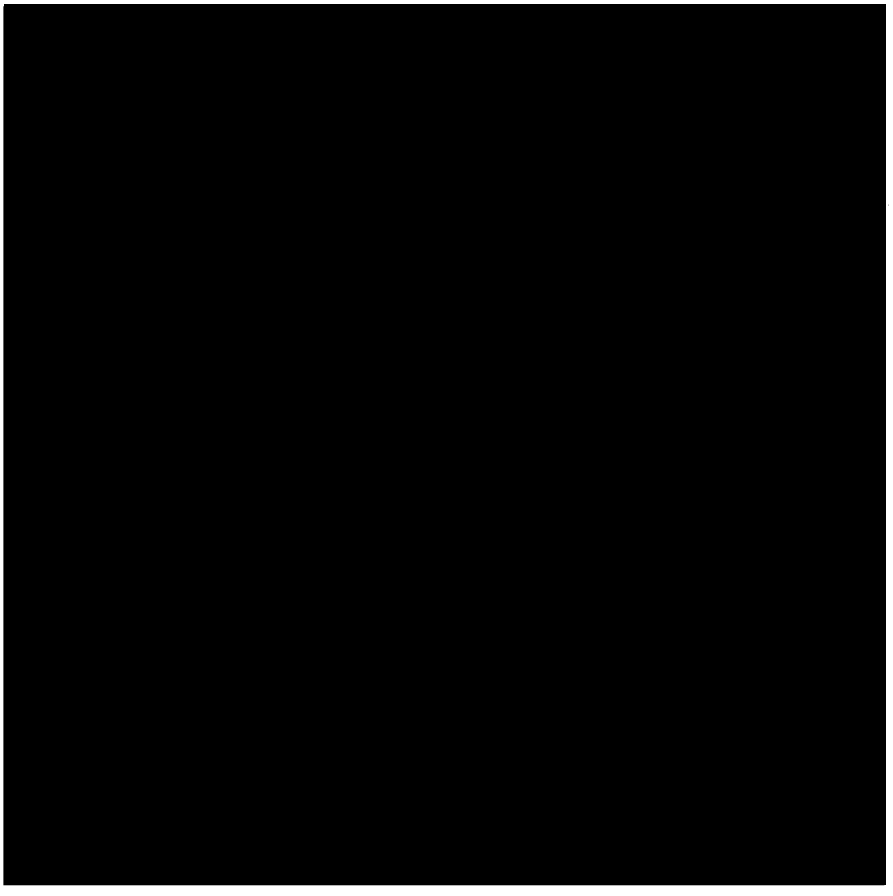
ENTROPY AND CERTAINTY IN LOSSLESS DATA COMPRESSION

by

James Jay Jacobs

A Dissertation
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved:



December 2009

The University of Southern Mississippi

ENTROPY AND CERTAINTY IN LOSSLESS DATA COMPRESSION

by

James Jay Jacobs

Abstract of a Dissertation
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

December 2009

ABSTRACT

ENTROPY AND CERTAINTY IN LOSSLESS DATA COMPRESSION

by James Jay Jacobs

December 2009

Data compression is the art of using encoding techniques to represent data symbols using less storage space compared to the original data representation. The encoding process builds a relationship between the entropy of the data and the certainty of the system. The theoretical limits of this relationship are defined by the theory of entropy in information that was proposed by Claude Shannon. Lossless data compression is uniquely tied to entropy theory as the data and the system have a static definition. The static nature of the two requires a mechanism to reduce the entropy without the ability to alter either of these key components. This dissertation develops the Map of Certainty and Entropy (MaCE) in order to illustrate the entropy and certainty contained within an information system and uses this concept to generate the proposed methods for prefix-free, lossless compression of static data. The first method, Select Level Method (SLM), increases the efficiency of creating Shannon-Fano-Elias code in terms of CPU cycles. SLM is developed using a sideways view of the compression environment provided by MaCE. This view is also used for the second contribution, Sort Linear Method Nivellate (SLMN) which uses the concepts of SLM with the addition of midpoints and a fitting function to increase the compression efficiency of SLM to entropy values $L(x) \leq H(x)+1$. Finally, the third contribution, Jacobs, Ali, Kolibal Encoding (JAKE), extends SLM and SLMN to bases larger than binary to increase the compression even further while maintaining the same relative computation efficiency.

ACKNOWLEDGMENTS

This work is dedicated to my family, especially my wife, Kelli, and my daughters, Jade and Amber. Their continual encouragement throughout this process made everything possible. I would like to take this opportunity to thank all of those who have assisted me in this effort, especially all the members of the committee for their constant support and encouragement. In particular, my advisor, Dr. Ali who introduced the topic of data compression and whose consistently supported the project in good times and bad. Dr. Kolibal's continual helpfulness and brilliance in data compression and mathematics. Dr. Pandey's sharing of his insight into theory of entropy in Physics. Dr. Burgess for his stochastic and Markovian processes as this insight allowed for a thorough breakdown of the interactions within the data compression environment. Dr. Seyfarth's introduction to computational algorithms as these concepts laid the framework for this dissertation and the future work related to it. Dr. Zhang for his insight on parallel processing. The general guidance of Dr. El-Sawi throughout the process. I also want to thank those in the School of Computing at USM and the office personnel, their devotion to the students and the school is a model that all should follow. Semper Fidelis.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

SFE	-	Shannon-Fano-Elias
SF	-	Shannon-Fano
PMF	-	Probability Mass Function
MaCE	-	Map of Certainty and Entropy
SLM	-	Select Level Method
SLMN	-	Sort Linear Method Nivellate
MSPM	-	Midpoint Spatial Mapping
JAKE	-	Jacobs, Ali, Kolibal Encoding

NOTATION AND GLOSSARY

General Usage and Terminology

The notation used in this text represents fairly standard mathematical and computational usage. In many cases these fields tend to use different preferred notation to indicate the same concept, and these have been reconciled to the extent possible, given the interdisciplinary nature of the material.

General Definitions

A code definition refers to the requirements defining the code. A code word refers to the unique string of bits within a given range defined by the code definition used to represent a symbol. A symbol refers to the entities being assigned the code word which is also referred to as encoding. Encoding refers to the process of assigning a symbol to a code word. Decoding refers to the process of retrieving a symbol from the encoding table by utilizing the code word.

General Notations

Entropy is denoted in multiple fields by S and a specialization of S denoted by H is used for the purpose of information entropy, also known as Shannon entropy. Entropy for an individual symbol is denoted by H_i for datum i and the probability of occurrence for an individual symbol i is denoted by P_i. Information or surprisal is denoted by $\mathbb{I}$. The length of a code or the level within the space is denoted by L and the width of the space is denoted by W. The level L representing equilibrium is denoted by L_e. Certainty of an event is represented by C. The probability of certainty P_c denotes the termination location of complete code word representing the symbol.

Acronyms

MaCE (Map of Certainty and Entropy), developed in Chapter 4, is a depiction used to illustrate the spatial relationships between entropy H, certainty C and the symbol. SLM (Select Level Method), developed in Sec. 5.2, is the first compression method proposed which uses the concepts from MaCE to select the level required by the symbol to recreate Shannon-Fano-Elias encoding. MSPM (Midpoint Spatial Mapping) uses the spatial concepts illustrated in MaCE to define the midpoints between the levels in the entropy and certainty space. SLMN (Sort Linear Method Nivellate), developed in Sec. 5.3, is the second proposed method which uses the concepts from SLM and MSPM to decrease the final entropy $L(x)$ to $L(x) \leq H(x) + 1$. JAKE (Jacobs, Ali, Kolibal Encoding) extends SLM and SLMN into other numerical bases to decrease the entropy below $H(x) + 1$.

Chapter 1

INTRODUCTION

1.1 Introduction to Data Compression

Data compression is the art of using encoding techniques to represent data symbols using less storage space compared to the space required for the original data representation. Various techniques have been developed to address this problem over the years and many have survived the test of time. The longevity of these techniques has led to many enhancements and specializations to accomplish the task. However, the ever growing need to store, transmit and retrieve data in a timely manner while preserving communication and storage resources has rekindled the demand for even more efficient data compression methods.

Data compression is an art form with the purpose of representing a symbol or series of symbols with fewer symbols as compared to the original data representation. An early example of the art form is in the development of Morse code [39]. Morse code was invented in 1838 and is designed to represent letters efficiently by assigning shorter code words to the most frequent utilized letters. In the Morse code a letter consists of one to five dots or dashes and the commonly occurring letter 'e' is assigned the dot code word. Although the concept was not applied to computing until later, the general idea remains.

A code word is the fundamental building block of data compression, building the relationship between the encoded message and the original data. This relationship is utilized to produce the encoded message by replacing the original data with the code words. The decoding process reverses the encoding by replacing the code words with the symbols. The code word is defined by the code definition used to produce the relationship and the various properties relating the code word, such as length and assignment order to the symbols. For data compression the encoding technique tries to find a code word that is shorter or smaller then the original representation. There is generally no requirement to mask the relationship between the code words and the original data, so the term encoding should not be misinterpreted as pertaining to cryptology. For computing purposes, the symbols can be anything from a simple character represented in ASCII to a complex series of bits in an image. The main requirement is that the symbol be uniquely representable for the encoding and decoding process in order to be able to reassemble exactly the original data being compressed. The space utilized by data compression can be any machine resource and is usually engineered in binary and the space available is represented in bits.

As alluded to in the previous paragraph, the two main applications for data compression are the storage and transmission of data. Both of these applications have finite resource limitations and data compression is needed to efficiently utilize the available resource. From this point of view data compression can be interpreted as an optimization or fitting process to the resources available. The only variables are the data or how the data is represented, since the resource storing or transmitting the data is static. This attribute requires a data-centric mechanism to preform the manipulation and achieve the desired results in terms of transmission throughput or data retention.

Data compression has a history almost as old as computing itself. It is typically traced back to the development of information theory in the late 1940's. One of the most notable figures in the field, due to his original writings on the subject, is Claude Shannon. His approach of using the entropy equation to model communication is the foundation for much of the work in this area. This paper discusses the pertinent parts of his work in detail in the following chapters.

Data compression is a required part of computing and an endeavor of great worth to the computing community. Data compression is a large field with varying requirements from the retention of data attributes to the length of the code words. This paper examines the base algorithms supporting the current approaches used to achieve data compression and proposes a new model of the problem, as well as a few methods that utilize this model. We will start with a brief review of the properties used to classify various data compression techniques to introduce these concepts.

1.2 Properties to Classify Compression Algorithms

We define some of the properties used to classify data compression algorithms in order to categorize the algorithms and define their relative strengths and weaknesses. The combination of these properties defines the requirements of the algorithm. Each property must be selected appropriately to achieve the desired compression. More importantly, the quality of the reproduced data source must be achievable based on the properties selected. Compression methods can be broadly grouped in several sub-categories: lossless or lossy; prefixed or prefix-free; variable or fixed length code; and dynamic or static.

The first category of data compression algorithms is defined by whether or not all the original data is completely represented in the compressed form in a manner that allows for reconstruction of the data after decompression. The terms associated with this category are lossless and lossy compression. More specifically, the terms refer to how the symbols are represented in their compressed form as the loss is usually implemented at the time the

encoding table is created. The encoding tables are used by the compression algorithms to store the code word to symbol relationship for use in the decoding and encoding of the data.

Lossy compression uses a method that removes part of the information when creating the encoded symbol. This method typically reduces substantially the size of the final encoding in contrast to lossless compression. Lossy compression is typically used on data sources that do not require all of the information to be utilized in their reproduced form. For instance, an audio signal is originally an analog signal representing the continuous fluctuation of air density. High fidelity conversion of this signal to digital itself already removes some of the information from the original analog signal without a noticeable difference. In some cases this loss is even welcomed as it makes the signal cleaner and more distinguishable. Converting the audio files to a lossy compression form, e.g., MP3 (MPEG-1 Audio Layer 3), removes even more of the information from the data stream while still retaining a respectable quality audio reproduction. In addition to MP3, most multimedia data sources like pictures, audio and video use some sort of lossy compression for the same reason. Some of the formats include JPEG (Joint Photographic Experts Group) for static images and MPEG (Motion Pictures Expert Group) for video images.

Lossless compression is the opposite of lossy in that it retains all of the information from the original data source, i.e., there is a direct correlation between representing the symbol in the original data and the reproduced representation of the symbol in the compressed data. The lossless requirement typically requires a larger compressed representation than the lossy method. Lossless compression is used in applications where data sources cannot tolerate loss in the reproduction of the data. For instance, a text message that losses some of the character data would be of little use to the end reader of the document. Of course, the use of error-recovery codes can eliminate the loss, but only at the expense of greater redundancy, yielding no net gain in compression! So, even though the size of the end result may be large, it is the dictates of the final requirements that determine whether or not lossy compression can be utilized. In data compression a combination of lossy then lossless compression is typically utilized to achieve maximum compression for data sources that allow lossy compression.

The second category of data compression algorithms is defined by the code words being generated by the encoding process. Two properties associated with defining the code word are prefixed or prefix-free. A prefix is utilized to define a difference between one root word and another with the addition of a leading appendage. A prefix, in the case of binary code words, is an appendage of zeros and ones tagged onto the front of a root code word in the form of zeros and ones. For example, if the root of the code word is the last digit in a string of binary numbers and we have the code of 110$\boxed{0}$ and 110$\boxed{1}$. The root of the code word is

represented by the values in the box and the prefix is represented by 110. A prefixed code allows for a prefix to exist within the code words.

The main disadvantage to prefixed codes is that they require some type of termination knowledge in order to differentiate one code word from the next. Two of the common ways to acquire this knowledge is through a fixed length standard or termination codes in the form of additional characters. For data compression this may be a drawback, depending on the method chosen to compress the data as both fixed length and the additional termination codes lead to extra utilization of the resource.

In contrast, prefix-free is a property that maintains code words that do not contain a prefix to another code word. The property maintains the ability for the code word to be unambiguous by maintaining a complete string or complete strings of zeros and ones that terminate at only one symbol. The uniqueness of the code word based on the termination of a recognizable string allows a prefix-free code word length to vary. The varying length is beneficial when it is possible to avoid the extra utilization of the resource required by prefixed code words. The main disadvantage to prefix-free code is that the prefix-free properties usage of the system may be disproportional in terms of code word lengths. In data compression a combination of prefixed and prefix-free code is typically utilized to achieve maximum compression.

The third category for data compression algorithms is defined by the data source being encoded. The terms associated with this category are static and dynamic and refer to the data source stability while the compression algorithm is encoding the data. For static compression algorithms the data distribution does not change from the time of sampling to the time of compression. The source being static allows the algorithm to make decisions on how to properly encode in advance based on the distributions. This may require the building of an encoding table which contains the symbol counts or relative percentages. The main disadvantage to the static compression model is that the algorithm usually requires two passes through the data set. The first pass builds the encoding table and the second pass encodes the data source into the encoding stream.

Dynamic compression allows for changing distributions within the source data stream. The encoding changes in response to the distribution modifications as new symbols are encountered or in response changes in relative distribution of neighboring symbols. The ability to adapt to changes in distribution allows for the algorithm to process the data in one pass by encoding the symbols as they are read. The key advantage to this approach is that one pass through the data allows for encoding in real time. A major disadvantage is its susceptibility to error which can ruin the code. Since the data is sampled in a one pass or streaming manner the error may be encoded and transmitted before the error is recognized.

The error may even reach the decoding phase before the error is recognized inhibiting the decoding process. Retry and sampling logic is implemented to address these concerns, increasing the overhead of the compression model. Static compression is also susceptible to error, but since the data is static the retry logic can be imposed at the time of error with less cost as compared to dynamic compression. Another disadvantage is the added overhead involved in the update to the encoding table which is required when a distribution changes. This requirement adds additional overhead as updates must be periodically sent to the receiver. Without the added ability to address the errors and decrease the overhead adaptive encoding is limited.

1.3 Objective

The list of properties in Sec. 1.2 is far from exhaustive as it only explains native differences in a variety of compression schemes, however, the list is complete enough for the objective of this study. The research preceding this dissertation evolved through a thorough analysis of the current methods used in the art of data compression in both the static and dynamic environments. The analysis revealed that the current solutions were based on three fundamental algorithms, Shannon-Fano, Huffman and Shannon-Fano-Elias. This included Arithmetic encoding which is the current standard in data compression and is based on the concepts of Shannon-Fano-Elias and the use of data modeling. The main complaint about the current solutions in both the static and the dynamic environments is the overall complexity of the methods. These complaints are the main reason Huffman is still used for compression even though Arithmetic encoding has been shown to produce better results. We tried to address the problem of complexity in the previous analysis and the process revealed that only marginal gains could be accomplished using the previous algorithms containing current enhancements. The research into these algorithms yielded insight on new ways to analyze the data compression environment and to use these concepts to provide new methods to perform data compression.

To accomplish this objective, Chapter 2 covers some of the basic terms related to data compression beginning with the theory of entropy and the theoretical ideal. The theory of entropy is geared around the concept of space, items represented by their probability within the space and the distance within the space separating the items, or known reference points. The entropy concept is fundamental to all the work in the field of data compression and is used throughout this dissertation to analyze the various methods. Also covered are the three major compression algorithms fitting the above scheme and an analysis of the strengths and weakness of these algorithms. This review of the base algorithms in data

compression points out the data centric view and how this view influences the approach to compression that the algorithms follow.

In Chapter 3 we re-examine in detail the base algorithms with regard to the unique approach used to create the algorithms and how this approach applies to the theory of entropy. We breakdown the previous algorithms operation in terms of entropy theory to examine their subcomponents to gain new insight into the methods ability to model the entropy ideals. The new analysis of the base algorithms includes the formulation of the equations of entropy that represent these algorithms and provides a quantification of the limits of these algorithms to compress data in terms of Shannon's theory. Chapter 3 also discusses the view taken in this work on the data compression environment. During this analysis the concept of certainty and the duality of the theoretical ideal are added to entropy theory in order to explain the relationships between the algorithms and the system.

The concepts are further developed in Chapter 4 as the model of compression environment of the base algorithms, Shannon-Fano, Huffman and Shannon-Fano-Elias, is examined to reveal the benefits each depiction has at describing entropy and the certainty. Chapter 4 uses the concepts exposed to introduce a new more holistic model of the compression environment based on the space available in the system and the space occupied by the data using the analysis in Chapters 2–3. The Map of Certainty and Entropy, MaCE, defines entropy and certainty as a two-dimensional space based on the entropy and certainty equations. MaCE shows the individual components that comprise entropy and certainty of the environment. The environment consists of both entropy inherent to the data and certainty inherent to the system. The development of this model reveals that it is possible to compare certainty to entropy and entropy to certainty to simplify the problem of data compression. MaCE is also used to illustrate the use of base manipulation to increase compression. This model opens multiple doors into the subject of data compression, and only a small portion of these are examined in this dissertation.

Chapter 5 uses MaCE to address some of the complexity of the problem of data compression and produces some new methods to accomplish this task. These new methods show the versatility of the proposed approach and its affect on both speed and compression related to existing compression algorithms. The methods, SLM (Select Level Method), SLMN (Sort Linear Method Nivellate) and JAKE (Jacobs, Ali, Kolibal Encoding), use these concepts to accomplish the task. SLM is developed utilizing MaCE model and is designed for speed to achieve a code length $L(x)$ with $L(x) \leq H(x) + 2$ with solely an arithmetic comparison model. The concept of equilibrium and the split-tree are defined in order to further enhance the computational efficiency of SLM.

The second method, SLMN, is developed utilizing MaCE and the concept of midpoints between levels in the binary space. The midpoints develop an integrated sorting algorithm of $O(n)$ to pre-process the symbols for the data compression method as well as a fitting function built on the midpoint concept to adjust the code of length $L(x)$ to $L(x) \leq H(x) + 1$.

Chapter 5 also analyses the meaning of the $+1$ in $H(x) + 1$ and this knowledge produces JAKE. JAKE is an extension of SLM and SLMN into other bases more suitable to represent the data within the constraints of the linear system. The concept of choosing a base adjusts the width of the compression environment to a dimension more suitable for the data distribution. The adjustment of the width provided by JAKE augments the compression optimization as SLMN and SLM both concentrate on minimizing the length of the code word. The width adjustment by JAKE allows the use of SLM and SLMN to obtain data compression closer to the theoretical ideal. We look at Arithmetic encoding with the understanding of the $+1$ with the purpose illustrating the power of changing the numerical base of the system to other than base 2. A comparison between Arithmetic encoding and Huffman encoding also shows how the granularity of the system affect on the overall compression.

In Chapter 6 we also analyze the results in terms of CPU cycles for SLM in comparison to Shannon-Fano-Elias and we analyze SLMN in comparison to Shannon-Fano-Elias and the theoretical ideal in terms of entropy, file size and compression ratio. The results are broken down into their respective parts in order to clearly see the relationship between the algorithms and the data. The test sets included best/worst case analysis as well as random and sequential data distributions. The analysis shows the advantages of the methods over the current algorithms and reveals prospects for improving upon the results. Chapter 7 contains conclusions generated from this dissertation and some of the future work resulting from this research.

Chapter 2

BACKGROUND INFORMATION AND CURRENT SOLUTIONS

2.1 Introduction to Data Compression Algorithms and Techniques

This chapter provides a review and the framework for the development of major prefix-free, lossless, compression algorithms based on static data sources. In order to accomplish this task a brief introduction to some of the major terms that apply to data compression and information theory is also required. Of great importance to the theory of data compression and information theory is the work of Claude Shannon in Sec. 2.2. Shannon's decision to use the theory of entropy to describe information is one of the keystones in the field and is now used to define and compare compression algorithms. In Sec. 2.3 we discuss the theory of entropy in general and continue the discussion as it relates to data compression and information theory in Sec. 2.4. In Sec. 2.4.1 we explain Shannon's concept of the theoretical ideal and the concept of surprisal. These topics lay the ground work for data compression and are used throughout this dissertation to explain and compare the various algorithms and methods.

In Secs. 2.5–2.7 we discuss three of the fundamental algorithms in data compression; Shannon-Fano, Huffman and Shannon-Fano-Elias as all three encoding methods adhere to the given requirements. Each algorithm compresses data in a unique fashion and each presents a different approach to the problem of data compression. The approach influenced how the algorithms were designed and how the algorithms applied to the theory of entropy.

2.2 Claude Shannon and Information Theory

Claude Shannon is considered the pioneer of information theory. Through his work and the work of others, three of the premier compression algorithms in use today were devised. Shannon's theory of information entropy and the theoretical ideal [39] play a significant role in this development. The concepts of information entropy and the theoretical ideal are used to define the meaning and to analyze the overall compression efficiency of the algorithms throughout the dissertation. Both of these concepts will be further explained in Secs. 2.3–2.4. To develop the concepts Shannon first visualizes the content of a message sequence being a stochastic process that can be examined as a Markov chain. These two terms deal with the state transition properties related to the message state distribution.

A stochastic process is one where all state transitions are probabilistic. A state in the case of data compression and information theory represents a single possibility within the group of possible states of the data and the transition is the movement from one state of the data to another based on the probability. Shannon uses this view to explain communication theory. Given a finite set of possible symbols to transmit, the next symbol is represented by the probabilities of all the possible symbols. This model is appropriate for data compression as the probabilities of the symbols determine the transitions in the compression model and typically there are multiple symbols represented by their probabilities of occurrence.

A Markov chain is a stochastic process adhering to the Markov property. The Markov property means that the next state is only dependent on the present state. The existence of this property allows a Markov chain to be built showing the transitions from one state to another. Shannon applies this theory to communication by adding the assumption that a symbol is produced for each state transition. In data compression the state transitions are based on the current symbols probabilities and the code word produced is in response to the state transition.

In addition to the above terms, ergodicity is a term used to describe a stochastic process that is not dependent on the initial conditions of the system. This means that it is impossible to predict the next state of the system based on the present condition and that the probabilities are sufficient to describe the system. This term can be applied to both data compression and communication.

The next step in Shannon's view is to find a measure that represents choice and uncertainty. Since the underlying assumption is that data compression and transmission adhere to the property of being an ergodic Markov chain, Shannon says we are looking for a measure $H(p_1, p_2, \ldots, p_n)$, where p is the probability of datum i and n is the number of items. The H measure he introduces adheres to the requirements in Property 1:
Property 1.

1. H is continuous in P_i;
2. If all the P_i are equal, i.e., $P_i = 1/n$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, uncertainty, when there are more possible events; and,
3. If a choice needs to be broken down into two successive choices, the original H should be the weighted sum of the individual values of H. This concept is reproduced in Fig. 2.1.

The left side of Figure 2.1 represents three possibilities with transition probabilities $P_1 = 1/2$, $P_2 = 1/3$ and $P_3 = 1/6$. On the right of Fig. 2.1, a modification of the original H with the weighted sum of P_2 and P_3 is represented. Both of the representations have the

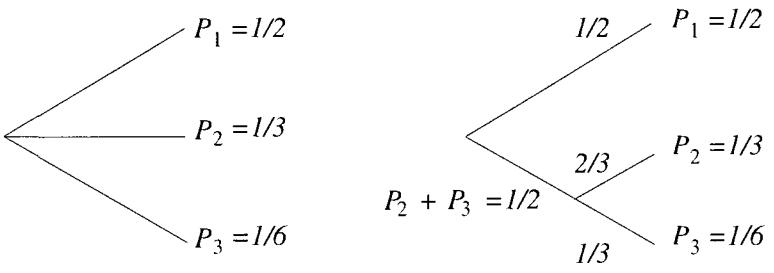


Figure 2.1: Shannon's decomposition of a choice from three possibilities. The third property of H is illustrated by the combination and division of P_2 and P_3.

same probabilities and are equivalent. The equation to represent this transition is $H(1/2, 1/3, 1/6) = H(1/2, 1/2) + 1/2H(2/3, 1/3)$. The $1/2$ coefficient represents the second set of choices only occurring half the time. The multiplication of $1/2 \times H$ of each component, $1/2H(2/3, 1/3) = H(1/3, 1/6)$, would represent the original choices.

Shannon concludes that only one H satisfies the requirements and the H is shown in (2.1). Shannon notes that the equation has the same form as the entropy equation used by other fields. The observation leads to the theory of entropy and its relationship to the fields of information theory and data compression.

$$H \equiv H(P_1, P_2, \ldots, P_n) = -k \sum_{i=1}^{n} P_i \log_2 P_i. \tag{2.1}$$

2.3 Entropy Theory

Shannon's assumption of the application of the entropy equation is used either directly or indirectly by multiple data compression algorithms. It is also used to describe the efficiency of data compression models including the proposed methods. This section is not meant to cover all aspects of entropy as it has a wide range of implications, it is meant to show the transition from the general form of the entropy equation to the form used in information theory.

In general, entropy can be viewed as the limits to do useful work imposed on a system. Shannon extends this concept to represent the amount of uncertainty that remains in a information system [39] as summarized in Sec. 2.2. We must analyze the general form of the entropy equation in order to understand the relationship between the concepts.

$$S = -k \sum_{i=1}^{n} P_i \log P_i, \tag{2.2}$$

where S is entropy, k is a constant of proportionality and P_i the i the known item's probability.

Before examining the details in (2.2), we need to define entropy in terms of the space, the items contained within the space and the work to be accomplished. The space in data compression corresponds to the resource available. The space in information systems is usually defined in binary and the space available is the bit space. The bit space is usually visualized using a binary tree to describe the possible paths from the root of the tree to the leaves. The paths from the root to a leaves represent the code words required to reach symbols stored in the tree. The combination of all the leaves represents the symbol space as all the symbols must be contained there to maintain the prefix-free property. In information theory and data compression the symbols represent the information contained within the space. The distance between the symbols and a known reference point represents the uncertainty or the entropy within the system. In order to quantify a relationship between these items, it is necessary to put the two terms in proportion. This term k is exhibited in (2.2).

Entropy is defined by a distance measure between the items contained in the space S. The proportionality constant k in (2.2) relates the probability represented by P_i of datum i to S. This relationship makes the constant of proportionality dependent on the unit of measure chosen to represent the system space. For example, for binary data probability P_i represents the symbol usage of this space. The constant to achieve equality is $k = 1/\log(2)$, this converts the symbol fraction in the equation to its representative number of bits. Some other common proportionality constants are Boltzmann's constant k_B for kinetic energy at the particle level and Euler's number (e) for fluid dynamics.

The $-\log()$ function in (2.2) deserves some attention. In the general entropy equation the logarithm is in base 10 and in order to represent other spaces the $\log()$ function is converted by k as previous explained. This discussion is about the general entropy equation, so the logarithm used is in base 10. Since $-\log(P_i) = \log(1/P_i)$ we also need to examine the fraction $1/P_i$. This quantity represents the ratio of the space used by item i in comparison to the whole, i.e., the reciprocal of P_i. P_i represents the space used by the datum i and the number of divisions of the whole possible by P_i. For example, if $P_i = 0.01$ then $1/P_i = 1/0.01$ and the number of divisions returned is 100. The number of divisions is the operand passed to the logarithm function to determine the number of expansions required to ideally represent P_i. The $\log()$ function is now operating on the reciprocal of P_i and the function returns the number of expansions of the base to equal the reciprocal. For example, $\log(100) = 2$ which represents the two expansions of base 10. If we look at this visually, the expansions are on the vertical and the divisions are on the horizontal. In a binary tree the expansions represent the level L of 2^L and the divisions of L are the number of nodes or expansions of the base to equal the reciprocal.

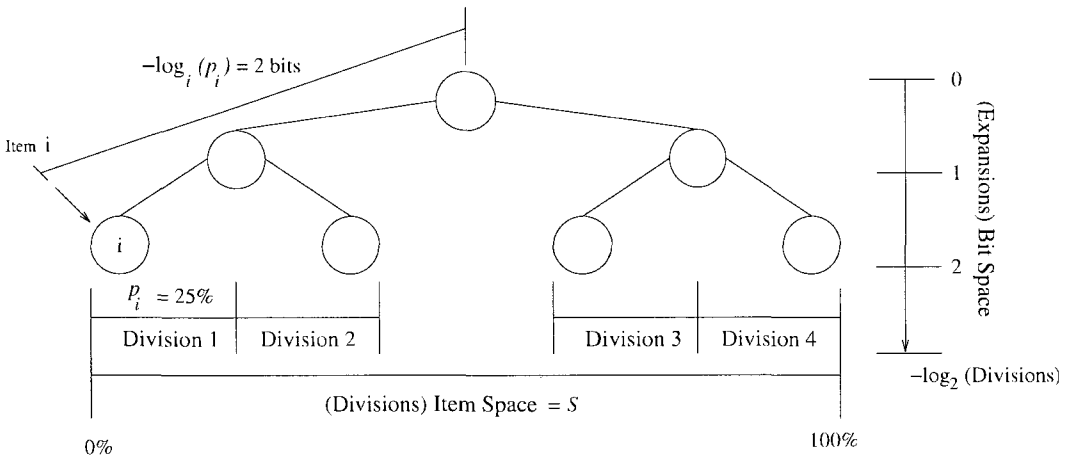


Figure 2.2: Entropy of a tree described in base 2. The probability of item i is represented by $P_i = 25\%$. The division of the space is represented by the nodes in the tree and the number of expansions are represented by the horizontal levels. The $-\log(P_i)$ represents the distance from the root to the item i.

This completes the explanation of the subparts of the entropy equation. Putting the parts back together, the constant of proportionality puts the equation into proportions relative to the space and the items contained within the space. After the k term is executed the log() function and the probabilities are now in proportion to the space available for work. The $1/P_i$ is a ratio used to describe the number of divisions that fit on the horizontal given item i's probability. The log() function converts the divisions on the horizontal into the number of expansions on the vertical required to reach the space given the base. The remaining P_i preceding the log() function is multiplied and returns the percentage of the space used by the item i given the number of expansions returned by the logarithm.

Figure 2.2 displays the concept in base 2. Base 2 is used for simplicity purposes, however, the concept applies to all bases. The P_i in the figure represents the probability of symbol i equal to 25%. The nodes represent the divisions of the space. At the level 2 there are $2^L = 4$ divisions. This is also represented by $1/P_i = 4$. The number of expansions to reach L is represented by $\log_2(1/P_i) = \log_2(4) = 2$.

The above explanation does not, by any means, explain all the uses of the entropy equation. After all, the meaning of an equation is how it is applied and not in the equation itself. The point of the above explanation is to set the framework related to the entropy equation for use with information systems. The explanation is geared around the concept of a space, with items represented by their probability within the space and the distance within the space separating the items or known reference points. The separation between the items or the known reference points can be viewed as the entropy.

2.4 Entropy in Information

In Sec. 2.3 we looked at how the entropy general equation can be used to calculate the entropy in a given space through the probabilities of the symbols. This section will customize the equation for information systems and explain the terms catered toward that endeavor. As stated earlier, Shannon was the first to propose the use of the entropy equation for information theory [39]. In general we defined entropy as the limits on the system to do useful work. In data compression and information theory the work involved is the transfer or storage of information via a code word to an information relationship. This relationship is the fundamental building block of both processes. The success of a process is represented by the certainty that the symbol resolves uniquely to the information.

In Sec. 2.2 Shannon defines the measure for information entropy adhering to Property 1 in terms of H. Since then the symbol H has become the customary representation to define information entropy, also referred to as "Shannon Entropy". This is in contrast to S for general entropy given previously. By substitution, where $k = 1/\log(2)$ and S is exchanged for H, the equation for information theory becomes (2.3).

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i. \tag{2.3}$$

Now that we have an equation to represent Shannon's view of entropy, we look at the terms again more closely in binary. According to Shannon, information entropy represents the average uncertainty or lack of information contained in the encoded message. So if we follow the framework provided by the spatial meaning of entropy we are looking for two things. The first is the space being utilized by the system and in the case of binary it is the bit space. This space is represented by a series of 0's or 1's (bits) with some limitation put on the length or number of the bit values representing the complete space. The second thing we are looking for is the items of known probability and we refer to them as symbols. These symbols can represent anything from a character to a piece of an image. By this simple exchange it is easy to see that the entropy equation relates to the transmission and storage of data in a computer. The space is represented by the binary values possible and the symbols represent the data to be encoded or stored.

In order to understand entropy the definition of certainty is also required. We define certainty as the ability to identify an individual symbol. This is accomplished by the complete code word. The complete code word is required to represent certainty as it is the path establishing both distance and direction to the symbol and the symbol itself. The final bit in the code word represents the space where the symbol is located and the string of bits represents the path to that location. The individual bits in the code word are the overhead

or entropy. Using any of the individual bits or subsequence of bits does not resolve to a symbol directly. This applies to the last bit as well even though the symbol is located at that bit location the preceding bits are required to reference it exactly. This means that all the bits in the code word represent uncertainty.

We define uncertainty of resolving a code word to a symbol by the complete entropy equation and the space it defines. Although the complete sequence of bits represents certainty, the bits themselves also represent uncertainty. All the bits preceding the final bit can represent paths to other symbols, however, the bits cannot represent symbols themselves due to the prefix-free requirement. Uncertain results will occur when referencing a location specified by any combination of the preceding bits.

For example, if the code word 000 represents the arbitrary symbol A. The final 0 represents the symbol and the three bits 000 are the overhead required to reach the symbol. The preceding two bits 00 cannot be used to represent other symbols but they can be used to represent a path to another symbol. For example, 001 can represent an arbitrary symbol B. The preceding two bits 00 can be interpreted as representing two symbols, however, the bits do not represent a symbol uniquely and the outcome remains uncertain. The use of the complete code word to represent a symbol places the symbol in a leaf in the binary tree and the complete code word and the symbol A it identifies represents certainty. This illustrates the meaning of entropy for binary as the uncertainty of identifying a symbol for any combination of bits other than the complete sequence of symbols and the symbol it represents. So the sum returned by the H equation is the sum of all the bits representing the code words of the encoded symbols and the symbols themselves.

For example, we can use the entropy equation $H = -\sum P_i \log_2(P_i)$ to calculate the average uncertainty in relation to a set of possible colors (symbols) and their probabilities. Suppose we have four arbitrary colors: red, white, blue and green represented by their known probabilities of occurrence $P_i = (0.5, 0.25, 0.125, 0.125)$ within the color space, c. We need to translate the color state percentages into the divisions being used by the symbol space using $1/P_i$, yielding the series $(1/0.5, 1/0.25, 1/0.125, 1/0.125) = (2,4,8,8)$. Applying the logarithm of the divisions we obtain $y_i = (1,2,3,3)$ as the number of expansions. Multiplying through by the probabilities returns the space used by each color i within the total color space. This final step is displayed in Example 2.5.

$$\text{Red} = 0.5, \quad \text{White} = 0.25, \quad \text{Blue} = 0.125, \quad \text{Green} = 0.125;$$

$$E(y_i) = \sum_{i=1}^{n} P_i y_i = (0.5 \times 1) + (0.25 \times 2) + (0.125 \times 3) + (0.125 \times 3) = 1.75. \qquad (2.4)$$

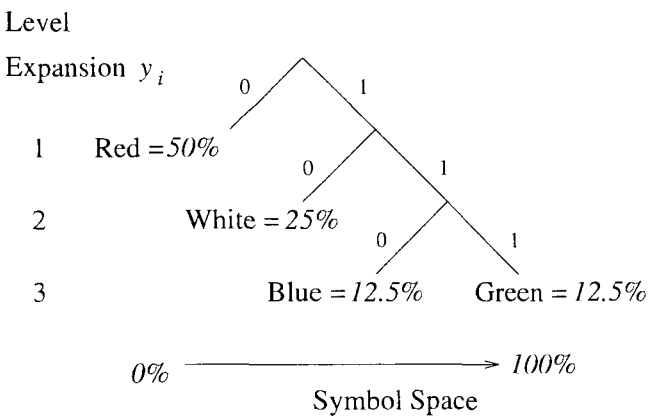


Figure 2.3: Colors expansion example. The path and the terminating percentages represent certainty and the path to the colors represents the entropy.

The example represents the application of (2.3) which produces an entropy value of 1.75 for the system. The code words assigned are defined by 0 for a left branch and 1 for right branch. The code words for each are: (red: 0, white: 10, blue: 110, green: 111). The probabilities of the colors and their respective code words represent the certainty of the system and the space described by the code word to reach the symbol represents the uncertainty of the system.

Looking closely at each of the individual terms in Example 2.5, we can see they represent the space used in each expansion of the tree. Red is using 0.5×1 or 50% of the space possible at expansion 1. White uses 0.25×2 or 25% of the space possible at expansion 2. Blue and Green each use 0.125×3 or 12.5% of the space possible at expansion 3. The expansion and percentage of the tree used is illustrated in Fig. 2.3. We will refer to the partial sums as H_i to stand for individual entropy values calculated in the complete entropy equation where $H = -\sum_{i=1}^{n} H_i$ and $H_i = -P_i \log_2(P_i)$. The equation is denoted in (2.7).

$$H = -\sum_{i=1}^{n} H_i, \tag{2.5}$$

$$H_i = -P_i \log_2(P_i). \tag{2.6}$$

2.4.1 Shannon's Theoretical Ideal

Example 2.5 shows that the H_i equation represents the space used by P_i, where P_i represents the percentage of the symbol space and $-\log_2(P_i)$ represents the length of the code word to reach the symbol represented in the leaf. The concept of assigning the symbol to the leaf is represented by the theoretical ideal proposed by Shannon. The theoretical ideal is part of the H equation and is defined by $-\log_2(P_i)$. The return value represents the information

denoted by of the number bits.

In [39] Shannon first proposed the concept of using the logarithm function to measure information contained in a message, basing this idea on the Hartley function which measures uncertainty of a system. The Hartley function states that the information revealed by picking a message x at random from a finite set of variables is equal to the $\log|X|$. In Hartley's case the log function is in base 10 and Shannon uses the constant of proportionality k to convert it to $\log_2$ for binary.

The same component $-\log_2(P_i)$ is also known as surprisal as it stands for the opposite of the probability or the "surprise" of the opposite of the probability occurring. This may seem contradictory, but as with most things, the opposite side is dependent on the point of view.

In actuality, the term $-\log_2(P_i)$ has both meanings as it represents the location in space where the information is ideally located in reference to the root node and $-\log_2(P_i)$ also represents the distance between the information and the root denoting the uncertainty or surprisal. The duality is explained by $-\log_2(P_i)$ representing both the number of bits in the complete code word representing the certainty of the symbol and also the substring of the code words representing the uncertainty. The equation for both is denoted in (2.8) using the symbol $\mathbb{I}$ to indicate the duality of the two statements.

$$\mathbb{I} = -\log_2(P_i). \tag{2.7}$$

In Secs. 2.2–2.4 we discussed the major components to the theory of entropy and its augmentation to information theory and data compression. This information will be used throughout the dissertation to explain various algorithms and concepts in the field of data compression. The goal of data compression is to remove the overhead as much as possible. The overhead is limited by the theory of entropy as it represents the required path from the reference point to the symbol being represented. The following Secs. 2.5–2.7 introduce the algorithms Shannon-Fano, Huffman and Shannon-Fano-Elias. These are considered entropy encoding methods and they fit the requirement of prefix-free, lossy compression algorithms of static data described in Chapter 1.

2.5 Shannon-Fano Encoding

The purpose of this section is to describe the details of Shannon-Fano encoding. An example of the operation of the algorithm is given as is a discussion of the efficiency of the algorithm in terms of entropy. The advantages and disadvantages are reviewed in order to make comparisons to the other algorithms. Lastly, the view of the bit/symbol space that

Shannon-Fano perceived in the development of the algorithm and the algorithm's relation to the entropy equation is discussed.

The Shannon-Fano algorithm was first published by Claude Shannon [39] in 1948 and later attributed to Robert Fano when it was published in his technical report [15]. The basis of the technique can be seen in the third property of H as defined in Properties 1. This property states that if a choice needs to be broken down into two successive choices, the original H should be the weighted sum of the individual values of H_i. This concept is also displayed in Fig. 2.1. This property allows an algorithm to combine and then subdivide a series of choices (possible symbols) based on their individual H_i values. Recall the H equation (2.3) is a sum of all the symbols individual entropy values H_i. The determining factor of H_i is the probability P_i of the individual symbol i. Knowing this it is possible to base the choice on the symbol's probability without actually calculating the H_i value itself, making the algorithm a little less computationally expensive and easier to analyze.

To continue the discussion using the third property as a guide and the substitution of P_i for the individual H_i values an algorithm is defined for data compression. What the property defines is the actions required for the algorithm to accomplish during the act of data compression. The algorithm needs to start with a combination of the choices based on their probabilities and then subdivide the combination based on the weighted sums of the choices at each division. The last part of the puzzle is to define the divisions used in the method. Since binary is being used the division is based on powers of 2. The process of dividing by two also leads to the decision of dividing the combined choices weighted sum by approximately half of the total sum. This process has the result of mapping the data elements to the binary tree and producing data compression based on the properties Shannon described. The process produces a very simple algorithm based on a divide and conquer methodology.

2.5.1 Shannon-Fano Encoding Algorithm and Example

The algorithm is a recursive subdivision of all the symbols based on a division by 2 at each choice. The algorithm begins with all symbols sorted based on their probabilities at the root of a binary tree. This node represents the combined weighted sum of all the probabilities as we have not made any divisions at this point. After the sorted list is placed at the root, the algorithm finds the point where the combined sum can be divided in approximately half. The two halves represent the combined weighted sum of individual H_i as discussed in the third property of H. The assignment of a 0 to the left side and a 1 to the right side of the tree will represent each division and start the formation of the code words representing the

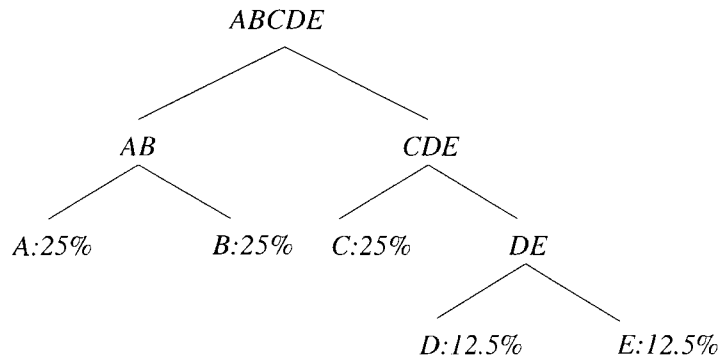


Figure 2.4: Shannon-Fano example. ABCDE represent a set of symbols that are recursively split based on their percentage using Shannon-Fano's algorithm.

symbols final location in the binary tree. The two steps of dividing the combined weights and assigning the 0 and 1 are then recursively executed on each of the divisions. This process continues until all symbols are located in a leaf node of the tree. The string of 0s and 1s from the root to the leaf represents the code word assigned to the symbol. Note: The assignment of 0 and 1 to a particular side is arbitrary, the only requirement being consistency on the choice in order to have the series represent a valid path to the symbol in the tree. The runtime of the algorithm is $O(n)$ with n number of symbols. The pseudo code is listed below:

1. Sort the symbols according to their probabilities.
2. Find the weighted sum of the two halves at approximately 50%.
3. Assign 0 to one division and 1 to the other.
4. Repeat steps 2 and 3 until all symbols are represented in a leaf.

For example, given five arbitrary symbols: A, B, C, D and E represented by their known probabilities of occurrence (A:0.25, B:0.25, C:0.25, D:0.125, E:0.125). Next, the algorithm calls for the symbols to be in sorted order. This is already done for convenience. Next the algorithm finds the point to divide the two halves represented by a weighted sum of approximately 50%. This divides the list into left half (A:0.25, B:0.25) and right half (C:0.25, D:0.125, E:0.125). Next the algorithm assigns the 0 and the 1 to their respective halves. The recursion splits the two halves again and again until the all the symbols are represented in a leaf on the tree. This concept is displayed in Fig. 2.4

2.5.2 Shannon-Fano Encoding Analysis

In the given example the symbols are mapped to their correct location in the bit space. This results in optimality with regards to the theoretical ideal (2.8) for each of the symbols. All the symbols are ideally mapped and the total entropy represented by $L(X)$ is equal to $H(X)$ where X represents the set of probabilities. Unfortunately, the Shannon-Fano algorithm does not always produce optimal results. The lack of optimality can be attributed to two reasons. The first is the binary requirement. Each of the divisions represents a power of 2 while no requirement of a power of 2 is made on the data set. The actual symbol percentage can be a value greater than or less than the space. Since the theoretical ideal of each of the symbol percentages cannot be mapped directly to the tree there is difference between the symbols actual length $l(x)$ as it is placed in the tree and the calculated $\mathbb{I}$ using the theoretical ideal. The lack of granularity by using binary to represent the symbol's percentage requires a certain amount of tolerance as the symbol must be mapped to a space represented by a power of 2. In general, Shannon-Fano produces an encoding with $L(X) \leq H(X) + 1$ where the $+1$ represents the tolerance for this difference. The significance of the tolerance will be explained in Sec. 5.4. It is sufficient to know the limitations for the time being.

The second reason for the lack of optimality has to do with the algorithm itself. The key assumption is to divide the combined symbol percentage in half recursively. This assumption only addresses the width of the tree represented in the entropy equation H. As stated earlier the H equation represents the bit space and the symbols space. The result of the H equation is the sum of all the code word lengths represented by $-\log_2(P_i)$ multiplied by the widths used by the symbol P_i. The code word length in the case of Shannon-Fano is not considered and only arbitrary assigned based on the division of the symbol space. The lack of consideration of the length leads to results that are suboptimal to the ideal that can be achieved. For example, given five arbitrary symbols: A, B, C, D and E represented by their known probabilities of occurrence (A:35%, B:17%, C:17%, D:16%, E:15%). The recursive division splits the symbol space based on the 50% rule and results in the left image in Fig. 2.5. If we use the H equation to calculate the actual entropy $L(X)$ we obtain $L(X)$ =(A:35% $\times$ 2) + (B:17% $\times$ 2) + (C:17% $\times$ 2) + (D:16% $\times$ 3) + (E:15% $\times$ 3) = 2.31. If we arrange the tree to resemble the right image in Fig. 2.5 we obtain a new actual entropy $\overline{L}(X)$ =(A:35% $\times$ 1) + (B:17% $\times$ 3) + (C:17% $\times$ 3) + (D:16% $\times$ 3) + (E:15% $\times$ 3) = 2.30. The second result is the actual optimal value obtainable given the binary representation for that arrangement of symbols. The original result is a tenth off of the optimal. In Sec. 2.6 we describe the Huffman algorithm which actually achieves the optimal result. This result could be worse if not for some of the subtitles associated with the algorithm. We will look

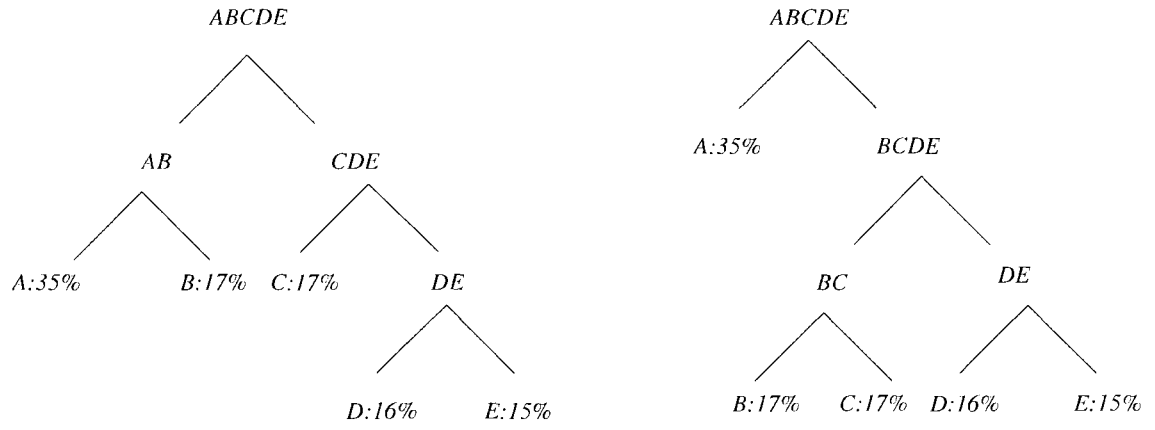


Figure 2.5: Comparison of Shannon-Fano to Huffman encoding example. On the left is the tree representing the compression produced by Shannon-Fano. On the right is the optimal encoding produced by the Huffman algorithm.

at the Shannon-Fano algorithm in more detail in Sec. 3.3.

Although Shannon-Fano does not produce an optimal encoding on all occasions it does stay within $H(x) + 1$, which is an achievement. The real advantage of the Shannon-Fano encoding algorithm is the speed and simplicity of the algorithm. The runtime of the algorithm is $O(n)$ with n number of symbols. Due to the recursive nature of the algorithm it is easy to implement in parallel. The disadvantages are the requirement of a sorted list of symbols and the final encoding does not always represent an optimal encoding. If speed is required, the optimality can be overlooked in favor for the processing time. The sort is a constant hindrance.

The two reasons behind the lack of optimality in terms of ideal entropy as described by Shannon and in terms of the actual optimality achievable by an algorithm given the binary requirement were explained. In Sec. 2.6 we look at the Huffman algorithm which does achieve optimal encoding with regards to the binary requirement.

2.6 Huffman Encoding

The purpose of this section is to describe the details of Huffman encoding. An example of the operation of the algorithm is given as is a discussion of the efficiency of the algorithm in terms of entropy. The advantages and disadvantages are reviewed in order to make comparisons to the other algorithms.

Huffman code was created in 1952 by David Huffman in a term paper assigned by Claude Shannon and was later published [22]. The Huffman algorithm uses a bottom up approach based on the greedy principle to build a binary tree. This approach produces

an algorithm that supersedes Shannon-Fano by producing optimal compression with only a slight variance in complexity. Although for certain applications there are better alternatives, Huffman encoding is still a premier compression algorithm after 55 years.

2.6.1 Huffman Encoding Algorithm and Example

The Huffman algorithm begins with symbols sorted based on their relative counts. The algorithm chooses the two minimum count symbols from the list and combines them under a single node represented by the combined count. This combined count is subsequently placed back in the proper place to maintain the sorted list. The next step repeats by combining the next two minimum counts which can either represent individual symbols or combined groups of symbols in a sub-tree. There is no regard to which is chosen, just that the minimum values are combined at each step. The constant choice of the minimums adheres to the greedy principle by always making the best choice at the time of decision. The combination of the symbols or the sub-trees eventually reaches the point where all symbols are represented under a common tree and completes the encoding sequence. The route from the root to the leaves represents the code word for each symbol. The sort as specified at the beginning of this explanation is not a requirement, it is utilized to diminish the overall complexity of the algorithm in both explanation and implementation. The addition of a sorted list and priority queues enhances the runtime of the algorithm to $O(n)$ with n number of symbols. The pseudo code is listed below:

1. Find the two symbols with the lowest count from the list of symbols. The list can be sorted or unsorted.
2. Combine the sums and reference both symbols under a common node.
3. Place the common node back in the list.
4. Repeat steps 1 thru 3 until all symbols are represented in tree. No symbols are orphaned.
5. The paths from the root to the leaf nodes represents the code words.

We can see the change in optimality by examining the same example that caused Shannon-Fano to reach less than optimal results. Given five arbitrary symbols: A, B, C, D and E represented by their known symbol counts (A:35, B:17, C:17, D:16, E:15). Note: the numbers are the same because both the counts and the probabilities equal 100. The numbers are in sorted order for convenience, but as stated this is not required. Next, the

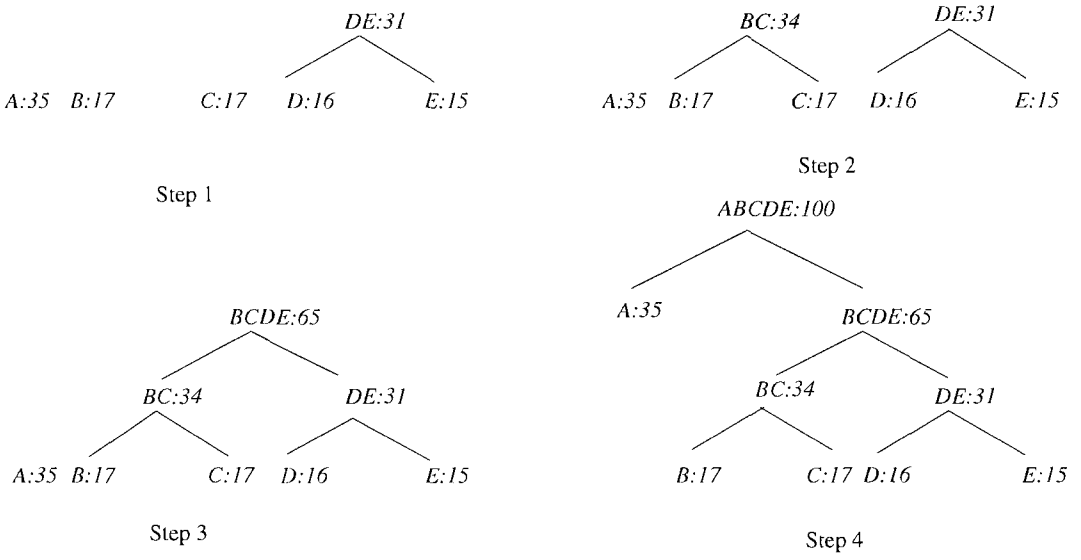


Figure 2.6: Illustration of the Huffman algorithm.

algorithm selects the two minimum values (D:16, E:15) and combines them into a single node with the representative counts (DE:31). Now the series looks like (A:35, DE:31, B:17, C:17). The algorithm repeats by finding the next two minimums (C:17, D:17) and combines them to (CD:34). The series is now (A:35, BC:34, DE:31). The algorithm repeats again by finding the next two minimums (BC:34, DE:31) and combines them to ($BCDE$:65). The next series (A:35, $BCDE$:65) consists of only two items which are combined under the root node. The final combination completes the tree with the root representing ($ABCDE$:100). The algorithm now has all the symbols placed in the correct location and the code is represented in binary by following the path from the root to the leaves of the constructed tree. The end result is an actual entropy $L(X)$ =(A:35% $\times$ 1)+(B:17% $\times$ 3)+(C:17% $\times$ 3)+(D:16% $\times$ 3)+(E:15% $\times$ 3)= 2.30. This concept is displayed in Fig. 2.6.

2.6.2 Huffman Encoding Analysis

As stated previously, Huffman encoding produces optimal code in terms of the entropy possible given the binary requirements as all the values stay within $H(x) + 1$. The binary requirement is responsible for the $+1$ requirement for the same reason as in Shannon-Fano. We can see the effect of the binary system on the algorithm by analyzing the difference between the entropy for the theoretical ideal and the actual encoding. The theoretical ideal entropy is calculated in (2.9). For this small example we can see the difference between the theoretical $H(X) = 2.233$ and the actual obtainable value $L(X) = 2.30$. The key advantage to using this algorithm is the optimality of compression. It does so in $O(n)$ with the additional space requirements and the complexity of priority queues. The disadvantage is the

algorithm requires a sort to achieve $O(n)$. In Sec. 2.7 we look at the Shannon-Fano-Elias algorithm which does not achieve optimal code, but it does not require the data to be sorted either. This is a significant advantage as it reduces the overall compression time to $O(n)$.

$$\begin{aligned} H(X) &= (A:35\% \times 1.515) + (B:17\% \times 2.556) + (C:17\% \times 2.556) \\ &\quad + (D:16\% \times 2.644) + (E:15\% \times 2.737), \\ &= 0.53025 + 0.43452 + 0.43452 + 0.42304 + 0.41055, \\ &= 2.233. \end{aligned} \tag{2.8}$$

2.7 Shannon-Fano-Elias Encoding

The purpose of this section is to describe the details of Shannon-Fano-Elias encoding. The discussion applies encoding to entropy and the theoretical ideal as discussed in Sec. 2.4. An example of the operation of the algorithm and a discussion of the efficiency of the algorithm in terms of entropy is given. The advantages and disadvantages are reviewed in order to make comparisons to the other algorithms.

The Shannon-Fano-Elias encoding method was proposed by Shannon, developed by Elias and later published by Abramson [2] to produce a prefix-free code. Shannon-Fano-Elias was developed on the basis of Shannon-Fano, the proof of Kraft's inequality [26] and Shannon's work on information theory. Shannon's work on information theory and Shannon-Fano code were explained Secs. 2.4–2.5. Section 2.7.1 explains the Kraft inequality.

2.7.1 Kraft Inequality

The Kraft inequality places conditions on the unique decodability of the code words. Recall that the prefix-free requirement describes a code where no code word is a prefix to another and if the code is prefix-free the code is uniquely codable. Kraft adds to this by explaining the sum of all the leaves in a tree cannot exceed 1, as described in (2.10).

$$\sum_{i=1}^{n} 2^{-L_i} \leq 1, \tag{2.9}$$

where L_i is the level of leaf i and n is the number of leaves.

Although the inequality does not guarantee a code is decodable, the code is required to meet the specifications in order to be decodable. From the results of the inequality a few items can be ascertained:

1. If the inequality holds with strict inequality, the code has some redundancy;

2. If the inequality holds with strict equality, the code is a complete code and decodable;

3. If the inequality does not hold, the code is not uniquely decodable; and,

4. If the inequality simply holds, more information is required.

The inequality states that if the sum of the leaves equals one, all the paths to the leaves are used and the code is complete. If we look at an ASCII encoding table the code is complete and the sum is equal to one. A prefix-free code can also have all the leaves equal to one, if the code is complete and each leaf has an integer valued exponent.

The Kraft inequality also states that if the sum of the leaves is greater than one the code is not uniquely decodable and at least one code word has to be a prefix to another. The Kraft inequality is useful to rule out a set of code words that do not meet the prefix-free requirement or tell if the set is complete.

In terms of space the Kraft inequality is a measure of the usage of the symbol space. If the sum of the usages is equal to one the code is complete because each symbol uniquely maps to leaves in a complete binary tree and the symbol space utilized is 100%. If the sum is greater than one, the code violates the prefix-free requirement by containing a symbol or symbols below a previously completed binary tree and, in this case, the symbol space usage exceeds 100%. If the inequality is less than one nothing can be ascertained because a code word could exist that does not meet the prefix-free requirement. The failure to meet the inequality does not state that the code violates the requirement. It means the symbol space usage is less than 100%, so the space requirement is not violated, however a prefixed code could exist.

2.7.2 Shannon-Fano-Elias Algorithm and Example

Knowing the preliminaries for Shannon-Fano-Elias, we can discuss the algorithm itself. The premise behind the Shannon-Fano-Elias algorithm is that the optimal code length is represented by a leaf (i) located at $-\log_2(P_i)$ where P_i is the probability of a symbol in index i. The length equation is the theoretical ideal indicated by Shannon [39]. Unlike the previous two algorithms, which do not directly apply the entropy equations to the problem, Shannon-Fano-Elias uses the theoretical ideal $\mathbb{I}$ to create a compression algorithm. The visual of the Shannon-Fano-Elias algorithm has a leaf which casts a shadow represented by the percentage of the symbol. This premise is illustrated in Fig. 2.7.

The functions developed for the algorithm are formally stated in (2.11)–(2.14). The algorithm uses a function $F(P_i)$ (2.12) to sum all the probabilities previously encountered. The result gives the base where previous division denoted by P_i ended. The results of $F(P_i)$

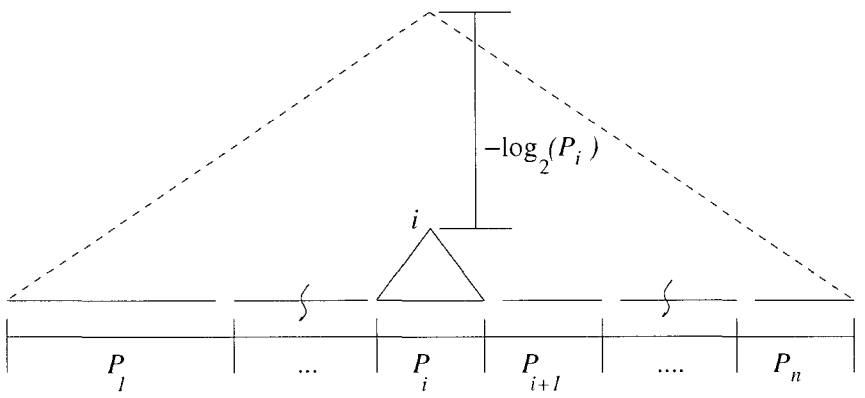


Figure 2.7: Shannon-Fano-Elias' view of the data compression environment. The base of the binary tree is divided by P_i for all i. The value $-\log_2(P_i)$ represents the length in bits to reach P_i.

are subsequently used in a second function $\overline{F}(P_i)$ (2.13). $\overline{F}(P_i)$ uses the previous sum as well as half of the current probability to produce a result which is converted to binary. This addition ensures the code word to be generated contains enough information to coincide with the length function $l(i)$. The binary representation of the sum returned from $\overline{F}(P_i)$ is used for the final code word representing the symbol. A third function $l(i)$ (2.14) uses the `ceiling` of $-\log_2(P_i)+1$ to calculate the length of the code word for the symbol. This length determines the number of leading bits which are used from the binary result $\overline{F}(P_i)_2$.

$$\text{Let } P = 1,2,3,...n, \text{ and } P_i > 0 \text{ for all } i, \tag{2.10}$$

$$F(P) = \sum_{1 \le i} P_i, \tag{2.11}$$

$$\overline{F}(P) = (P_i)/2 + \sum_{1 < i} P_i, \tag{2.12}$$

$$l(P) = \lceil -\log_2 P_i + 1 \rceil. \tag{2.13}$$

The following example uses the same input from the Shannon-Fano and Huffman encoding examples. Five arbitrary symbols: A, B, C, D and E represented by their known probabilities of occurrence (A:35, B:17, C:17, D:16, E:15). The results of each of the sub-equations shown is in the following table Table 2.1

Each of the calculations in Table 2.1 is executed as described in (2.11)–(2.14). $F(P)$ and $\overline{F}(P)$ are calculated in order with the result being converted into binary $\overline{F}(P)_2$. The length $l(P)$ is calculated using the $\log_2()$ and the `ceiling` function. The final code is represented in $code(P)$.

Table 2.1: Shannon-Fano-Elias tabulated results for given data set A:35, B:17, C:17, D:16, E:15.

i	1	2	3	4	5
P_i	0.35	0.17	0.17	0.16	0.15
$F(P)$	0.35	0.52	0.69	0.85	1
$\overline{F}(P)$	0.175	0.435	0.605	0.77	0.925
$\overline{F}(P)_2$	0.00101...	0. 01101...	0.100110...	0.110001...	0.111011...
$l(P)$	3	4	4	4	4
$code(P)$	001	0110	1001	1100	1110

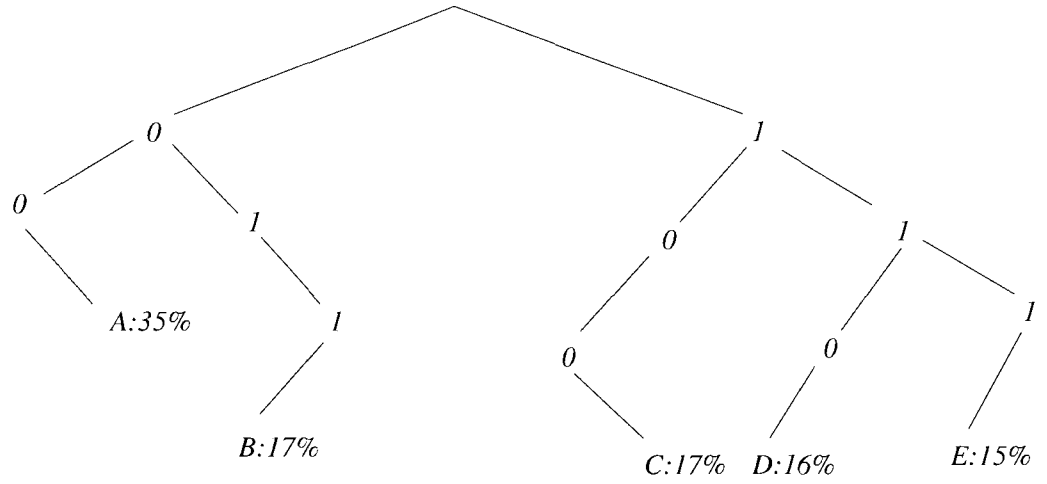


Figure 2.8: Tree created by Shannon-Fano-Elias encoding for comparison with the same values as Shannon-Fano and Huffman encoding.

2.7.3 Shannon-Fano-Elias Encoding Analysis

Compared to Shannon-Fano and Huffman encoding there are a number of calculations required to map symbol percentage P_i. Also, note in Fig. 2.8 that the tree is far from complete and in comparison to either the Huffman Fig. 2.6 or Shannon-Fano Fig. 2.4 encoding it is not as efficient. There is wasted space due to the `ceiling` and the $+1$ that are required for the algorithm to avoid conflict and $\overline{F}(P)$ defines the width. If we compare the entropy values to Shannon-Fano ($L(X) = 2.31$) and Huffman encoding ($L(X) = 2.30$) we can also see that the results are not equivalent. $L(X)$ =(A:35% $\times$ 3) + (B:17% $\times$ 4) + (C:17% $\times$ 4) + (D:16% $\times$ 4) + (E:15% $\times$ 4) $= 3.65$

The purpose of both the `ceiling` and the +1 value in the length equation is to avoid

conflicts in the resulting binary codes and adhere to the Kraft inequality. For example, the series 25, 25, 25, 24, 1 results in the code 00, 01, 10, 11 and 111001 if simply the $-\log_2(P_i)$ is used to place the values. The 11 and the 111001 violate the prefix-free requirement and the Kraft inequality. With the fitting functions the values would resemble 001, 011, 100, 101 and 1111111. The net effect of the operation is to double the base which insures there is enough space to fit the values without any conflicts and the code adheres to the Kraft inequality. The benefit gained by the added space is that symbols can remain unsorted as there is more than enough space to represent them. The combined operations in (2.11)–(2.14) are commonly referred to as the fitting function. This added fitting function has the effect of increasing the overall entropy to $L(X) < H(x) + 2$. The addition of the fitting function also mandates that Shannon-Fano-Elias can never return an optimal compression.

As compared to Shannon-Fano and Huffman encoding, the advantage of Shannon-Fano-Elias coding method is the simplicity of the method as it has few moving parts. The method does not require the data to be sorted as the percentage values can be operated on in any order which results in computational time savings. The main disadvantage of Shannon-Fano-Elias is that the overall compression has a code length of $H(x) + 2$. This is 1 bit longer on average than Shannon-Fano and Huffman encoding. Also, the $\log_2()$ function is moderately expensive computationally with regard to the simple comparison models of Shannon-Fano and Huffman encoding methods.

In summary, Shannon-Fano-Elias is an algorithm based on the theoretical ideal proposed by Shannon. The algorithm divides the space by the symbol percentage P_i in order to map the symbols to the `ceiling` of $\log_2(P_i) + 1$. The `ceiling` and the $+1$ are used to avoid conflict when mapping the symbols based on the Kraft inequality. The approach that influences the algorithm was a divides entropy by the symbol percentage P_i. The end result is an algorithm that achieve $L(x) \leq H(x) + 2$.

2.8 Summary

We examined Shannon's development of entropy and the theoretical ideal. We examined the concepts relationship to information theory and data compression in order to explain the limits imposed on the system to do useful work. In both information theory and data compression the work involved is the unique representation of a code word to a symbol. In data compression an addition requirement is to minimize the length of the code word representing the symbol with the shortest length. The theory of entropy defines the efficiency of the algorithm to remove the excess overhead in terms of bits of the code word. The theoretical ideal defines the ideal limit of the length of the code word to symbol relationship. Both

of these concepts are key measurements and ideals required for any development of data compression algorithms. In this chapter we also examined three prefix-free and lossless compression algorithms for static data: Shannon-Fano, Huffman and Shannon-Fano-Elias.

The following summarizes these key points. Shannon-Fano's algorithm splits the complete data set based on the properties of H from the top. The algorithm stays within $H(x) + 1$, however, the algorithm does not always produce optimal code. The Huffman algorithm's combines the symbols in a bottom-up greedy fashion in order to produces optimal encoding with regards to binary. The approach is also the most expensive in terms of work applied in comparison to Shannon-Fano and Shannon-Fano-Elias. Shannon-Fano-Elias focuses on splitting the data from the bottom and refits to handle the conflicts. The key concept used by Shannon-Fano-Elias is the use of the theoretical ideal $\mathbb{I}$. The use of the theoretical ideal applies directly to entropy theory. The algorithm does not require a sort and is the most computational expensive algorithm of the three examined. The absence of a sort causes the algorithm to be bound by $H(x) + 2$.

Chapter 3

CURRENT ANALYSIS OF CORE ALGORITHMS

3.1 View of the Data Compression Environment and Resulting Algorithms

In Secs. 2.5–2.7 we discuss three of the fundamental algorithms in data compression. Each algorithm compresses data in a unique fashion and each presents a different view of the data compression environment. The analysis of the algorithms in Chapter 2 revealed three entities that are of importance: the first was the concept of the duality of the theoretical ideal in Sec. 2.4.1; the second concept was entropy and certainty in Secs. 2.2–2.4; and the third concept was the definition of the environment as a two dimensional space in Sec. 2.3. In this chapter, we expand upon these topics to determine how the algorithms apply to the theory of entropy, the entropy equation and the theoretical ideal as discussed in Secs. 2.2–2.4.1.

In Sec. 3.2 we further define and clarify the meaning of certainty and entropy as these are the two main components in data compression. This section explains the two types of entropy and their nature within the environment. In Sec. 3.3 we analyze Shannon-Fano's traditional model of the data compression environment and illustrate how this model lead to the algorithm that divides entropy by the certainty of the system. In Sec. 3.4 we examine the Huffman algorithm's lack of a complete depiction of the environment until the completion of the method. This results in a solution based on entropy addition. In Sec. 3.5 we examine Shannon-Fano-Elias and the model of the data compression environment represented by the entropy equation H. The equation and the environment is divided by P_i resulting in a solution based on entropy division.

3.2 Certainty and Two Types of Entropy

The purpose of this section is to define more precisely describes certainty and uncertainty for information theory and data compression. Certainty is defined by a code word to symbol relationship and is the fundamental relationship required by both disciplines. Certainty only exists when a code word resolves to a unique symbol. The code word has to be unique in both the sequence of elements and the length of the sequence. In addition, the end of the sequence must terminate at a single symbol. Any other combination results in an uncertain results. The prefix-free requirement mandates that a sequence resolving to a symbol must

not contain a subsequence of elements that represents another symbol. The subsequence can represent a path to another symbol, but it cannot represent another symbol. Since certainty is defined in the environment as a group of complete code words that represent symbols, we can perceive certainty as only existing at a specific points. The points are the division between the entropy and certainty. This division can be seen as the last node in the sequence of bits representing the symbol. This point also represents uncertainty as it is also a subsequence of a code word. This fact is represented in the duality of $\mathbb{I}$ explained in Sec. 2.4.1. Since certainty can only be represented by specific points, certainty cannot be represented continuously. Given the above constraints, the combination of a unique symbol and a unique code word is the a finite relationship within the data compression environment. Since certainty can only be defined by specific points in the environment all other points in the environment must represent uncertainty or entropy. Entropy is the second main component in the environment.

In contrast to certainty, uncertainty is continuous in both the symbols space (width) and the bit space (length). The bit space is continuous even though the actually bit values cannot represent the continuous nature. This relationship is exemplified in the difference between the theoretical ideal $\mathbb{I}$ which is continuous and the actual binary lengths obtainable by the system which are finite. We know from the properties of H that the symbol space is continuous. Since the data compression environment is the combination of the bit and symbol space, uncertainty represents all of the space in both dimensions.

Entropy in information exists in two types. The sub-sequence of a code word that represents a symbol is one type of entropy. This entropy is measured in the entropy equation. The second type of entropy is when a sequence does not terminate at a code word. This form of entropy is not currently measured. Although we can obtain this measurement by comparing to the theoretical ideal the measurement is not entirely accurate as the theoretical ideal cannot be represented in a real system. Multiple code words can resolve to a single symbol without affecting the overall entropy as long as the sequence is unique and only resolves to one symbol. The ability to have multiple code words resolve to a symbol uses the second form of entropy to produce redundant code words to represent a single symbol. It maybe necessary to develop a method to measure the second type of entropy as it would actually represent the under utilization of the actual system.

Given the definitions of entropy and certainty, the act of assigning information (entropy) to certainty (the system) defines the transmission and storage. The added constraint for data compression with the combination of the definitions of certainty and entropy make the solution of data compression problem a minimal mapping of the uncertainty of the data to the certainty represented by the system.

3.3 Shannon-Fano's Traditional View and Entropy Division by Certainty

Shannon-Fano's algorithm is based on the traditional model of a binary tree. The model is based on repetitive expansion of the representative symbol space based on powers of 2. The expansion starts at a singular node referred to as the root node. The root node has the option to spawn up to two nodes referred to as children and the root becomes the parent. Each child has the option to spawn up to two children of its own or can terminate the expansion of the tree as a leaf node. The progression builds a tree's ancestral hierarchy from the root to the leaf nodes. The number of expansions from the root can be represented by the $\log_2()$ function which returns the inverse of the expansion.

Shannon-Fano's algorithm maps directly to this model by starting with a single root node representing all of the symbols. The algorithm doubles the symbol space available and allocates the symbols to each half of the newly expanded spaces based on the symbols combined percentage. The algorithm terminates when the space available in the leaf nodes is equal to the number of symbols (one leaf and one symbol). The condition where one leaf node relates to one symbol represents certainty and is denoted by the sequence and length of a code word following the path to certainty. The Shannon-Fano's model not only expands the space, but it also divides the entropy associated with the grouping of all the symbols. Recall, there are two types of uncertainty. One type is where the symbol does not exist and the other case is where a single symbol is not discernible within a group of symbols. To explore this relation further we must examine the algorithm's relation to the entropy equation itself.

The Shannon-Fano algorithm starts with all the symbols grouped at the root node and this condition represents 100% entropy with all of the symbols represented simultaneously in one node. The process continues by recursively dividing the uncertainty by the certainty represented by the node's probability of occurrence. In this case the probability representing certainty is not the probability of a symbol P_i as all the symbol probabilities are combined and represent uncertainty. The certainty for the Shannon-Fano algorithm is represented in the binary tree's structure. In the binary tree the bit sequence and length represents the path to the leaf, sequence terminating node, that will represent the symbol and the certainty of a result. The relationship has certainty represented by the nodes contained within the tree. Each node has a probability P_c of occurrence based on the length and width of the tree or sub-tree. P_c is represented by the percentage located in the nodes in Fig. 3.1. The length and width represent the bit sequences possible given the tree area defined by the bit/symbol space. For the algorithm P_c represents certainty when the code word terminates at the leaf node containing P_c.

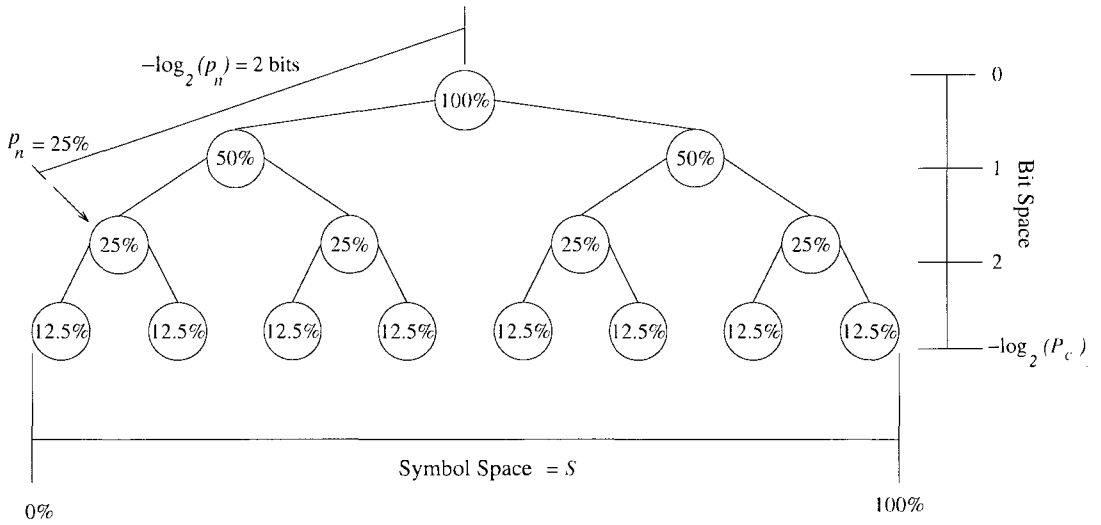


Figure 3.1: Shannon-Fano's view of the entropy space. The recursive division of the symbols places P_i at P_n and ideally $P_i = P_n$. The percentage in the node is the probability P_c, which represents the certainty of the node n occurring within the bit/symbol space.

The knowledge of P_c is the basis for the algorithm as it splits the sum of the probabilities even though the concept of P_c is not directly utilized. The algorithm endeavors to match P_i to an equal P_c by dividing the $\sum P_i$ by 2. The sum of the symbol probabilities $\sum_{i=0}^{n} P_i$ represents uncertainty. The recursive division of the partial sums of P_i representing uncertainty divided by the certainty represented by P_c continues and builds the bit sequence and length of the code word to represent P_i. In terms of the entropy equation, the Shannon-Fano process recursively divides the uncertainty until certainty is represented by a symbol in a leaf node.

To represent this change mathematically, we will use C to represent certainty where $C = -\sum P_c \log_2(P_c)$. This equation represents the certainty inherent in the definition of the binary tree. In Fig. 3.1 each node contains a percentage P_c. The root node represents 100% certainty by definition. The two children of the root node represent half of the certainty of the parent as each child represents half of the possible bit sequence from the previous expansion of the tree. For example, if the left child represents 0 then the right child represents 1 and combined they represent 100% of the possibilities 0 and 1. Note: Since we are working with the definition of certainty, each child node represents half of the parents probability, however, when working with uncertainty P_i the resulting children may not represent exactly half of the parent probability.

Now that we have an equation to map certainty and we know Shannon-Fano is based on entropy division, we can divide the uncertainty by the certainty equation and reveal the process in those terms. At each step of the algorithm the partial $\sum P_i$ is located at P_c, there-

fore the number of bits for both the symbol and the certainty is defined by $-\log_2(P_c)$. The substitution of $-\log_2(P_i)$ with $-\log_2(P_c)$ creates the defining equation of the relationships between certainty and uncertainty as $(-\sum P_i \log_2(P_c)))/(-P_c \log_2(P_c))$. The $-\log_2(P_c)$ function cancels which leaves $\sum P_i/P_c$. This division of the entropy, $\sum P_i$, by the certainty, P_c based in the division by 2 continues until one symbol is represented in one leaf node. If P_i is equal to P_c the mapping is optimal. Unfortunately, there is no requirement to have $P_c = P_i$ which leads to the number of bits used $-\log_2(P_c)$ at the final location not equal to $-\log_2(P_i)$. This inability to map to the correct length leads to less than optimal results in terms of $H(x)$. By mapping the symbols to the tree using this concept Shannon-Fano is dividing the combined symbol percentages recursively in two without regard to the bit space as defined by the $-\log_2()$ function. The disregard given to the $-\log_2()$ function relates to a lack of optimality in some conditions.

In summary, the relationship between the traditional view and Shannon-Fano is how they both relate to an expanding environment. They both start at the root node and expand to same termination point where there is a one-to-one mapping between a leaf in the tree and a symbol. This corresponds to certainty and entropy as the divisions of certainty are represented by the nodes and the uncertainty of the symbol combinations is recursively divided until the symbol is represented in the leaf nodes. Since the Shannon-Fano algorithm divides the uncertainty by 2 at each node and the certainty is divided by 2 at each node the result is a near optimal solution of a symbol's certainty P_i mapped to the leaf node representing certainty. The overall effect is that uncertainty decreases as the tree expands and terminates in certainty with a single symbol in a leaf and the path representing the complete code word.

3.4 Huffman and Entropy Addition

Huffman did not derive his algorithm directly from information theory, but his algorithm does adhere to the equations for entropy and the theoretical ideal. In Sec. 3.1 we analyzed the view of Shannon-Fano and determined it was a expansionist view which recursively divided the entropy over the certainty of the tree. Huffman encoding does not have a complete view of the tree until the algorithm is complete. Huffman's encoding model is local to only individual symbols or sub-trees represented by the value of the symbols contained in the sub-tree. In order to analyze Huffman encoding in terms of entropy we must reduce the H equation to consist of only two terms.

Since the algorithm is always grouping the symbols in groups of two, it is essentially reducing n values to 2 when calculating the P_i for a given symbol. This reduces the problem

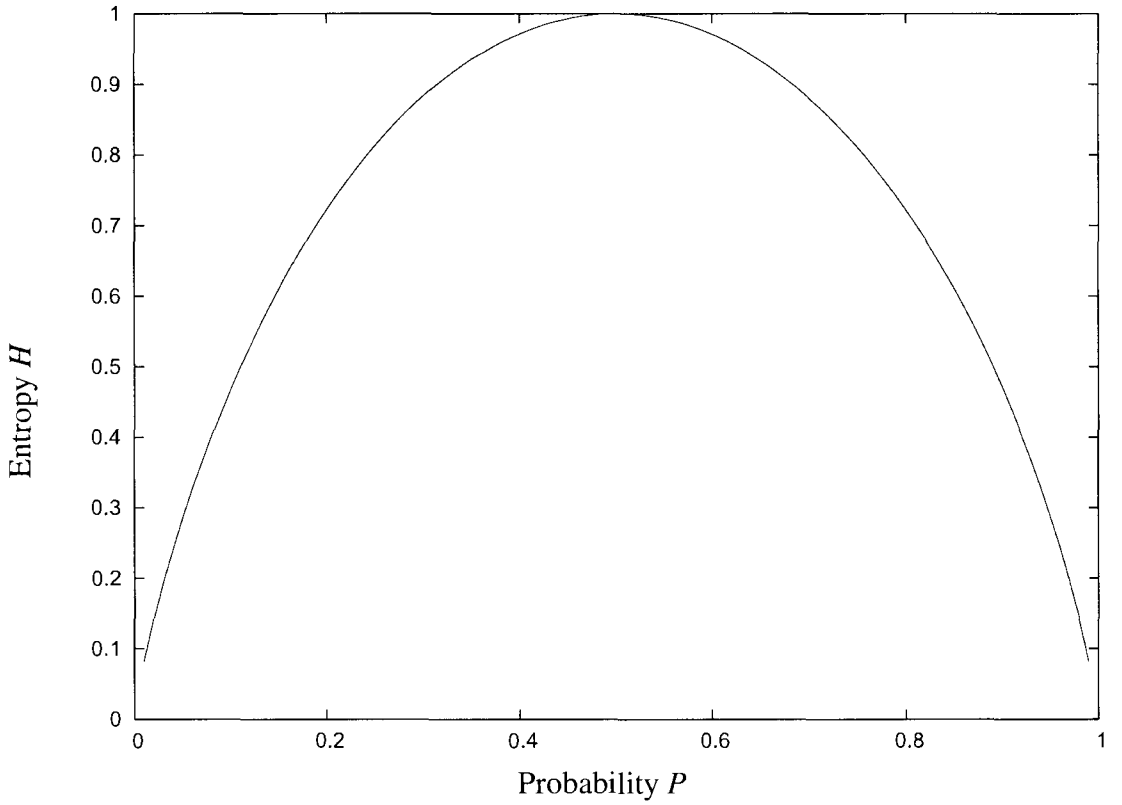


Figure 3.2: Graph of two probabilities using the entropy equation $-(P\log_2(P) + (1-P)\log_2(1-P))$ demonstrating the relative growth of entropy H measured in bits. The system consists of two possibilities P and $(1-P)$, as certainty increases for P the entropy decrease to 0. As P decreases in certainty $1-P$ increases in certainty.

to a P and $1-P$ problem as used by Shannon to describe the interesting properties [39] of H. The graph of the concept is replicated in Fig. 3.2.

Figure 3.2 displays the H in bits, where $H = -(P\log_2(P) + (1-P)\log_2(1-P))$. As Shannon explains the most uncertain situation occurs when the P_is are all equal and this condition exists when each symbol has the maximum distance from the root node or certainty. In the graph this condition is represented at the top of the curve where H is equal to one. At this point both P and $1-P$ have a certainty of 50% and an uncertainty of 50%. The combined uncertainty is 1.00 or 100%. To minimize uncertainty one of the two probabilities must approach 100% and H approaches zero.

The graph of two probabilities applies to Huffman encoding since it is only dealing with two symbol counts. The algorithm ensures that the P_i values in the theoretical ideal are minimum by selecting the two minimum counts. The $P_i\log_2(1/P_i)$ returns the maximum values of all possible values of the H equation. Since each of the values is a local maximum it ensures that the entropy equation for the two is the maximum as displayed in

the graph. This is important because the algorithm is building the tree bottom up and optimal compression requires that the symbols with the most entropy be located at the bottom of the tree. The algorithm does not apply any of this because the $\log_2()$ function is used to calculate the number of expansions required to fit the values in the entropy equation, i.e., the length. Since we are only dealing with two values, the length is one, so the calculation is not needed.

The only thing left to simplify in H for two values is the probabilities which represents symbol count Sc divided by the total symbol count TSc, i.e., Sc/TSc where TSc is the sum of the two symbol counts. Since the algorithm is only dealing with two values the probabilities can be reduced to only the symbol counts for the algorithm. If we look at this step mathematically, the total symbol count in the probabilities would simply cancel in a comparison between two values to reveal the counts themselves. For example, $2/3 <> 1/3$ in a comparison is equivalent to $2 <> 1$ as far as the outcome is concerned.

This simplifies the equation to a comparison between two values as described by the Huffman algorithm. Since the algorithm is selecting the minimum valued symbols, it is selecting the symbols representing the greatest entropy. The selection of the two minimum values also means that the values are as close to equal as possible, and the selection ensures that the width is also the minimum for each combination. By selecting them in pairs the algorithm is balancing the relationship between P and $P-1$ as depicted in Fig. 3.2. Therefore, the selection of the minimums at each step ensures both the length and the width of the resulting subtree is minimal. The combination of the choices results in the complete algorithm.

Previously, we understood Huffman encoding as selecting the two minimum symbol counts, but the above explains the selection in terms of entropy. By canceling out the terms in the entropy equation in the above discussion we can see that Huffman encoding adheres to entropy ideals. The algorithm is actually selecting the maximum entropy values at each decision and building a tree that models the entropy equation precisely. The greedy selection of the maximum entropy values ensures that the final codes are optimal.

3.5 Shannon-Fano-Elias' Entropy View and Entropy Division by Entropy

In Sec. 3.3 we analyzed the view of Shannon-Fano and determined it was an expansionist view which recursively divided the entropy over the certainty of the tree nodes. In Sec. 3.4 we analyzed Huffman's algorithm and determined it builds a tree bottom up with all the sub-trees containing maximum entropy. The third approach, Shannon-Fano-Elias, develops a concept based on an already existing tree which contains both entropy and the

symbols within the space. The space is defined by $-\sum P_i \log_2(P_i)$ directly from the theory of entropy to represent both entities. Recall from Sec. 2.7 that the Shannon-Fano-Elias approach divides the space by P_i. In Sec. 2.4 we broke down the complete entropy equation into $H_i = -P_i \log_2(P_i)$ which represents the individual entropy summation values within H. Dividing this space by P_i equals $H_i/P_i = (-P_i \log_2(P_i))/P_i) = -\log_2(P_i)$. The division reduces the equation to a vector containing a sequence of bits. In Sec. 2.4.1 we observed that $-\log_2(P_i)$ has the duality of representing both information and the uncertainty. The information is represented by the full length of $-\log_2(P_i)$, the sequence of bits and the symbol itself. The entropy is represented by the distance represented by $-\log_2(P_i)$ from the root to the symbol. By dividing the equation by P_i the algorithm is extracting the information and the entropy from the space. The additional requirements of taking the `ceiling` and the $+1$ are simply to avoid conflicts in uniquely mapping a symbol to a leaf. The conflicts are related to the algorithms' disregard to the symbol space interactions. Recall, that the symbol space is on the horizontal and the divisions used by Shannon-Fano-Elias are based on the vertical component represented by the $-\log_2(P_i)$ bits. Equation (3.3) summarizes the mathematical process:

$$H_i = -P_i \log_2(P_i), \tag{3.1}$$

$$H_i/P_i = -P_i \log_2(P_i)/P_i, \tag{3.2}$$

$$H_i/P_i = -\log_2(P_i). \tag{3.3}$$

In summary, Shannon-Fano-Elias is a approach to data compression that separates the symbols, represented by P_i, from the environment, represented by $-\sum P_i \log_2(P_i)$. In the process the algorithm is left with $-\log_2(P_i)$ as a vector defining the distance in bits representing entropy and the location of the leaf node representing certainty. The fitting function avoids conflicts between symbols being mapped. Of the three base algorithms this is the only one to apply directly to entropy theory by using the theoretical ideal. The use of entropy theory provides the algorithm the structure to build the relationships between entropy and certainty.

3.6 Summary

In this chapter we re-examined the three prefix-free and lossless compression algorithms for static data: Shannon-Fano, Huffman and Shannon-Fano-Elias. We started the discussion by examining certainty and the two types of entropy. Certainty is defined as a finite point denoted by a code word representing the symbol and the symbol itself. The first type of entropy is represented by the code word length and the subsequence of bits defining

the code word in order represent the symbol. The subsequence of bits cannot represent another symbol, but the subsequence can represent a path to other symbols. The first type of entropy is measured by the entropy equation H defined by Shannon. The overlap of certainty and the first type of entropy results in the duality of $\mathbb{I}$ in Sec. 2.4.1. The second type of entropy occurs when a code word defined by the system does not resolve to a symbol. The second type of entropy is not currently measured although an approximation is possible the exact measure is not clearly defined. The definitions make it clear that the focus of data compression is the mapping of the entropy of the data to the certainty of the system with minimum overhead. All data compression algorithms try to accomplish this act. We continued the discussion using the guidance of the definitions and the theory of entropy discussed in Chapter 2 to analyze the base algorithms in more detail.

By examining the algorithms we observed that each of the three had a unique model of the compression environment and their models influenced how each algorithm was developed. The three algorithms interactions with entropy was also examined and we illustrated how each algorithm applied to the entropy equation H and the theoretical ideal $\mathbb{I}$ as defined by Shannon. The following summarizes these key points.

Shannon-Fano's use of certainty, defined by the binary tree and the binary tree's nodes, to divide the entropy represented by the combination of symbols. This algorithm resolves to optimal results when $P_i = P_c$. The use of certainty gives the ability to speed up the data compression time by having a predefined structure to compare the uncertainty of the combined symbols to the certainty of the structure. The use of a predefined structure is of great importance for the topic in Chapter 4.

The Huffman algorithm's focus is the most complete with regard to entropy as it starts at the bottom and builds a minimum length code that is optimal with regards to binary. Huffman encoding shows that the ideal approach for accomplishing optimal compression should be conscious of both the length and the width of the tree with regard to the entropy equation when forming the solution. To accomplish this Huffman encoding requires the building of all the sub-components in terms of entropy and then builds the entire structure of certainty to define the code word to symbol relationship. Since Huffman encoding has no concept of predefined certainty or a predefined structure, this approach is the most expensive of the three in terms of work applied.

Shannon-Fano-Elias focuses on splitting the data from the bottom and refits to handle the conflicts. The algorithm does not require a sort and is the most computational expensive algorithm of the three examined. The key concept used by Shannon-Fano-Elias is the use of the theoretical ideal $\mathbb{I}$. The use of the theoretical ideal applies directly to entropy theory and provides the algorithm the structure to build the relationships between entropy and

certainty. As with Shannon-Fano the use of a structure to define certainty gives Shannon-Fano-Elias the advantage in terms of asymptotic times as it requires only $O(n)$ operations.

From the previous algorithms it is clear that the development of a data compression algorithm is tied directly to the view of the environment consisting of two sub-environments: certainty and uncertainty. Certainty is defined by the structure used to facilitate the code word to symbol relationship. Uncertainty is inherent to the symbols or the combination of symbols and the purpose of information theory is to define the relationship between certainty and uncertainty. Data compression adds the additional requirement of minimizing entropy or the uncertainty and the theoretical ideal defines the limit between the two interactions.

Chapter 5 explores the views held by the previously discussed algorithms in more detail. The analysis gives greater insight into the specifics defining the view and how the view's focus either limits or exemplifies the characteristics of the environment. Both of the sub-environments are depicted in a similarly defined space in order to see the relationship between entropy and certainty. Chapter 5 uses a novel illustration to address some of the complexity of the problem of data compression and produces some new methods to accomplish this task.

Chapter 4

MAPPING OF CERTAINTY AND ENTROPY

4.1 Illustrating Certainty and Entropy

In Chapter 2 we introduced the concept of entropy as defined by Shannon and how it relates to the field of data compression. We also examined three different algorithms for prefix-free and lossless encoding of static data. In Chapter 3 we further categorized certainty and the two types of entropy within the data compression environment. We analyzed each of the algorithms data compression model and how the model created differences within the three algorithms. We also analyzed how the algorithms applied to the entropy equation proposed by Shannon and the certainty of the system to produce the encoding.

We continue this discussion in Sec. 4.2 focusing on the development of a model that describes both entropy and certainty in the data compression environment. Sec. 4.2 examines the illustrations of Huffman, Shannon-Fano and Shannon-Fano-Elias in greater detail in order to extract the benefits and drawbacks the depictions have describing both certainty and entropy. The analysis leads to Sec. 4.3 which defines the requirements and describes the details of a new model illustrating the certainty and entropy of the binary system. Section 4.4 uses the new model to map the area representing entropy within the space. We describe entropy by the dimensions defined by the H equation and determine the orientation of entropy in the bit/symbol space. Sec. 4.5 combined both certainty and entropy in the depiction of MaCE. The purpose of the combination is to allow the mapping of entropy to certainty and certainty to entropy for the purpose of data compression. An example of the use of MaCE is supplied in Sec. 4.6.

In Sec. 4.7 we add the symbol to MaCE and describe how it relates to the entropy and certainty of the system. We show how to calculate the space used by the symbols in both the bit and symbols spaces. In the discussion the topic of expansion of the base is explored and the concept is utilized to describe compression based on base manipulation. The mapping of certainty and entropy illustrates that it is possible to compare entropy to certainty and certainty to entropy for the purpose of data compression. In addition, the use of MaCE and the mapping of the symbol illustrates the use of base manipulation to increase this compression.

4.2 Analysis of the Previous Models

Vision is one of the fundamental ways that we perceive the world and the choices we make when navigating an environment. When looking at a problem from the top, we are looking for a path to the bottom. When looking at the problem from the bottom, we are looking for a path to the top. When looking at the problem from the side, we have a choice to go up or down from the location. In the real world the choice is usually dictated to us. In contrast, in the virtual environment we have a choice on where to start and how to navigate through the environment to get the desired result. In the following Secs. 4.2.1–4.2.3, we analyze the bottom-up (Huffman), top-down (Shannon-Fano), and sideways (Shannon-Fano-Elias) approaches to compression.

4.2.1 Analysis of the Previous Model: Huffman

Huffman's model focuses on only one sub-tree consisting of two nodes at each step of the algorithm. The resulting algorithm builds the tree adhering directly to the entropy equation with each sub-tree being optimal. This algorithm simplified the H equation by reducing $-\log_2(P_i)$ to 1 and the probabilities P_i to the symbol counts. The reductions allowed the development of an encoding algorithm to build a optimal tree with minimum entropy from the bottom up. The major benefit of this approach is the production of optimal results, but the algorithm is the most expensive of the three compression algorithms in terms of work applied.

The problem is the total entropy environment is not clear until the algorithm is complete. Although we could map the entropy on the resulting space created by the algorithm, we cannot map the entropy for the general case based on this model. The lack of a complete depiction of the entropy equation makes it difficult to visualize the interactions the bits and symbols have with entropy except in the micro environment containing two symbols and a single bit representing two options, 0 and 1, as illustrated in Fig. 3.2.

4.2.2 Analysis of the Previous Model: Shannon-Fano

The Shannon-Fano approach uses the decomposition of a choice defined in properties of H as it models the uncertainty created by the combination of symbols to the certainty of the binary tree. Shannon-Fano's model consists of all the symbols combined at the root and the expansion of the tree creates the necessary leaf nodes to represent the symbols. The resulting algorithm divides the symbols in half using an expansionist view to determine the required divisions of the symbols. The algorithm terminates when a single symbol exists in a leaf node. The traditional depiction is represented in Fig. 4.1.

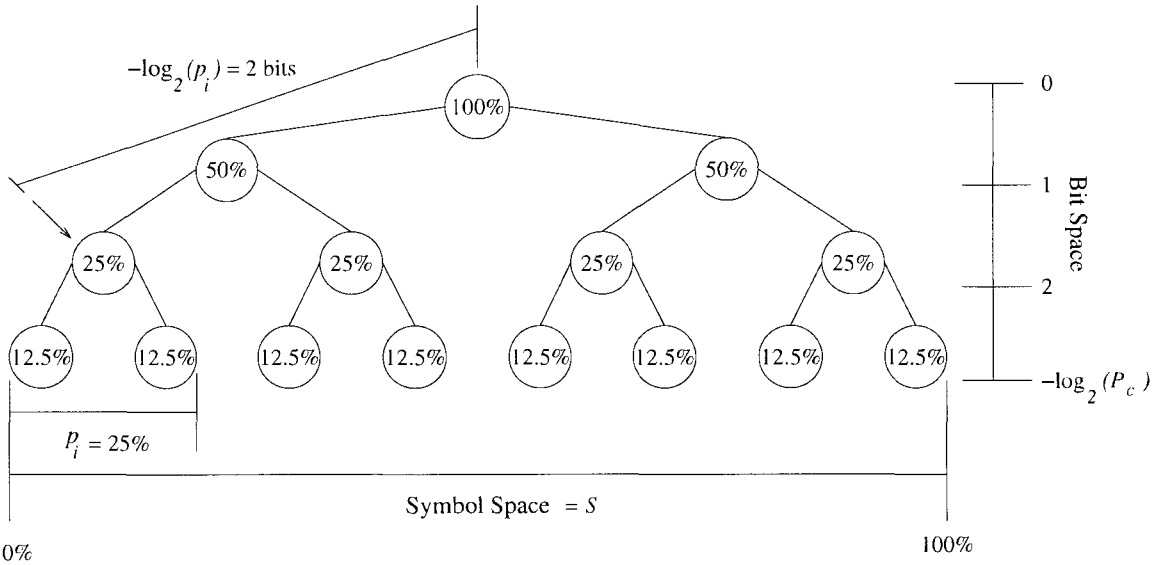


Figure 4.1: Traditional depiction of the binary tree. The nodes represent the possible symbol locations, which is important for the algorithm, but the model does not represent entropy very well. The depiction shows the horizontal divisions clearly but the continuous nature of the symbol space is not represented.

Shannon-Fano's model and the traditional tree are applied in tandem to recursively divide the uncertainty of the combined symbols over the certainty represented by the structure. When describing the Shannon-Fano algorithm this relationship is advantageous as it allows for an easy visualization without the complexity of the description as required in the previous discussions. The additional use of the certainty of a tree percentage P_c in the description is also a key benefit as it gives a framework to map entropy.

The problem with this perspective is that it does not show the complimentary nature of the bit and symbol spaces clearly. In the traditional model the bit space is on a diagonal and the symbol space is horizontal as illustrated in Fig. 4.1. With the addition of a scale the divisions of the vertical space (bit space) can be clearly defined as rows of nodes, but the divisions on the horizontal (symbol space) at each level are difficult to determine with the nodes diagonal from the parent.

The circular nodes of the traditional model are a good way to represent individual symbols for the Shannon-Fano algorithm as a one-to-one mapping is required by the algorithm. The problem with the circular nodes is that they divide the symbol space leaving room between them. This is a problem because the symbol space is a continuous entity from 0 to 1.0 and the continuous nature cannot be clearly visualized with the circular nodes. Another drawback to the one-to-one mapping is the inability to illustrate the effects of the space used by the symbol other then the node in which it is placed. The symbol uses both the

node and the preceding nodes representing the code word. The code word sequence may have redundant paths, but with the traditional approach the paths from node to node are displayed only as a single line. The problem with the single line representation is that redundant paths to various symbols are not represented in the visualization. This is important as the paths represent the entropy created by mapping a symbol to a leaf node.

4.2.3 Analysis of the Previous Model: Shannon-Fano-Elias.

Shannon-Fano-Elias' model is based on the entropy equation where the space is defined by the symbol probabilities P_i (width) and $-\log(P_i)$ (length). The resulting algorithm divides the equation by P_i to produce code words based on $-\log_2(P_i)$. The division by P_i separates the data compression environment into vectors representing the bits space for each symbol. The division of the bit space is in contrast to Shannon-Fano which divides the symbol space. The perpendicular nature of the division results in Shannon-Fano-Elias having the opposite problems in comparison to the traditional model in defining the space. Shannon-Fano-Elias' depiction is represented in Fig. 4.2.

The Shannon-Fano-Elias' model is advantageous at describing the continuous nature of the symbols space. The visual allows one to see the leaf to P_i relationship described by the algorithm. In addition, the divisions on the horizontal are clear as they separate the symbol space based on the percentages of the symbols. The perpendicular nature of $-\log_2(P_i)$ function to the symbol space is also present but only exists for one symbol.

The problem with the model is the perpendicular nature of the bit and symbols spaces for the entire data compression environment is not clearly shown. Although it is possible to draw multiple $-\log_2(P_i)$ functions within the visualization that was not the point of the design. The creation of the individual vectors also removes the path redundancies required to show entropy interactions. This makes it impossible to see the interactions between the symbols and the code words within the environment. In addition, the division of the vertical space is not clearly represented which makes it difficult to define the depth of a node in the Shannon-Fano-Elias model.

4.3 Map of Certainty and Entropy (MaCE) of the Symbol and Bit Space

Each of the models in Secs. 4.2.1–4.2.3 has benefits describing the algorithms they represent, and each has drawbacks when trying to describe the other algorithms or the entropy space as a whole. The purpose of this section is to design a more comprehensive model that illustrates the entropy of the data and the certainty of the system using a combination of the previous models. In Sec. 3.2 we defined certainty and two types of entropy. Since certainty

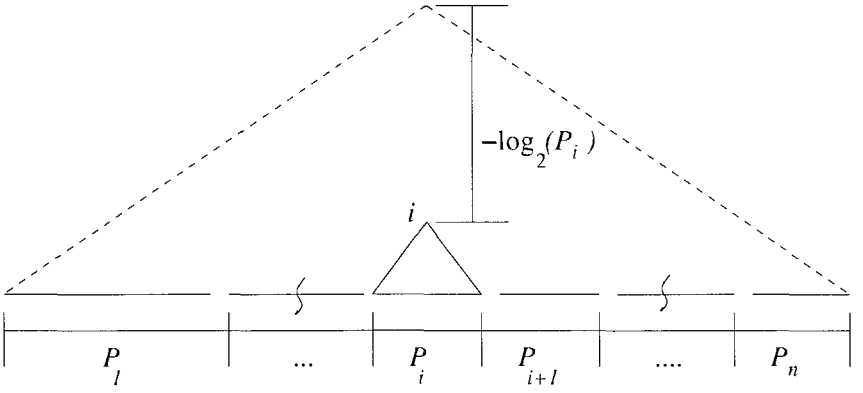


Figure 4.2: Shannon-Fano-Elias' model of the data compression environment. The model works well to illustrate the continuous nature of the symbol space and the use of $-\log(P_i)$ to represent a symbol. The model does not show the perpendicular nature of the space or the placement of the actual symbols.

and uncertainty are actually defined in two different spaces, entropy for data and certainty for the system, there are two goals to accomplish in the development of the model for the purpose of data compression. One is to represent certainty defined by the system in a visual representation. The second is to define an equivalent visualization to define entropy. Once both visualizations are equivalent we can relate entropy to certainty or certainty to entropy when desired.

We begin by combining the models of Shannon-Fano and Shannon-Fano-Elias to maximize the benefits they have at describing the space. A combined depiction would clearly define the divisions for both the symbol space (horizontal) and the bit space (vertical). In addition, we need a mechanism to show the space available and the space used in the bit/symbol space. The mechanism describes both the entropy of the data and the certainty of the encoding environment. Recall the duality of the symbol representation occupies both the symbol space and the bit space. The duality of the symbol representation in both needs to be represented in order to analyze the interactions between the two dimensions. The code word to symbol relationship needs to be readable. The code word represents both the path to the symbol and the entropy, so this relationship needs to be visible. The dimensions for both the length and the width of the area must be visible and the dimensions must be perpendicular to conform with the H equation. If all the requirements are met in the visualization of the data compression environment, the end result will be a model of the symbol and bit space that allows for a full description of the entropy and certainty interactions. This model of the data and the system is named MaCE, the Map of Certainty and Entropy.

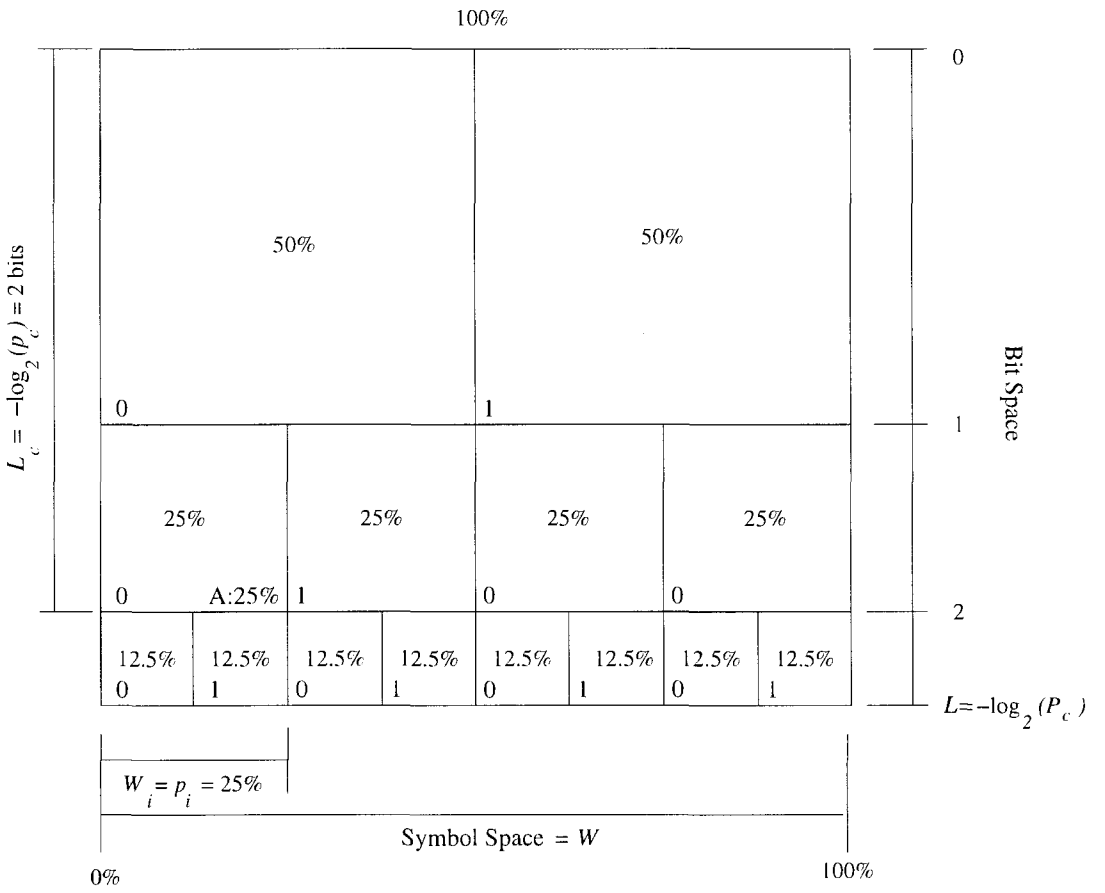


Figure 4.3: MaCE with percentages. The percentages, P_c, represent the space available or used in the node location. The divisions of the vertical space represent the individual bits. The divisions on the horizontal represent the space available to represent the symbols.

4.3.1 Design of the Map of Certainty and Entropy

The essence of the design of MaCE is based on a two-dimensional representation of the relationships between certainty and entropy. The use of a two-dimensional depiction is not very different from the two-dimensional representation of a binary tree except all the dimensions are perpendicular based on the mathematical description of entropy space. MaCE as displayed in Fig. 4.3.

The vertical dimension represents the length L of the bit space and is divided to represent the individual bits. The combination of bits in a vertical division from top to bottom are used to represent a symbol given a specific level L_i, where i identifies a specific symbol. The length $L_c = -\log_2(P_c)$, where P_c is a division of the width of the symbol space. The relationship between length and width of the space also has L representing the number of expansions from the base. This is the same as the traditional model, except the expansions are denoted by the lines dividing the space at each level as the total width remains constant.

In Fig. 4.3, the bit value is displayed in the lower left corner of the divided space. This assignment may not be necessary depending on the algorithm implemented, as the order can be assigned based on the percentage of the space used, similar to the Shannon-Fano-Elias approach.

The horizontal dimension represents the symbol space, and the width always equals 100%. The horizontal dimension contains 2^L squares at each level representing the expansion described by the number of level L in the vertical dimension. The squares represent the nodes from the traditional model and the leaf nodes which can be used by the symbols.

The product of the two dimensions represents the possible binary words that can represent a symbol. Recall that $-\log_2()$ represents the bit space and the sum of all the individual probabilities P_c represents the symbol space. Since the symbol space is always represented by 100% at each level the change between each level is in the number of divisions that represent the possible nodes. This represents the expansion of the base and the division of the width simultaneously by expanding the number of divisions of the width.

In Fig. 4.3, the symbol is located in the lower right hand corner of the symbol square if the square is occupied, and the symbol's percentage is located next to the symbol, representing the percentage of the symbol space the symbol would ideally represent. A symbol is thus represented in both the vertical and the horizontal spaces by a square. Both of these dimensions are required to represent certainty of a location in the ideal tree.

As noted, inside each square is a percentage P_c which represents the space available or used by the symbol. Recall from Sec. 3.3, that P_c represents the certainty defined by the code word representing the symbol at that location. This is appropriate because the map we are defining is of certainty and we will be mapping uncertainty (entropy) onto this space. The percentage is equal to 2^{-L} for each of the squares at level L.

For example, 25% represents a square that can ideally host a symbol of 25%. If we analyze this in binary the square represents 25% of the 2^L possible combinations of the code word, where $L = 2$. The possible code words are the four two bit numbers $(00, 01, 10, 11)$.

The representation can be broken down into individual component to represent individual symbols. L_i represents the length of the code word to represent symbol denoted by i. $L_i = -\log_2(P_i)$ in bits. W_i can represent the width of the symbol denoted by i, however W_i is equal to P_i so this is not usually necessary. The bit values denoted in the lower left corner of each square are combined from top to bottom within the space used by i to create the code word.

For example, symbol A is equal to $W_i = P_i = 25\%$ and is placed in the space on $L = 2$ in the first square representing $P_c = 25\%$. $L_i = -\log_2(25\%) = 2$ bits. The hatched area represents the space used within the bit/symbolpace to represent A. The bit values in the

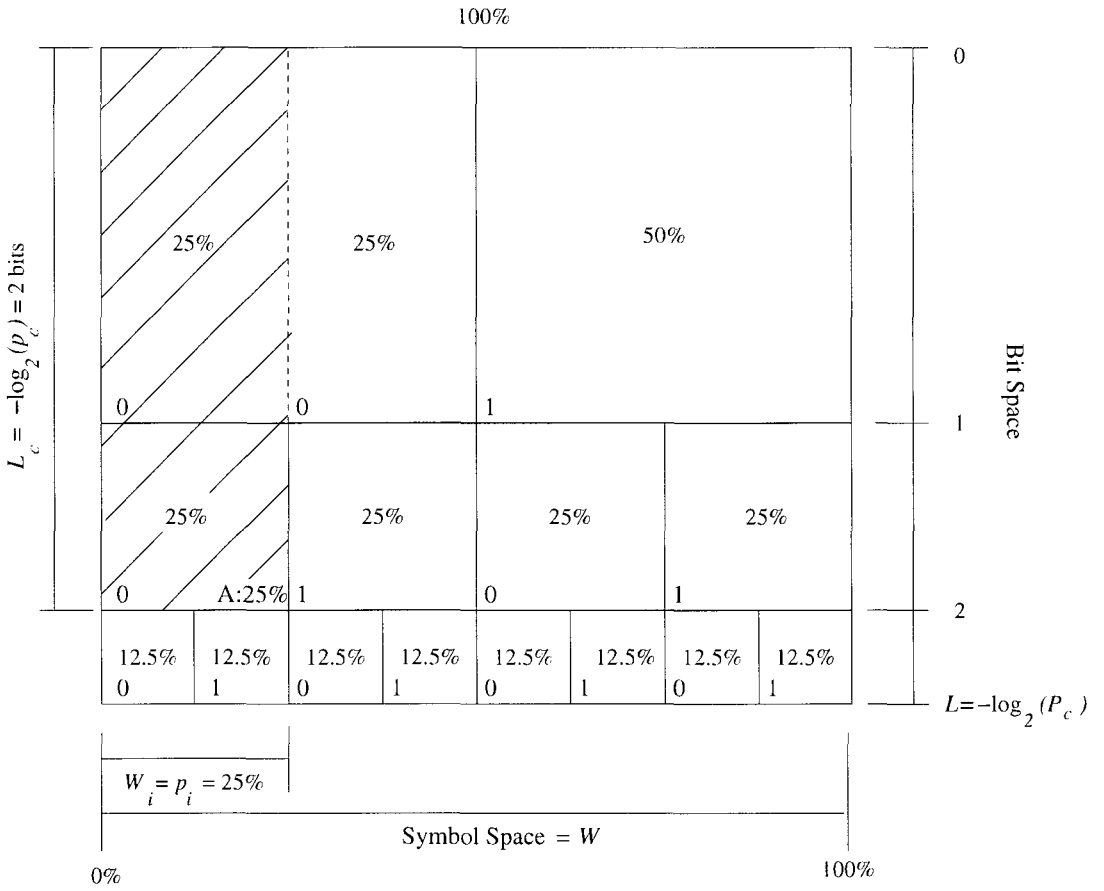


Figure 4.4: MaCE example with percentages. The hatched area represents the space used to code 'A' in both the bit space and the symbol space. $A : 25\%$ uses the square $P_c = 25\%$ represented by code word 00.

hatched region represents the code word, in this case 00 will identify A. The percentage in the preceeding squares is reduced to show the space used in that square and a second percentage is added to represent the space remaining, in this case the square represent by partial code word 0 has 25% used and unused, as illustrated in Fig. 4.4

The design of MaCE takes advantage of the combination of the Shannon-Fano and the Shannon-Fano-Elias models along with the additional requirements to represent certainty and entropy in data compression. The design of MaCE has the advantage of describing the length and width of the data compression environment and all the information required to see the entropy and certainty interactions within the map. A quick look at MaCE shows all the information available in regards to space and the information stored in the space, making it easier to analyze the entropy and certainty interactions within the space. MaCE describes the horizontal and vertical divisions cleanly. The hatched area provides the mechanism for showing the space available and the space used in the bit/symbol space. Further

advantages of this construction will be made evident when we begin to map entropy onto the two-dimensional space. Before the entropy mapping is possible, we must figure out the appropriate orientation by which to map the entropy equation H to MaCE.

4.4 Entropy in Two-Dimensional Space

Previously, we have been discussing the results of the H equation as bits of entropy. Looking back at Example 2.5 we observed that entropy is defined by each $H_i = -P_i \log_2(P_i)$, representing the space used at the level in the tree defined by $-\log_2(P_i)$ and the symbol P_i. This observation represented the percentage of the symbol space used by the color c. In Sec. 2.4 we noted all the bits in the code word represents entropy as referencing anything but the complete code word resolved in uncertainty. This would seem to indicate that entropy is also defined in the bit space. This apparent contradiction exists because entropy exists in both spaces.

Entropy exists in the symbol space as none of the symbols represents 100% when multiple symbols exist. The complete symbol space consists of a combination of the all symbols probability P_i with a range from 0 to 100%. Entropy H is continuous in P_i and this combination of all P_is defines the symbols space as a continuous entity from 0 to 100%.

Entropy exists in the bit space because only the complete code word resolving to a symbol represents certainty. All other variations of the code words available within the bit/symbol space resolve to an uncertain result of either no symbol or too many symbols as described in Sec. 3.2. Entropy is continuous in the bit space as ideally the code words would be represented by the continuous function $-\log(P_i)$, however the actual system representing certainty is finite and discontinuous.

We examine Fig. 4.3 to determine how to represent entropy in terms of both spaces, in which the function $-\log_2(P_c)$ defines the length of the bit sequence and P_c represents the percentage of the symbol space. Since entropy has the same two perpendicular dimensions, it makes sense to illustrate entropy in MaCE as described in Sec. 4.3, and the construction shows that the entropy equation H can be represented as a two-dimensional space. This may seem odd, as information is not typically considered two-dimensional, but the representation of the data in the binary tree is two-dimensional as well. The dimensions relate to the height and width of the binary tree and the two dimensions fully describe the information content of the space. When mapping the symbols onto the space, the symbols are all located in leaves of the tree due to the prefix-free requirement. The sum of the leaves represents the width of the space and the sum of the symbol percentage always equal 100% to adhere to the lossless requirement. The route from the root of the tree to the leaf is rep-

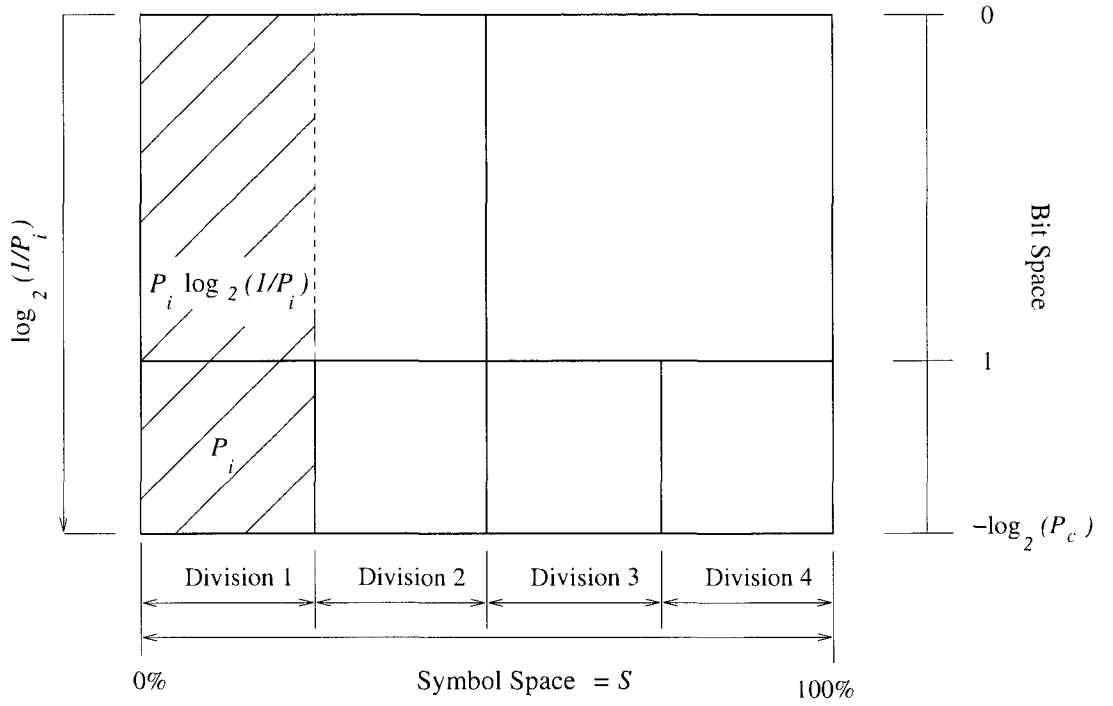


Figure 4.5: Entropy map: The area representing a single symbols entropy, H_i, is shown in the hatched region.

resented by a string of bits representing the code word identifing the symbol location in the tree. The number of bits in the code word represents the number of expansions required to assign the symbol to the leaf and the height of the space. Multiplying the symbol percentage P_i by the number of bits required results in the area used by the code word to represent the symbol, and this area represents the space that *cannot* be used for another symbol. The area used by a single symbol entropy is H_i as described in (2.7).

In the traditional model of the binary tree this relationship to entropy and the two-dimensional space was not easy to discern as the multiple paths representing the space were represented by a single line. In contrast, MaCE clearly defines the entropy area, as shown by the hashed area in Fig. 4.5. From the depiction of $H_i = P_i \log_2(1/P_i)$ we can see the orientation of $\log_2(1/P_i)$ is on the vertical (bit space) and P_i is on horizontal (symbol space) and the product represents the area. The dimensions correspond directly to the area defined by the hatched area in Fig. 4.4 to represent symbol A. The concept can be expanded to represent the complete entropy equation. The entropy equation describes the total area used by encoding all the symbols P_i to P_n, and is depicted in Fig. 4.6. The final representation also agrees with the bits per symbol definition of entropy.

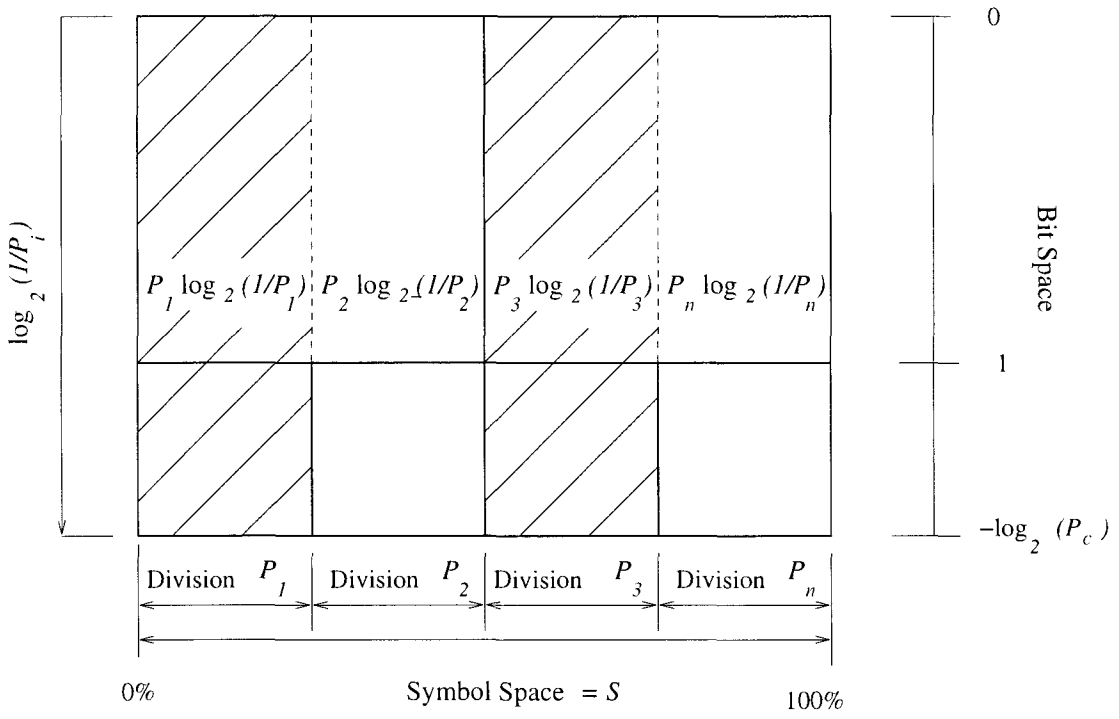


Figure 4.6: Entropy map complete: The area representing each H_i entropy for four symbols is shown in the striped regions.

4.5 Mapping Certainty and Entropy

The majority of the discussion so far has been in regards to entropy or certainty not the combination of the two. As explained previously, certainty is represented by the certainty of the binary structure (the system). Certainty is defined by the rules that define the binary tree. We know for certain that binary 0 represents the first space to the left and the 1 represents the first space to the right from the root. The relationship of 0 and 1 to the define certainty exists because we defined them in our definition of the binary system to represent the two possible states. We say this certainty is by definition.

In the concept of probability, the 0 represents 50% of the binary representation for a single bit code word. Its counter part, the 1, represents the other half of the binary representation and the other half of the one bit code word. This concept was represented in Fig. 3.1. We also use certainty to define MaCE in Sec. 4.3.1. The node percentages contained in both depictions represented the percentage of certainty P_c. The length from the root to P_c was definded by $-\log_2(P_c)$. Combining these concepts, the area that certainty represents is defined $-\sum P_c \log_2(P_c)$, and is displayed in Fig. 4.7.

The second depiction of the space is in terms of entropy. We defined this view earlier in Sec. 4.4, as displayed in Fig. 4.5. Looking closely at the two figures the only difference is

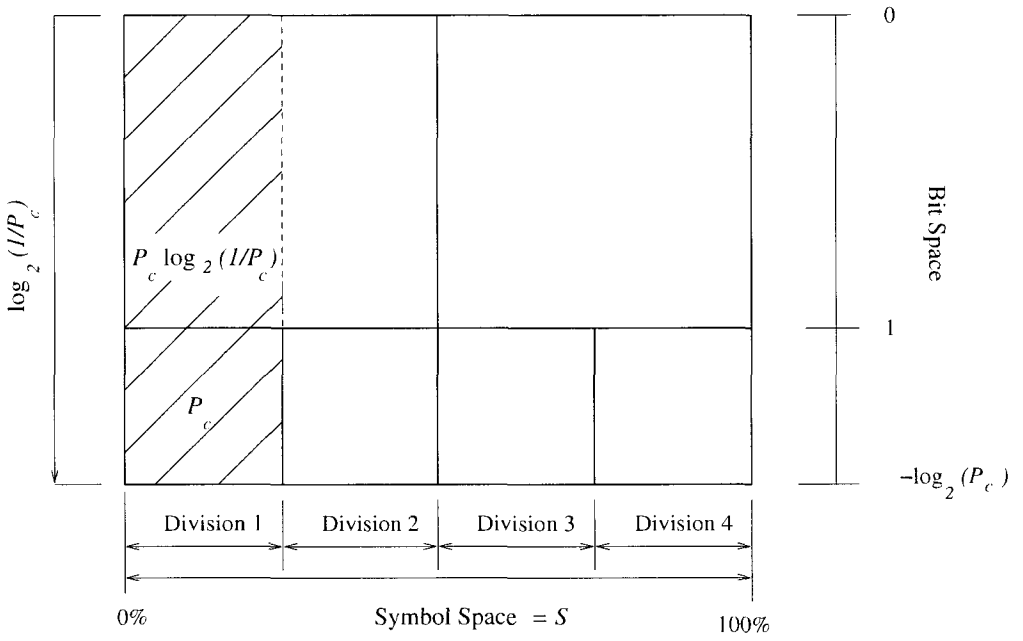


Figure 4.7: Certainty map: P_c represents the percentage of certainty represent by the square region occupied by P_c. The value of $P_c \log_2(1/P_c)$ represents the space used to represent the certainty in P_c.

the P_c representing certainty defined by the binary tree and the P_i representing the symbol. More importantly, this demonstrates that both entropy and certainty can be represented in the same depiction. What this relationship shows is that the goal for minimizing entropy can be accomplished by assigning the uncertainty represented by P_i to the most certain location represented by P_c where $P_i \approx P_c$. Another possibility is mapping P_i representing the symbol to the certain location represented by $-\log_2(P_c)$.

4.6 Example of MaCE

The purpose of MaCE is to illustrate the interaction between entropy and certainty. The following examples illustrate its effectiveness. We examine the same example used to illustrate the three base algorithms in Chapter 2. Given five arbitrary symbols: *A*, *B*, *C*, *D* and *E* represented by their known symbol counts (*A*:35, *B*:17, *C*:17, *D*:16, *E*:15). Note: the numbers are the same because both the counts and the probabilities equal 100. We use MaCE in Fig. 4.8 to illustrate the theoretical ideal overlaid on the binary system. The ideal entropy $H(X)$ is the sum of the rectangular areas calculated previously in (2.9). By using MaCE we can observe how the ideal and the system used to represent the symbols relate to each other. We can also see by the depiction in Fig. 4.8 that none of the symbols P_i would map ideally to the P_c represented by the binary system. The quest of the compression

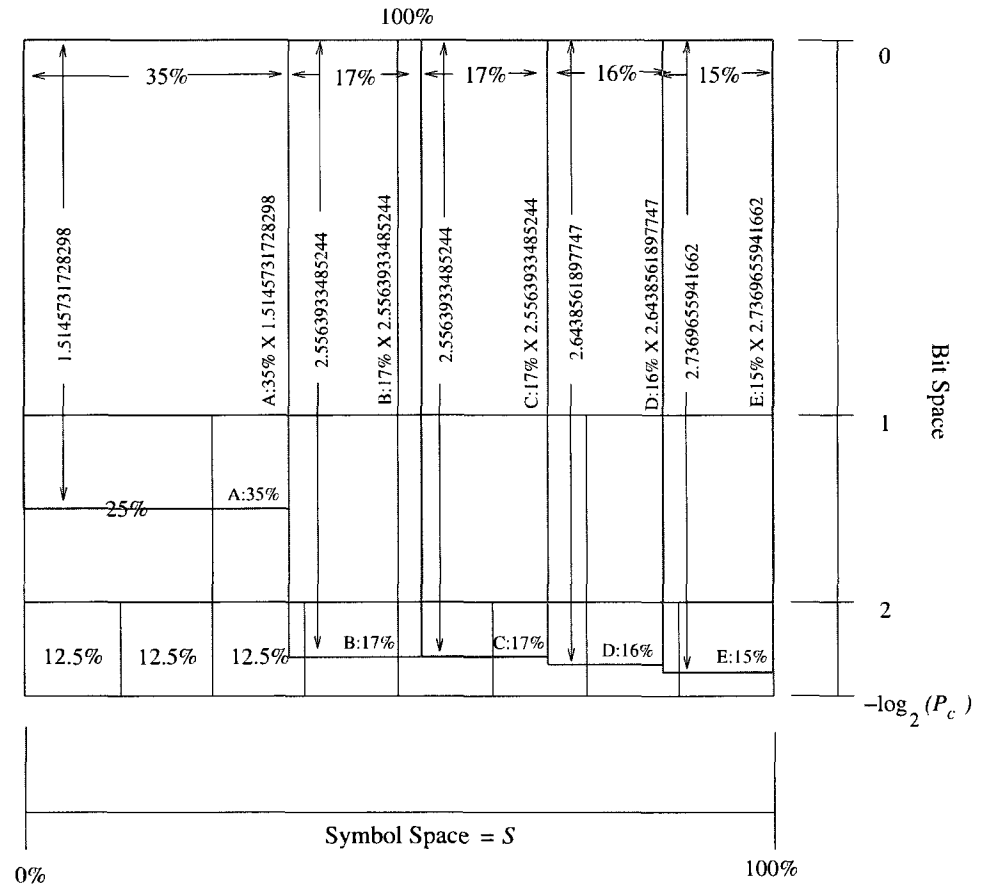


Figure 4.8: MaCE example with the ideal H to represent 5 symbols. The color rectangular areas illustrates the space required to ideally map the 5 symbols $A-E$. The vertical bit space and the horizontal symbol width are displayed. The area within each rectangle represents the space the symbol would ideally use in $W \times L$.

algorithm is to efficiently arrange the symbols as close to the certainty as possible.

The Huffman algorithm accomplishes the task with the best solution in binary. Huffman produces an actual entropy $L(X)$ =(A:35% $\times$ 1)+(B:17% $\times$ 3)+(C:17% $\times$ 3)+(D:16% $\times$ 3)+(E:15% $\times$ 3)= 2.30. We illustrate the Huffman solution with MaCE in Fig. 4.9. MaCE illustrates the overshoot of the 4 smaller symbols $B-E$ and the under shoot of symbol A when the symbols are aligned to the binary system. The illustration also alludes to why Shannon-Fano can achieve similar entropy results to Huffman encoding. The overall entropy is the same as long as the overshoots and undershoots balance between the different code words. The individual values H_i may vary greatly, however, the end result in compression is the same if there is a balance.

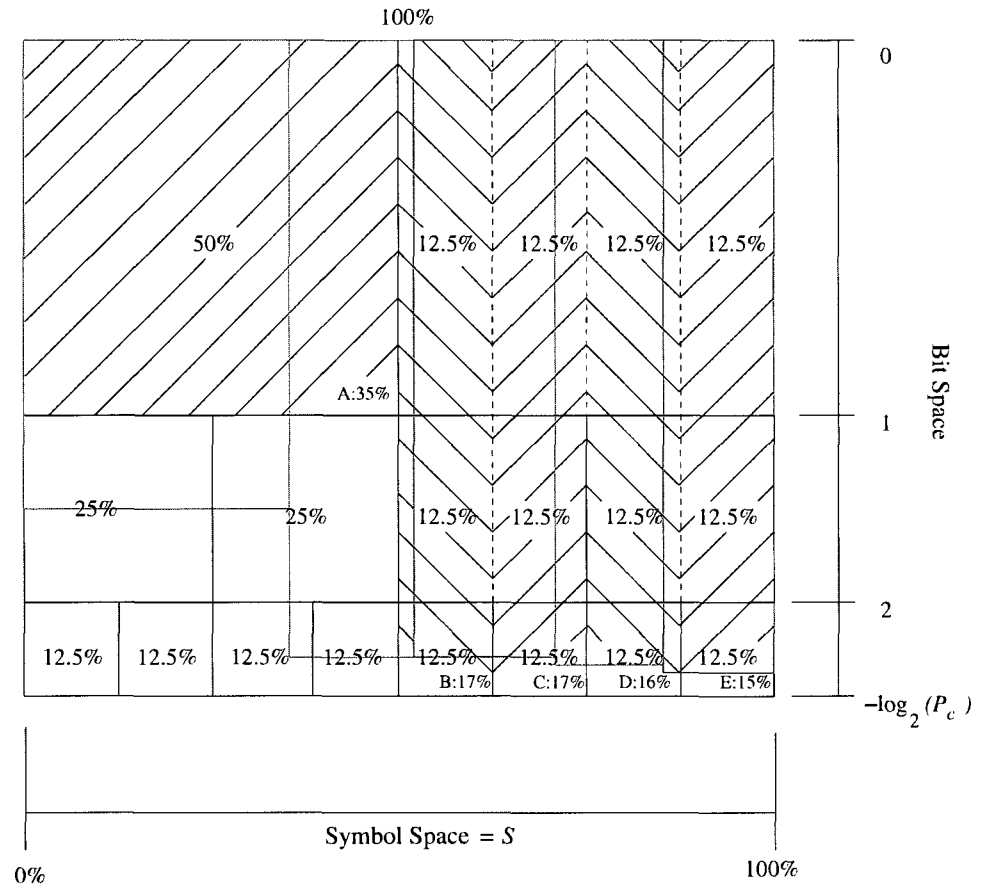


Figure 4.9: MaCE example illustrating the compression obtained by Huffman encoding. The color rectangular area illustrates what is required to ideally map the 5 symbols $A - E$. The hatched marks illustrate the compression acquired by Huffman and the best possible arrangement given the binary requirement.

4.7 MaCE and the Symbol

Previously, we have been concentrating on the certainty and the entropy related in representing the symbols and have neglected the symbols themselves. In MaCE we denote the representation of the symbol in the lower right corner of the surrounding rectangle delimiting the certainty and/or entropy. The symbol itself is located in the number of bits from bottom up to the bound represented by the limit defined by Shannon's theory examined in Chapter 2. The key to seeing this relationship is accomplished by adding space at the bottom of MaCE, below the limits defined by H_i for each symbol. The bottom of the graph is referred to as the base. Recall, the base b is the number of divisions of the symbol space. If well put this in numerical terms we are changing the base by adding the additional bit values to the length. By adding space to the bottom of the illustration we are expanding the base for MaCE illustration and changing the base numerically.

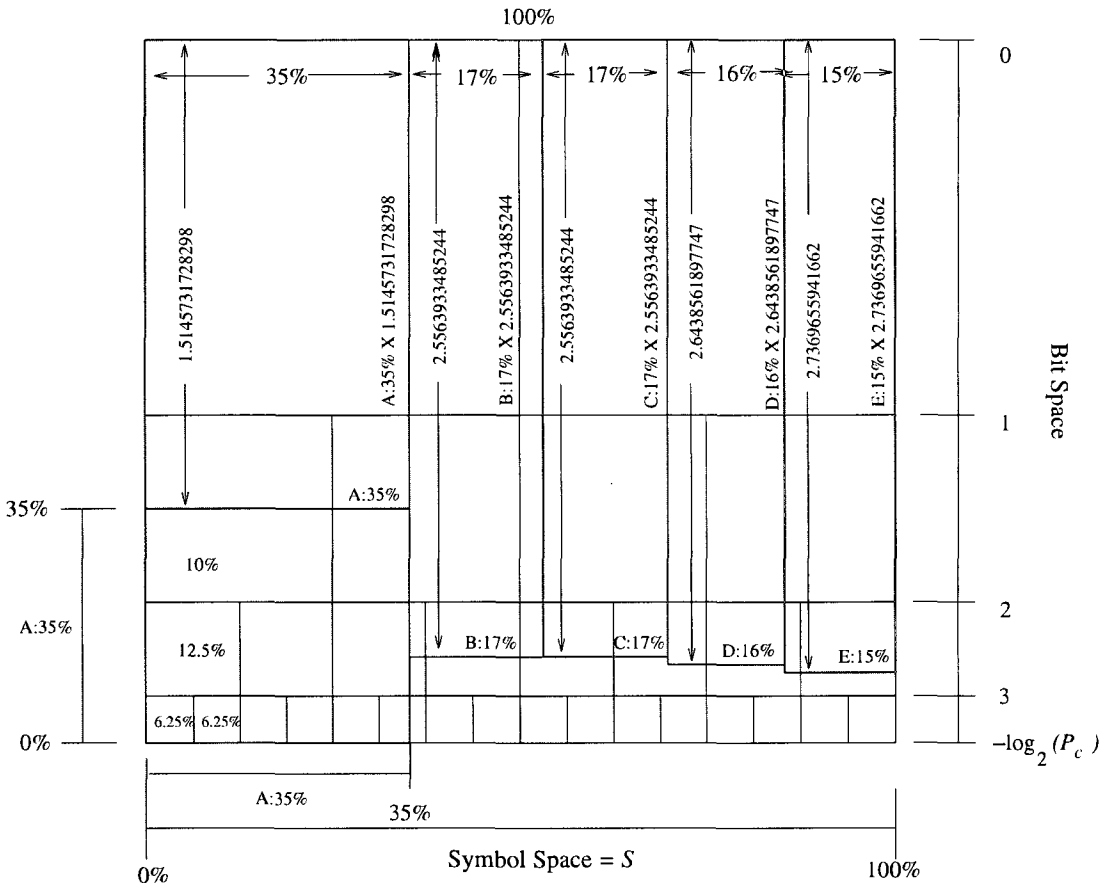


Figure 4.10: MaCE example including a symbol. The illustration shows the color rectangular areas required to ideally map the 5 symbols $A - E$. The hatched marks is what is possible given the binary requirement. This is the arrangement and acquired by Huffman encoding.

The symbol occupies the vertical space (bit space) from the base to the limit defined by $-\log(P_i)$ and the width P_i in the horizontal space (symbol space) as discussed in Chapters 2–4. The vertical portion is defined by starting at the base and the percentage is increasing from 0% to P_i. In Fig. 4.10 we chose $b = 16$ in order to acheive a more uniform appearance. The summation 0% to P_i of the vertical space used by the symbol includes all of the bit values displaced. The addition includes the adjacent spaces at the base that subdivides the higher square as the adjacent bit is one of the displaced bits. This summation of the adjacent spaces insures that all of the space is accounted for in the map for that symbol. Since we are only concerned with the vertical length from the base, the summation only has to represent one path. For example, if we used either of the spaces denoted by 6.25% in Fig. 4.10 we need to add the adjoining 6.25%. We can then continue up the path to the boundary between entropy and certainty, adding the space used at each block. In this case, using Fig. 4.10 as a guide, we add $(6.25\% + 6.25\%) + 12.5\% + 10\% = 35\%$.

The visualization of the expansion has another importance as the space represented by the expansion from one base to the other represents the space possible for compression by reversing the process. This is important as it shows the space that can actually be compressed by changing from a higher base to lower one. The relationship between the base and the length also indicates that in the process of mapping an ideal symbol, i.e. a symbol with $P_i = P_c$, a base can be chosen to represent the symbol entirely in its range and domain. For example, compare Fig. 4.10 in base 16 to Fig. 4.9 in base 8. The space removed is some of the overhead not required to represent the symbols, however, this simple translation will only be optimal when the $P_i = P_c$ for the P_c of the base. Other methods must be applied to address non-equally distributed symbol combinations.

4.8 Summary

The purpose of this chapter was to introduce MaCE. The concept illustrates the entropy of the data and the certainty of a system definition to equivalent representations in both space and orientation. The construction allows for the expansion of the space available and the divisions to represent a symbol to be clearly defined. The orientation of entropy as bit spaces per symbols space is determined by using the entropy equation and MaCE, and allows for representing certainty and entropy. This allows for the continuous nature of entropy and the discrete nature of certainty to be mapped simultaneously. The use of MaCE illustrates that it is possible to compare entropy to certainty and certainty to entropy for the purpose of data compression. The addition of the symbol to MaCE illustrates the use of base manipulation to increase this compression. Both of these developments will be utilized in Chapter 5 to create new methods for data compression.

Chapter 5

USE OF MaCE FOR DATA COMPRESSION

5.1 Using MaCE

Chapters 2–4 examined three different approaches to the mapping of entropy. These three major approaches have influenced countless variations of compression schemes over the years. Previously, we have shown how Shannon-Fano was a top down solution which divided the entropy by the certainty of the system to map the symbols. Shannon-Fano-Elias divided the entropy space from the bottom by P_i leaving the vector of bits to represent the code word. Shannon-Fano-Elias produces the final code by using this technique along with the decimal representation of the binary values to represent certainty. The purpose of this chapter is to describe an approach that utilizes the definition of certainty as well as entropy to accomplish the data compression. This chapter introduces several new methods, SLM (Select Level Method), SLMN (Sort Linear Method Nivellate) and JAKE (Jacobs, Ali, Kolibal Encoding), using these concepts and the visualization approach to quantifying these relationships provided by MaCE. SLM creates Shannon-Fano-Elias code in less time, and more importantly from a different direction, then the traditional method. The use of a different path allows for SLMN to create a new code with entropy less than $H(x)+1$. This concept is extended to create JAKE which combines these concepts to increase the data compression beyond $H(x)+1$ with less arithmetic computation than the current methods.

Looking at the data compression problem from the side is similar to the difference between Riemann and Lebesgue integration. In Riemann integration, the domain is subdivided in small elements Δx and we evaluate the function $f(x)$ on each element. Summing $f(x)\Delta x$ over all such elements and taking the limit as Δx gets small, returns the integral. This approach to integration is easy to explain and visualize and is the common way to explain integration in calculus class. The view relates to Shannon-Fano-Elias dividing the symbols space as the domain and the results are the number of bits in the range space. In Riemann integration the sum of the vertical divisions results in the final area defining the solution. In Shannon-Fano-Elias the sum of the bits in the vertical space for each of the code words defines the final solution.

In Lebesgue integration, which provides a more comprehensive view, Lebesgue divides the range (vertical) space instead of the domain (horizontal) and defines the integral using this approach. The result was a theory that allowed more functions to be integrated. We

can see that division of the bit space (vertical) or the range using the spatial concept is also possible in MaCE. The division of the environment by the range or bit space will leave only the symbol space and has a similar effect in reducing the complexity of the problem.

In Sec. 5.2 the first method, SLM, is introduced which uses MaCE to divide the space and the entropy equation in an complementary direction in comparison to Shannon-Fano-Elias. The division is accomplished by using bit length of certainty for the divisor which has the effect of reducing the space and the entropy equation to only consist of P_i. The reduction of the entropy equation allows for direct comparison between the symbol percentage representing entropy and the optimal percentage representing certainty. The end result is to reach the same location and compression with an easier implementation by eliminating the need for the logarithm an `ceiling` function. In Sec. 5.2 we also introduce the concept of equilibrium in order to increase the efficiency of SLM and the division of the space.

Section 5.3 introduces a second method, SLMN, which uses the framework to divide the symbol space and sort the symbols based on percentage. The sorting technique is a bucket sort based on the midpoint percentage defining the expansion of the base. The data being sorted and the use of the midpoints allows for SLMN to place the data closer to the optimal location based on the comparisons. The combination reduces the entropy to less than $H(x)+1$ for SLMN in comparison to Shannon-Fano-Elias which has the efficiency of less than $H(x)+2$.

Section 5.4 explains the significance of $H(x)+1$ and how the knowledge gained from MaCE can be used to reduce the total entropy. The knowledge in this section allows for the concepts of SLM and SLMN to be extended in JAKE to allow the compression in any base which increases the compression and the computational efficiency of data compression. Section 5.6 explains how the encoding and decoding table can be used in a symmetric manner to both encode and decode messages.

5.2 Select Level Method

Shannon-Fano divides the combined entropy of the symbols $\sum P_i$ over the certainty of the space represented by P_c. In Shannon-Fano, $\sum P_i$ and P_c determine the length of the code word used to represent a symbol by recursively dividing the partial $\sum P_i$ by P_c until a one-to-one mapping is produced. The width is based on the length as it is only increased when additional divisions are required by the algorithm.

In contrast, Shannon-Fano-Elias divided the entropy space by P_i leaving the vector of bits $-\log_2(P_i)$ to represent the length of the code word based on the theoretical ideal $\mathbb{I}$

(2.8). In Shannon-Fano-Elias the reciprocal of P_i determines the length of the code word by applying the $\log_2()$ function to the reciprocal of each symbols probability P_i. The `ceiling` function and the $+1$ values increase the length to avoid conflict with other symbols being mapped. $\overline{F}(P)$ supplies the width offset to supply the offset from 0 to map the symbol.

Both of these methods have a common thread of P_i when mapping the symbols, only the mapping methods differ. The difference in the methods is clear when examining the simplification of the entropy equations representing them. The entropy equation simplification for Shannon-Fano is represented in (5.1) and Shannon-Fano-Elias is represented in (5.2), i.e.,

$$H_i/P_c = (P_i \log_2(1/P_c))/(P_c \log_2(1/P_c)) = P_i/P_c, \tag{5.1}$$

$$H_i/P_i = (P_i \log_2(1/P_i))/(P_i) = \log_2(1/P_i). \tag{5.2}$$

The simplification of the equations in (5.1)–(5.2) makes it evident what remains to define the length when the respective algorithms divide the space. Shannon-Fano's length is defined by P_i/P_c and Shannon-Fano-Elias' length is defined by $\log_2(1/P_i)$. SLM uses a combination of these two approaches to produce a method which is less computationally expensive then Shannon-Fano-Elias while producing the same encoding table. By combining these aspects of the previous algorithms, SLM uses P_i to make comparisons to P_c and appropriately map the symbols to the space. In Sec. 5.2.1 we must reduce the entropy equation to consist of only P_i in order to have comparable entropy and certainty spaces.

5.2.1 Reduction of the Entropy Equation and Entropy Space

As mentioned in the Sec. 5.1, SLM reduces the space in an complementary direction in comparison to Shannon-Fano-Elias. Recall that the entropy space is two-dimensional, represented by the length in bits and width represented by the $\sum P_i = 100\%$. Shannon-Fano-Elias divided the width of the space by P_i which removed the width component of the space and the remainder was the length in bits. SLM divides the space by a vector representing the length of bits which reduces the space to P_i. In order to execute the reduction of the entropy equation, SLM uses MaCE percentage P_c and divides the equation by $-\log_2(P_c)$, where $P_c = P_i$, where P_c is the certainty percentage representing the percentage of the space in the binary tree represented by $-\log_2(P_c)$ bits. When $P_c = P_i$ the logarithm function returns the same number of bits. This cancels the length component leaving the width represented by P_i. The reduction is represented

$$H_i/(-\log_2(P_c)) = -P_i \log_2(P_i)/(-\log_2(P_c)) = P_i. \tag{5.3}$$

Notice at the completion of (5.3), we are left with only the P_i. The $\sum P_i = 1$ represents the complete horizontal symbol space of the entropy map. This is in contrast to Shannon-Fano-Elias in (5.2) which divides by P_i and the remainder is $-\log_2(P_i)$. The use of P_c is similar to the approach used by Shannon-Fano except in SLM we use only $-\log_2(P_c)$, not the complete space. The reason this simplification is possible is because SLM is only operating with one H_i not the combination of several H_i's.

To complete the process, SLM only needs to compare the P_i to P_c. We have already defined P_c for certainty using MaCE and can determine the location of P_c by length and width. The definition allows for the reduction of the certainty space to a single component. The simplification of both the entropy space and the certainty space represented by MaCE allows for a simple comparison between P_i and P_c to determine the appropriate mapping. The length in terms of bits is determined by the comparison and width is an offset in the symbol space previously described for Shannon-Fano-Elias in Secs. 2.7–2.7.3. First, we need to show that both cancellations map to the same location. We accomplish this task by looking at the spatial representation of the Shannon-Fano-Elias operations in MaCE.

5.2.2 Visualization of the Reduction of the Entropy Equation and Entropy Space

Figure 5.1 uses MaCE with the bit values representing Shannon-Fano-Elias model after the cancellation of P_i. From the image we can see the ideal entropy area is described by the solid hatched region. This represents $H_i = -P_i \log_2(P_i)$ with $P_i = 25\%$. The height $-\log_2(P_i)$ of the ideal entropy area is 2 bits long and the width is symbol percentage P_i. The 1 values within the region represents the bit each block represents.

The dashed hatched area represents the space used by the fitting functions of `ceiling` and $+1$ and the symbol itself. The logarithm and `ceiling` function ensures the equation returns the max of the bit range and the $+1$ places the location at $l(x)$. The combination of the two functions increase the length to 3 bits. The $\overline{F}(P)$ offsets the space from the left by $P_i/2 = 12.5\%$. placing the symbol in the location represented by code word 001. $P_i/2$ ensures that only one node is used in the sub-tree denoted by the $+1$s. In this case the code word of 000 is never represented and results in increased entropy and shows an example of the second type of entropy described in Sec. 3.2. The second type of entropy is where the code word does not represent a symbol.

SLM is using (5.3) and MaCE to reach the same location. In Fig. 5.2 MaCE illustrates the percentage values representing the space used after the cancellation of $-\log_2(P_c)$. In this figure we have that code word 00 represents 25% of the bit / symbol space. This space would be the ideal location to place the symbol. MaCE also shows that any percentage

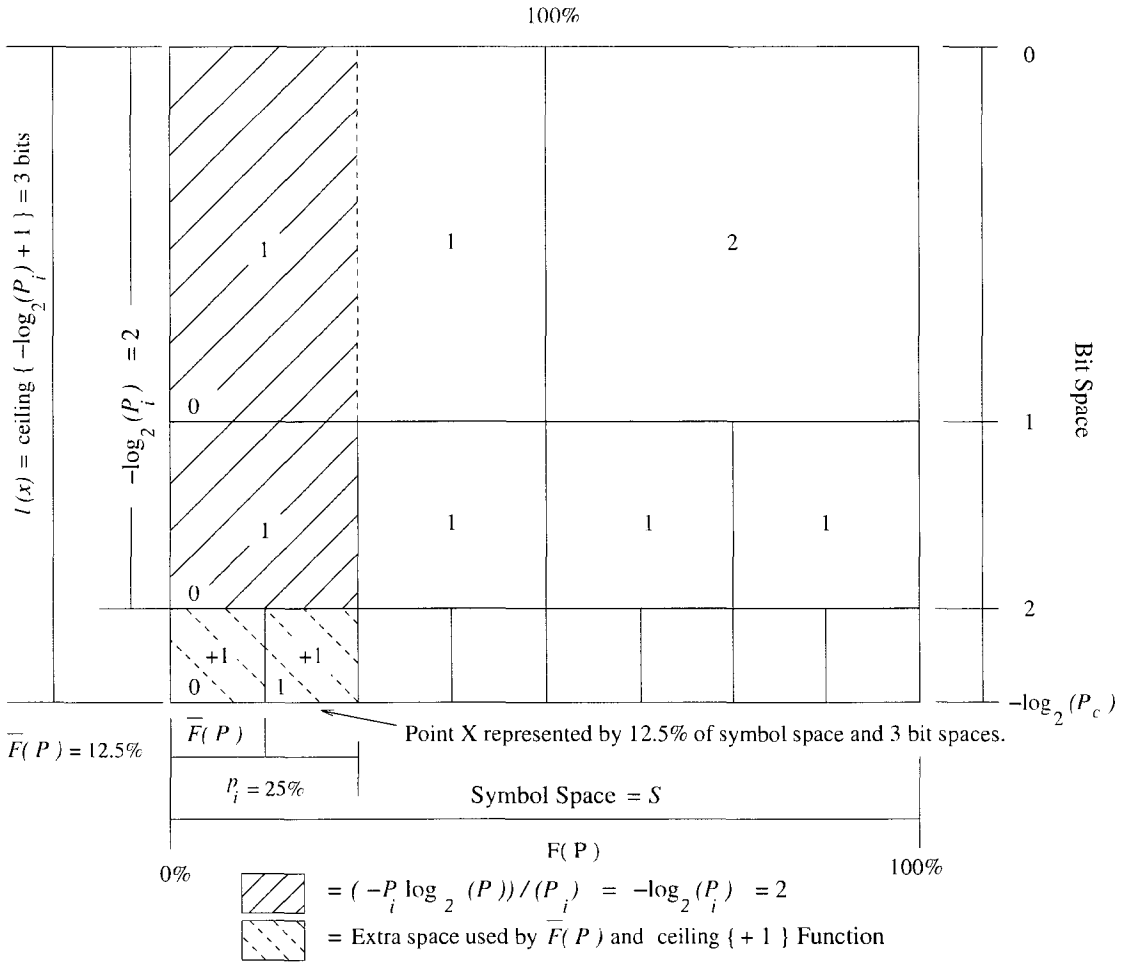


Figure 5.1: Shannon-Fano-Elias: The intersect point. Point X represents the location where the offset used by SFE defines the space to represent symbol P_i.

within the block would also match the requirement of the $\lceil -\log_2(P_i) \rceil$. The range of the block is from $[25\%, 12.5\%)$ or in bits $[2, 3)$.

The range indicates by using a comparison model that we only need to compare P_i to the higher percentage of the block P_c. For example, $P_i = 25\% \leq P_c = 25\%$ will place the value in the location requiring 2 bits 00. This comparison will replace both the logarithm and the `ceiling` functions within the algorithm. All that remains is the $+1$ to increase the string to bit length to 3 bits the same a Shannon-Fano-Elias. The $\overline{F}(P)$ offsets the space from the left by $P_i/2 = 12.5\%$ placing the symbol in the location represented by code word 001. This maps the symbol to the same location and represents the same bit sequence as Shannon-Fano-Elias. We can also see this visually by comparing Fig. 5.2 to Fig. 5.1.

The change from Shannon-Fano-Elias to SLM only modifies how calculating $l(p)$ is accomplished. The change works correctly because both the $-\log_2(P_i)$ and P_i intersect at the same integer values when $P_i = P_c$. Since we cannot represent any of the sub-locations

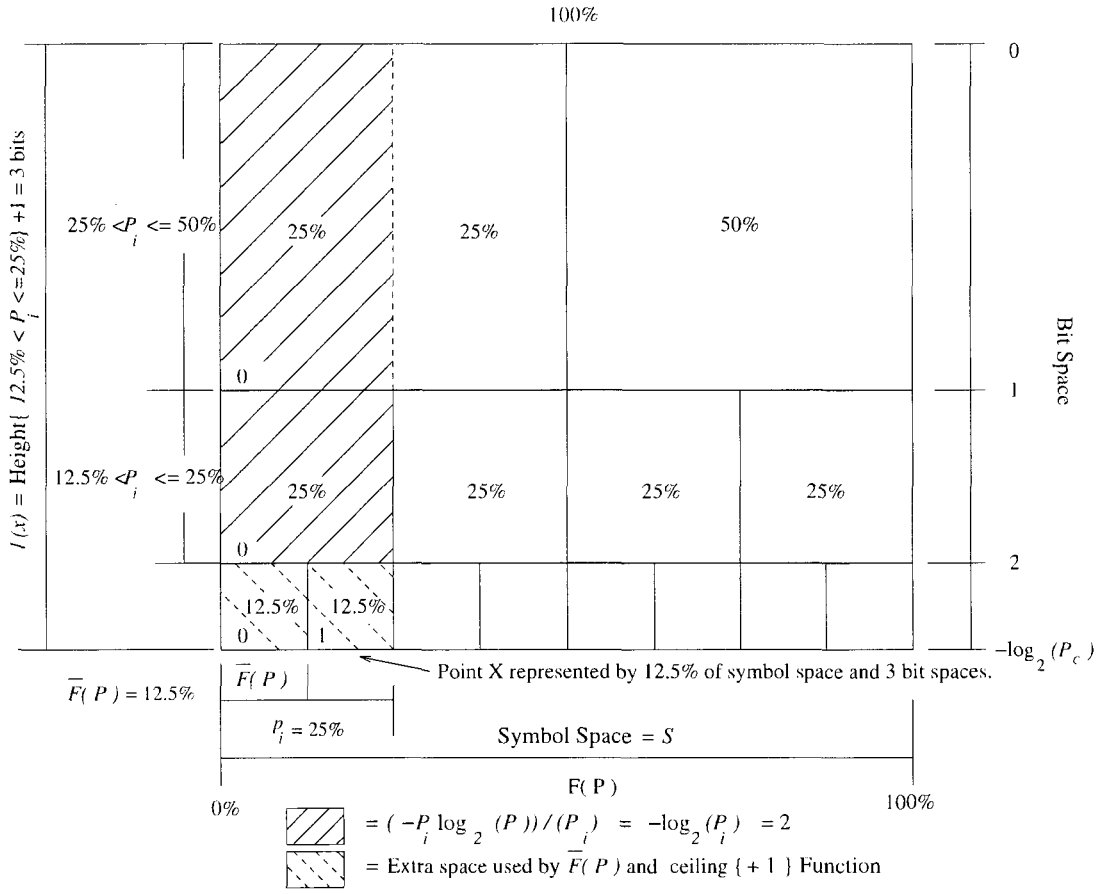


Figure 5.2: SLM: The intersect point. Point X represents the location where the offset used by SLM defines the space to represent symbol P_i.

within the block, the $\lceil -\log_2(P_i)+1 \rceil$ only represents integer values as well. This allows for the comparison to map the symbols to the integer values based on the range of percentage denoted within the block without the use of the logarithm. The integer values represent the length of the code word $l(p)$. The rest of the implementation of Shannon-Fano-Elias remains the same.

Now that both MaCE and the entropy map consist of spatial percentages, we can simply compare the symbol components representing uncertainty to the MaCE components representing certainty and place the values in the same location as Shannon-Fano-Elias.

5.2.3 Comparison Model for SFE Code

By changing the focus to certainty instead of entropy and dividing the space by $-\log_2()$ for SLM the logarithm function and the `ceiling` function are replaced by comparisons. The space in MaCE is represented by the code word where $P_c + 1 < P_i \leq P_c$. $P_c + 1$ represents the level below P_c. This change allows for a less computational implementation of a com-

pression algorithm while mapping the symbol to the same location. Since both SLM and Shannon-Fano-Elias map the symbols to the same location the entropy will be the same as well as any details regarding the usage of the encoding tables.

For the implementation of SLM we need to replace the statement that calculates the length of the code word with an a function that returns the same length value using the comparison model. By examining the pseudo code for Shannon-Fano-Elias (in Algorithm 1), we can see that the length $l(P_i)$ is determined by $l(P) = \lceil -\log_2 P_i + 1 \rceil$ from (2.14).

Algorithm 1 Shannon-Fano-Elias pseudo code.
The code uses $\lceil (-\log_2(P_i) + 1) \rceil$ to determine length of code word.

for all Symbols i to n **do**

$\overline{F}(P_i)$ (2.13) - Combine half current probability P_i + previous sum $\sum P_i$

$F(P_i)$ (2.12) - Increment previous sum by P_i

$l(P_i)$ (2.14) - Determine length of code word using $\lceil (-\log_2(P_i) + 1) \rceil$

$\overline{F}(P)_2$ - Convert $\overline{F}(P_i)$ to binary

Return code word with bit length $l(P)$ of the leading binary values of $\overline{F}(P)_2$

end for

The pseudo code for SLM (in Algorithm 2) represents this change with the IF/ELSE function. The pseudo code for the IF/ELSE function is displayed (in Algorithm 3). The individual components of the IF/ELSE block model the range each spatial block represents in MaCE as displayed in Fig. 5.2. Each block has the range from 2^{-L} to 2^{-L+1} where L is the level in the space. In the IF/ELSE block this is represented by the IF statements. Each IF statement represents a level L in the space. The length is returned which represents the largest bit represented in the block plus one.

For example, the second IF represents the spatial block with the range $[25\%, 50\%)$ or in bits $[2, 1)$. The 2 is the largest value in the block. Recall, from Shannon-Fano-Elias the `ceiling` is taken of this range resulting in 2 for this block as well. At this point, Shannon-Fano-Elias would add the $+1$, however, for the IF/ELSE we simple add it into the return value. In this case, the IF/ELSE block returns 3.

5.2.4 IF/ELSE Function Implementation

Section 5.2.3 creates a method that uses the reduction of the entropy equation in Sec. 5.2.1 and an IF/ELSE block to accomplish the same encoding as Shannon-Fano-Elias. The IF/ELSE block introduced is based on a top down linear comparison where each division

Algorithm 2 Select Level Method pseudo code.

SLM code replaces $\lceil -\log_2(P_i) + 1 \rceil$ used in Shannon-Fano-Elias with IF/ELSE function.

for all Symbols i to n **do**

$\overline{F}(P_i)$ (2.13) - Combine half current probability P_i + previous sum $\sum P_i$

$F(P_i)$ (2.12) - Increment previous sum by P_i

IF/ELSE function Fig. 3 - Determine length of code word using IF/ELSE block

$\overline{F}(P)_2$ - Convert $\overline{F}(P_i)$ to binary

Return code word with bit length $l(P)$ of the leading binary values of $\overline{F}(P)_2$

end for

Algorithm 3 IF/ELSE Block for SLM.

Code to replicates the length value returned by $\lceil -\log_2(P_i) + 1) \rceil$.

```
if Symbol Percentage ≥ 50% then
  Return length = 2
else
  if Symbol Percentage ≥ 25% then
    Return length = 3
  else
    if Symbol Percentage ≥ (2^-L)% then
      Return length = L + 1
    end if
  end if
end if
```

represents a level L. The problem with the top down approach is the symbol space increases in multiples of the base, 2^L, at each addition to L and the top down approach is starting at the small end and working down thru an increasing number of symbols. The top down approach has the effect of looking for the a space to store the symbol by first looking at the levels with the least amount of possible spaces. To compound the problem the top down approach does the same thing for all n symbols. A better approach would divide the space more equally and start at a location with the most comparisons first. We must introduce the concepts of equilibrium and the split tree to acquire a more suitable starting point.

5.2.5 Equilibrium and Split Tree Properties

The second property of H, defined by Shannon, states that when all the choices are equal entropy is at the maximum. The individual symbols are at maximum entropy when they are the farthest from the root node. By combining these two observations we define equilibrium and the equilibrium level L_e as the level where all the symbol percentages are equal and farthest from the top of the space. The depth of L_e is first level where all n symbols can be represented. Mathematically it is the level L_e where $2^{L_e} \geq n$.

Equilibrium has a unique property in that it is the *only* level in the space that can represent all n symbols without adding additional entropy to the space. Recall from Sec. 3.2 that there are two types of entropy. Entropy as the distance from the root to the symbol and entropy were a code word does not resolve to a symbol. If a symbol is represented by a code word the first type of entropy is inevitable with more than one choice. The second type of entropy is avoidable if all paths lead to a symbol. In terms of the Kraft inequality the $\sum_{i=1}^{n} 2^{-L_{ei}} = 1$. If $n = 2^L$ and all the symbols have an equal percentage, all n symbols will ideally be located on level L_e. Since the previous level $L_e - 1$ can only represent $n/2$ values the range for mapping the symbol to level L_e is $[(n/2+1), n]$ symbols. If there were only $(n/2)$ symbols the ideal location would have L_e representing the next higher level. The concept of equilibrium is illustrated in Fig. 5.3.

The reason that the equilibrium property is important is because this level represents the most comparisons that may be required for any level in the space. The level above equilibrium $L_e - 1$ can only represent $n/2 - 1$ symbols. The -1 represents the symbol required to allow the mapping of the other $n/2+1$ symbols still required to be mapped. The level below equilibrium $L_e + 1$ can represent $2n$, however, half of the code words would not represent symbols. This mapping or any other mapping below equilibrium would result in increased entropy that is avoidable. So the maximum comparisons is at $2^{L_e} = n$ as $2^{L_e-1} = n/2 - 1$ and $2^{L_e+1} = n - 2$. For this reason the equilibrium level makes a good initial dividing point for the IF/ELSE block.

The equilibrium level used as the initial split has other implications as the level above represents a negative growth rate. Each level above equilibrium represents half of the possible spaces of the level proceeding it. In other words, we are reversing the expansion by traveling toward the root by repeatedly dividing the number of spaces by 2. The decrease in space below L_e is not nearly as steep as it progresses from $n/2 - 1$ at $L_e + 1$ to only representing 2 symbols as L_n. The observation shows that the space to represent the number of symbols represented in the levels above L_e diminishes more rapidly than the number of symbols that may be placed on the level below L_e.

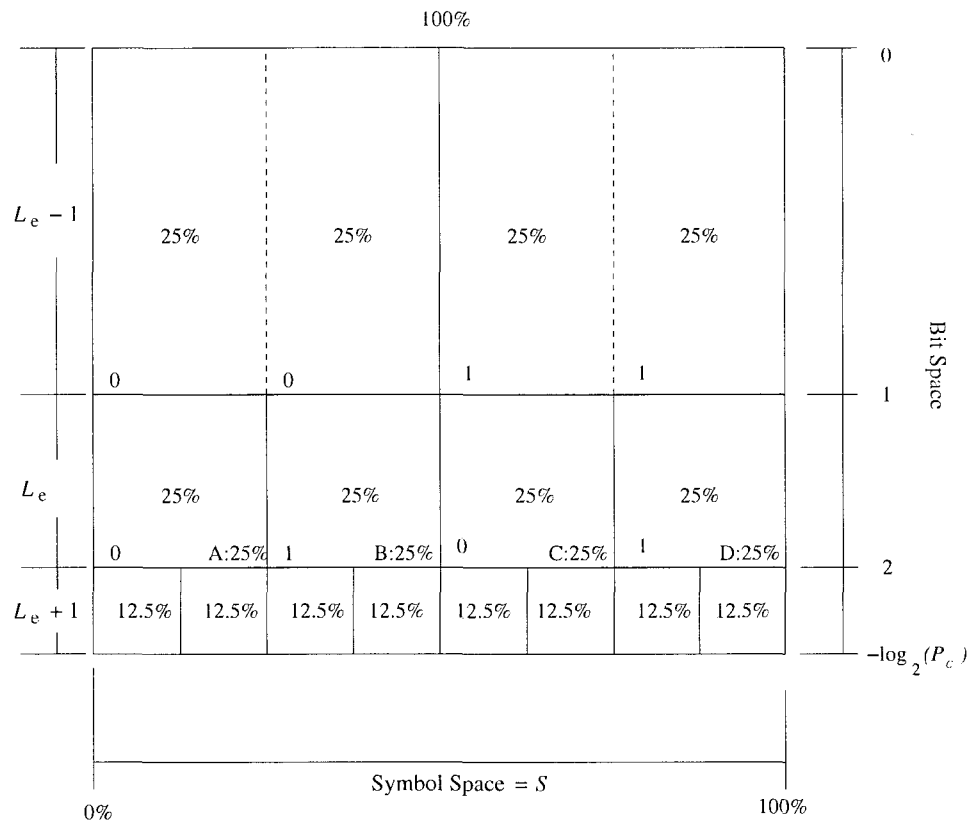


Figure 5.3: Equilibrium illustration. The value L_e is the first level that can represent all n elements.

The last implication is based on equilibrium and the displacement and expansion of the space. When the equilibrium level is full with n symbols, any symbol moved up a level displaces 2 which need to move to the next lower level. The displacement of 2 is due to a node currently representing a symbol being split and that symbol and one other node on L_e becoming the two children on the next level $L_e + 1$. The movement of the two displaced nodes to the next level represents expansion of the equilibrium space. This also applies if the equilibrium level is not full, but instead of displacing a symbol to the next level, the empty space available on L_e is used to compensate for the movement. The combination of the implications means more symbols will be placed below equilibrium then are placed above equilibrium. Equilibrium divided the vertical dimension of the tree. These observations mean that the implementation of the IF/ELSE block should give precedence to the equilibrium level and below. We illustrate this concept in Fig. 5.5.

In addition to equilibrium another property divides the tree space. We will refer to this as the split tree property and it divides the tree on the horizontal. We know that the tree is split at the top and we represent half with a 0 and half with a 1. Then the space is recursively divided by 2. What is of importance to compression is the relation between

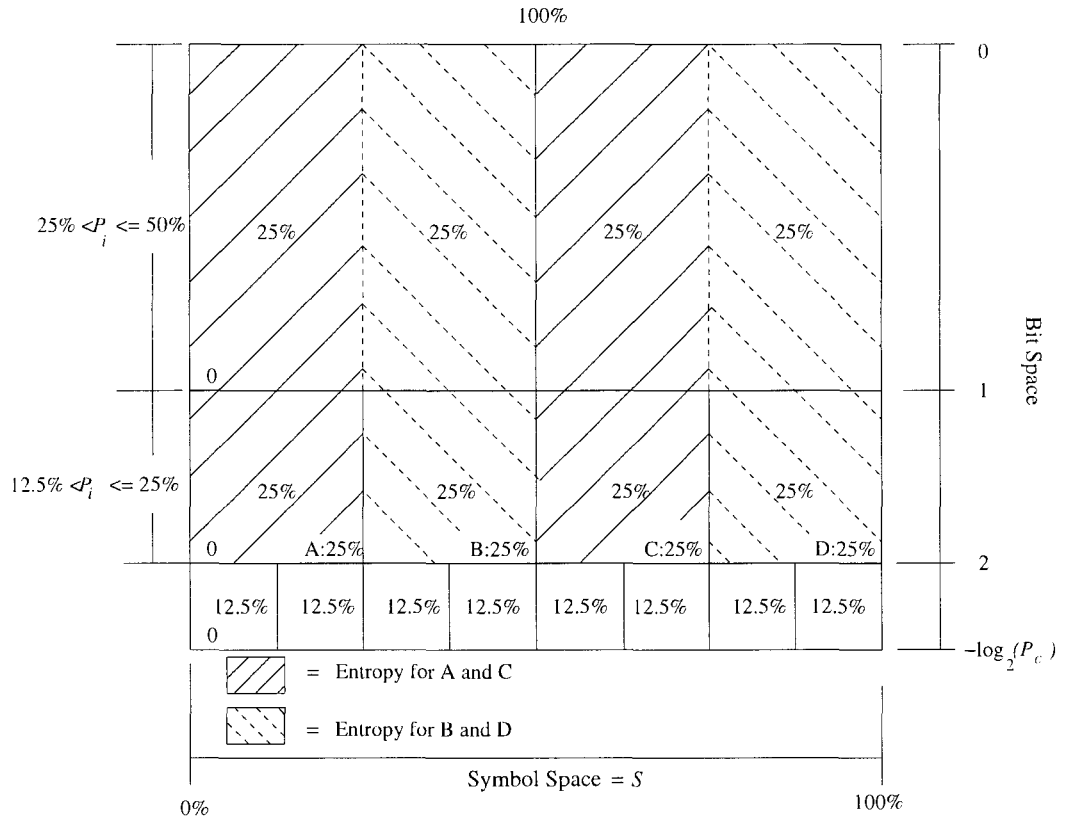


Figure 5.4: Equilibrium: Individual entropy values are equal and at the maximum. The hatched area represents the entropy space for symbols A and C. The dashed hatched area represents the entropy space for symbols C and D.

the two halves of the tree space and equilibrium. When looking at the space at equilibrium level there is a set number of spaces to represent the n symbols. If the number of spaces 2^L is equal to n the level is full and any movement of a symbol upward in the space requires an expansion of the space on the next level equal to accommodate the two symbols displaced. Where the split tree becomes a factor to this movement is in regard to the number of values that can move up and down in the space. Since the tree is split, half of the symbols are on the 0 side and the other half are on the 1 at equilibrium with n symbols filling all the spaces. The minimum number of symbols that can be represented on either side is 1. When looking at this at equilibrium, one value displaces $2^{L_e-m} - 1$ symbols when moving m levels higher in the space. The displacement represents a shift of all the adjacent symbols to the right and with a full tree 2 symbols must move down expanding the space. The maximum displacement is $n/2 - 1$ as 1 symbol must remain on the left side of the tree. The expansion represents the symbols moving to the levels below L_e. The displacement allows of the prefix-free requirement to be maintained and the expansion is required to adhere to the lossless requirement. An example of the displacement and expansion is displayed in

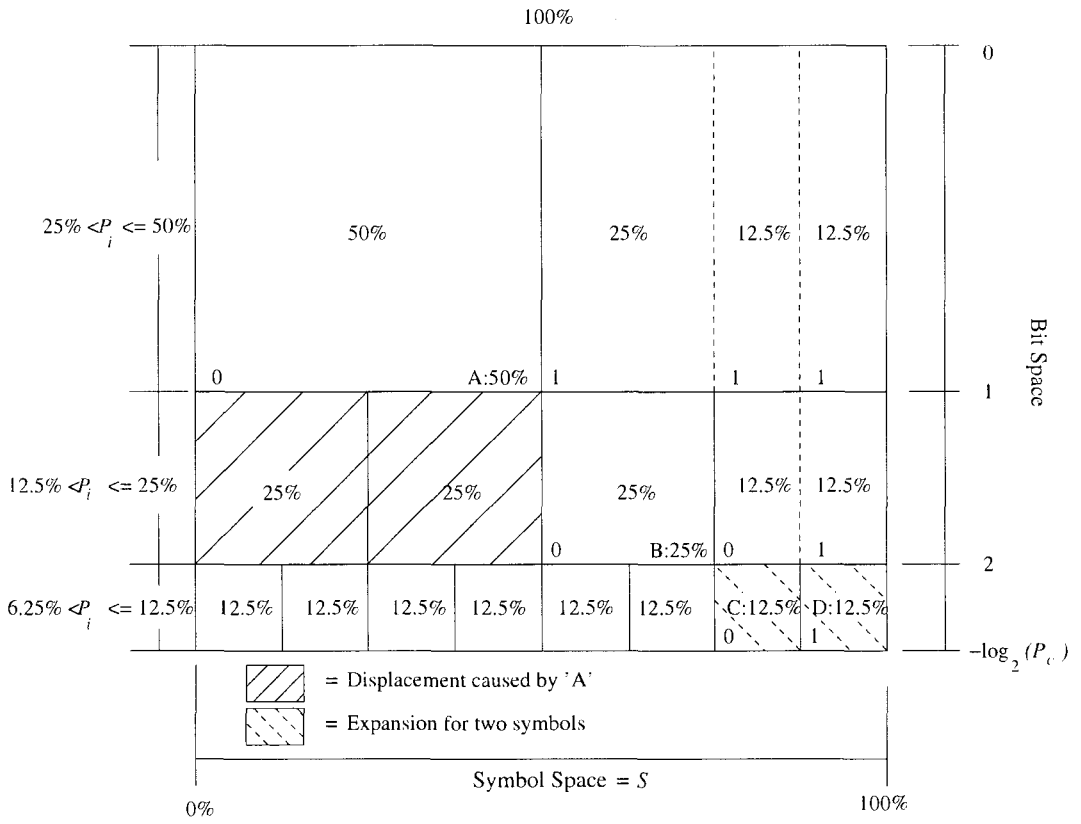


Figure 5.5: Split tree displacement/expansion. The hatched area represents the displacement caused by moving A up from equilibrium to a 50% space. The displacement caused and expansion represented by the dashed hatched area in order to represent symbols C and D.

Fig. 5.5.

Figure 5.4 has all the symbols (A, B, C, D) represented on level 2 and all the symbols represented 25% of the space and we show the expansion and displacement in Fig. 5.5. In Fig. 5.5 the symbol A represents 50% of the space. Immediately below the new square 0 represents the shadow or the displacement created by the movement of A. Notice the two squares that are displaced but only one symbol has been mapped. In order to maintain the prefix-free property additional space must be created. The space is created by vacating the space previously used by symbol D and moving the two symbols C and D to the level $L_e + 1$. The expansion is visible in the dashed hatched region. For the implementation of the IF/ELSE block this means that more sybmols will be placed below equilibrium and precedence should be given to equilibrium and below in the function. We use these concepts in SLM and SLMN to increase the efficiency of the methods.

5.2.6 Modified IF/ELSE Block

For the IF/ELSE block the combination of the split tree and equilibrium indicates that precedence should be given to the equilibrium level and below when implementing the function and to accommodate these factors the IF/ELSE has been modified (in Algorithm 4).

The first modification to SLM is to create an outer IF/ELSE block to determine if the symbol percentage is less than or equal to equilibrium percentage. The addition of the block allows L_e and below to have precedence in the IF/ELSE function. The second modification is to reverse the order of the comparison of all the symbol percentage above L_e. The modification allows for the ELSE portion of the outer block to operate on the larger levels first. The change also increases the efficiency of the IF/ELSE function.

Algorithm 4 Revised IF/ELSE Block for SLM

The IF/ELSE block is used to split the space at equilibrium L_e and give precedence to equilibrium L_e and below.

---

if Symbol Percentage $\geq 2^{-L_e}\%$ **then**
  if Symbol Percentage $< 2^{-(L_e-1)}\%$ **then**
    Return length $= L+1$
  else
    if Symbol Percentage $< 50\%$ **then**
      Return length $= 3$
    else
      Return length $= 2$
    end if
  end if
else
  if Symbol Percentage $\geq 2^{-(L_e+1)}\%$ **then**
    Return length $= L+1$
  else
    if Symbol Percentage $\geq 2^{-(L_e+2)}\%$ **then**
      Return length $= L+1$
    else
      Return length $= L+2$
    end if
  end if
end if

5.2.7 SLM Example

We can see the relationship between SLM and Shannon-Fano-Elias by examining the same example used in Chapter 2. Given five arbitrary symbols: A, B, C, D and E represented by their known symbol probabilities (A:35, B:17, C:17, D:16, E:15). The numbers are in sorted order for convenience, but as stated this is not required. SLM selects the first value, A:35 and using the comparison model maps the symbol directly to the length of 3 bits and the offset describe by Shannon-Fano-Elias in (2.14) is used to set the width. The combination of width and depth produces the same code word 001 to represent the symbol. SLM continues by selecting each of the remaining values, B:17, C:17, D:16, E:15 and uses the comparison model to map the symbols directly to the length of 4 bits and the offset describe by Shannon-Fano-Elias. The codes for the four symbols acquired by SLM are 0110, 100, 1100 and 1110. The final result is the same code detailed in Sec. 2.7.2 with the same entropy $L(X) = (A : 35\% \times 3) + (B : 17\% \times 4) + (C : 17\% \times 4) + (D : 16\% \times 4) + (E : 15\% \times 4) = 3.65$ as Shannon-Fano-Elias. The advantage is in the speed produced by using less comparisions to reach the same location. This concept is illustrated in Fig. 5.6, which clearly shows the inefficient use of the system resource, this lack of compression is corrected in SLMN.

5.2.8 Comparison Model Pros and Cons

In both SLM and Shannon-Fano-Elias we are creating a unit vector by dividing entropy by a known value. In Shannon-Fano-Elias the known value is P_i which results in a unit vector for the length. The `ceiling` function and the $+1$ complete the length in bits.

SLM uses the known value of P_c, where $P_i \geq P_c$ and the range of P_i is $P_c = 2^{-l} \leq P_i < 2^{-l+1}$. The division of P_c results in a unit vector for length. A comparison between P_i and P_c determines the length in bits for the range and the $+1$ implicitly added to the return value.

In both SLM and Shannon-Fano-Elias the simplification of the entropy equation allows for a comparison of a vector to an ideal dimension in the entropy and certainty space. It may be transparent, but the logarithm function used by Shannon-Fano-Elias is a series of comparisons to determine the appropriate number of expansions to represent the value P_i as explained in Sec. 2.3. The comparisons are in addition to recursive multiplications to determine the mantissa. The advantage of SLM over Shannon-Fano-Elias is the number of comparisons is not mandated, as it is in the logarithm function, by an outside attribute. Furthermore, the number of comparisons can be fitted to the data to enhance the speed of the compression.

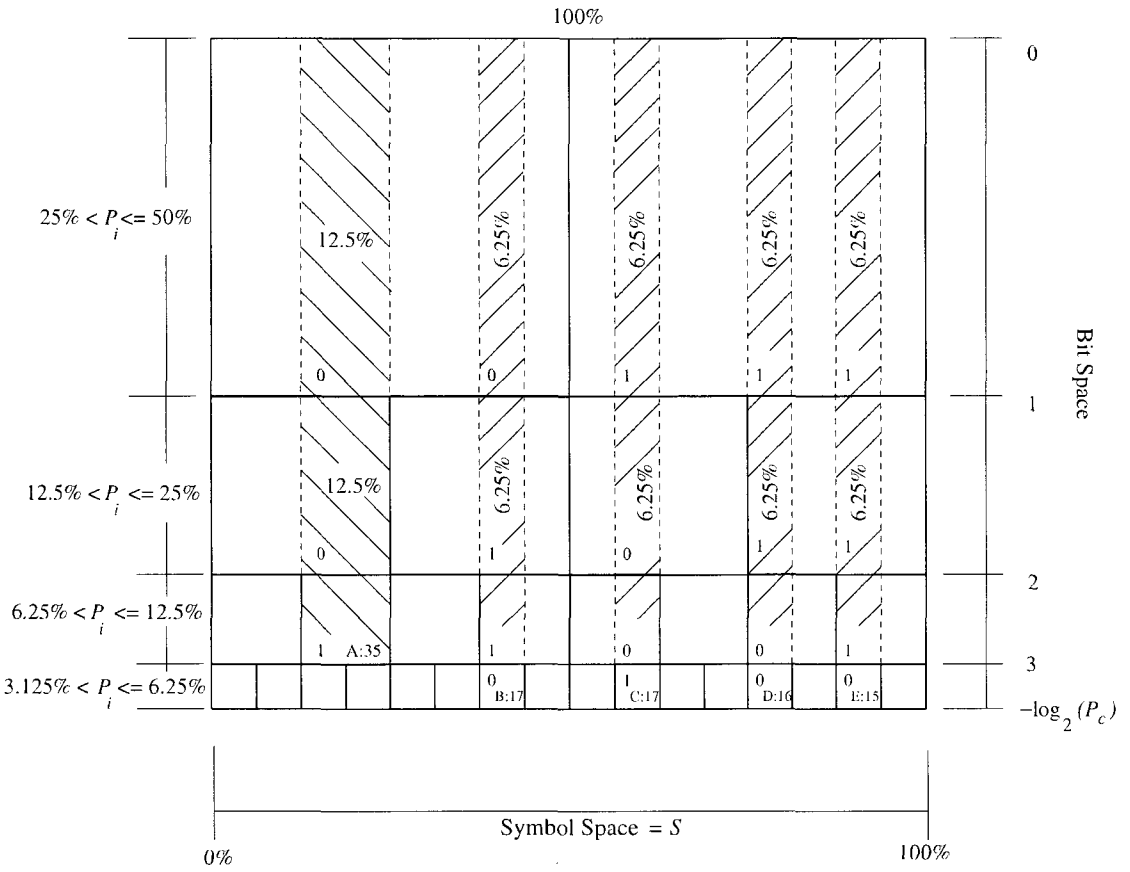


Figure 5.6: Illustration of SLM algorithm. The hashed area in MaCE represents the use the certainty of the system to represent the entropy of the data elements. The comparisons on the left represent the logic to determine the location.

The variability of the comparisons could also be viewed as a disadvantage as the implementation of an IF/ELSE block is biased toward an average expected distribution to minimize the number of comparisons required. The disadvantage would be more prominent if not for the current usage of prefix-free variable length compression algorithms. The two-step compression model mentioned in Sec. 1.2 first uses an algorithm that generates a fixed length code word. The second step uses the prefix-free algorithm based on the encoding table produced by step one. The use of the preceding algorithm to generate a fixed length code word has two effects that are beneficial.

First, the fixed length algorithms focus is to have an equal distribution of the symbol percentage values to maximize the compression. The closer to equilibrium the combinations of the characters become to the fixed length code words the better the compression in terms of entropy. The mapping to equilibrium is done by combining strings of characters together under a common code word. The effect is beneficial for SLM because the preceding algorithm generates a series of symbols with percentages closer to equilibrium which

is where SLM has the least amount of comparisons.

The second effect is that we know in advance how to implement the IF/ELSE statements based on the fixed length that the preceding algorithm generated. SLM implementation is centered around equilibrium having the greatest number of comparisons possible. Since the bit values are fixed the equilibrium value is also fixed. This removes the variability of the position of equilibrium based on the number of symbols in the data set. Knowing the equilibrium value allows for the implementation of SLM to be designed around equilibrium to minimize the number of comparisons without having to modify the code or the added computation require to handle cases that do not exist.

For example, if the goal is to compress the data to an 8 bit fixed length scheme, the algorithm used in step one would create 256 code words of fixed length with a value range from 00000000 to 11111111. Each one of the code words would have a combination of symbols that map close to equilibrium defined by the 8 bit representation. Although a few values may vary from the norm, the majority will fall in the sweet spot of SLM.

The code created by SLM has the same problem as Shannon-Fano-Elias in terms of compression ratio. By only concentrating on one dimension of the two-dimensional space the result returns less then optimal results. Sec. 5.3 uses the framework to decrease the entropy $L(x) \leq H(x) + 1$.

5.3 SLMN: Decrease Entropy to $H(x) + 1$

Although ideally we can represent entropy exactly, this is not the case in the real world. In the real world the limitations are placed on the mapping to the system representing the symbols in the space. The lack of granularity of the system requires having to adjust to the differences between the actual ability of the system and the theoretical ideal location to achieve results close to optimal compression.

The purpose of SLMN is to demonstrate the usefulness of MaCE to decrease entropy. SLMN uses the concept of midpoints introduced in Sec. 5.3.1 to make decisions on where to place a given symbol. The method begins by dividing the symbols over the bit space based on the symbol percentage. Each division is kept in sorted order to increase the efficiency of the compression. Once the data is sorted, the second step adjusts the symbol location based on the difference between the symbol percentage and the optimal percentage. This step also includes the formation of the code word. The code word is generated similar to Shannon-Fano-Elias in that we use the decimal value to denote the location.

In [37] the focus was to design a system that combined compression and encryption. The authors propose a mechanism to construct the code by keeping track of the space avail-

able in the tree and mapping to the first available location with enough space. The tracking mechanism replaced the need `ceiling` and the $+1$ resulting in a better compression but still achieving $H(x)+2$. SLMN uses a tracking mechanism as well, but the goal is to reach $H(x)+1$.

In SLMN, we use the combination of the probability P_i of symbol i and the difference between the previous symbol percentage P_{i-1} and the actual percentage P_c representing the node where the symbol is mapped to determine the location. We use the comparison model of SLM and the concept of midpoints to select the location. The net effect is averaging out the differences between the P_i and P_c to achieve a compression of $H(x)+1$.

5.3.1 Midpoint Spatial Mapping (MSPM)

The purpose of a midpoint is to define a position between two locations of interest. In this case, the points of interest are the levels above and below P_i as defined by $\lceil -\log_2(P_i) \rceil$ and $\lfloor -\log_2(P_i) \rfloor$. The midpoint has some useful properties when interested in measuring distance. The midpoint provides a third point of reference to do comparisons and provides a mechanism to quickly determine which endpoint is closer to a given location in the range.

For example, given an arbitrary symbol: $(A : 0.45)$ known probability of occurrence 45%. $P_i = 45\%$ and the $\lfloor -\log_2(P_i) \rfloor = 1$ and $\lceil -\log_2(P_i) \rceil = 2$. The midpoint in terms of bits is 1.5. The $-\log_2(0.45) \approx 1.15$ and by using a comparision to the midpoint we know that the point is closer to the floor than the ceiling.

Applying this concept to MaCE requires the determination of the midpoints for the map based on percentages rather then using the logarithm function and comparing bits. The naive approach would simply add the two adjacent percentages and divide by 2. For example, the first level $L_1 = 50\%$, the second level $L_2 = 25\%$ and combined $((L_1 + L_2)/2) = 37.5\%$. The problem with this approach is that the division assumes that the growth between levels is linear and the resulting midpoint is incorrect. The vertical growth rate is actually logarithmic which makes the midpoint $2^{-1.5} \approx 0.3536$. The approximation is due to rounding to 4 decimal places for the purpose of illustration.

We use a less computation approach based on the growth rate to avoid the computation of the exponentiation. The approach is based on the fact that even though the growth rate is logarithmic, the rate is constant at $-\log_2()$. This means that once we calculate the first midpoint we can calculate all the subsequent midpoints by simply dividing by 2. For example, from midpoint concept we have the first midpoint as $2^{-1.5} \approx 0.3536$ and the second as $2^{-2.5} \approx 0.1768$ which is half of the first. The original value can be stored as a constant and all subsequent midpoints are calculated by division rather then using

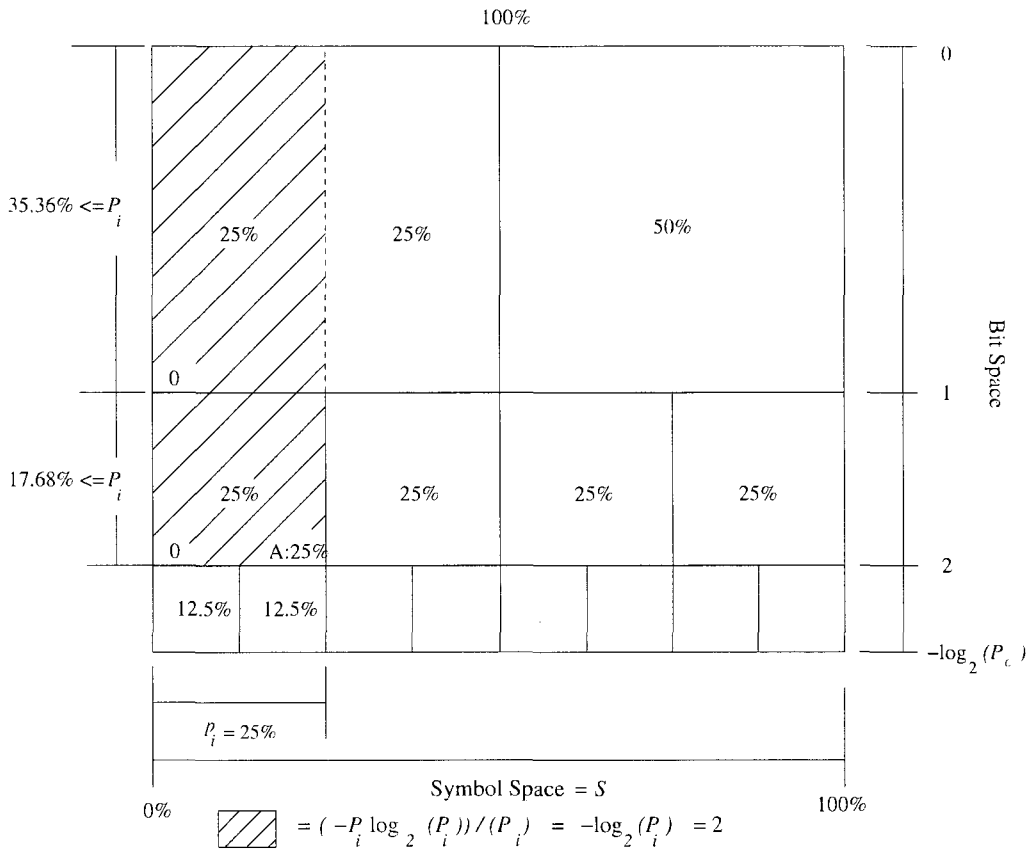


Figure 5.7: SLMN: Midpoint mapping. The hatched are represents the entropy for A: 25%. The comparision to the midpoints is displayed on the left side of the space. Based on the comparisions A is placed in the two bit position 00.

the exponentiation. Although the rounding error in the approximation maybe an issue for other applications, any rounding done by a computer for the exponentiation or the repeated division will equate in equivalent results.

Of even greater importance is that we actually have midpoints between the levels in MaCE by which we can map P_i to the closest location. This concept allows us to modify SLM to map the symbols based on the proximity to the level by using the midpoints.

5.3.2 Using Midpoints to Enhance Select Level Method to $H(x)+1$

SLM used the percentages of certainty P_c to determine the location suitable for a comparable algorithm to Shannon-Fano-Elias. SLMN will expand upon this concept by using the midpoints as in Sec. 5.3.1 to map the symbols closer to the location of P_c. By mapping the symbols closer to the P_c the entropy will decrease to $H(x)+1$.

Figure 5.7 shows that the comparison model has changed in reference to Fig. 5.2. The comparison on the left are based on the midpoints in Fig. 5.7 while it is based on the oper-

ations defined by $\log_2()$ in Fig. 5.2. The comparison to the midpoint allows for a better fit of the symbol P_i to the space available represent by P_c. In the figure $A : 25\%$ is mapped to a two bit position in contrast to SLM and Shannon-Fano-Elias which mapped the symbol to a three bit position. The application of SLMN has the effect of decreasing the entropy required to map the symbol. It removed the excess entropy related to 000 not being utilized and decreases the length if the code word from 001 to 00.

The side effect of this process is the removal of the fitting function of the `ceiling` and $+1$. This removal has a benefit and a drawback. The benefit is the decrease in entropy in representing the symbol. The drawback is that without the fitting function we need to address the conflict that can arise. We could use a technique similar to the one used in [37]. By keeping track of the space used the symbols could be mapped without conflict. Another option is to refit the data by taking another pass through the encoding table and resolving any conflicts. The refitting process can also be used to assign the code word to the symbols. If the space in any level is exceeded the lowest percentage symbol can be moved down a level to resolve the conflict.

Since we are changing the code in SLMN, we need to insure that the code is decodable and encodable. First, we need to determine if the resulting code violates the Kraft inequality summarized in Sec. 2.7.1. The fitting portion of the method traverses level by level down through the space and uses two counters to accomplish this task. The counters represent the nodes used by the previous symbols and the space available given the depth 2^L. If more than one symbol remains to be placed, the level is incremented to increase the space available. This insures that the Kraft inequality is not violated as the width of the space from 0 to 1.0 is the quantity Kraft is measuring and the counter insures that this constraint is not violated.

Unfortunately, this alone does not insure the values meet the prefix-free requirement. We must also insure that the mechanics of the method does not place values below previously mapped symbols. SLMN uses the values of the certainty P_c of the spatial nodes to accomplish this task. The spatial nodes represent the width of symbol space used by mapping the symbol to a location. The method uses the certainty percentage that the symbol displaced as the offset for the next symbol. The offset insures that the subsequent symbols cannot be placed in a level below the space previously occupied. This concept is similar to Shannon-Fano-Elias except instead of using the symbol percentage as the offset we are actually using the spatial value P_c of the symbol space as the offset. Since we cannot place a value below a used space and all subsequent values are to the right of the location, the Kraft inequality and the encoding table contain only values that are codeable.

Another benefit to SLMN is that the data can be partially sorted based on the symbol

percentage. If the granularity is not high enough to completely sort the data we can find the lowest value in a level to move to subsequent levels. This will decrease the time it takes for the bucket sort portion of the algorithm.

Algorithm 5 SLMN Pseudo Code

SLMN uses the concept of midpoints to sort the symbols and then assigns the location based on the proximity of P_i to the closest P_c. The variable D_{pc} is the sum of the differences from P_i to P_c. .

---

for all Symbols i to n **do**
  Bucket sort the symbols based on the midpoints
end for
for all Non-Empty Levels **do**
  for all Symbols on level l to n_l level contents **do**
    if $D_{ic} < 0$ **then**
      Calculate average difference $= D_{ic}$
    else
      Calculate average difference $= D_{ic}/n - (i-1)$
    end if
    while Midpoint $>$ average difference **do**
      Move down a level
    end while
    if Level Not FULL **then**
      Place Symbol at level dictated by P_i and average difference
    else
      Place Symbol at level below that dictated by P_i and average difference
    end if
    Assign Code Word based on P_c
    Calculate D_{ic}
  end for
end for

5.3.3 SLMN Example

We can see the compression improvement for SLMN over SLM by examining the same example in Sec. 5.2.7. Given five arbitrary symbols: A, B, C, D and E represented by their known symbol counts (A:35, B:17, C:17, D:16, E:15). SLMN first sorts the data based on the level percentages provided by MaCE. The initial sort is a linear sort based on the concepts derived in Chapter 4 and provides the knowledge required to make the algorithmic decisions. The method takes a second pass over the elements to calculate the difference between the P_i and the P_c at the mapped level for symbol i. The method selects the first symbol, A:35 and using the comparison model maps the symbol directly to the length of 2 bits. The offset of zero is used as the value represents that no symbol has been previously mapped, the symbol space is empty. The zero offset is different from Shannon-Fano-Elias and SLM as they used $\overline{F}(P) = (P_i)/2 + \sum_{1<i} P_i$ to set the width which created the inefficiency in the compression. The combination of width of 0 and length, 2 bits, produces the code word 00 to represent the symbol. The difference, D_iC is calculated as the difference between P_c of the location and the probability P_i of the symbol i. The total previous difference is used only when the previous overhead is negative. In all other cases an average difference is used to adjust subsequent symbols in the space. In this case $D_iC = 35\% - 25\% = 10\%$ and the difference is passed to the next iteration. The average difference addesses the problems with the 32 bit system to map all the values and has the effect of leveling out the previous differences. The use of the average comes at a cost of optimality of the compression, but insures we stay within the system constraints.

SLMN continues by selecting the next symbol (B : 17) and combines the value of P_i with the average difference $10\%/4 = 2.5\%$, so $17\% + 2.5\% = 19.5\%$. The combined total using the comparison model maps the symbol directly to the length of 2 bits. The offset of $P_c = 25\%$ represents the width used and the combined width and length is equivalent to 01 in binary. In this case $D_iC = 27\% - 25\% = 2\%$ and the difference is passed to the next iteration.

The third symbol (C:17) is selected and the value of P_i is combined with the average difference $2\%/3 = 0.667\%$, so $17\% + 0.667\% = 17.667\%$. The combined total using the comparison model maps the symbol directly to the length of 3 bits. The offset of $P_c = 50\%$ represents the width used and the combined width and length is equivalent to 100 in binary. In this case $D_iC = 19\% - 12.5\% = 4.5\%$ and the difference is passed to the next iteration. At this point it is noted that the value deviates from Shannon-Fano code with a code length of 2 as this is caused by the average difference being used rather then the full value.

The fourth symbol (D:16) is selected and the value of P_i is combined with the average

difference $6.5\%/2 = 3.25$, so $16\% + 3.25\% = 19.25\%$. The combined total using the comparison model maps the symbol directly to the length of 3 bits. It is noted at this point that 19.25% would map to the higher level, but we are fitting top down and will not place the symbol in the higher location to avoid conflicts with prefix free and the 32 bit system requirement. However, this could be used as an indicator to correct previous symbol locations using back propagation or pointer techniques to increase the compression efficiency. The offset of $P_c = 62.5\%$ represents the width used and the combined width and length is equivalent to 101 in binary. In this case $D_iC = 19.25\% - 12.5\% = 6.75\%$ and the difference is passed to the next iteration.

The fifth symbol (E:15) is selected and the value of P_i is combined with the average difference $6.75\%/1 = 6.75\%$, so $15\% + 6.75\% = 21.75\%$. The combined total using the comparison model maps the symbol directly to the length of 3 bits. It is noted at this point that 21.75% would map to the higher level as well. The offset of $P_c = 75.0\%$ represents the width used and the combined width and length is equivalent to 110 in binary. In this case $D_iC = 21.75\% - 12.5\% = 9.25\%$. The remaining difference means we did not get a ideal mapping of all, i.e., 1.0, of the symbol space to all, i.e., 1.0, of the bit space.

The final result is the code with the entropy $L(X) = (A : 35\% \times 2) + (B : 17\% \times 2) + (C : 17\% \times 3) + (D : 16\% \times 3) + (E : 15\% \times 3) = 2.48$ with the ideal of 2.23284. The advantage is the ability to produce code within $H(x) + 1$ with very little computational effort via the comparision model. This concept is displayed in Fig. 5.8. The visual clearly shows added efficiency in terms of compression over Shannon-Fano-Elias. The slice remanining unfilled in this case is due to the averaging function. Figure 5.9 shows SLMN without the averaging function and produces the same code as Shannon-Fano in Sec. 2.5.1.

5.3.4 Pros and Cons of SLMN

The two main differences between SLMN and SLM is the inclusion of the linear sort and the new fitting function. Recall that the by-product of the extra space mandated by the fitting function for Shannon-Fano-Elias was the algorithm did not require a sort. The reason the sort is important to the other algorithms is the ability to work on the assumption that the next symbol percentage is greater or less than the current symbol dependent on order chosen. The algorithm without this assumption needs enough space to accommodate any character and the use of the `ceiling` and the $+1$ or the equivalent in SLM. The algorithms within $H(x) + 1$ achieve the greater compression by using the sort and the knowledge it generates.

SLMN integrates the sort into the algorithm using the midpoints of the spatial percent-

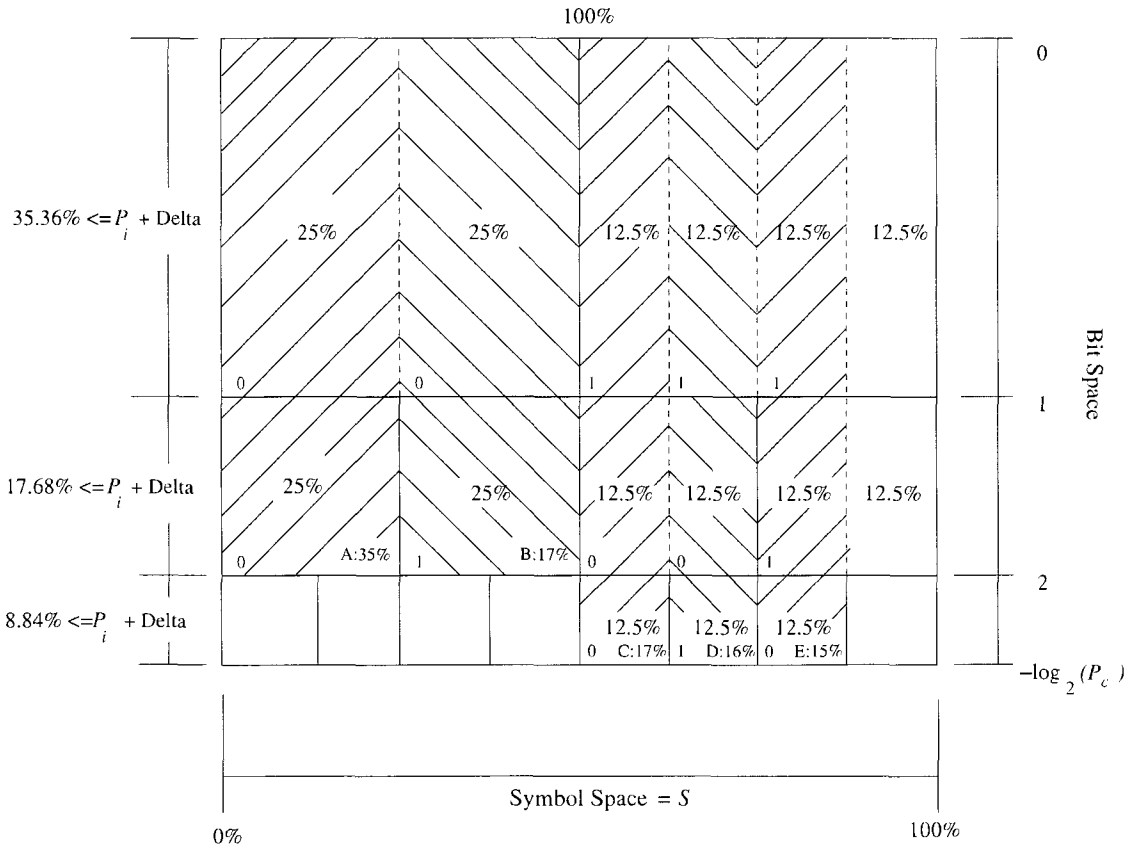


Figure 5.8: Illustration of SLMN algorithm with average. The hashed area in MaCE represents the use of the certainty of the system to represent the entropy of the data elements. The comparisons on the left represent the logic to determine the location based on the P_i and P_c values.

age. This alone will pre-processes and compress some of the data points. The integration helps speed up the total time by using the sort portion to partially build the encoding table. For example, the best case is when the data is sorted all on one level. Where the algorithm only does n iterations of the bucket sort, $\log_2(n)$ iteration to move to equilibrium and n iterations to place the level. The last n iterations could be short circuited, if required, by checking the size available at level to the number of symbols contained. The main benefit to this model is that the sort is not a comparison sort, so that this portion of the algorithm can be accomplished in $O(n)$ time versus $O(n\log_2(n))$.

The second difference in SLMN is the new fitting function. The function resolves the conflict by making an extra pass through the data. This fitting function could also be tied to a series of flag values to minimize the fitting operation to only conflicts. SLMN's fitting function is also different in its purpose as it is not simply to resolve conflicts. The fitting algorithms purpose is also to create the code words as SLMN defines it's code words based

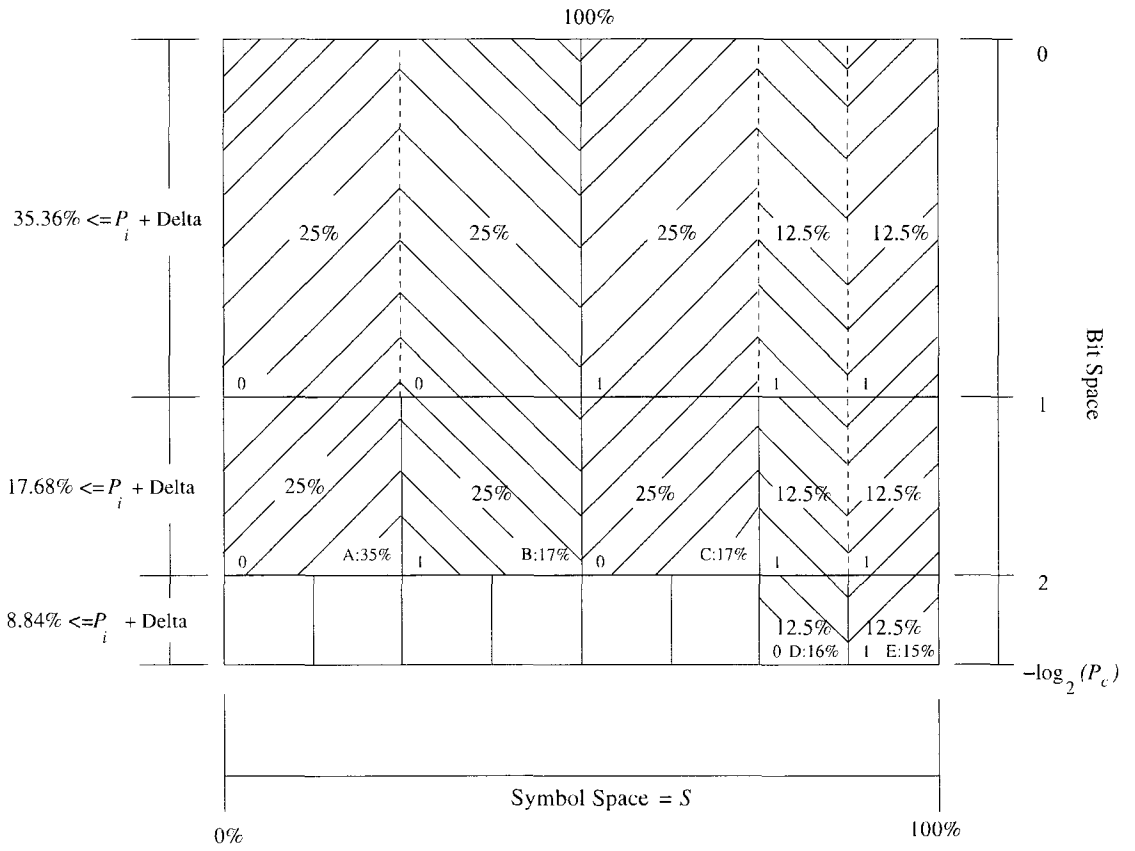


Figure 5.9: Illustration of SLMN algorithm without average. The hashed area in MaCE represents the use of the certainty of the system to represent the entropy of the data elements. The comparisons on the left represent the logic to determine the location based on the P_i and P_c values. Direct comparison between SLMN with and without the average shows the effect of the 32 bit requirement even on small n values.

on the certainty space utilized by the symbols. This is in contrast to Shannon-Fano-Elias which utilizes the symbol probability or the uncertainty to define the code words.

The first benefit to SLMN is the achievement $H(x) + 1$ by fitting the data closer to the certainty represented by the bit space. SLMN accomplishes the task with a comparison model based on the midpoints between levels in the space. The second benefit is SLMN does not require the data to be sorted as the sort is integrated into the algorithm. Since the data is in sorted order when the code words are assigned we get the benefit of a incremental code similar to Shannon-Fano. This has the advantage of easy implementation of a lookup table to accomplish the encoding and decoding of the actual data.

Averaging of the differences has both a positive and negative effect. The positive effect is that the average evens out the length of the code word similar to Shannon-Fano code. The code lengths being relatively flat also enables an efficient lookup. The negative effect is that the code does not reach its full potential. In certain cases the symbols maybe assigned a

longer code word based on the average which leaves a difference in values. The drawback to the approach is the extra step to either keep track of space or refit. Depending on the approach chosen the severity of the drawback varies. The refit comes at the cost of overall speed and the tracking mechanism saves some of the speed at the cost of space.

5.4 Beyond $H(x)+1$

We consider entropy in the physical world as a series of particles in a solution. If the particles are equally distributed throughout the solution, the suspension can be viewed as in equilibrium where the uncertainty represented by the solution is an equal measure from the certainty of the particle. There are two ways to increase the certainty and decrease the uncertainty of this system without altering the number of particles. One is to move the particle around in the system reducing the distance between the particles and this, in effect, reduces the uncertainty between the particles and certainty remains constant. The second way to decrease the entropy by reduction of the solution used to separate the particles, while the number of particles remain constant in both relative distance from each other and in number. The reduction of the solution reduces the uncertainty of the system as a whole and changes the scale by which the certainty is measured. These same concepts apply to data compression as a symbol represents the particle and the code word space represents the solution. In the previous chapters we have been concentrating on the first way to reduce entropy in a system as we have been dealing with a constant binary system to represent the data. In this section we examine the concept of changing the system to reduce the entropy. First, we look at the current meaning of $H(x)+1$.

Entropy is used to classify compression algorithms, but without understanding its meaning, it is of little use. Entropy as described earlier taken to be the limit of compression. But what about the $+1$ portion of the equation $H(x)+1$? Ideally, all the symbols would map to the same depth of the space denoted by the theoretical ideal $\mathbb{I}$. For this to occur the space would have to be divided sufficiently to represent each symbol at the precision required. In binary, this is not the case, as the granularity increases in powers of 2 and the lack of granularity is the primary reason the $+1$ exists. The $+1$ represents the portion of one bit that cannot be mapped to the certainty imposed by the binary structure. We can analyze the two extremes to visualize the effects of binary of the data distribution. One extreme has all symbols at equilibrium and the other extreme is when one value represents almost certainty or $P_i \approx \%100$.

The first extreme is when the symbol percentages are all equal and the condition contains maximum entropy. Take the case of two symbols. When both symbols are equal they

represent 50% of the space or half a bit each with a total entropy between the two equal to 1 bit. This condition can be demonstrated by using the H equation and plugging in the values for $1/2$. $L(x) = -(1/2\log_2(1/2)) + -(1/2\log_2(1/2)) = 1$. Both sides of the equation have an equal amount of uncertainty and combined that equals 1 or 100% uncertainty. Both the theoretical ideal $\mathbb{I}$ and $L(x)$ equal 1, so the $H(x) = L(x)$. This makes sense and no additional space is used by either symbol. The entropy of the symbols is mapping exactly to the certainty of the system with no loss in efficiency. The certainty of the system must be able to map the equilibrium condition, otherwise the lossless requirement cannot be meet.

For the second extreme, we will incrementally increase one of the two values to 100%. As we modify the first symbol higher in percentage the second symbol naturally has the decrease equivalently. For example, using $P_i = 3/4$ results in $H(x) = -(3/4\log_2(3/4)) + -(1/4\log_2(1/4)) = 0.5$. The overall uncertainty decreased by $\approx 19\%$. This trend continues as the first symbol increases toward 100% the logarithm approaches 0 and the second symbols P_{i+1} also approaches 0, hence the entropy of the entire system approaches 0. However, in the actual binary system it is required that we use one bit to represent the two symbols. The actual entropy is calculated by $100\% \times 1\text{bit}$ and $0\% \times 1\text{bit} = 1 \times 1 + 0 \times 1 =$ 1bit of entropy, this means $H(x) \approx 0$ and $L(x) \approx 1$.

What this demonstrates is when the two symbols are equal, actual entropy and theoretical entropy are equal as well, because the system is able to handle the mapping of entropy onto the certainty of the system. However, as demonstrated, as the two symbols diverge the entropy becomes less, but the actual model is unable to compensate. Granted there is no need to run a compression algorithm on 2 symbols when the space is in binary, but the example illustrates the importance of the $+1$ as a limiting factor. This property is expanded to include more than only two symbols by simply adding the relationships of all the values as done in the overall H equation.

This relationship also shows that regardless of the mapping method, the end result will always have the $+1$ limitations. The reason for understanding this limit is to give insight on how the limit can be addressed.

5.5 JAKE

Since the limit of $+1$ relates to the ability to map P_i to P_c and P_c is a property of the system, we must change the system representing the symbols to decrease this limitation. P_i is only limited by the ability to represent the probability in the system, i.e., the floating point representation. P_c is represented by the division in the system that represent the finite nature of certainty in the cases exampling binary. The relationship between P_i and

P_c illustrates that by choosing the correct base the actual entropy can be made closer to theoretical entropy. For example, changing the base to decimal will theoretical change the limit to 0.1 for a prefix-free code. By changing the base and applying the same framework, it maybe possible to get even better compression. This is the same concept as removing the solution in the physical environment. As we remove the solution the entropy decreases and the certainty of the particles increase. This is a trivial solution if all the particles are of equal weight. In data compression the interest is in symbols of varying P_i so the task is to find the most $P_c \approx P_i$.

Changing the base gives the initial compression, but will only return ideal results if all the symbol percentages map directly to the space. Since we are not interested in the trivial solution, the methods proposed in this dissertation are applicable. The beauty of the comparison model, used for SLM and SLMN, is that in order to map a symbol in another base all that is required is an original set of elements to define the base. In binary, we have two possible elements an 0 and 1. In decimal, we have ten elements 0 thru 9. In hexadecimal, we have sixteen elements 0 thru F. The wider the base the less entropy is inherent to the system. Once we have the original base, the expansion is the sequence of elements is expanded in powers of the base. We introduce JAKE (Jacobs, Ali, Kolibal Encoding) to extend both SLM and SLMN into other bases.

In the real world not all symbol percentages are equal so the concept of simply changing a base does not work completely. Also, choosing a default base may not be the best solution as the granularity of the base may be too much or too little for the given requirements. Since the largest value represents the closest relationship to certainty it indicates that this value could be used to choose the base. Choosing the base where the largest $P_c = P_i$ or the majority of P_c's $= P_i$'s can be utilized to map the ideal. The reason is the largest difference in entropy in the system is the distance from the root node to the first level of the space. After this level all the other entropy distances decrease logarithmically. By choosing the base to represent the largest P_i we are insuring that the system will have the granularity required to map the entropy of the symbols to the entropy of the system. Choosing the correct implementation can model entropy as close as possible, the only limitation is the actual system requirements. The change of base via JAKE for SLM and SLMN increases the efficiency of the compression. The change in base also has the potential of increasing the efficiency of the algorithm as the IF/ELSE block represents the depth of the space and that depth is decreased by the base change. The Pseudo code for JAKE is displayed (in Algorithm 6) and the IF/ELSE implementation is displayed (in Algorithm 7).

In addition to using the largest $P_i \approx P_c$ or the most $P_i \approx P_c$ we are able to choose other system parameters based on the requirement for compression and flexibility. Since the sys-

Algorithm 6 Pseudo code for JAKE

Code to returns the length value returned required in any base.

Change the base where the most $P_l \approx P_c$

Use SLM or SLMN logic to create encoding with modified IF/ELSE

Algorithm 7 IF/ELSE block for JAKE.

Code to returns the length value returned required in any base b.

The code shows the implementation without the concept of equilibrium for simplicity.

```
if Symbol Percentage ≥ b^-1 % then
    Return length = 2
else
    if Symbol Percentage ≥ b^-2 % then
        Return length = 3
    else
        if Symbol Percentage ≥ b^-L % then
            Return length = L + 1
        end if
    end if
end if
```

tem proposed is not limited to other requirements, we could use a intermediate base such as 60 to use a number with multiple prime factors. The reason the number of prime factors is important is due to the same reason the size of the base is important. Prime numbers are only divisible by themselves and 1, so any number that does not have a factor represented within the system will inherently not be exactly representable. This results in increasing the ending entropy difference between $L(x)$ and $H(x)$. The length of the expansion can be acquired using the same comparison model defining SLM and the additional fitting of SLMN can also be used. We have already seen this in action for binary. Shannon-Fano-Elias, SLM and SLMN use the decimal value to represent certainty in order to map the uncertainty defined by the length $-\log_2(P_i)$ while the Huffman and Shannon-Fano algorithms do not have this flexibility. This relationship is the main reason we are comparing the results to Shannon-Fano-Elias and not the other two algorithms in Chapter 6.

Table 5.1: JAKE tabulated results in base 3 with actual and ideal code lengths.

i	1	2	3	4	5	6	7
P_i	0.34	0.11	0.11	0.11	0.11	0.11	0.11
$l(P)$	1	2	2	2	2	2	2
$code(P)$	0	10	11	12	20	21	22
Ideal Length	0.982	2.009	2.009	2.009	2.009	2.009	2.009

Table 5.2: Huffman tabulated results in base 2 with actual and ideal code lengths.

i	1	2	3	4	5	6	7
P_i	0.34	0.11	0.11	0.11	0.11	0.11	0.11
$l(P)$	2	3	3	3	3	3	3
$code(P)$	00	010	011	100	101	110	111
Ideal Length	1.556	3.184	3.184	3.184	3.184	3.184	3.184

5.5.1 JAKE Example

The following example uses seven arbitrary symbols: A, B, C, D, E, F and G represented by their known probabilities of occurrence (A:34, B:11, C:11, D:11, E:11, F:11, G:11). The results of each of the sub-equations for JAKE is in Table 5.1 and the resulting code for Huffman encoding is in Table 5.2

The table illustrates the calculations for JAKE in base 3 and Huffman encoding in base 2. The ideal compression is also illustrated for both bases. Using the results we can calculate entropy and the relative efficiency. JAKE in base 3 achieved $L(x) = 1.66$ and with the ideal 1.659909274984051. Huffman encoding in base 2 achieved $L(x) = 2.63089395544898$ and the ideal $H(x) = 2.66$. The efficiency E of the compression for JAKE is $E = 1.659909274984051/1.66 = 0.99995$. The efficiency E for Huffman is $E = 2.63089395544898/2.66 = 0.98906$. The results show that JAKE would have a greater compression then Huffman in this case. Of greater importance is that the complexity would not be any greater than with SLM and SLMN as the only difference is the midpoints used for the comparisons. MaCE in base three is displayed in Fig. 5.10.

For the example we expanded to 7 characters to illustrate the effect of changing the

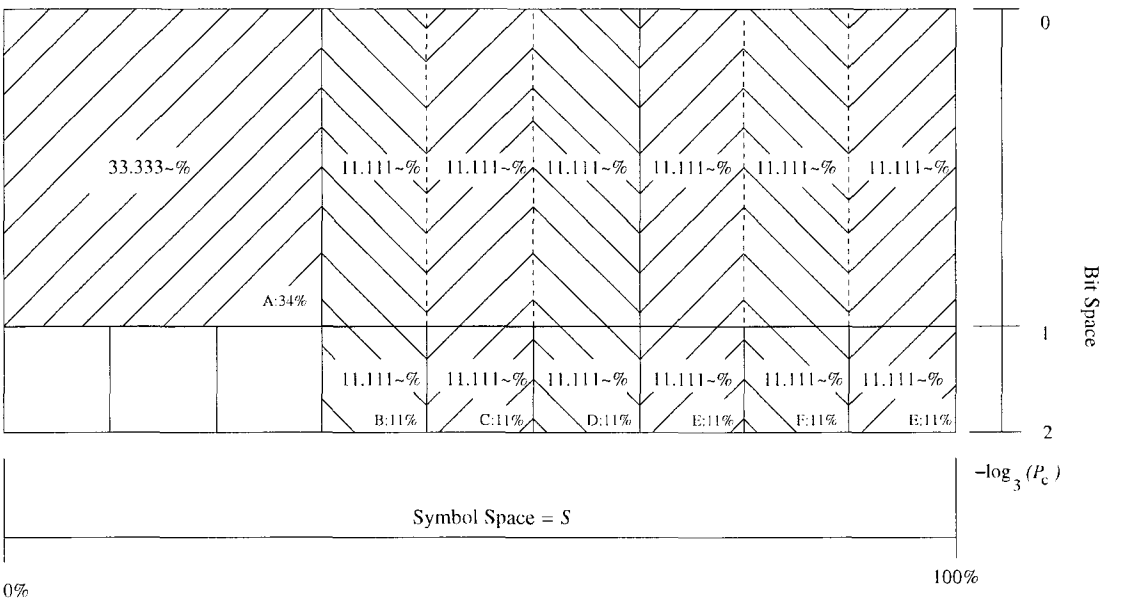


Figure 5.10: JAKE in base 3. MaCE illustrates the mapping of the symbols A:34, B:11, C:11, D:11, E:11, F:11, G:11 to their ideal locations in base 3.

base. The reason we did not use the previous example from the other algorithms is due to the 5 characters with the given percentages is not ideal for base 3. It was a better choice to stay in base 2. The choice of base 3 and 7 characters was also only to illustrate the point. In compression we will not be dealing with only 7 characters and base 3 is not an ideal base to use for JAKE or information systems. Also, Huffman does have the ability to compress in base 3, although that is not the typical implementation. The comparison illustrates that the change of base does have an effect on the compression efficiency in comparison to the theoretical ideal. Since the entropy and certainty area and the subcomponents are complementary, the ideal base chosen must try to match the P_i for as many of the symbols as possible. The concept of midpoints can help in this decision.

5.5.2 Arithmetic Code

The relationship between the algorithm and the $H(x)+d$, where d represents the limit of the system, is observed in Arithmetic encoding. Arithmetic encoding is the leader in lossless data compression in terms of actual data compression achieved, however, it does so with great complexity. Arithmetic encoding inherits some of its complexity from Shannon-Fano-Elias encoding in Sec. 2.7 from which it is lossy based. The majority of the complexity comes from the models required by Arithmetic encoding to reach optimal compression and the algorithms computational operations on m number of characters. The encoding of all m characters instead of n symbols and the use of models by Arithmetic encoding is

a departure from the previous algorithms covered in Chapter 2. This poses unique challenges when comparing the proposed methods to Arithmetic encoding that are still being researched. The implementation and example of Arithmetic encoding is covered in Appendix A. Only the analysis of the pros and cons of the method are covered here as the focus is to analyze the effect of the value of d in $H(x)+d$.

5.5.3 Arithmetic Code Analysis

Arithmetic encoding is significantly different from the data compression algorithms introduced in Chapter 2. These differences allow Arithmetic encoding to address the $H(x)+d$. The first difference is the lack of a one-to-one relationship between the symbol and the code word. Arithmetic encoding encodes one character (single instance of symbol) at a time and assigns the character a decimal code word. This means that each character in the data set will be assigned a different code, unlike the previous methods that assign each symbol the same code. The intermediate codes are generally not used, but are of interest for analysis. The final result is a single decimal number to represent the entire encoding of the document.

The lack of a one-to-one encoding allows the encoding process to remain in decimal and not convert to binary until the operation is completed. In Sec. 2.7, Shannon-Fano-Elias did most of the calculations in decimal and the use of binary was only to generate the code word for the symbol. Arithmetic encoding stays entirely in decimal and the conversion is not necessary in theory. The use of decimal over binary does have an advantage as Arithmetic encoding is not required to deal with full bit values except for the final generation of the code word. This allows an easier manipulation of the line between certainty and entropy in addressing the $H(x)+d$. However, since we are dealing with modern computers based on binary logic the decimal numbers themselves are represented in binary.

The third difference for Arithmetic encoding is the ability to change the base used with the same algorithm. As with JAKE the ability to change the base in which the algorithm is operating allows for Arithmetic encoding to pre-compress the data using the symbol to entropy relationship discussed in Sec. 4.7. This ability substantially reduces the work applied in the actual compression. It also allows for flexibility of the algorithm to adapt to changing environments and the ability to address the d in $H(x)+d$.

Lastly, Arithmetic encoding is heavily dependent on the model representing the data. The model supplies the certainty required to get the desired results close to the ideal. The algorithms in Chapter 2 acquire certainty from the sort and the binary system although the certainty may only be partially modeled. Arithmetic encoding does not require a sort or

the encoding table to have predetermined knowledge which makes the statistical model of utmost importance when making decisions on mapping entropy to the certainty of the system. An example is given in Appendix A to show the actions required for Arithmetic encoding.

In the given example, Arithmetic encoding did reach the optimal encoding as the model corresponded directly to the data. We can see from the example that Arithmetic encoding is capable of reaching close to $L(x) \approx H_M(x)$, where $H_M(x)$ is the limit defined by Shannon's theory of entropy given model M. In other examples it is also possible to show $L(x) = H_M(x)$. However, if the model M does not relate to the data, then results may not be optimal. In [6] they analyze the efficiency and determine the average length produced by Arithmetic encoding converges on the optimal with a large message sequence given the correct model. There are several models including Order-n Markov models, adaptive models and other stochastic modeling techniques. This indicates that the model select is of utmost importance at achieving $H(x) + d$.

In [6] the discussion continues to make a comparison to Huffman encoding. The advantage Arithmetic encoding has over Huffman encoding is the ability to represent a symbols with a fraction rather then full bits as is required by Huffman encoding. Arithmetic code is better at handling symbol sets that have large percentage values. Huffman encoding using binary starts to approach the $+1$ limitation in these conditions as demonstrated in Sec. 5.4. The comparison between Huffman and Arithmetic encoding also shows the relationship to $H(x) + d$. The farther Huffman encoding is from the larger entropy values in base 2 the better the overall results. Arithmetic encoding in decimal has the opposite effect as the key advantage is the granularity of the decimal system at the top of the space as shown by MaCE.

Direct comparison between JAKE and Arithmetic encoding is left for future work as the objective was not to compete with Arithmetic encoding. The insight into Arithmetic encoding is important to demonstrate the power of the base on compression. Further work will need to be accomplished in order to define additional relationships and enhancements to either JAKE or Arithmetic encoding.

5.6 Symmetry of Encoding and Decoding

An encoding table is simply a table mapping a symbol to a code word. In order to actually encode the message to the code words and decode the message back we must utilize this reference. This issue needs to be addressed for completeness. Typically, for static data the information is compressed once and decompressed many times which usually means the

algorithms are willing to sacrifice compression time for decompression time. Due to this difference in this type of data compression usage an asymmetric algorithm is tolerated in one direction. For real time data symmetry in encoding and decoding is more desirable as the data is usually sent and discarded once decoded.

Most encoding techniques use a coding table as do the methods in this dissertation. To create a symmetric set of methods to encode and decode the symbols for the method present a virtual tree structure is implemented on both sides of the table. To encode the symbol from the original message to the encoded message the symbol space is recursively divided until the symbol is reach in the table index. The index is subsequently used to reference the code word which replaces the symbol in the encoded message. This approach is a $\log_2(n)$ operation. No space is required for the tree as it is virtually represented in the recursive division of the table.

To decode the code word to the symbol we use the decimal representation of the binary word. The decimal number allows for easy comparisons based on the decimal value of the code word and the decimal values dividing the range space. Since the message is encoded using a decimal to binary length conversion, then each of the code words also represents a unique decimal number. This allows for a similar approach for decoding the message as encoding the message. We know the encoded symbols are divided across the range 0 to 1 so we use the decimal representation to split the code words until the table is reached. Again, using the index of the value reached to replace the binary string with the symbol at the index. This again will take $\log_2(n)$ time to decode the message. No space is required for the tree as it has a virtual representation in the recursive division of the table.

5.7 Summary

This chapter introduced three new methods for data compression. SLM uses MaCE to create a less computationally intensive method to create Shannon-Fano-Elias code. The basis of this method is the reduction of the entropy equation to contain only the probability of the symbol P_i. This allows for a direct comparison of P_i to the certainty defined by the system P_c to map the symbol to the appropriate length. The equality of the comparison model to Shannon-Fano-Elias is verified visually using MaCE.

The concept of equilibrium is also utilized in order to enhance the computational speed of SLM. We note that equilibrium represents the most comparisons possible for any level in the symbol space. We also note that the negative growth of the comparisons above equilibrium is steep compared to the negative growth below equilibrium. The concept of the split-tree, as defined by the binary values of 0 and 1, is also explored to further explain

the difference in the growth rates above and below equilibrium. The knowledge of the growth rates and the split-tree allows for a modification of the IF/ELSE comparison logic to enhance efficiency of SLM. The pros and cons of SLM are examined in comparison to Shannon-Fano-Elias and in terms of usage.

SLMN uses MaCE to construct a method with an integrated sort to reach $H(x)+1$ in $O(n)$ total time. The concept of midpoints is defined which gives the ability to directly compare values to a structure defining certainty. The midpoint goes to a single point of comparison to determine placement of the symbol in the structure. By placing the values closer to the theoretical optimal the difference between theoretical entropy and the actual entropy is reduced to $H(x)+1$. The pros and cons of SLMN are also explored.

We also explain the $+1$ requirement to the entropy algorithms developed in binary. From this knowledge it is possible to define new algorithms that can decrease the entropy by defining compression systems that closely emulate the data characteristics. SLM and SLMN are extended in JAKE to change the base used in order to address the $+1$ in $H(x)+1$. Arithmetic encoding was examined as the method has the ability to use multiple bases to increase the compression efficiency. In the final sections we explain the concept of symmetry of encoding and decoding information and explain a method to encode and decode in $\log_2(n)$ time.Chapter 6 covers the experimental results of SLM in terms of CPU cycles and SLMN in terms of entropy and compression.

Chapter 6

EXPERIMENTAL RESULTS OF SPATIAL METHODS

6.1 Results of SLM and SLMN

Chapter 5 introduces two methods which utilize the knowledge gained from MaCE. This chapter analyzes the results of SLM in comparison to Shannon-Fano-Elias in terms of CPU cycles and shows the computational advantage to the comparison model. We also compare SLMN to Shannon-Fano-Elias and the theoretical ideal in terms of compression in order to demonstrate the versatility of the mapping methods.

6.2 Test Environment

SLM and SLMN are evaluated on an Intel Core 2 Duo @ 2.00 Ghz and 4 GB RAM. The operating system for the machine is Windows Vista 64 bit and the code was compiled using Bloodshed Dev-C++ Version 4.9.9.2. The methods processing speed was measured by using the kernel function `QueryPerformanceCounter()` [33]. The dynamic-link library (dll) function is included in Windows.h header file and is used to query the performance-counter for the current CPU counter. The counter increments at each processor cycle and a subsequent call to `QueryPerformanceCounter()` and a calculation of the difference, after the target computation is concluded, reveals the number of CPU cycles required for the computation. One issue with this method is the inability to measure the effect of interrupts caused by other processes on the machine or the operating system during the time of execution. In order to mitigate the effects of the interrupts only the best of 100 repeated cycles of the algorithm with the same data set are used for analysis. The H equation for entropy is utilized to measure the compression and a calculation using the data percentage and the number of bits in the encoding table is used for the size comparison. The compression ratio is calculated based on the original file size and the compressed file size.

The test data consists of a static 256 number of symbols representing an 8 bit input due to the current usage for prefix-free algorithms. We can assume the number of characters is static as the current compression model is a two step process using a fixed length compression scheme followed by a variable length prefix-free scheme. The ability to optimize the prefix-free algorithm to this process is possible since the number is static and the ability to optimize is a notable trait for the methods that have the ability.

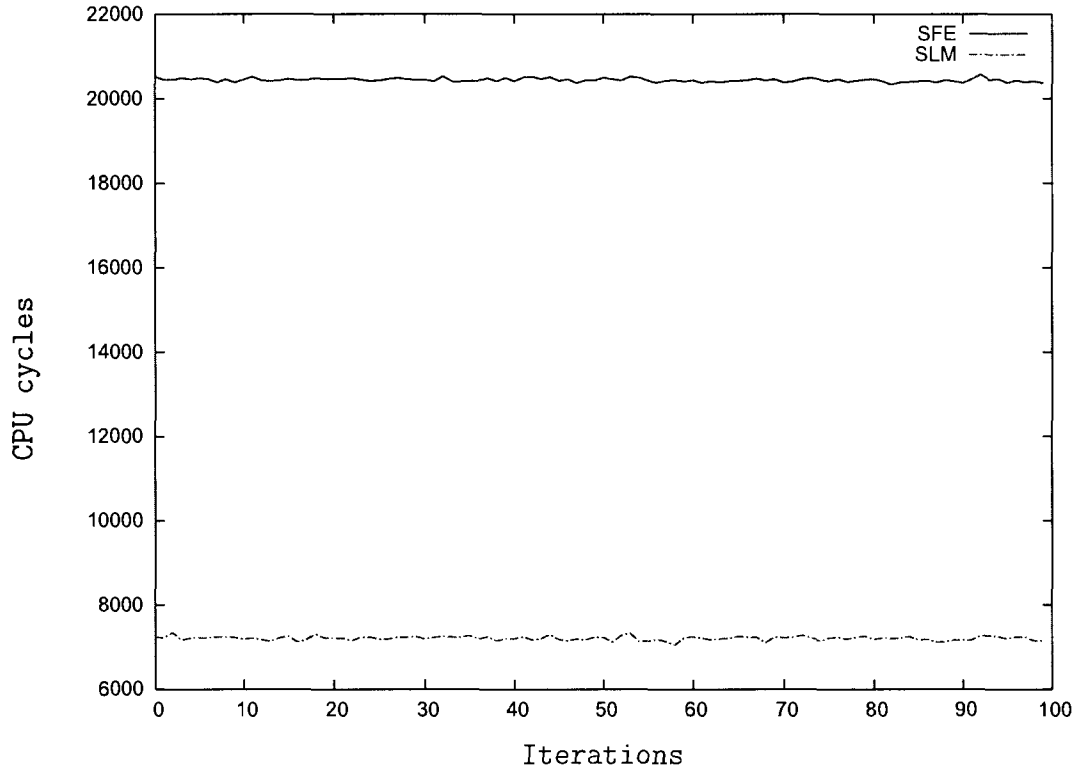


Figure 6.1: Total number of CPU cycles for SLM and Shannon-Fano-Elias. The dashed line represent SLM and the solid line represents SFE.

6.3 CPU Time Comparison between SLM and Shannon-Fano-Elias

In Sec. 5.2 SLM more efficiently creates Shannon-Fano-Elias code. The method uses the concepts of MaCE to reduce the number of computations required by the logarithm and `ceiling` functions utilized by the Shannon-Fano-Elias algorithm. The method also uses the concept of equilibrium and the split tree to prioritize the order of operations to the levels with the most possible comparisons. The combination of the computational reduction and priority given to the levels with the largest number of comparisons shows a significant improvement in processing time.

Figures 6.1–6.3 graph the results of SLM and Shannon-Fano-Elias in terms of CPU processing time in CPU cycles. Figures 6.8–6.13 separate the composite graphs by the individual methods and place the graphs on the same page for quick analysis. The cycle comparison focuses on the difference between the logarithm and `ceiling` functions used by Shannon-Fano-Elias with the comparison model used by SLM by isolating the time samples to these segments of the source code. The focus is possible as SLM only differs from Shannon-Fano-Elias for those operations and the rest of the code is identical between the two methods. The logarithm function for Shannon-Fano-Elias is a software version used to compare like technologies without caching and other tricks to increase the speed of

the logarithm function.

The data set consists of 256 symbols randomly assigned count values and the test included a total of 100 random data sets for each iteration. The statistics were taken over 100 iterations of the same data set to mitigate the effects of the interrupts and the results published are the lowest value of the statistics: minimum, maximum and total CPU cycles. Figures 6.1–6.3 display the comparisons between the two methods in terms of total CPU cycles, minimum number of CPU cycles required by a single operation and the maximum number of CPU cycles required by a single operation.

Figure 6.1 represents the total number CPU cycles taken for each of the 100 iterations. The total is the summation of the best result for each of the 100 iterations of the same random generated symbol to symbol count relationship. The summation includes system interrupts and it notes that the best total may not correspond to the same data set where the best minimum and maximum CPU cycles are acquired. A direct correlation between the average CPU cycles and the totals cannot be ascertained due to the inability to make the relationship between the data elements in each metric. The total CPU cycles for SLM is about one third the number of CPU cycles for Shannon-Fano-Elias which is a considerable improvement. The results for both are relatively flat at this resolution, so both of the results will be analyzed in more detail in their separate graphs.

Figures 6.8–6.9 displays separate graphs for total CPU cycles of the two methods. The separation of the totals into two graphs shows the volatility of the methods using the random data sets. Figure 6.8 shows the CPU cycles range from approximately $7050 - 7325$ for SLM, while Fig. 6.9 shows Shannon-Fano-Elias with a range from approximately $20325 - 20575$ CPU cycles. The range of the total relates to approximately a 300 cycle range between data sets for SLM, while Shannon-Fano-Elias is more constrained at approximately 250 for the same data sets. We also observe that many of the fluctuations in the number cycles occur simultaneously between both methods, although as stated the totals may not have a one-to-one correspondence due to the sampling method. The number of CPU cycles used by SLM is about a third the number used by Shannon-Fano-Elias even with the greater volatility for SLM.

Figure 6.2 represents the minimum number of CPU cycles required to encode a single symbol using SLM and Shannon-Fano-Elias. Although the minimum is an interesting statistic, it cannot represent the encoding process as a whole as both of the algorithms have a variability dependent on the symbol percentage. The speed of Shannon-Fano-Elias is primarily dependent on the number of digits and the numerical size of the mantissa for the logarithm and `ceiling` functions the algorithm utilizes. SLM is primarily dependent the number of comparisons required to reach a resolution of level by the IF/ELSE block.

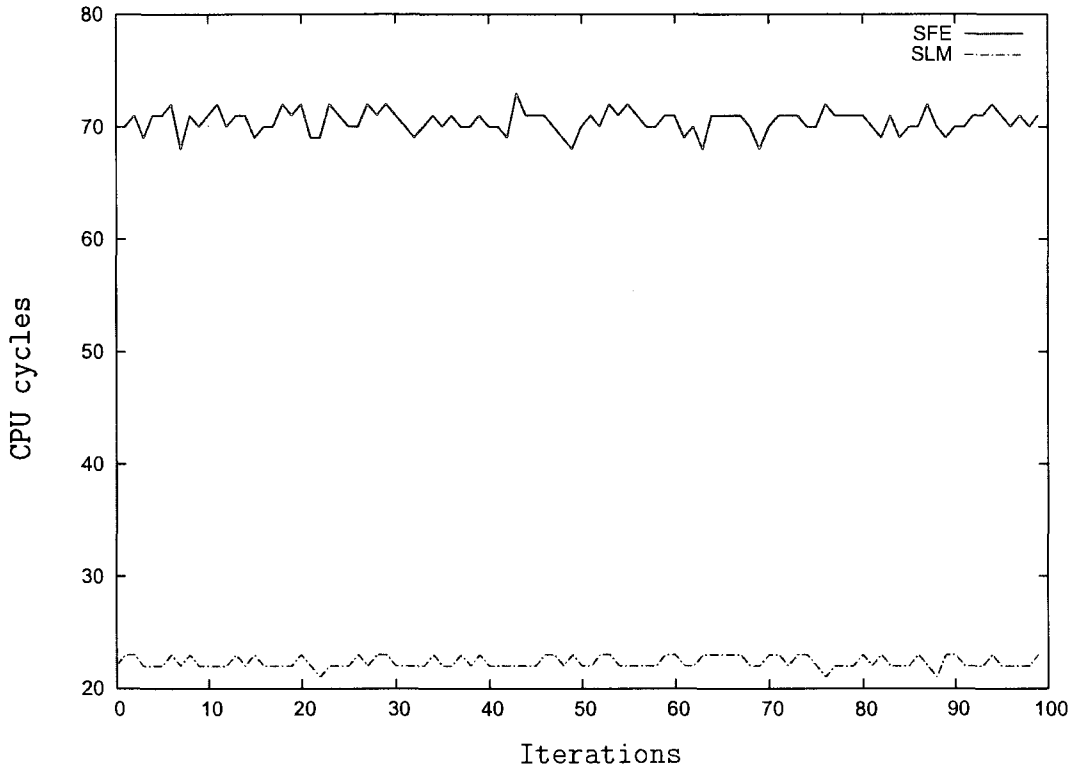


Figure 6.2: Minimum number of CPU cycles for SLM and Shannon-Fano-Elias.

Similar optimizations for both algorithms are possible to achieve results closer to the minimum and with these optimizations the minimum for SLM will require less comparisons on equivalent data sets then the logarithm function used by Shannon-Fano-Elias resulting in a better overall time. The results for both are flat at this resolution, so both the results will be analyzed in more detail in their separate graphs.

Figures 6.10–6.11 display separate graphs for minimum CPU cycles of the two methods. Figure 6.10 shows a volatility of 2 cycles with a range from $21 - 23$ CPU cycles for SLM. Figure 6.11 shows a volatility of 5 cycles with a range from $68 - 73$ CPU cycles for Shannon-Fano-Elias. These results concur with the total CPU usage as the minimum for SLM is approximately a third of the cycles taken Shannon-Fano-Elias. A minimum of $21 - 23$ CPU cycles is a metric to aspire to in future developments.

Figure 6.3 represents the minimum of the maximum CPU cycles required to encode a single symbol using SLM and Shannon-Fano-Elias algorithm. The minimum of the maximums was taken in order to reduce the effect of interrupts as the actual maximum of all iterations may contain one or more system interrupts within the time slice. The technique appears to work well as the results are relatively flat as with the minimum and totals. Their are some spikes within the maximums which maybe the result of smaller interrupts, but we expect more volatility in the maximums than the minimums due to the variability in the symbol percentage. The results for both methods in this case are not as flat as the previ-

ous graphs. However, for completeness the results will be analyzed in more detail in their separate graphs.

Figures 6.12–6.13 display separate graphs for minimum of the maximum CPU cycles of the two methods. Figure 6.12 shows a volatility of 60 cycles with a range from $35 - 95$ CPU cycles for SLM and Fig. 6.13 shows a volatility of 50 cycles with a range from $175 - 225$ CPU cycles for Shannon-Fano-Elias. The maximum results for Shannon-Fano-Elias correspond directly with the totals as the range is approximately one-hundredth of the total cycles used by Shannon-Fano-Elias. The results for SLM are significantly less then the total, which may indicate that this method is more susceptible to interrupts. More study will be required to determine the actual cause of the difference. The comparison shows SLM with the advantage compared to Shannon-Fano-Elias and the further study would only show if the efficiency can be further increased for SLM.

Figures 6.12–6.13 have another interesting feature to note. The graph for SLM in Fig. 6.12 is fairly flat, while the graph for Shannon-Fano-Elias in Fig. 6.13 has a oscillation. The oscillation of Shannon-Fano-Elias is in response to the length and numerical size of the mantissa being the dominate factor in the logarithm function. The mean of the oscillation represents the median variation of the mantissa. SLM is not dependent on the specifics of the value, only the value itself to make the IF/ELSE comparisons. The number of comparisons will be relatively flat in most cases.

All the comparison graphs show the advantage SLM has over Shannon-Fano-Elias and all three show that SLM takes approximately a third of the time to reach the same point dictated by the algorithm.

In addition to the random data test, the two extremes of the data distributions are also tested represented by the best case and worst case for SLM. The best case occurs when all values are equal by design explained in Sec. 5.2.6 and the worst case occurs when one value dominates the distribution and the remaining values are placed on the lowest level possible. The best case was tested by assigning all 256 symbols the same count value and the same methodology as the random test was applied to determine the best statistics. The best case for SLM requires 6819 total CPU cycles, a minimum of 22 CPU cycles and a maximum of 42 CPU cycles for one comparison. The best case for Shannon-Fano-Elias requires a total of 12940 CPU cycles with a minimum of 45 and a maximum of 170 CPU cycles. The best case for SLM requires half as much time as Shannon-Fano-Elias to complete the table creation for the same data set. The advantage over Shannon-Fano-Elias was diminished in this test case in comparison to the times of the random data, however, the number of CPU cycles is the best of all the testing.

The worst case for SLM was tested by assigning one symbol a count of 16384 and all

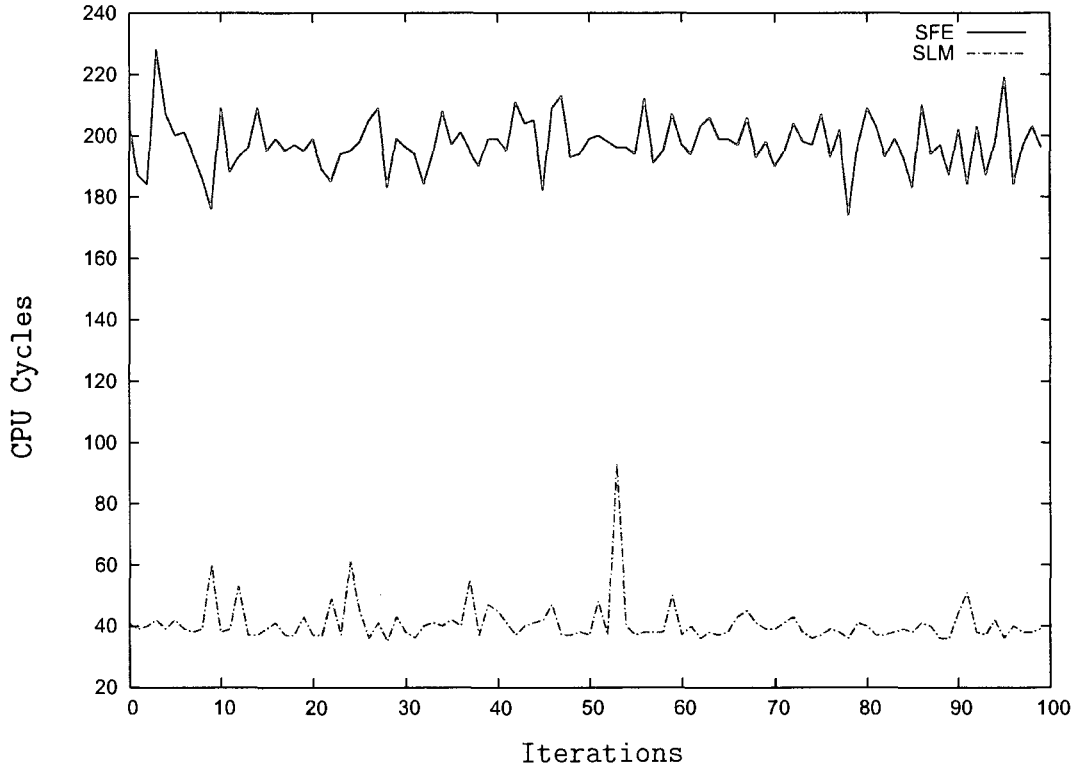


Figure 6.3: Maximum number of CPU cycles for SLM and Shannon-Fano-Elias.

other symbols the count of 2, which results in the first symbol being located in the second level and the rest located at the bottom of the space in the 31st level. The CPU cycles required in the worst case had a total of 12718 CPU cycles with a minimum of 39 and a maximum of 71 CPU cycles. In contrast, Shannon-Fano-Elias requires a total of 20869 CPU cycles with a minimum of 75 and a maximum of 196 CPU cycles. The worst case for SLM required approximately 60% of time requires by Shannon-Fano-Elias to complete the table creation for the same data set.

In summary, the direct comparisons of the range for the total number of CPU cycles for Shannon-Fano-Elias was between 20350 and 20575 to complete each of the 100 random generate sample data sets, while SLM's range was between 7050 and 7550 CPU cycles. On average SLM requires one third of the CPU cycles required by Shannon-Fano-Elias. The minimum number of CPU cycles required for one length procedure containing the logarithm and `ceiling` functions for Shannon-Fano-Elias was between 68 to 73 CPU cycles, while SLM required 21 to 23 CPU cycles to complete a length procedure using the IF/ELSE comparisons. The ratio of the two minimums corresponds directly to the ratio of the method totals for both algorithms. Shannon-Fano-Elias requires a maximum of 175 to 230 CPU cycles and SLM required 40 to 95 CPU cycles. Although the results for Shannon-Fano-Elias correspond directly to the total and the minimums, the ratio did not hold for SLM. The range of the maximum would seem to indicate that the method is more susceptible

to interrupts as that is currently the only known cause of the disparity. Further study is required to determine the direct cause, but the result would only increase the efficiency of SLM over Shannon-Fano-Elias.

6.4 Entropy Comparison between SLMN, SFE, and Theoretical Ideal

In Sec. 5.3 SLMN was introduced to create a more efficient code than Shannon-Fano-Elias code in terms of space. The method uses the concepts of MaCE to reduce the number of computations required by the sort function. A traditional comparison sort requires $O(n \log n)$ operations while the bucket sort using MaCE will require $O(n)$ operations. The sort is integrated into the algorithm based on the concept of midpoints which is also used to determine the final position in the space. The difference from the actual symbol percentage and the certainty percentage is used to make future decisions on placement of subsequent symbols. The method uses the concept of equilibrium and the split tree to prioritize the comparison to the levels with the most possible comparisons. The combination of the reduction and the priority given to the larger levels show a significant improvement in processing time in SLM. The replacement of the Shannon-Fano-Elias fitting function with the midpoint comparison model reduces the code to $H(x) + 1$.

Figure 6.4 displays the results for Shannon-Fano-Elias, SLMN and the theoretical ideal in terms of entropy. The graph shows that the difference between the theoretical ideal and the results for SLMN stay with the $+1$ requirement. The data was based on a static 256 symbols count with varying distributions in symbols count between each iteration. The analysis of the peaks and valleys of the three results shows the modeling is pretty accurate over the span, but there are conditions when the one model may diverge slightly representing subtle differences in the results. The minimum entropy difference for Shannon-Fano-Elias from the theoretical ideal is 1.64 bit spaces per symbol while the minimum entropy difference for SLMN from the theoretical ideal is 0.11 bit spaces per symbol. The maximum entropy difference for Shannon-Fano-Elias from the theoretical ideal is 1.53 bit spaces per symbol while the minimum entropy difference for SLMN from the theoretical ideal is 0.26 bit spaces per symbol. A markable improvement in terms of entropy were both the minimum and the maximum difference between SLMN and Shannon-Fano-Elias is over 1.25 bit spaces per symbol.

Figure 6.4 also shows another interesting result as both the theoretical ideal and SLMN represent compression of the data set while the entropy results for Shannon-Fano-Elias actually shows the opposite. The number of symbols is equal to 256 and if all values were encoded in ASCII text the entropy would be equal to 8 bit spaces per symbol. The results

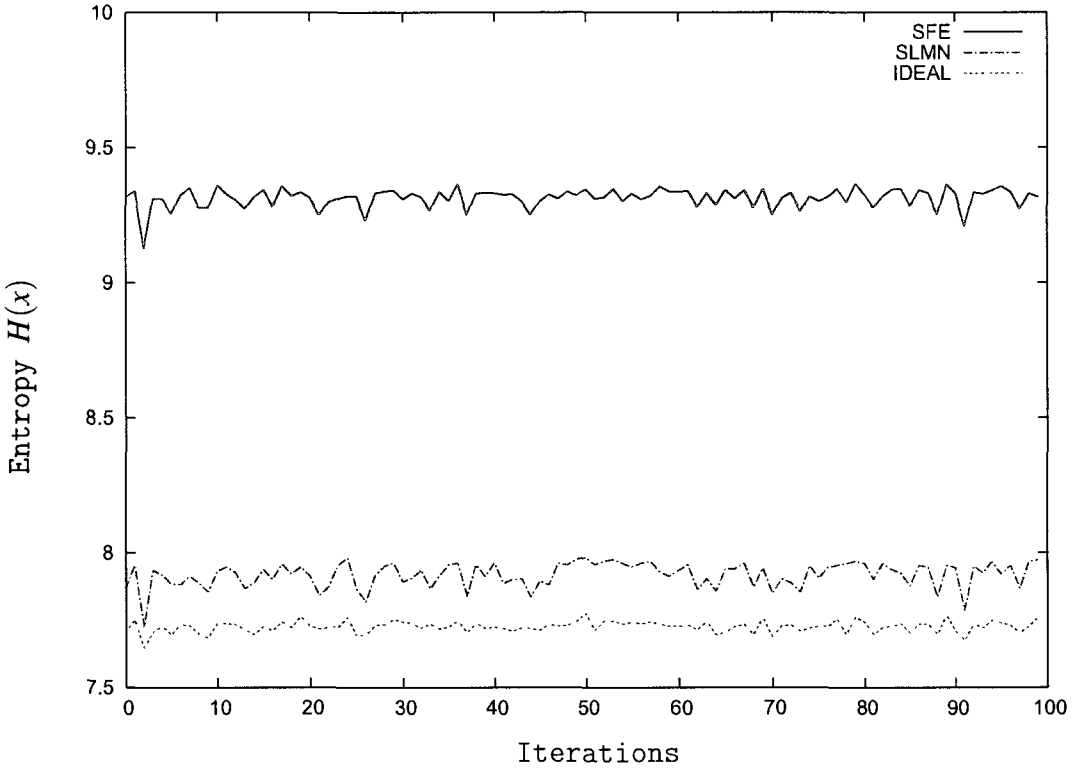


Figure 6.4: Entropy Comparison between SLMN, Shannon-Fano-Elias and the theoretical ideal.

in Fig. 6.4 show that Shannon-Fano-Elias produces between 9.12 and 9.36 bit spaces per symbol which means that throughout the span of 100 random data sets Shannon-Fano-Elias added ≈ 1.12 bits per symbol and produces a larger file then the original data file that the algorithm was suppose to compress. This scenario will occur because of the fitting function that always increases the space in order to avoid the conflicts as explained in Sec. 2.7.3. The problem with the fitting function is also displayed when the file to be compression has all equally weighted values. For example, with 256 symbols of all equal counts Shannon-Fano-Elias will produce an encoding with 9 bit spaces per symbol of entropy while both SLMN and the theoretical ideal will produce and encoding of 8 bit spaces per symbol and the original file contained 8 bit ASCII. This shows that Shannon-Fano-Elias has the potential to actually increase the file size it is trying to compress.

The condition where all the symbol counts are equal exhibits one of the extreme conditions in the possible data distribution. The other extreme is when all the values are approximately one half of the previous symbols count. In order to test this condition, on a 32 bit system, the symbol count was limited to 30 ranging in value from 1073741824 to 2. The results show that SLMN and the theoretical ideal both require 2 bit spaces per symbol, while Shannon-Fano-Elias requires 3 bit spaces per symbol. The end result in both the extreme conditions is Shannon-Fano-Elias requiring extra bits per symbol to represent the

same data where SLMN would produce optimal compression in both extremes.

The final test for SLMN is designed to show how the algorithm handles increasing entropy from the base case where one symbol count dominates the distributions and the other values increase over time. In order to accomplish this task, the test used 256 symbols with a static total count of 65536 and the distribution starts with one symbol having a count of 65281 and the remaining 255 symbols having a count of 1. The increase in entropy from this condition is accomplished by recursively dividing the sum of the counts of the adjacent indexes by 2 until the symbol count to be split is equal to 1, at which point the process is started again from the starting index. The remainder of 1 for the odd value counts is added to the leading index value in the sum to maintained a sorted order. For example, with the initial distribution of $(65281, 1, 1, 1, \ldots)$, the first division would result in $(32641, 32641, 1, 1, \ldots)$ and the second $(32641, 16321, 16321, 1, \ldots)$. For the first iteration the process restarts when the following is reached $(\ldots, 5, 3, 2, 1, 1, \ldots)$.

The process was applied with 50000 iterations and the results are shown in for entropy in Fig. 6.6 and in file size Fig. 6.7. Figure 6.6 shows the increase in entropy as the symbol count is distributed more evenly with each iteration. The minimum entropy difference from the theoretical ideal for SLMN from the ideal is 0.00237 bit spaces per symbol and for Shannon-Fano-Elias the minimum is 1.25 bit spaces per symbol. The maximum entropy difference from the theoretical ideal for SLMN is 0.963 bit spaces per symbol and for Shannon-Fano-Elias the maximum is 1.990 bit spaces per symbol. The condition for the maximum difference in entropy from the theoretical ideal occurred for both methods on the very first iteration when one symbols count dominated the distribution and the minimums occurred at various points in the distribution. A cycle is noted in the graph of SLMN of increasing wavelength as the number of iterations increase.

Figure 6.7 shows the increase in file size as entropy increases for both Shannon-Fano-Elias and SLMN along with the static size of the file. The file size is $65536 \times 8 = 524288$, which is displayed as the original file size in the figure. The file sizes for both Shannon-Fano-Elias and SLMN are both calculated using the percentage representing the symbol P_i, the total number of symbols representing the file n, the number of bits b in the encoding table. Although this may not be 100% accurate due to rounding error the end result will be comparable to a live data set. Using this data we can see the relationship between entropy and the file size by comparing Fig. 6.6 and Fig. 6.7. They follow the same general plot which concures with the explanation of entropy as it represents the uncertainty of the symbols P_i mapped onto the certainty represented by a P_c times b_i and the sum represents the final file size. The failure of the mappings and the system to model entropy can also be observered in Fig. 6.6 as the difference between the theoretical ideal and the methods

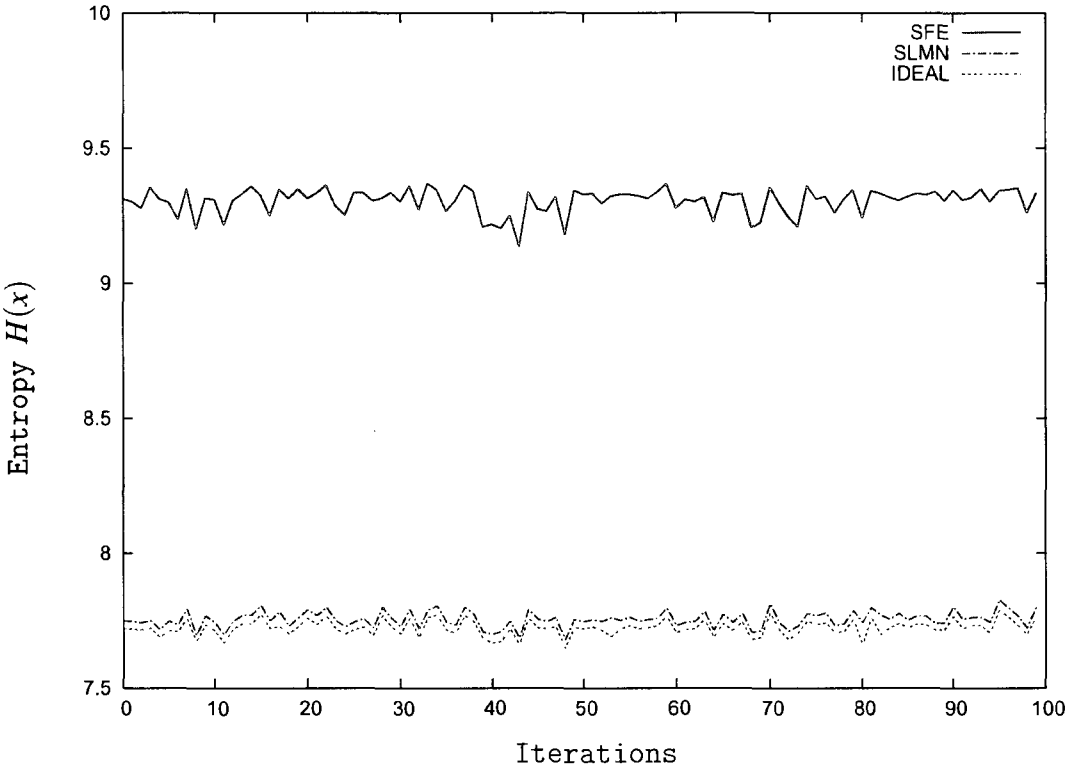


Figure 6.5: Entropy Comparison between SLMN, SFE and the theoretical ideal for SLMN using only the direct difference used as an offset.

results.

The compression ratio for the two methods was calculated using the original file size $F_o = 524288$ and the calculated file size based on the encoding. The test data starts with a single symbol with the count of 65281 and the other 255 symbol counts equal to 1 and total count symbol of 65536. The size F_o is divided by each of the compressed files sizes for SLMN $F_{SLMN} = 67576$ and Shannon-Fano-Elias $F_{SFE} = 134897$ to determine the ratios r. The ratios acquired by the calculations are: $r_{SLMN} = F_o/F_{SLMN} \approx 7.7585$ and $r_{SFE} = F_o/F_{SFE} \approx 3.8866$.

The file size concurs with the theory of entropy where the starting condition represented approximate 0.0679 bit per symbol space for the theoretical ideal and the actual obtained by SLMN is 1.03113 and the file size is approximately a one-to-one relationship between symbols and entropy. If we multiply the original number of symbols $n \times L(x) = 65536 \times 1.03113 = 67576$ negating floating point errors. In short, entropy is defining how many bits on average it takes to represent a symbol because of the duality of the meaning of the length between the root and the symbols representing certainty and uncertainty. Theoretically, we should be able to reach the $n \times H(x) = 65536 \times 0.0679 = 4450$ negating floating point errors. The difference shows the failure of base 2 to represent the symbols exact percentage as $P_c \neq P_i$ in the mapping.

Although this graph is far from exhaustive, there is nothing to indicate a failure for SLMN to adhere to the $H(x)+1$ specification. The sequential test creates a continual cycle with an elongating wavelength which suggests that the method would not deviate sharply and would come close to level with the ideal at some point. We can be fairly confident that this point is equilibrium and as the test of this condition resulted in the entropy for SLMN and the theoretical ideal to be exactly equal. The bounds of the distribution of equilibrium and the slope equate to the ideal and the 100 random values close to the equilibrium also concur with SLMN remaining in $H(x)+1$. SLMN does a good job at maintaining the relationship between the theoretical and the actual results which means there maybe better way to model the difference. It is noted that the inability of the current implementation to model the percentage past beyond 32 bits does have an effect on the results. The random example the entropy was reduced to $0.02-0.08$ from the $0.11-0.26$ in the current model by only passing the average difference to the following symbols in the list. The results are shown in Fig. 6.5 and the difference is evident when comparing the figure to Fig. 6.4. The 32 bit limit was exceeded in complete tests used to test the unmodified version of SLMN. The limit caused errors to occur during testing which keeps the complete test results from being published and further testing and development is required to determine the best course of action.

Another possibility of improvement to the method is to change the method to only average the negative difference after equilibrium is reached. This change may result in lower entropy values as the larger percentage. Those above equilibrium would receive a better location in terms of entropy and the lower percentages values would simply receive a lower level as a result. The precedence given to the larger values in terms of space used in the system goes with the assumption that the larger values would have the greatest effect on the difference between the actual certainty of the system and the symbol percentage. Further testing is required to determine the best course of actions in this regard. Other improvements to increase compression will also be possible with time and there is room for improvement in the terms of speed of the algorithm as the method was implemented with ease of understanding and the desire to produce an encoding with $L(x) \leq H(x)+1$, which was demonstrated in the results.

6.5 Summary of Results for SLM and SLMN

The results from the experimental results of SLM and SLMN show a significant improvement over Shannon-Fano-Elias in terms of CPU cycles and file size. The minimum, maximum and total CPU cycles were graphed and examined in detail for SLM and the experi-

ments included a best and worst case as well as a random and sequential data set analysis. The analysis shows a significant improvement in all three categories by SLM in comparison to Shannon-Fano-Elias with a software implementation of the logarithm function. The results in all categories shows SLM uses less then $1/2$ the number of CPU cycles required by SLM and in most cases examined SLM used $1/3$ the number of CPU cycles.

SLMN is examined in comparison to Shannon-Fano-Elias and the theoretical ideal in terms of entropy, file size and compression ratio. The minimum, maximum and total entropy values were graphed and examined in detail. It is noted that Shannon-Fano-Elias has the possibility to actually create a larger file size if the compression is within one bit of the original data representation due to the fitting functions. SLMN does not have this problem as the limits for both the upper and lower bounds were tested using the best and worst case data sets. It was shown through the experiments that SLMN stays within the limit of $H(x)+1$ and through random and sequential analysis there is no indication that the method will deviate beyond this limit. We also observe the $+1$ requirement in the initial condition of the sequential test and compare this to the theoretical ideal throughout the testing. In addition to the results, we used graphs to display the improvement related to the failure of the 32 bit system to model the data in the sequential data set. The systems limiting factor was the inability to have a range of values that exceeded 2^-31. The inability required the code tested to have additional specification to avoid this situation which reduces the compression efficiency of the model. However, the random values do not exhibit this problem as they stayed within the system requirement. The results show a substantial improvement in data compression and the solution to the 32 bit limitation is left for future work.

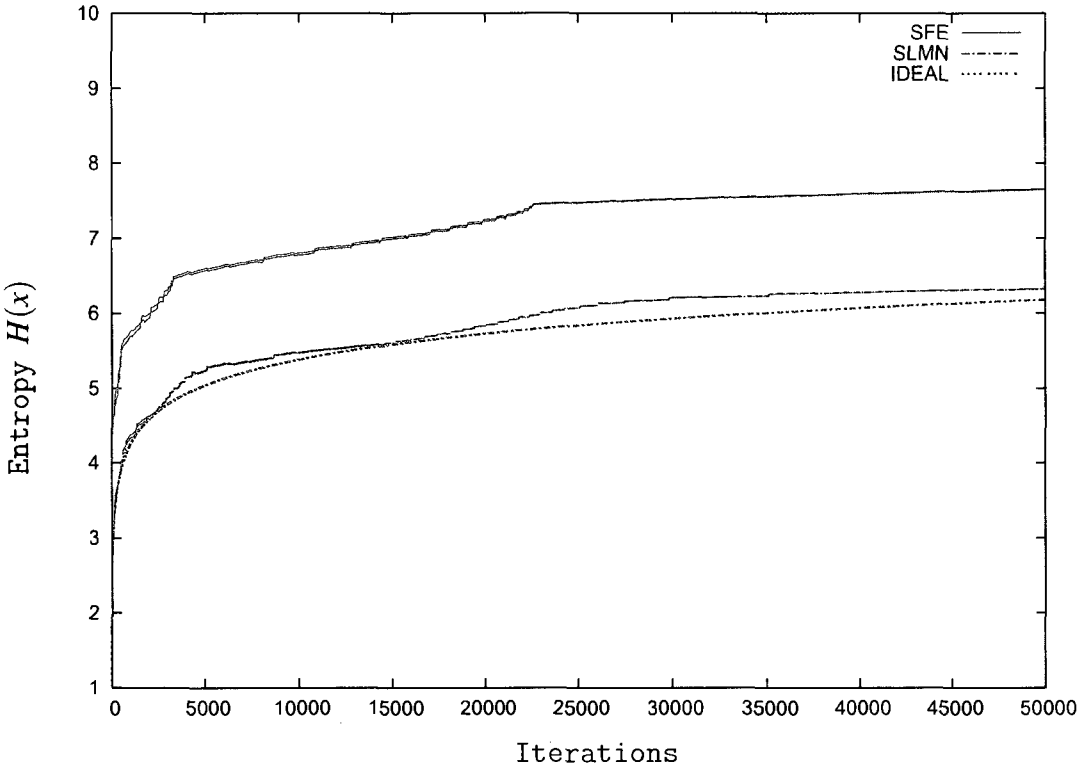


Figure 6.6: Entropy comparison between SLMN and Shannon-Fano-Elias.

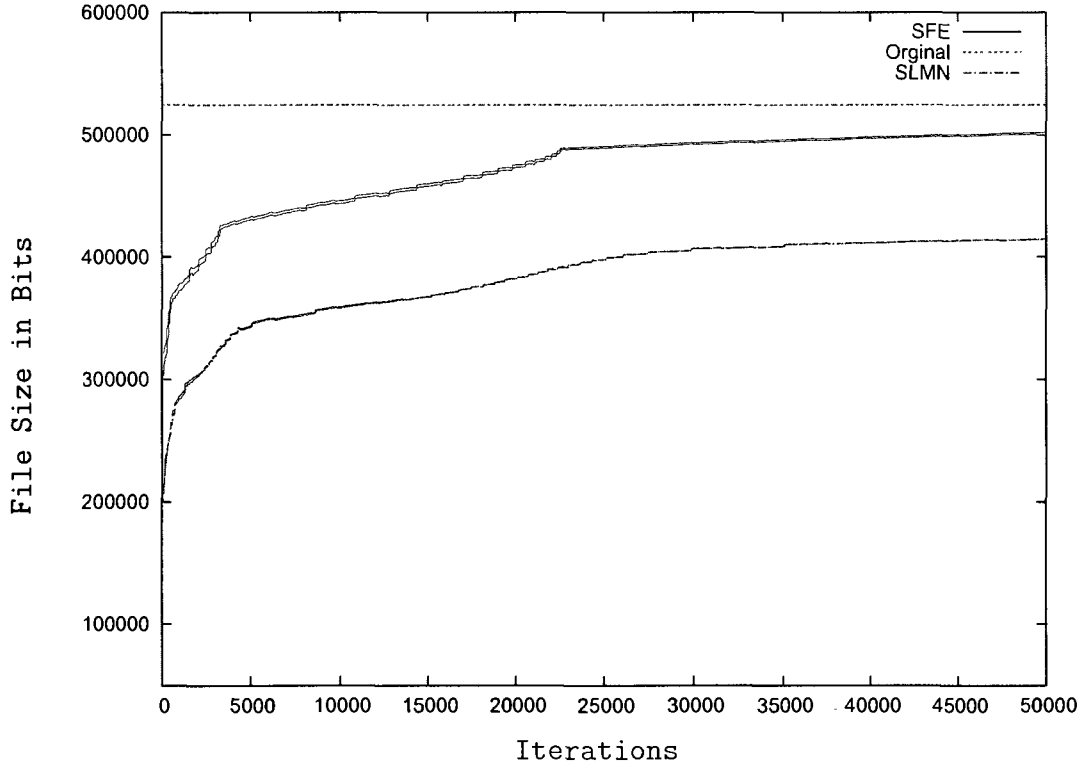


Figure 6.7: File size comparison between SLMN and Shannon-Fano-Elias.

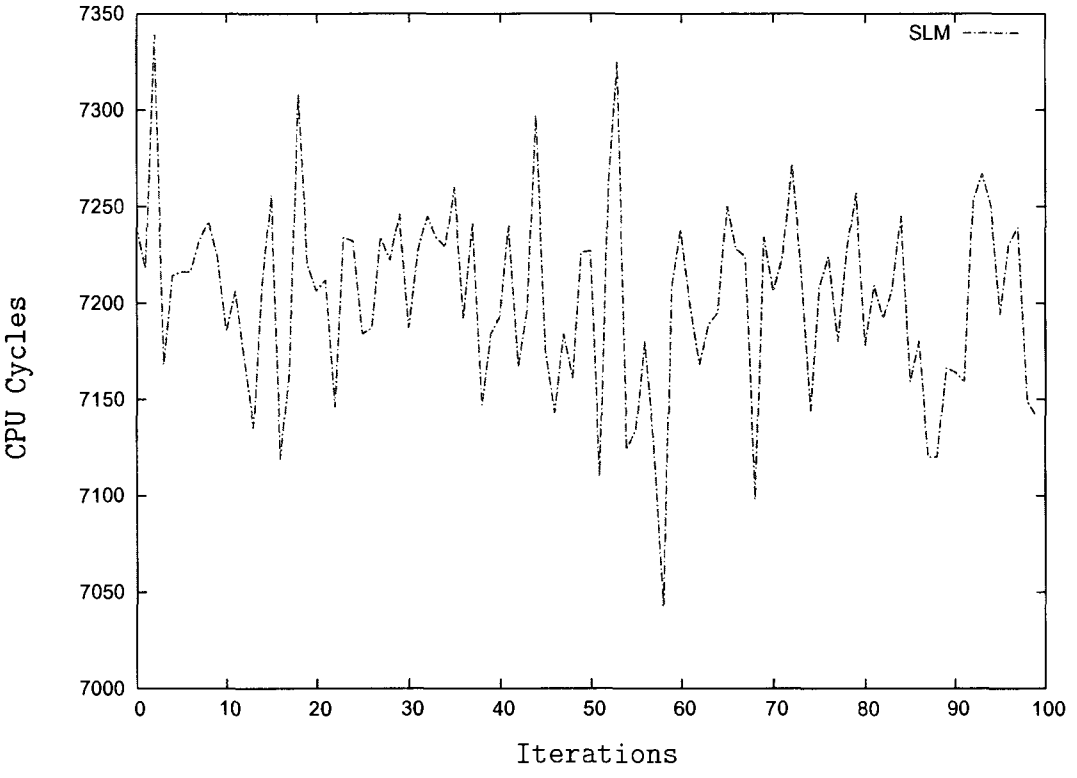


Figure 6.8: Total Number of CPU cycles for SLM.

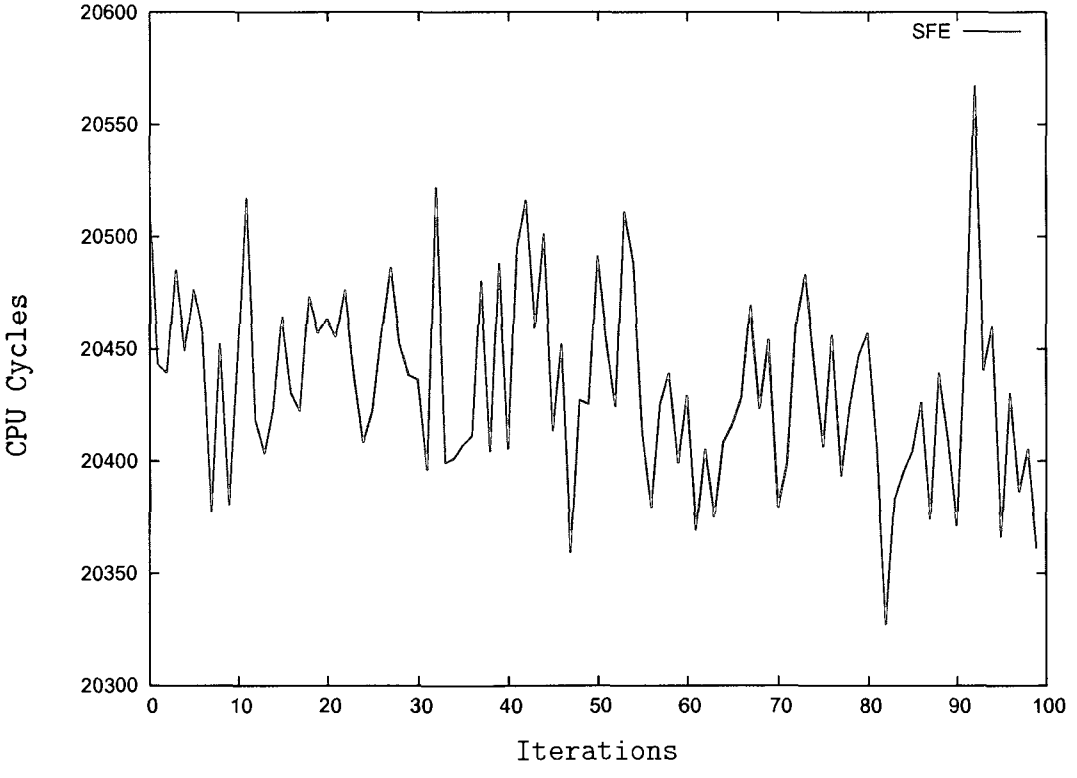


Figure 6.9: Total CPU cycles for Shannon-Fano-Elias.

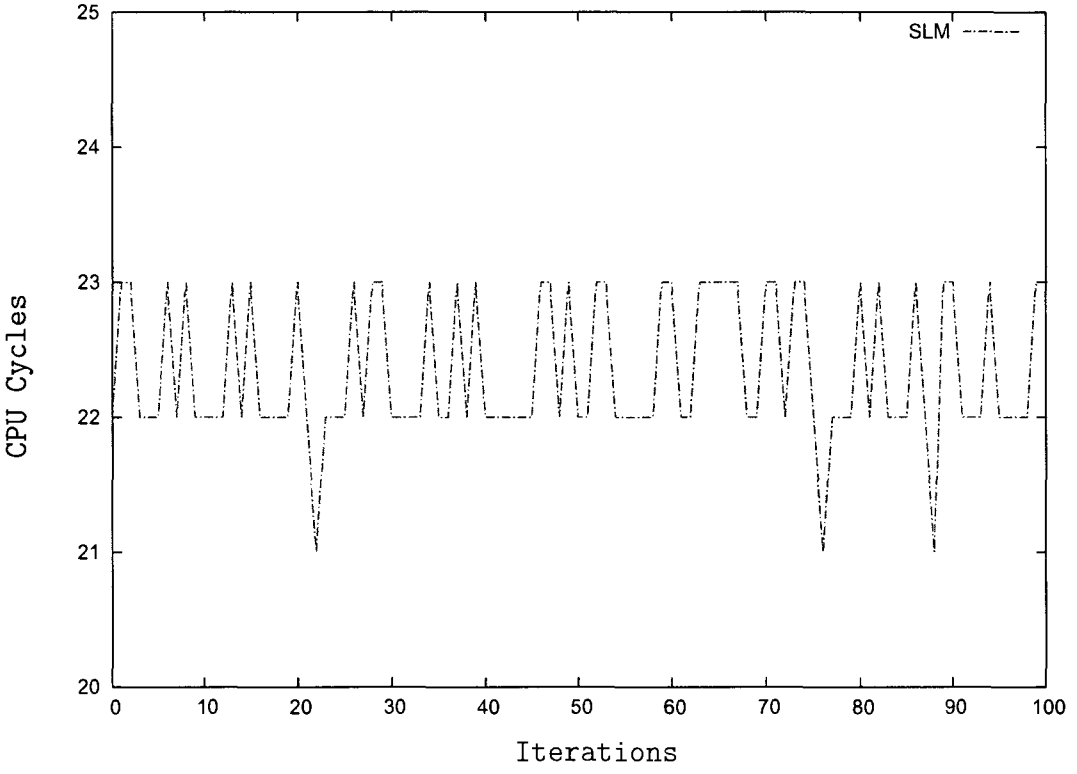


Figure 6.10: Minimum number of CPU cycles for SLM.

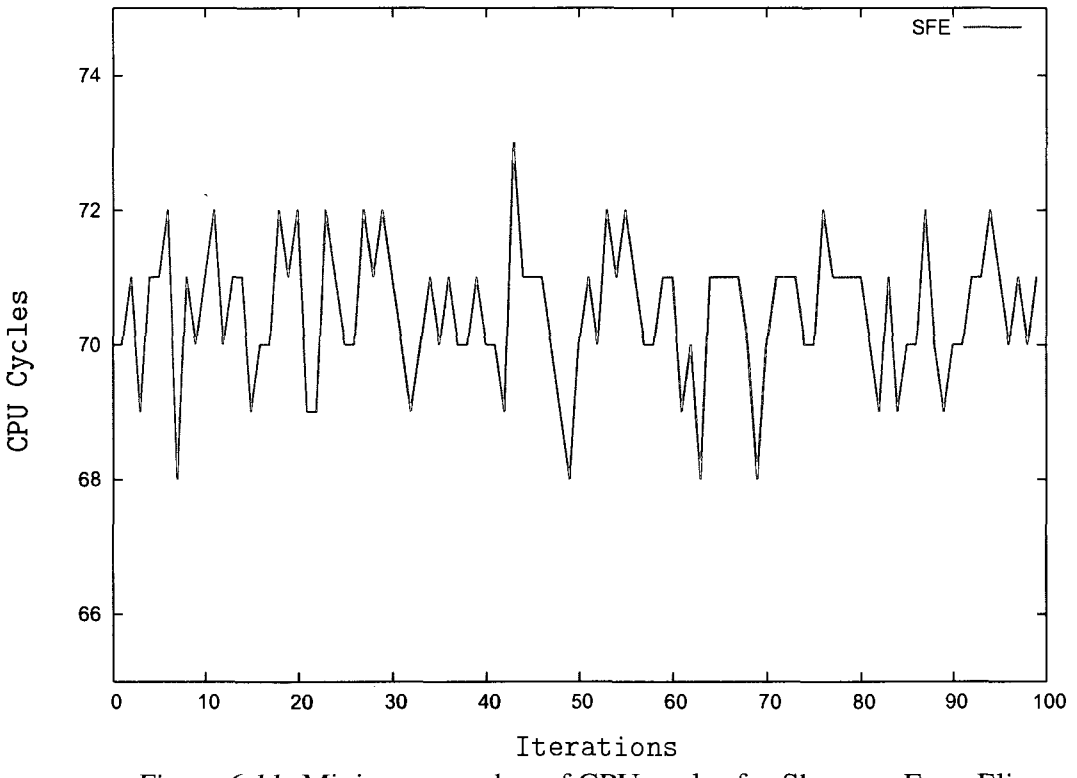


Figure 6.11: Minimum number of CPU cycles for Shannon-Fano-Elias.

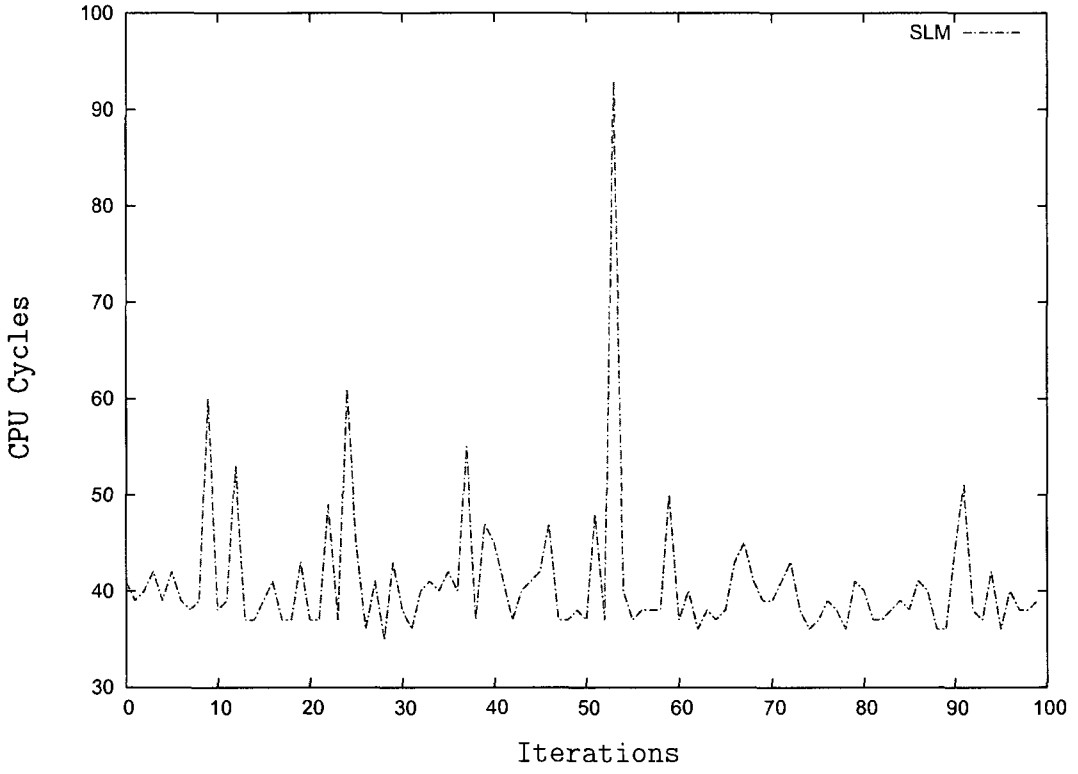


Figure 6.12: Maximum number of CPU cycles for SLM.

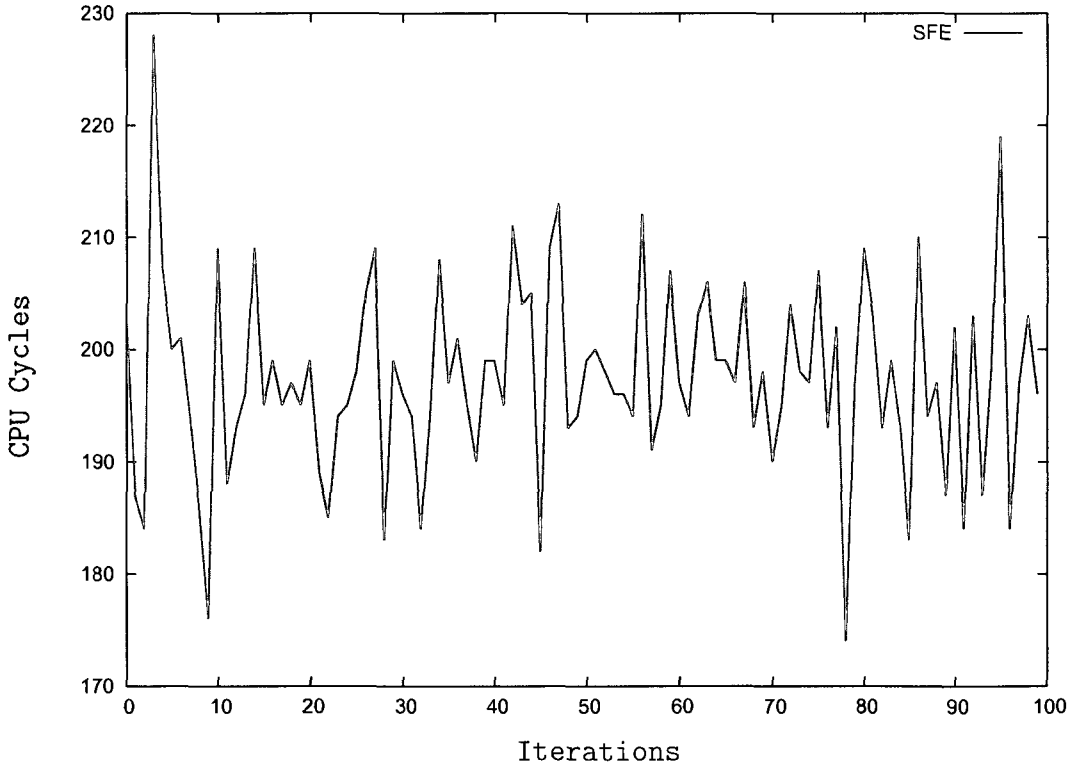


Figure 6.13: Maximum number of CPU cycles for Shannon-Fano-Elias.

Chapter 7

CONCLUSION

7.1 Conclusion

This dissertation represents a fundamental shift in perspective from the previous views used to construct data compression algorithms for prefix-free, lossless compression algorithms. The foundational algorithms of Shannon-Fano, Huffman and Shannon-Fano-Elias were examined in detail in Secs. 2.5–2.7 in order to make their views clear. The examination of these algorithms included the relationship between the algorithms in terms of information theory and the entropy equation, $H(x) = -P_i \log(P_i)$, defined by Claude Shannon, as summarized in Secs. 2.2–2.4. The relationship between entropy in general defined by S and the transition of S to H is examined to understand the relationships between the equations. The rudiments of the entropy equation were defined in order to further understand the relationship between the equations and the space they represents. The concept of the theoretical ideal is the part of the entropy equation that receives special attention in information theory. The theoretical ideal, $-\log(P_i)$, represents the information that is contained within a message. This was proposed by Shannon as an extension of the Hartley function. The duality of this portion of the equation is also explained with the concept of "surprisal" as the same length function, $-\log(P_i)$, that defines the location of the information or certainty also defines entropy (uncertainty).

In Chapter 3 the models of the data compression environment were examined to show the relationship between the model and the algorithms. The algorithms examined were Shannon-Fano, Huffman and Shannon-Fano-Elias encoding. The data compression environment consists of both the certainty relationship between the code words and the existence of the specific symbol to adhere to the lossless compression property. The data compression environment also consists of uncertainty inherent to the complete data set and the binary environment not relating to a specific symbol. The views of environment for each of the algorithms varies, which relates directly to the produced solution for each of the algorithms. The relationship between the model of the environment and the entropy equation was examined in detail in order to understand how the final algorithm accomplishes data compression. The analysis shows the process the algorithms use to map the entropy of the data onto the certainty of the system and the efficiency related to the mapping. We show that Shannon-Fano encoding uses certainty of the binary system to divide

the entropy of the symbols, Huffman encoding combines the maximum entropy symbol(s) and builds the certainty in the system in a bottom-up fashion and Shannon-Fano-Elias divides the space containing both entropy and certainty by the symbol percentage to extract length represented by the theoretical ideal. All the algorithms use a unique model of the environment and the model influences how the algorithm is designed as well as the efficiency of the algorithm in both speed and compression efficiency.

In Chapter 4 we continue the discussion about the models of the data compression environment with more focus on the ability to represent the complete environment related to data compression and through this analysis a list of requirements to define the space. This list is utilized to define MaCE which consists of a two-dimensional space that is able to map both the certainty of a system and the entropy of the data in a common visual. This allows for a easy examination of the contents and interactions within the space. Via this model the orientation and the two types of entropy are defined along with the certainty of the system representing the symbols. The analysis of the two types of entropy shows that the discontinuous certainty and the continuous uncertainty can be mapped using arithmetic comparisons between the percentages P_c, representing the certainty of the space in the system, and P_i, representing the uncertainty of the data.

Chapter 5 uses MaCE and entropy concepts developed in Chapter 4 to create three methods which improve the efficiency of Shannon-Fano-Elias in both CPU cycles and data compression. The first method, SLM, is used to increase the computational efficiency by dividing the entropy space and the equation by $-\log(P_c)$, representing certainty, to enable the ability to use the arithmetic comparison model. Once the reduction is accomplished an IF/ELSE model is implemented to determine the appropriate encoding of the symbol. The efficiency of the algorithm is further enhanced using the concepts of equilibrium and the split tree which gives the insight to prioritize the IF/ELSE block to the levels with the greatest number of comparisons and enhance the efficiency of the methods. SLM addresses the speed of data compression using the spatial concepts and maintains the same compression as Shannon-Fano-Elias.

SLMN is the second method introduced in Chapter 5 and is designed to show the ability of the spatial concepts to increase the compression of the data to less than $H(x)+1$. The midpoints are introduced in order to make this possible as they define the reference points to use the comparison model. The defined midpoints are utilized to create a bucket sort which decreases the sorting complexity from $O(n\log(n))$ to and order $O(n)$. The significance of the sort in data compression is to allow the ability to map more precisely the symbols to the correct location without the added space required by an algorithm without a sort, such as Shannon-Fano-Elias. The addition of the sort and the midpoints allow for a compression

less than $H(x)+1$.

In Sec. 5.4 the meaning of the $+1$ in $H(x)+1$ is discussed as it pertains to the inability of the system defining the certainty percentage P_c to map the uncertainty represented by the symbol percentage P_i uniquely. The variance between P_c and P_i for all symbols i results in the inefficiency of the compression algorithm. The ability to understand the meaning of the $+1$ allows for SLM and SLMN to be extended to any base with greater efficiency in both computation complexity and compression ratio than the current algorithms. The extension is the third method, JAKE, which adds the ability to change the base. JAKE is introduced with the general pseudo-code and requirements of the IF/ELSE block modification. In Sec. 5.5.2 we use Arithmetic encoding to demonstrate the power of changing the base and using decimal in data compression. In the future we hope to use the techniques and visualizations developed in this work to address some of this complexity. The symmetry of the encoding table used to encode and decode data is also discussed, since the purpose of the encoding table is to make a mapping between the code word and the symbol. A symmetric scenario is explained to accomplish both actions in $\log(n)$ for each symbol or code word.

Chapter 6 examines the experimental results of SLM and SLMN concepts in comparison to Shannon-Fano-Elias and the theoretical ideal. The comparisons between SLM and Shannon-Fano-Elias are in terms of CPU cycles to gage the relative speed of the methods. The minimum, maximum and total CPU cycles were graphed and examined in detail. The experiments includes a best and worst case scenario as well as a random and sequential data set analysis. The analysis showed a significant improvement in all three categories by SLM in comparison to Shannon-Fano-Elias with a software implementation of the logarithm function. The results show SLM uses less then $1/2$ the number of CPU cycles required by Shannon-Fano-Elias and, in most cases examined, SLM used $1/3$ the number of CPU cycles.

SLMN is examined in comparison to Shannon-Fano-Elias and the theoretical ideal in terms of entropy, file size, and compression ratio. The minimum, maximum and total entropy values were graphed and examined in detail. It is noted that Shannon-Fano-Elias has the possibility to actually create a larger file size if the compression is within one bit of the original data representation due to the fitting functions. SLMN does not have this problem as the limits for both the upper and lower bound were tested using the best and worst case data sets. It was shown through the experiments that SLMN stays within the limit of $H(x)+1$ and through random and sequential analysis there is no indication that the method will deviate beyond this limit. We also observed the $+1$ requirement in the initial condition of the sequential test and compared this to the theoretical ideal through the testing. In addition to the results, we display a graph of an improvement related to the

failure of the 32 bit system to model the data in the sequential data set. SLMN code tested required an additional condition statement to avoid this situation. This addition reduced the compression efficiency of the model. The testing of the random values does not exhibit this problem as they stay within the 32 bit system requirement. The results show a substantial improvement in data compression and is left for future work.

The shift in perspective provided by the analysis of the lossless compression outlined in this dissertation opens many new avenues that have not been explored. MaCE SLM, SLMN and JAKE represent the start of this endeavor. Through these methods it is possible to employ parallelism to increase the algorithms throughput; dynamic compression techniques to accomplish real time compression; and the use of the modeling techniques to increase the speed of other algorithms.

For future work, the further development of JAKE is of highest importance. The general algorithm has the potential to increase the efficiency in both time and space over other compression algorithms currently used in the field. The testing of the binary compression of SLMN and SLM confirms this potential. Most of the improvement will be related to specifics to which the method is applied, such as data distribution and preprocessing of the data via other fixed length compression methods, such as the algorithm Lempel-Ziv-Welch (LZW). All future work also applies to JAKE as it is an extension of SLM and SLMN. The benefit of JAKE is the added compression inherent to the change in base reducing the entropy and the reduction in the range of comparisons thereby making JAKE even more efficient in higher bases.

Dynamic compression can be accomplished through a simple bubble up algorithm using SLM and midpoints to determine the appropriate compression. Each of the levels L refers to the pipe containing a set number of code words possible determined by base b^L. The use of a code word in one level cancels the certainty representable by the subsequent code using P_c as the offset. The updates of the compression scheme are not tied to the data distribution directly so and incremental update is possible. SLMN can also be employed to increase the compression efficiency.

SLM allows for easy parallelism as the pipes representing each level can be divided amongst processors to load share and increase the throughput. The parallelism can be applied to either the dynamic or the static cases. The ability to use incremental update allows for a variety of parallelization and update schemes to be utilized for the process.

Hardware implementation is another area of importance. As the ability to use a comparison model reduces the number of hardware resources required to accomplish data compression. The ability to scale the compression to the hardware is a distinct advantage as the resources on the hardware components are limited and often desired for more robust appli-

cation. Any improvement on common tasks, such as data compression, is felt throughout the industry as this reduction frees the resource for the prime purpose of the device.

In summary, the development of SLM, SLMN and JAKE in this dissertation represents a new avenue to address the uncertainty and certainty of the information systems as it relates to data compression. SLM and SLMN have shown that the new approach is possible and represents improvement over the current algorithm Shannon-Fano-Elias. JAKE represents the ability to extend SLM and SLMN to larger bases in order to increase the compression with little added complexity. The ability of JAKE to extend to other bases without a substantial increase in complexity allows for even a greater array of solutions to be accomplished in addition to the future work outlined.

Appendix A

ARITHMETIC ENCODING EXAMPLE

Arithmetic encoding is a derivative of Shannon-Fano-Elias algorithm which uses decimal code words, modeling and single character encoding to achieve greater compression then the algorithm in Chapter 2. The arithmetic method uses recursive segmentation of the range space to create a single decimal number to represent the encoded message. The algorithm does not require sorted data as it use modeling techniques to determine the appropriate encoding of a character. Arithmetic encoding's use of modeling and the single character encoding to a decimal number allow it to acquire better compression then those proposed in Chapter 2. Arithmetic encoding uses the $-\log()$ function transpose the decimal number to determine the code length in binary.

This example is based on the demonstration shown in [9]. The initial condition is a series of symbols represented by their probabilities P_i. The P_i values are used to create sub-ranges in an interval from $0-1.0$. For example, with three symbols A, B, and C represented by their known symbol counts (A:2, B:1, C:1). The ranges for each symbol are (A:[$0-0.50$), B:[$0.50-0.75$), C:[$0.75-1.00$)). We refer to the individual highs and lows for each character as hi_c and lo_c. For example, the highs are (hi_A, hi_B, hi_C) and the lows (lo_A, lo_B, lo_C) for the given character set.

The process of Arithmetic encoding begins with a full entropy environment and calculates the encoding range for one character c on each iteration. The encoding ranges sub-section the environment to represent each character via several calculations. The main calculation is the range r. The range r for each character c is calculated as the difference between the hi_{pc} and lo_{pc}, where hi_{pc} and lo_{pc} are the high and low values from the previous iteration. The pc subscript represents previous character's encoding range. The difference between hi and lo represents the remaining range r to represent c. The starting condition of $lo = 0$ and $hi = 1.0$ with range 1.0 representing a complete environment.

In addition to the range calculation, a combination of scaling and shifting is used to fit the next character. Scaling is accomplished by the multiplication of the r times the previous high and low values, hi_{pc} and lo_{pc}, in the calculation of the range for c. Shifting is accomplished by the addition of the previous calculated low value lo_{pc}. A loop structure is used to iterate through all n characters. The equations for the lo and the hi are in (A.1)–

(A.2) and the pseudo code for the algorithm is represented (in Algorithm 8).

$$lo = lo_{pc} + (r \times lo_c), \tag{A.1}$$

$$hi = lo_{pc} + (r \times hi_c), \tag{A.2}$$

where $r = hi_{pc} - lo_{pc}$.

Algorithm 8 Arithmetic encodingseudo code.

The code subtracts the range representing the character for each iteration.

$lo = 0$ - initialize lo to zero

$hi = 0$ - initialize hi to one

for all Characters i to n **do**

$r = hi_{pc} - lo_{pc}$

$lo = lo_{pc} + (r \times lo_c)$

$hi = lo_{pc} + (r \times hi_c)$

end for

The results of the series of calculations for the given symbols is represented in Table A.1. Since there is only three symbols and four characters in the series it is a very short example, but it shows the functionality of the method. The first calculation for B has a range of $0 - 1.0$. The previous low $lo_{pc} = 0$ and the previous high $hi_{pc} = 1$. The high and low for B is $hi_B = 0.75$ and $lo_B = 0.50$. The range $r = hi_{pc} - lo_{pc} = 1 - 0 = 1.0$. By substituting the values into (A.1)–(A.2) the range representing B is acquired. The low $lo = lo_{pc} + (r \times lo_B) = 0 + (1 * 0.50) = 0.50$ and the high $hi = lo_{pc} + (r \times hi_B) = 0 + (1 * 0.75) = 0.75$. This is shown in the table as lo and hi on the row denoted by B. The following iterations use the same substitution method for the remaining sequence A, C, A to finish the table.

The output number can be any number in the final range $0.59375 - 0.609375$ and typically the smallest value in the range is used as this produces the shortest codeword. If we use 0.59375 to represent the final encoding we can convert it to binary for transmission and storage. The decimal number 0.59375 is equal to 100110, which is 6 bits long. If we encoded the same sequence using Huffman encoding the encode values are (A:0, B:10, C:11). The series of bits to represent the message is 100110 which is 6 bits. SLMN would result in the same code as Huffman. JAKE could also use base 3 but the conversion to binary would result in 6 bits as well. The theoretical ideal for the value is 6 bits in binary and with this arrangement none violate entropy theory.

Table A.1: Arithmetic encoding tabulated results for example

Character c	Range r	lo	hi
Initial	1	0	1
B	1	0.5	0.75
A	0.25	0.5	0.625
C	0.125	0.59375	0.625
A	0.03125	0.59375	0.609375

It is interesting to note that in this example arithmetic code could drop the zero and actually be represented with 5 bits. This may seem to be a contradiction to entropy theory, but it is only possible because of the relationship to truncating the trailing 0 in decimal relates to the same number. In this case the choice is trivial to get the result as the percentage is the first value, however, in general you would need to search the range to find this value and the search adds to the complexity of the problem and only saves 1 bit on average. Typically this step is not used [38].

Appendix B

COMPUTER RESULTS SELECT LEVEL METHOD

Table B.1: SLM tabulated results in CPU cycles. The first number is total CPU cycles T, the second number is maximum CPU cycles T_M, and the third number is minimum CPU cycles T_m.

T T_M T_m	T T_M T_m	T T_M T_m	T T_M T_m	T T_M T_m
7240 41 22	7206 37 23	7193 45 22	7238 37 23	7178 40 23
7218 39 23	7212 37 22	7240 41 22	7198 40 22	7209 37 22
7339 40 23	7146 49 21	7167 37 22	7168 36 22	7192 37 23
7168 42 22	7234 37 22	7196 40 22	7189 38 23	7206 38 22
7214 39 22	7232 61 22	7297 41 22	7195 37 23	7245 39 22
7216 42 22	7184 45 22	7175 42 22	7250 38 23	7159 38 22
7216 39 23	7187 36 23	7143 47 23	7228 43 23	7180 41 23
7232 38 22	7234 41 22	7184 37 23	7224 45 23	7120 40 22
7242 39 23	7222 35 23	7161 37 22	7098 41 22	7120 36 21
7224 60 22	7246 43 23	7226 38 23	7234 39 22	7166 36 23
7185 38 22	7187 38 22	7227 37 22	7206 39 23	7164 44 23
7206 39 22	7227 36 22	7110 48 22	7224 41 23	7159 51 22
7170 53 22	7245 40 22	7264 37 23	7272 43 22	7253 38 22
7135 37 23	7234 41 22	7325 93 23	7218 38 23	7267 37 22
7209 37 22	7229 40 23	7124 40 22	7144 36 23	7250 42 23
7256 39 23	7260 42 22	7134 37 22	7208 37 22	7194 36 22
7119 41 22	7192 40 22	7180 38 22	7224 39 21	7230 40 22
7164 37 22	7241 55 23	7127 38 22	7180 38 22	7239 38 22
7308 37 22	7147 37 22	7043 38 22	7228 36 22	7149 38 22
7220 43 22	7184 47 23	7208 50 23	7257 41 22	7141 39 23

Table B.2: Shannon-Fano-Elias tabulated results in CPU cycles. The first number is total CPU cycles T, the second number is maximum CPU cycles T_M, and the third number is minimum CPU cycles T_m.

T T_M T_m	T T_M T_m	T T_M T_m	T T_M T_m	T T_M T_m
20519 204 70	20463 199 72	20405 199 70	20429 197 71	20457 209 71
20443 187 70	20455 189 69	20495 195 70	20369 194 69	20406 203 70
20439 184 71	20476 185 69	20516 211 69	20405 203 70	20327 193 69
20485 228 69	20438 194 72	20459 204 73	20375 206 68	20383 199 71
20449 207 71	20408 195 71	20501 205 71	20408 199 71	20395 193 69
20476 200 71	20422 198 70	20413 182 71	20416 199 71	20404 183 70
20458 201 72	20456 205 70	20452 209 71	20428 197 71	20426 210 70
20377 194 68	20486 209 72	20359 213 70	20469 206 71	20374 194 72
20452 186 71	20452 183 71	20427 193 69	20423 193 70	20439 197 70
20380 176 70	20438 199 72	20425 194 68	20454 198 68	20409 187 69
20446 209 71	20436 196 71	20491 199 70	20379 190 70	20371 202 70
20517 188 72	20396 194 70	20453 200 71	20399 195 71	20458 184 70
20418 193 70	20522 184 69	20424 198 70	20461 204 71	20567 203 71
20403 196 71	20399 194 70	20511 196 72	20483 198 71	20440 187 71
20423 209 71	20401 208 71	20488 196 71	20443 197 70	20460 198 72
20464 195 69	20407 197 70	20412 194 72	20406 207 70	20366 219 71
20430 199 70	20411 201 71	20379 212 71	20456 193 72	20430 184 70
20422 195 70	20480 195 70	20425 191 70	20393 202 71	20386 197 71
20473 197 72	20404 190 70	20439 195 70	20425 174 71	20405 203 70
20457 195 71	20488 199 71	20399 207 71	20447 197 71	20361 196 71

Appendix C

COMPUTER RESULTS SORT LINEAR METHOD NIVELLATE

Table C.1: SLMN tabulated entropy results.

The values represent entropy for 100 cycles described in Sec. 6.4

7.86380	7.91228	7.96010	7.93293	7.95853
7.95406	7.84012	7.88523	7.95390	7.89994
7.72379	7.87160	7.89782	7.86263	7.95979
7.93388	7.95314	7.90271	7.90293	7.93608
7.91575	7.97988	7.83246	7.85599	7.92153
7.88369	7.86408	7.89295	7.93945	7.87584
7.88055	7.81574	7.87736	7.93949	7.95131
7.91116	7.91459	7.95915	7.96027	7.94478
7.88731	7.94926	7.95318	7.87107	7.83399
7.85343	7.95978	7.97330	7.94218	7.95165
7.92998	7.88998	7.98061	7.84791	7.94362
7.94562	7.90460	7.95326	7.90124	7.78498
7.92536	7.93395	7.96615	7.88744	7.94725
7.86632	7.86423	7.97317	7.85153	7.92429
7.88729	7.91335	7.95853	7.95017	7.96575
7.93600	7.95269	7.94587	7.90334	7.91873
7.89889	7.95981	7.95999	7.94436	7.95114
7.95587	7.83586	7.96597	7.95308	7.86657
7.91995	7.95336	7.92818	7.95883	7.96540
7.94534	7.90986	7.91055	7.96819	7.97296

Table C.2: Shannon-Fano-Elias tabulated entropy results.

The values represent entropy for 100 cycles described in Sec. 6.4

9.31661	9.31230	9.32998	9.33386	9.32337
9.33913	9.24889	9.32322	9.33848	9.27578
9.12432	9.29598	9.32599	9.27849	9.31808
9.30988	9.30933	9.29948	9.33098	9.34409
9.30820	9.31883	9.25001	9.28549	9.34559
9.25358	9.31716	9.30014	9.34239	9.28164
9.32333	9.22697	9.32437	9.31097	9.34137
9.34896	9.32922	9.30854	9.34150	9.33014
9.27474	9.33684	9.33506	9.27634	9.25136
9.27607	9.33967	9.32065	9.34607	9.36301
9.35761	9.30634	9.34462	9.25050	9.32823
9.32462	9.32901	9.30854	9.31110	9.20707
9.30287	9.31405	9.31400	9.33127	9.33294
9.27199	9.26623	9.34593	9.26250	9.32674
9.31628	9.33563	9.29973	9.31741	9.33961
9.34116	9.29964	9.32848	9.29941	9.35527
9.27822	9.36435	9.30640	9.31626	9.33391
9.35522	9.24868	9.31989	9.34741	9.27079
9.31840	9.32838	9.35405	9.29530	9.32959
9.33236	9.33152	9.33710	9.36444	9.31484

Table C.3: Ideal tabulated entropy results.

The values represent entropy for 100 cycles described in Sec. 6.4.

7.70450	7.72666	7.72152	7.72579	7.74072
7.75072	7.71548	7.71590	7.73052	7.69676
7.64363	7.72142	7.70634	7.71233	7.72038
7.70403	7.72227	7.71891	7.74344	7.72531
7.72554	7.75914	7.71685	7.69385	7.73482
7.69560	7.69179	7.71088	7.70434	7.70080
7.73161	7.69257	7.73141	7.72407	7.73583
7.72846	7.72775	7.72516	7.73276	7.73316
7.69699	7.73171	7.72800	7.69518	7.69593
7.68302	7.75013	7.74244	7.75776	7.76634
7.73683	7.74047	7.77255	7.68691	7.70810
7.73633	7.73548	7.71000	7.72673	7.67341
7.73162	7.71884	7.74430	7.73035	7.73441
7.71258	7.73272	7.74485	7.70813	7.71932
7.69453	7.71522	7.73286	7.71821	7.74775
7.72495	7.72232	7.74031	7.72577	7.73745
7.70796	7.74454	7.73466	7.72551	7.72805
7.74166	7.70214	7.74302	7.75353	7.70031
7.72172	7.73558	7.73221	7.69653	7.72591
7.76002	7.71760	7.72687	7.76044	7.75793

BIBLIOGRAPHY

[1] J. Abrahams. Code and parse trees for lossless source encoding. *Compression and Complexity of Sequences 1997*, pages 145–171, 1997.

[2] N. Abramson. *Information Theory and Coding*. McGraw-Hill, New York, New York, 1963.

[3] J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4):320–330, 1986.

[4] E. Bergman and S. Klein. Fast decoding of prefix encoded texts. *Data Compression Conference*, pages 143–152, 2005.

[5] M. Biskup. Guaranteed synchronization of Huffman codes. *Data Compression Conference*, pages 462–471, 2008.

[6] E. Bodden, M. Clasen, and J. Kneis. Arithmetic coding revealed – a guided tour from theory to praxis, 2001.

[7] A. Broder and M. Mitzenmacher. Pattern-based compression of text images. *Data Compression Conference*, page 300, 1996.

[8] M. Buro. On the maximum length of Huffman codes. *Information Processing Letters*, 45(5):219–223, 1993.

[9] A. Campos. Arithmetic coding @http://www.arturocampos.com/ac_arithmetic.html. World Wide Web electronic publication, July 1999.

[10] R. Capocelli and A. De Santis. Tight upper bounds on the redundancy of Huffman codes. *IEEE Trans. Inf. Theory*, 5:1084–1091, 1989.

[11] C. Chen, Y. Pai, and S. Ruan. Low power Huffman coding for high performance data transmission. *2006 International Conference on Hybrid Information Technology*, 1:71–77, 2006.

[12] S. Chen and J. Reif. Fast pattern matching for entropy bounded text. *Data Compression Conference*, page 282, 1995.

[13] K. Cheung and A. Kiely. An efficient variable length coding scheme for an iid source. *Data Compression Conference*, page 182, 1995.

[14] B. Ergude, L. Weisheng, F. Dongrui, and M. Xiaoyu. A study and implementation of the Huffman algorithm based on condensed Huffman table. *2008 International Conference on Computer Science and Software Engineering*, 20:42–45, 2008.

[15] R.M. Fano. *Transmission of Information.* M.I.T. Press, Cambridge, Mass, 1949.

[16] R. Franceschini and A. Mukherjee. Data compression using encrypted text. *Third International Forum on Research and Technology Advances in Digital Libraries*, page 130, 1996.

[17] R. Gallager. Variations on a theme by Huffman. *IEEE Trans. Inform. Theory*, 24(6):668–674, 1978.

[18] B. Geoghegan. Historiographic conceptualization of information: A critical survey. *IEEE Annals of the History of Computing*, pages 66–81, 2008.

[19] M. Golin and H. Na. Generalizing the kraft-mcmillan inequality to restricted languages. *Data Compression Conference*, pages 163–172, 2005.

[20] A. Grigoryan, S. Dursun, and E. Regentova. Lossless encoding based on redistribution of statistics. *International Conference on Information Technology: Coding and Computing*, 2:620, 2004.

[21] R. Hashemian. Direct Huffman coding and decoding using the table of code-lengths. *International Conference on Information Technology: Computers and Communications*, page 237, 2003.

[22] D.A. Huffman. A method for the construction of minimum-redundancy codes. *PIRE*, 40(9):1098–1101, September 1952.

[23] X. Kavousianos, E. Kalligeros, and D. Nikolos. Optimal selective Huffman coding for test-data compression. *IEEE Transactions on Computers*, pages 1146–1152, 2007.

[24] S. Klein and D. Shapira. Huffman coding with non-sorted frequencies. *Data Compression Conference (dcc 2008)*, page 526, 2008.

[25] D.E. Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.

[26] L.G. Kraft. *A device for quantizing, grouping, and coding amplitude-modulated pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

[27] L. L. Larmore and D. S. Hirschberg. A fast algorithm for optimal length-limited Huffman codes. *Journal of ACM*, 37(3):464–473, 1999.

[28] L.L. Larmore and T.M. Przytycka. Constructing Huffman trees in parallel. *SIAM Journal on Computing*, 24(6):1163–1169, 1995.

[29] D.A. Lelewer and D.S. Hirschberg. Data compression. *ACM Computing Surveys*, 19:261–296, 1987.

[30] R.L. Milidú, E.S. Laber, and A.A. Pessoa. A work efficient parallel algorithm for constructing Huffman codes. *Data Compression Conference*, 0:277, 1999.

[31] A. Moffat and J. Katajainen. In-place calculation of minimum-redundancy codes. *4th Intl. Workshop on Algorithms and Data Structures*, pages 393–402, 1995.

[32] A. Moffat and A. Turpin. On the implementation of minimum-redundancy prefix codes. *IEEE Trans. Comm.*, 45(10):1200–1207, 1997.

[33] Microsoft (MSDN). Queryperformancecounter function @http://msdn.microsoft.com/en-us/library/ms644904(VS.85).aspx. World Wide Web electronic publication, August 2009.

[34] H. K. Reghbati. Special feature an overview of data compression techniques. *Computer*, 14(4):71–75, 1981.

[35] J. J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal Research Development*, 20:198–203, 1976.

[36] X. Ruan and R. Katti. Reducing the length of Shannon-Fano-Elias codes and Shannon-Fano Codes. *MILCOM*, pages 1–7, 2006.

[37] X. Ruan and R. Katti. Using an innovative coding algorithm for data encryption. *submitted to the IEEE Transactions*, NA.

[38] A. Said. Introduction to arithmetic coding – theory and practice @http://www.hpl.hp.com/techreports/2004/HPL-2004-76.pdf. World Wide Web electronic publication, April 2004.

[39] C.E. Shannon and W. Weaver. A mathematical theory of communication. *BSTJ*, 27:379–423, 1948.

[40] H. Tanaka. Data structure of Huffman codes and its application to efficient encoding and decoding. *IEEE Trans. Inform Theory*, 33(1):154–156, 1987.

[41] A. Turpin and A. Moffat. Efficient approximate adaptive coding. *Data Compression Conference*, page 357, 1997.

[42] J. Vitter. Design and analysis of dynamic Huffman codes. *J. ACM*, 34(4):825–845, 1987.

[43] T. Welch. A technique for high-performance data compression. *IEEE Computer*, pages 8–18, 1984.

[44] I. Witten, R. Neal, J., and Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):50–540, 1987.

[45] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Computer*, 23:337–342, 1977.