Balancing Compressed Sequences

by

Saamaan Pourtavakoli
B.Sc., University of Tehran, 2008

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

Balancing Compressed Sequences

by

Saamaan Pourtavakoli
B.Sc., University of Tehran, 2008

Supervisory Committee

Dr. T. A. Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. Sima, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. T. A. Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. Sima, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

The performance of communication and storage systems can be improved if the data being sent or stored has certain patterns and structure. In particular, some benefit if the frequency of the symbols is balanced. This includes magnetic and optical data storage devices, as well as future holographic storage systems. Significant research has been done to develop techniques and algorithms to adapt the data (in a reversible manner) to these systems. The goal has been to restructure the data to improve performance while keeping the complexity as low as possible.

In this thesis, we consider balancing binary sequences and present its application in holographic storage systems. An overview is given of different approaches, as well as a survey of previous balancing methods. We show that common compression algorithms can be used for this purpose both alone and combined with other balancing algorithms. Simplified models are analyzed using information theory to determine the extent of the compression in this context. Simulation results using standard data are presented as well as theoretical analysis for the performance of the combination of compression with other balancing algorithms.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to express my gratitude to:

Prof. T. Aaron Gulliver, my supervisor for all the support he gave me through this degree.

Drs. Andrew Rowe and Mihai Sima, for their insightful feedback and comments.

DEDICATION

To my mother who showed me one can love no matter what, and one can keep going no matter what.
To my father who showed me how one can have honor and kindness of heart.

To friends, family, and whomever I learned from, even by a grain...

Chapter 1

Introduction

1.1 Balanced coding and modern storage systems

In some digital storage and/or transmission systems, it is desirable to have balanced words. A binary word with m bits is said to be balanced if it has $m/2$ zeros and $m/2$ ones for even m, and $(m-1)/2$ or $(m+1)/2$ ones for odd m (only the case of even sized words are considered in the remainder of this thesis without loss of generality). The input from a user is assumed to be arbitrary, so a reversible mapping scheme is needed to transform the input into a balanced output. Such a mapping is known as a coding scheme. A coding module that performs such a task in a system operates on an input stream of bits (symbols) and produces (encodes it into) an output stream of bits (symbols). *Block codes* are a class of codes that take an input of a fixed size and map it into an output of a (different) fixed size. The input bits (symbols) that are processed at the same time will be called the *input word* or *user word* and the resulting group of output bits (symbols) are referred to as the *output word* or *code word*. One method for measuring the performance of such codes is the *code rate*. The size of a word can be defined as the number of bits (symbols) in it, and for input and output word sizes m and n, the code rate is defined as m/n. It varies from zero to one and is considered a measure of the efficiency of a code. The higher the code rate, the more efficient the code. The goal of code design is to impose some restrictions on the output sequence and to keep the code rate as large as possible.

Many magnetic and optical storage systems benefit from *balanced codes*. In optical disks (CDs and DVDs), balanced codes are employed to reduce the interaction between the data on the disk and the servo systems that follow the data tracks on the

disk. Also, common disturbances such as fingerprints have low frequency components and may cause incorrect readouts. These artifacts can be removed by highpass filtering, but it is necessary for the written data to not have low frequency components [7]. Balanced codes can help this problem by removing the DC component of the sequence.

Holographic storage is another optical storage system which is currently being researched for next generation storage systems, and is very promising in terms of its very high speed and capacity. The performance of holographic storage systems can be improved if the data pages are balanced (or almost balanced) for several reasons. These reasons will be discussed in the following section. In order to design a proper code, it is necessary to analyze this system, learn its characteristics, and identify the design criteria. Although the analysis presented in this thesis is general, some considerations are introduced from the properties of holographic system as a practical application.

1.2 Holographic storage systems

In holography, the light scattered from an object is recorded. Later it is possible to reproduce this scattered light so that it appears as if the object is still at the same position relative to the recording media as when it was recorded. In simple terms, the scattering properties of the object are recorded, so that if the position and/or orientation of the viewing system changes, the image changes in the same way as if the object were still present. As can be seen from Figure 1.1, when recording, two light beams arrive at the hologram. One comes from the object (signal beam), and the other is focused directly at the hologram and is called the reference beam. The interference between the two beams, when they come together, makes a fringe pattern of light and dark areas. On the hologram, this interference pattern is captured as a pattern of varying refractivity (refractive index) in the holographic medium (typically a holographic crystal or polymer).

In 1963, van Heeden of Polaroid first proposed the idea of storing data in three dimensions [12]. He tried to use holography, a fundamentally different approach from CDs and DVDs, to store hundreds of gigabytes on a disk the size of a CD. In the early 1970s, photorefractive media were investigated for use in holographic data storage and theoretical investigations indicated a potential density of 10^{13} b/cm with data transfer rates of gigabits per second. Unfortunately, at that time input and output interfaces

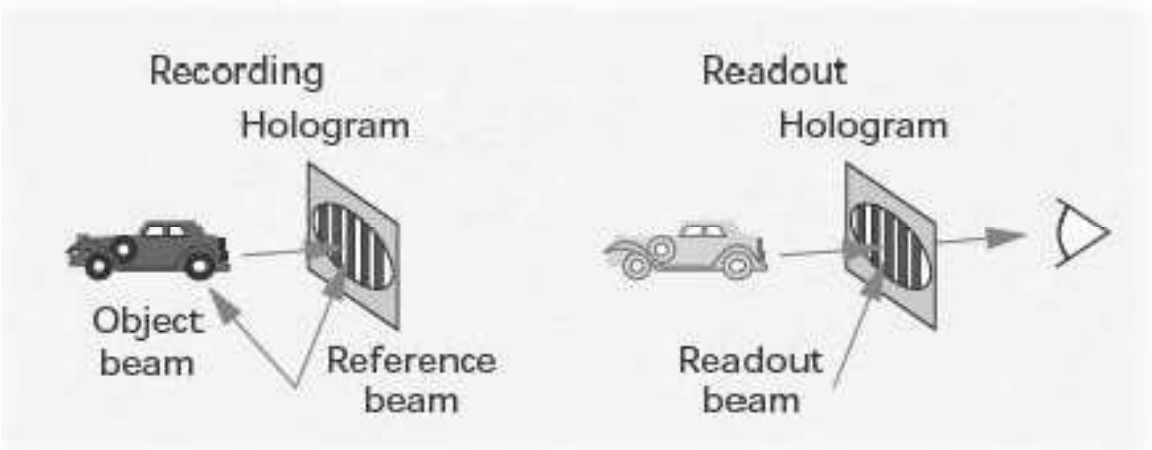


Figure 1.1: Holography; Object image is captured in the holographic medium, and when readout the viewer can observe an image as if the object was still there [11].

and processing systems were not available to take advantage of these high data rates. It was also difficult to fix the information in the crystals and avoid data erasure upon readout. Much later in the mid 1990s, joint work by Stanford University and IBM led to the development, building, and testing of a high capacity holographic storage system including hardware implemented holographic channel decoding electronics for transfer rates exceeding 10 Gb/s and capacities of more than 100 Gb per 6.5-in-diameter disk [5].

Typical main components of a holographic storage system are coherent light source, SLM (Spatial Light Modulator), photosensitive holographic medium, detector array (typically CCD), optical apparatus (to manage light beams properly), and electronic control systems as well as input/output interfaces. Figure 1.2 shows a typical system with electronic control units. The light source is usually a laser beam which is split into two separate beams. One is used as the reference beam in the storage (writing) process and also as the readout beam in the retrieval process. The other passes through the SLM and is the signal beam. SLM is essentially an array of pixels where each pixel can be either set to block or pass light (similar to a controllable transparency for an overhead projector), using an electrical control system. It is also called a *page composer* because the binary data is composed of pages of ON and OFF point sources.

The holographic storage system architecture is largely determined by the type of recording medium. Broadly speaking, holographic storage material is divided into two classes; those based on thin (a few hundred micrometers) photosensitive organic media and and those based on thick (a few millimetres to centimetres) inorganic photorefractive crystals [5]. Multiplexing in holographic storage refers to techniques for storing multiple pages of data in the same volume of storage material. Each page should be accessible separately and the readout quality of each page acceptable even with possible cross-talk and interference from other pages. There are several methods to achieve this such as angular, wavelength, phase encoded, shift, and spatial multiplexing. The output device detects the optical signals and transforms them into electrical signals. It is typically an array of detector pixels such as a charge-coupled device (CCD) or a CMOS pixel array.

If each data page input to the SLM is balanced (or at least approximately balanced), then the overall light intensity will not vary much from page to page, ensuring that the intensity ratio between the object beam and the reference beam is relatively constant. In this way, an exposure schedule for writing multiple holograms within a

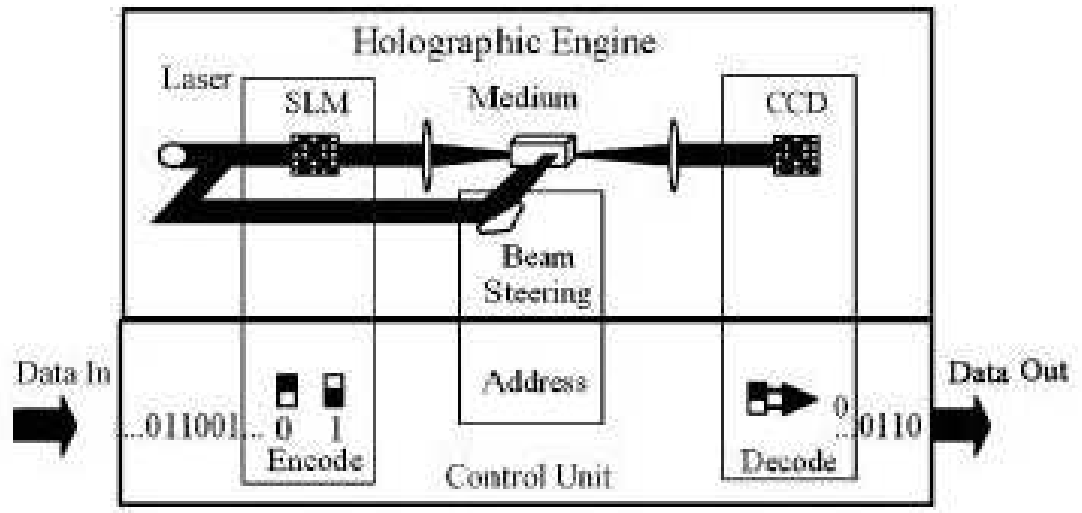


Figure 1.2: A typical holographic storage system including electronic control units [5].

stack can be set accurately. Also, the balanced condition is necessary for detection using a global binary threshold [1].

The advantages of using a balanced code in a holographic storage system are as follows. Knowing the number of ON pixels (number of 1s), which is possible if constant weight codes are used, eliminates the need for a threshold for demodulation by just using sorting. Note that the highest code rates (among constant weight codes) are possible with balanced codes. Further, reducing the effects of noise due to nonlinearities in the system sometimes requires *uniform recording*. This is when the beam ratio (ratio of the reference and signal beam) is kept fairly constant which in turn requires that the weight, or the number of ON pixels, in a given hologram (page) be constant [4].

1.3 Practical approaches to balanced coding

The authors in [4] showed that using balanced coding and increasing the input word size can decrease the symbol error rate (SER) for a given signal to noise ratio (SNR) and constant capacity. A very simple modulation code was analyzed in [1] and shown to significantly improve the storage system performance. Every bit is encoded into a balanced 1×2 array. It produces balanced pages and is very easy to encode/decode, but it has a very poor code rate.

The author in [7] divided the approaches that have been used in practical balanced coding into four groups, *zero-disparity code*, *low-disparity code*, *polarity bit code*, and *guided scrambling*. Here *disparity* is defined as the difference between the number of 1s and 0s.

Zero-disparity codes, as the name suggests, are one to one translations that map each input word into a balanced code word which has an equal number of zeros and ones (hence "zero-disparity"). Such code words can be concatenated (without any extra considerations), and the resulting sequence will still be balanced. On the down side, designing such codes with a high code rate and low complexity, particularly for longer codewords, is a challenging task. However, Knuth [9] proposed an easily implementable encoding technique with zero-disparity codewords which is capable of handling (very) large blocks. A thorough overview and analysis of his technique is provided is the following chapters.

Low-disparity codes are somewhat similar to the previous category. Codewords are grouped into pairs with equal and opposite disparity (a codeword that has a

certain number, say k, 1s more than 0s is paired with another code word with k 0s more than 1s). The balanced words in the code-book are assigned to input words using a one to one mapping. The remaining input words are assigned to one of the corresponding pairs. This will give the encoder a choice. In this case, the encoder chooses the codeword from the assigned pair that brings the accumulated disparity closer to zero, and the choice will then be signalled to the decoder.

Polarity bit codes, are slightly different than the previous category. For each block of input bits the encoder decides to send either the unmodified block or the complement of the block. The choice is again made in a way that brings the disparity closer to zero. A "polarity bit" is appended to the block to show which choice was made. This method is particularly interesting because there is no complex mapping involved, which means no look-up tables.

The last of the four techniques is guided scrambling. Guided scrambling is a member of a larger class of related coding schemes called *multi-mode* codes [7]. Each input word is augmented with different substrings and then scrambled using a given scrambling mechanism. The encoder tries a large selection of augmenting substrings and sends the resulting string which is balanced or brings the accumulated disparity closer to zero (or satisfies some other desirable criteria). The decoder descrambles the received string and removes the augmenting substring to get the input word. This approach relies on the fact that for a large enough set of augmentation substrings, there is a high probability that strings exist after scrambling which satisfy the desired criteria. Further, design parameters such as the scrambling mechanism, the set of augmentation substrings, etc., make it possible to "guide" the scrambling process to ensure a codeword with the desired properties.

1.4 How this thesis is organized

In this thesis we focus on two topics. First, we look at using compression as a means of achieving (almost) balanced words and establish that compression can help with balancing sequences. Second, we analyze one of the best practical balancing schemes and show how its performance is affected if the input has been compressed.

The remaining chapters of the thesis are organized as follows. In Chapter 2, a simple yet enlightening case of a random memoryless binary source is examined and expressions for the disparity and weight distribution of the output sequence are derived. This is accompanied by empirical results on the effects of applying well-

known compression programs to standard test data. The next chapter presents an overview of the balancing algorithm presented in [9] as an example of a practical and effective balanced coding method. An overview is also given of another more complex method presented in [6] that improves the former method. In Chapter 3, the first method above is analyzed in depth and expressions are derived for the performance of this method when combined with compression. Results using standard test data are presented and analyzed. The next chapter examines the approach in [6] as a technique that achieves the theoretical upperbound in [9], as well as its performance when combined with compression. Finally, concluding points and future work are discussed in Chapter 5.

Chapter 2

Balanced Properties of Compressed Sources and Data

2.1 Compression, source entropy, and balancing sequences

Typically, the output of an arbitrary (binary) source can be compressed and thus the entropy per bit is increased (in this thesis, for simplicity, entropy per bit is refered to as entropy when there is no confusion). Intuitively, increased entropy per bit means a (statistically) more random message or a more equal probability of zeroes and ones. Therefore with a perfect compression algorithm that achieves the maximum entropy and for an infinite sequence, a balanced output is expected. With a finite sequence and available compression algorithms, there is a higher chance for an unbalanced message to be compressed a lot compared to a close to balanced message. In the former case, the space saved can be used for various purposes or just left as a gain. In the latter case, the lack of compression is not a significant issue since the message is already almost balanced.

Assume a memoryless binary source (bits are independant) with entropy H. We know that

$$H = p\log_2(\frac{1}{p}) + (1-p)\log_2(\frac{1}{1-p}), \tag{2.1}$$

where p is the probability of a zero being generated and $(1-p)$ the probability of a one being generated. Because of symmetry, we only consider $p \in [0, 0.5]$ without loss of generality and define $p = h^{-1}(H)$. Figure 2.1 shows $H = h(p)$ over the interval

$[0, 1]$ and $p = h^{-1}(H)$ the inverse function after limiting p to $[0, 0.5]$.

For an m-bit sequence generated by such a source, the probability of its weight being equal to w is

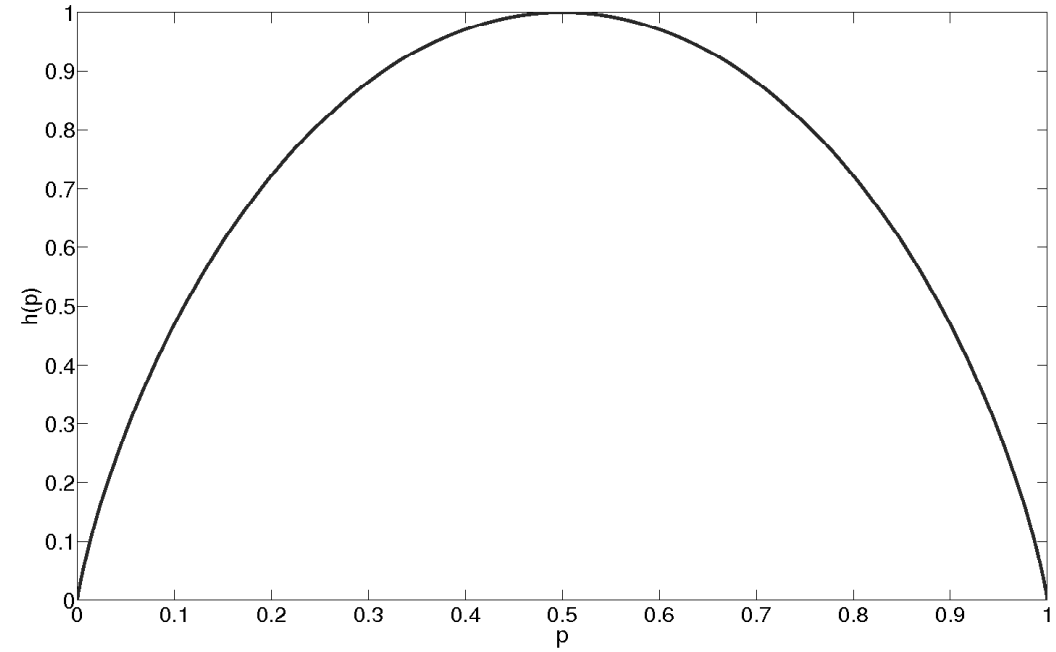
$$\begin{aligned} Pr(m, w, p) &= \binom{m}{w} p^w (1-p)^{m-w} \\ &= \binom{m}{w} [h^{-1}(H)]^w [1 - h^{-1}(H)]^{m-w}. \end{aligned} \tag{2.2}$$

Now if we fix $w = m/2$ we can see the effect of an increase in source entropy on the probability of a balanced sequence being generated. Figure 2.2 shows the results with a word size of $m = 50$ bits. This shows that the probability of a generated word being balanced dramatically increases as the source entropy increases. Figure 2.2 also shows how the probability of a generated word being "almost balanced" changes as the entropy increases. Note that here we call a word "almost balanced" if its weight remains in a *small* interval, say a symmetrical interval of length $2K$ for a positive integer K, around the balanced weight. The probability of being almost balanced is given by

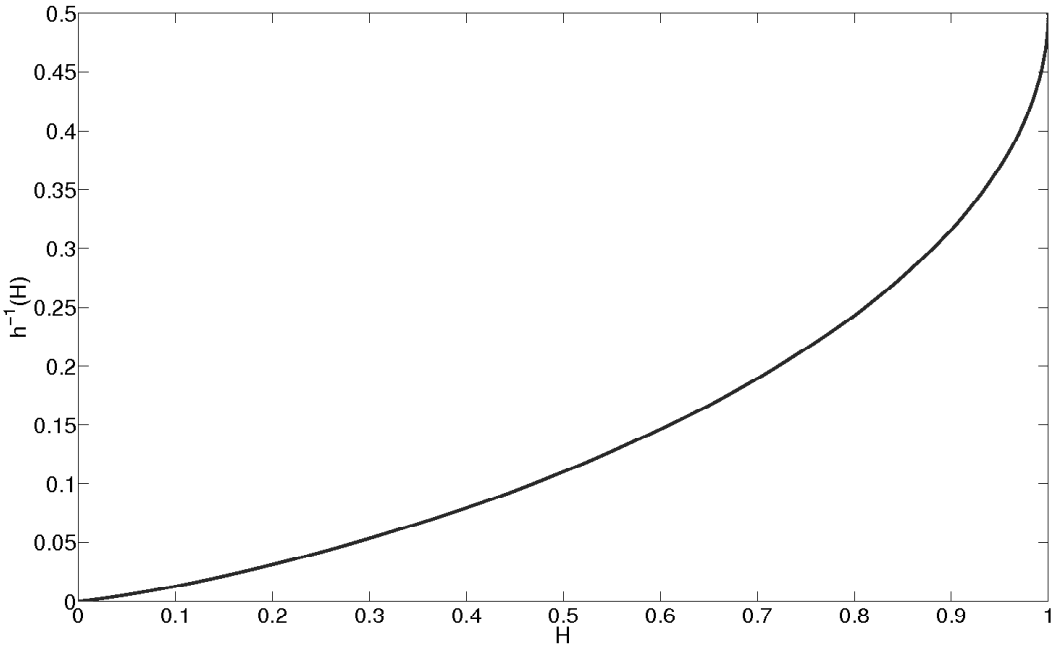
$$\begin{aligned} Pr_K(m, p) = \\ \sum_{w=\frac{m}{2}-K}^{\frac{m}{2}+K} \binom{m}{w} [h^{-1}(H)]^w [1 - h^{-1}(H)]^{m-w}. \end{aligned} \tag{2.3}$$

In Figures 2.2 and 2.3 the deviation interval is set to 1% of the word size (m). Figure 2.2 shows that as the entropy approaches 1, $Pr_K()$ goes from 0% to about 52%. Finally, Figure 2.3 shows the probability of the generated sequence being almost balanced for four different word sizes. It only focuses on the region where the probabilities of almost balanced words are well above zero (source entropy close to one). This corresponds to a bit probability of $p \in [0.35, 0.50]$. We see that only for larger word sizes does the probability of almost balanced words get close to 1 and the higher the word size, the faster this happens.

It is worth mentioning that in holographic storage systems which we are interested in one page will be accessed (read/written) at a time. Therefore a page is a meaningful word size in these systems. The page typically consists of 2^{20} bits ($2^{10} \times 2^{10}$ pixels) which provides us with a large word size. Further, increasing the word size in some coding schemes results in an increase in the code rate, which is desirable. Figure 2.4



(a)



(b)

Figure 2.1: Plot (a) shows the entropy per bit of a memoryless binary source H as a function of p, the probability of a zero ($H = h(p)$). Plot (b) shows the inverse function after limiting p to $[0, 0.5]$.

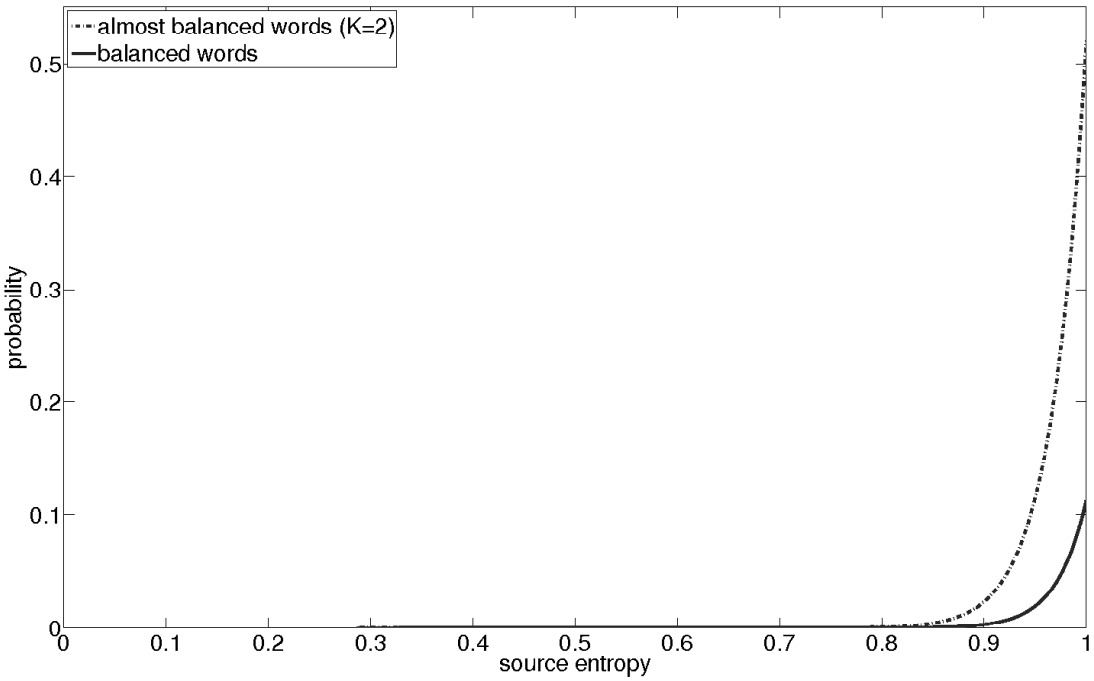


Figure 2.2: Probability of a sequence of size $m = 50$ bits, generated by a source with entropy H, being balanced and "almost balanced" as a function of H.

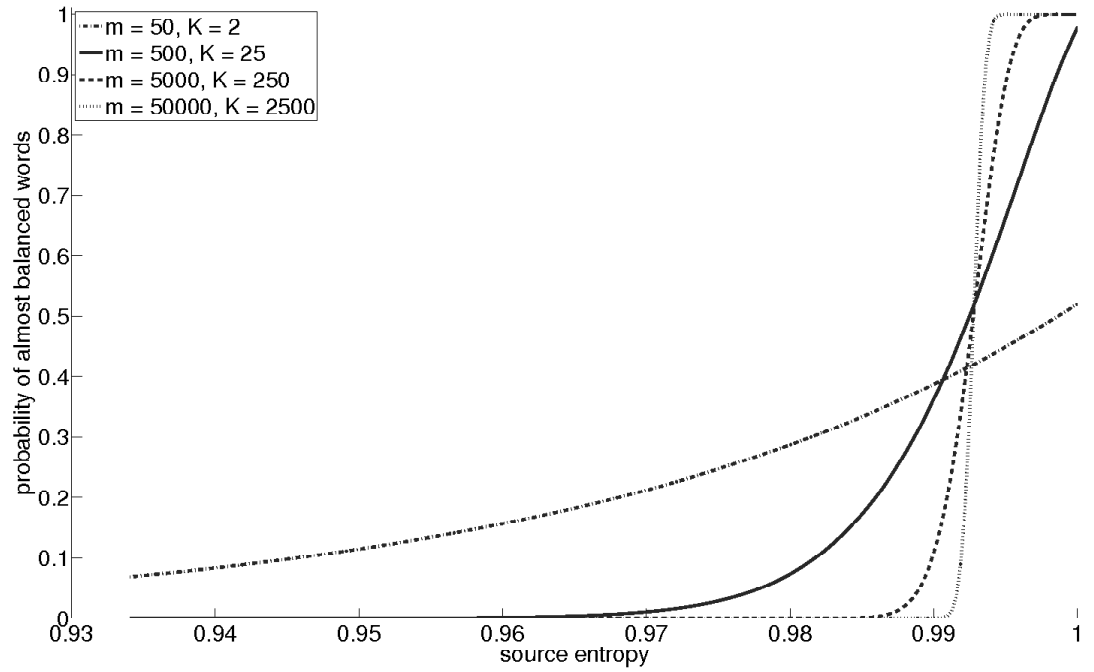


Figure 2.3: Probability of a generated sequence being almost balanced versus the source entropy for sequences of different sizes.

shows, the code rate as a function of the block size for the Knuth coding algorithm [9] (which is discussed in detail in the next chapter).

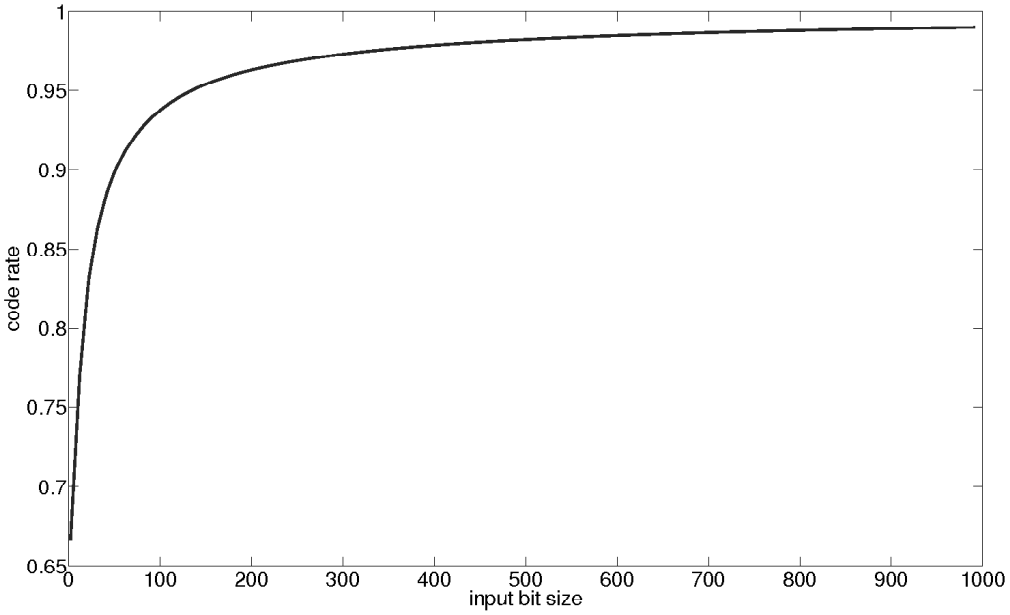


Figure 2.4: This graph illustrates how the code rate increases as the input word size increases. The word size stops at 1000 which is enough to show that as m increases the code rate tends to one.

2.2 Simulation results

Previous balancing methods in the literature such as Knuth's method [9], Immink-Weber's method [6], or others generally assume that every possible word (2^m possibilities for a word with m bits), has the same probability of appearing in the input stream. However, given that today's standard and readily available compression algorithms are fast, high performance, and have several advantages such as saving space, and possibly generating more balanced outputs, it is logical to assume that we have to deal with the output of a compression algorithm. Therefore, the performance of previous methods for the construction of balanced codes may not necessarily remain the same when applied to compressed sources. To investigate this, we chose the GNU Zip ("gzip") program as the compression tool.

GNU Zip is open source free software, created by Jean-Loup Gaily and Mark Adler, which uses a combination of LZ77 and Huffman coding and is the standard compression tool available in any distribution of the Linux operating system [10].

Aside from being a good general compression tool, it is currently used in communication systems. A good example of this is in HTTP/1.1 streams where it is used to improve the performance by encoding the header and/or payloads [8].

The test data used were well known corpora composed of "real world" user generated data (computer files), such as the Calgary Corpus, the Canterbury Corpus, and the (Canterbury) Large Corpus. The files found in these corpora have small sizes. However, larger file sizes are preferred because of our particular interest in holographic storage systems and their typically large pages of data which can in turn be translated into very large message words. Remember that having large code words can be beneficial as shown in Figure the previous section. Another corpus used in our analysis is from the Silesian University of Technology in Poland which contains much larger files compared to the classic corpora (more information on this corpus can be found at [3]). Details about the files in each corpus and their sizes are tabulated in Tables 2.1-2.4.

A critical property that controls the balancing performance is the distribution of message weights or equivalently the distribution of the massage imbalance. In order to get some empirical data on this distribution, we compressed each file with "gzip," then divided the zipped file into words of 2^{20} bits. The weight of each word was calculated and normalized by the size. Tables 2.5 and 2.6 show this result for the two smallest (Calgary and Canterbury). Because of their small size, all files from these corpora produced one to three words, therefore the results are just tabulated. For the Silesia and Large corpora, histograms of the normalized distribution of the weights of the described words were constructed. Figures 2.5-2.10 show these distributions for word sizes of 2^{20} and 2^{18}. In the case of the Silesia Corpus (Figures 2.7-2.10) there are three image files (namely x_ray, sao, and mr), that have slightly different characteristics and because of this, one histogram is provided for only these three together and another histogram is provided for the rest of the files. As we can see, they have different average weights but the distribution about the average is similar for the two histograms. For the purposes of comparison, a histogram of the uncompressed test files was also prepared. The files were broken into words of 2^{20} bits and then the weight of each word normalized by the word size. The result is the histogram in Figure 2.11 that shows a distribution not concentrated about the average.

There are a few observations to be made by looking at Figures 2.5-2.10 and comparing them to Figure 2.11. As we can see, after being compressed, messages are

File name	Description	File size (B)
Alice29.txt	English text	152089
asyoulik.txt	Shakespeare	125179
cp.html	HTML source	24603
fields.c	C source	11150
grammar.lsp	LISP source	3721
kennedy.xls	Excel Spreadsheet	1029744
lcet10.txt	Technical writing	426754
plrabn12.txt	Poetry	481861
ptt5	CCITT test set	513216
sum	SPARC Executable	38240
xargs.1	GNU manual page	4227

Table 2.1: List of files in the Canterbury Corpus.

File name	Description	File size (B)
bib	Bibliography (refer format)	111261
book1	Fiction book	768771
book2	Non-fiction book (troff format)	610856
geo	Geophysical data	102400
news	USENET batch file	377109
obj1	Object code for VAX	21504
obj2	Object code for Apple Mac	246814
paper1	Technical paper	53161
paper2	Technical paper	82199
pic	Black and white fax picture	513216
progc	Source code in "C"	39611
progl	Source code in LISP	71646
progp	Source code in PASCAL	49379
trans	Transcript of terminal session	93695

Table 2.2: List of files in the Calgary Corpus.

File name	Description	File size (B)
E.coli	Complete genome of the E. Coli bacterium	4638690
bible.txt	The King James version of the bible	4047392
world192.txt	The CIA world fact book	2473400

Table 2.3: List of files in the (Canterbury) Large Corpus.

File name	Description	File size (B)
dickens	English text; Collected works of Charles Dickens	10192446
mozilla*	Tarred executables of Mozilla 1.0 (Tru64 UNIX edition)	51220480
mr	Image; Medical magnetic resonance image	9970564
nci	Database; Chemical database of structures	33553445
ooffice	exec; A dll from Open Office.org 1.01	6152192
osdb	Database; Sample database in MySQL format from Open Source Database Benchmark	10085684
reymont	Polish PDF; Text of the book Chopi by Wadysaw Reymont	6627202
samba*	Tarred source code of Samba 2-2.3	21606400
sao	Bin data; The SAO star catalog	7251944
webster	Html; The 1913 Webster Unabridged Dictionary	41458703
xml	Html; Collected XML files	5345280
x_ ray	Image; X-ray medical picture	8474240

Table 2.4: List of files in the Silesia Corpus. The files marked with an asterisk (*) were not used in simulations.

Test file	Compressed bits	Weight	Percentage
Alice 29.txt	435480	220422	50.62
Asyoulik.txt	391608	198005	50.56
cp.html	63992	32051	50.09
fields.c	25144	12434	49.45
grammar.lsp	9968	5019	50.35
kennedy.xls	1048576	535090	51.03
	605656	308546	50.94
lcet10.txt	1048576	527492	50.31
	110504	56066	50.74
plrabn12.txt	1048576	523930	49.97
	513088	255073	49.71
Ptt5	451544	235243	52.10
Sum	103392	53985	52.21
Xargs.1	14048	6809	48.47

Table 2.5: Average bit weight in percentage for the Canterbury Corpus.

Test file	Compressed bits	Weight	Percentage
Bib	280504	140078	49.94
Book1	1048576	526298	50.19
	1048576	522695	49.85
	409856	203726	49.71
Book2	1048576	529721	50.52
	04920	304548	50.35
Geo	547944	290939	53.10
News	1048576	7527315	50.29
	110144	55420	50.32
Obj1	82584	41151	49.83
Obj2	653048	335059	51.31
Paper1	148616	74131	49.88
Paper2	238024	120404	50.58
Paper3	144776	72313	49.95
Paper4	44288	22527	50.86
Paper5	39960	20125	50.36
Paper6	105856	53223	50.28
Pic	451536	235235	52.10
Progc	106200	52673	49.60
Progl	130184	65000	49.93
Progp	89968	44434	49.39
Trans	151880	75263	49.55

Table 2.6: Average bit weight in percentage for the Calgary Corpus.

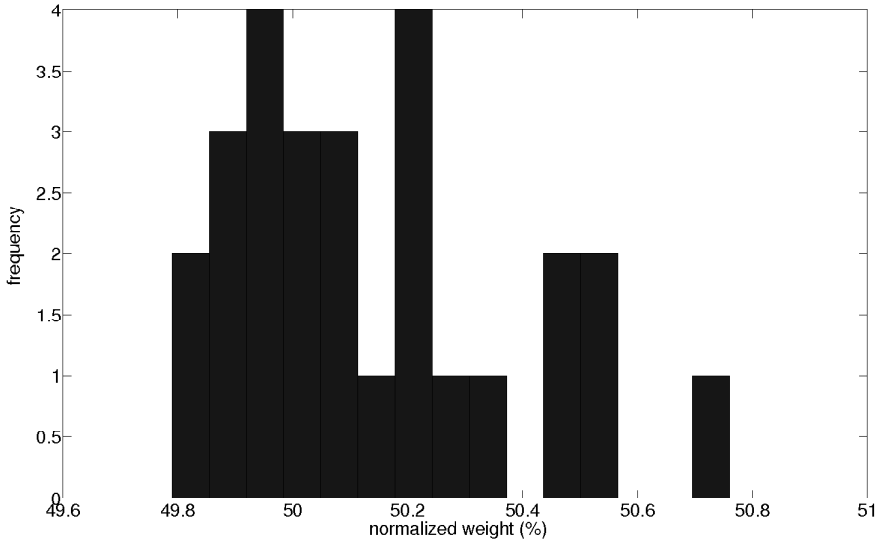


Figure 2.5: Normalized weight distribution for the Large corpus. All files were used producing a total of 27 words of 2^{20} bits. The average normalized weight is 50.13%.

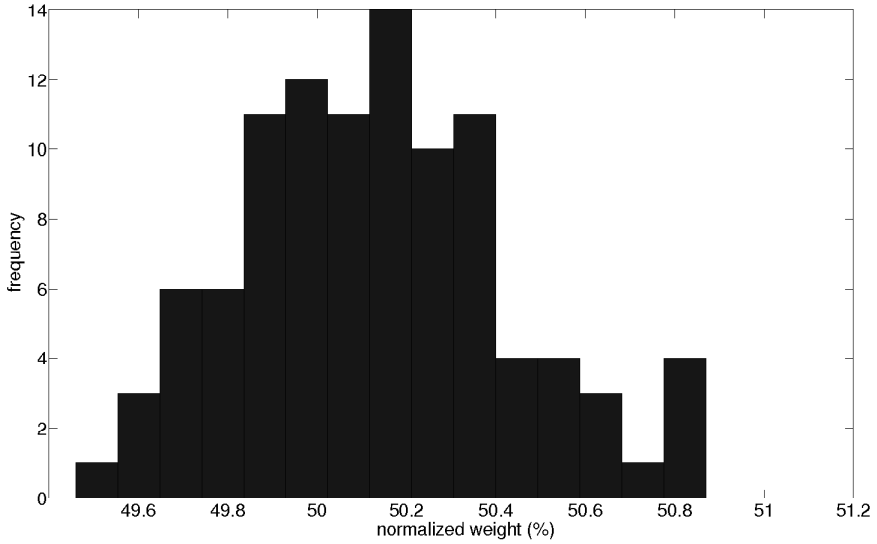


Figure 2.6: Normalized weight distribution for the Large corpus. All files were used producing a total of 101 words of 2^{18} bits. The average normalized weight is 50.13%.

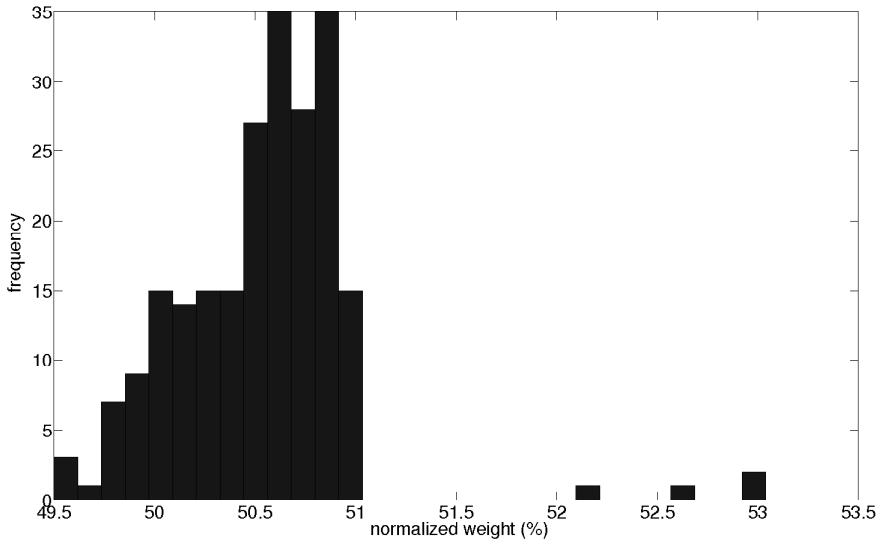


Figure 2.7: Normalized weight distribution for the Silesia Corpus. Only non-image files were used (x_ray, sao, and mr were excluded), producing a total of 223 words of 2^{20} bits. The average normalized weight is 50.54%.

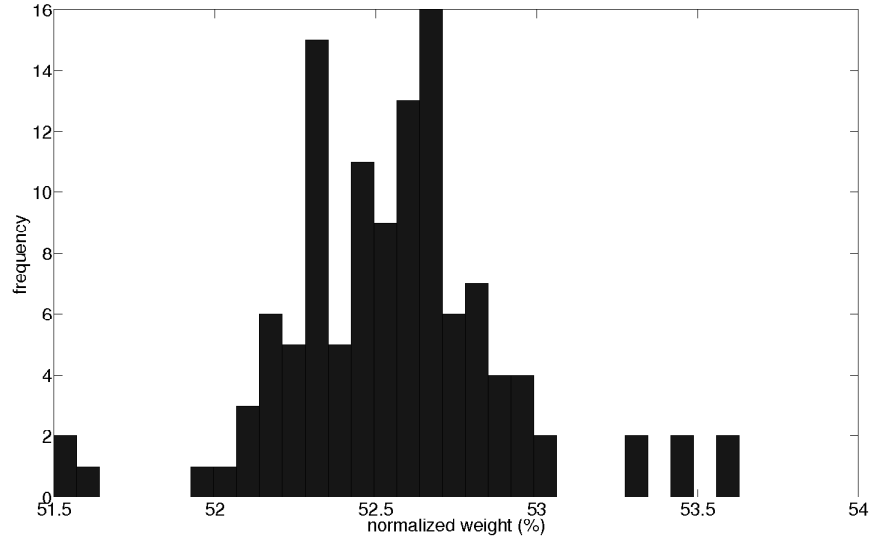


Figure 2.8: Normalized weight distribution for the Silesia Corpus. Only image files (x_ray, sao, and mr), were used producing a total of 117 words of 2^{20} bits. The average normalized weight is 52.56%.

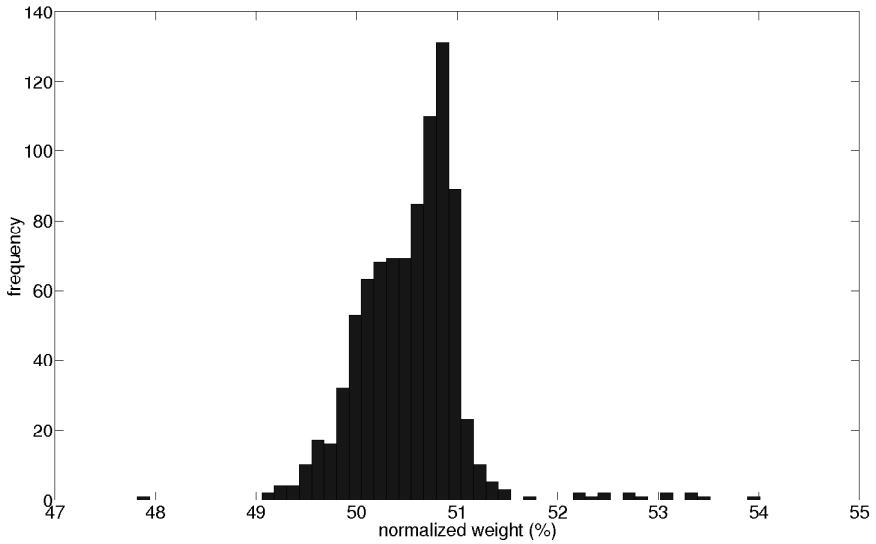


Figure 2.9: Normalized weight distribution for the Silesia Corpus. Only non-image files were used (x_ray, sao, and mr were excluded), producing a total of 879 words of 2^{18} bits. The average normalized weight is 50.54%.

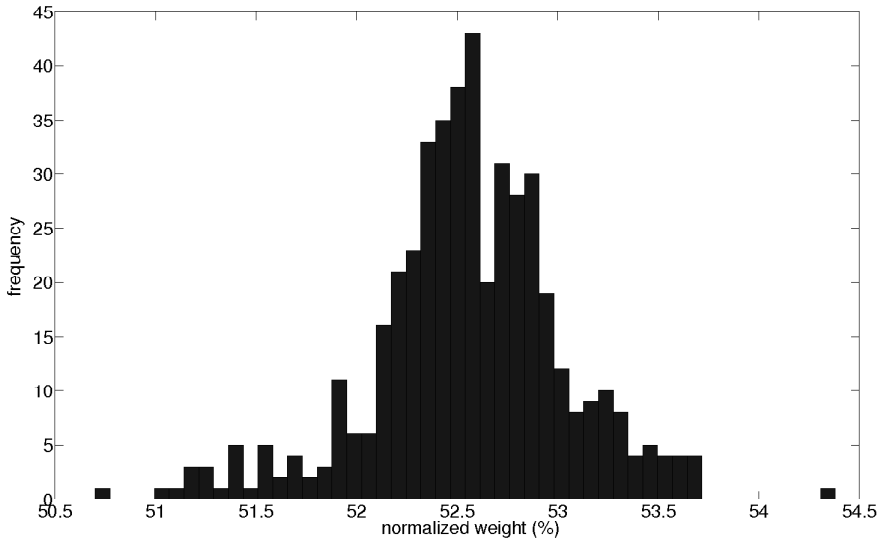


Figure 2.10: Normalized weight distribution for the Silesia Corpus. Only image files were used (x_ray, sao, and mr), producing a total of 461 words of 2^{18} bits. The average normalized weight is 52.56%.

very close to being balanced. The imbalance is almost always smaller than 2-3%. Another observation is that the distributions show some skewness which of course

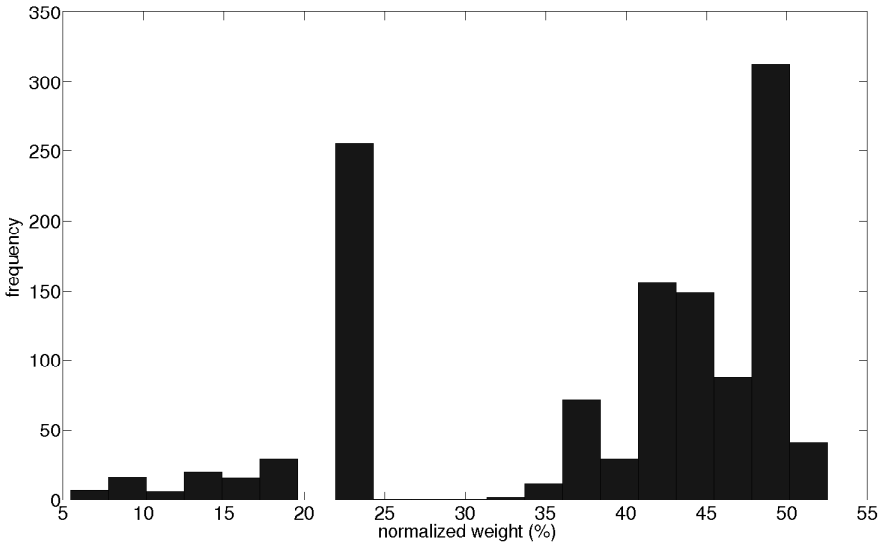


Figure 2.11: Normalized weight distribution for the uncompressed (original) files from all corpora. All files were used producing a total of 1212 words of 2^{20} bits. The average normalized weight is 37.09%.

shows characteristics of the compression tool and perhaps the test data.

The test data was chosen to be the kinds of user data that are commonly dealt with, and also vary in type to make the experiments generic enough. It can be established that the weight distribution of the words after compression is dramatically different than what is usually assumed in the balanced code literature. In the next chapter, we analyze how the performance of well-known balanced code construction methods can be improved based on the results in this chapter. In particular, the aforementioned graphs are used as typical weight distributions of the sources to be balanced.

Chapter 3

Applying The Knuth Method to Compressed Data

3.1 An overview of the Knuth method

Knuth [9], proposed a novel and very simple method for the construction of a balanced code. He also discussed different methods for encoding the auxiliary data (the data added during encoding so that unique decoding is possible afterwards). The corner stone of his method is that for a user word, b, of m bits, if every bit starting from the first is flipped sequentially there will be at least one location at which the resulting message is balanced:

$$b_1 b_2 b_3 \cdots b_m \longrightarrow \overline{b_1 b_2} \cdots \overline{b_k} b_{k+1} b_{k+2} \cdots b_m \tag{3.1}$$

where b_j is the j-th bit of the user word $\mathbf{b}$. If there is more than one bit position that results in a balanced word, the smallest index is chosen, which is called k here. The balanced word generated is then ready to be sent over the channel and only the bit position where the flipping ends (or possibly some other auxiliary data to recover the user word at receiver) has to be encoded. Note that a channel is the physical medium that transports the message from a transmitter to a receiver. This transportation can be direct as with a wire or indirect as in a computer hard disk with storage and retrieval processes.

Showing that the encoded message will still be uniquely decodeable is straightforward. Suppose we have a word $\mathbf{u}$ with m bits (m even). We define $\mathbf{u}^k$, to be $\mathbf{u}$ with its first k bits flipped and $weight(\mathbf{u})$ to be the number of ones in $\mathbf{u}$. To apply Knuth's

method, we find the smallest k such that $\mathbf{u}^k$ is balanced or $weight(\mathbf{u}^k) = m/2$, and then encode k as a balanced prefix, $\mathbf{r}$, with a certain (fixed) number of bits and send $\mathbf{rv}$ where $\mathbf{u} = \mathbf{u}^k$. However, if we use the weight of the input word as auxiliary data we encode $weight(\mathbf{u})$ in $\mathbf{r}$ and send $\mathbf{rv}$. We can easily show, by contradiction, that the latter is uniquely decodeable as well. When decoding, $weight(\mathbf{u})$ is obtained from decoding $\mathbf{r}$, and $\mathbf{u} = \mathbf{v}^k$ where

$$weight(\mathbf{v}^k) = weight(\mathbf{u}). \tag{3.2}$$

Assume there is a k' such that $k' < k$ and

$$weight(\mathbf{v}^{k'}) = weight(\mathbf{u}). \tag{3.3}$$

Denote a substring of an arbitrary sequence $\mathbf{u}$, starting from the j_1-th bit and ending at the j_2-th bit (inclusive), by

$$\mathbf{u}[j_1 : j_2]. \tag{3.4}$$

It can be verified (Figure 3.1 will be helpful in that regard) that from

$$weight(\mathbf{v}^k) = weight(\mathbf{u}) = weight(\mathbf{v}^{k'}) \tag{3.5}$$

we have

$$\begin{aligned} weight(\mathbf{u}[k'+1:k]) &= weight(\mathbf{v}^{k'}[k'+1:k]) \\ &= weight(\overline{\mathbf{u}[k'+1:k]}). \end{aligned} \tag{3.6}$$

It then follows that

$$weight(\mathbf{u}[k'+1:k]) = (k-k')/2. \tag{3.7}$$

and from there

$$weight(\mathbf{v}^{k'}) = weight(\mathbf{v}^k) \tag{3.8}$$

and because

$$weight(\mathbf{v}) = m/2 \tag{3.9}$$

then

$$weight(\mathbf{u}^{k'}) = m/2 \tag{3.10}$$

as well. This means that $\mathbf{u}^{k'}$ is also balanced, which contradicts k being the smallest

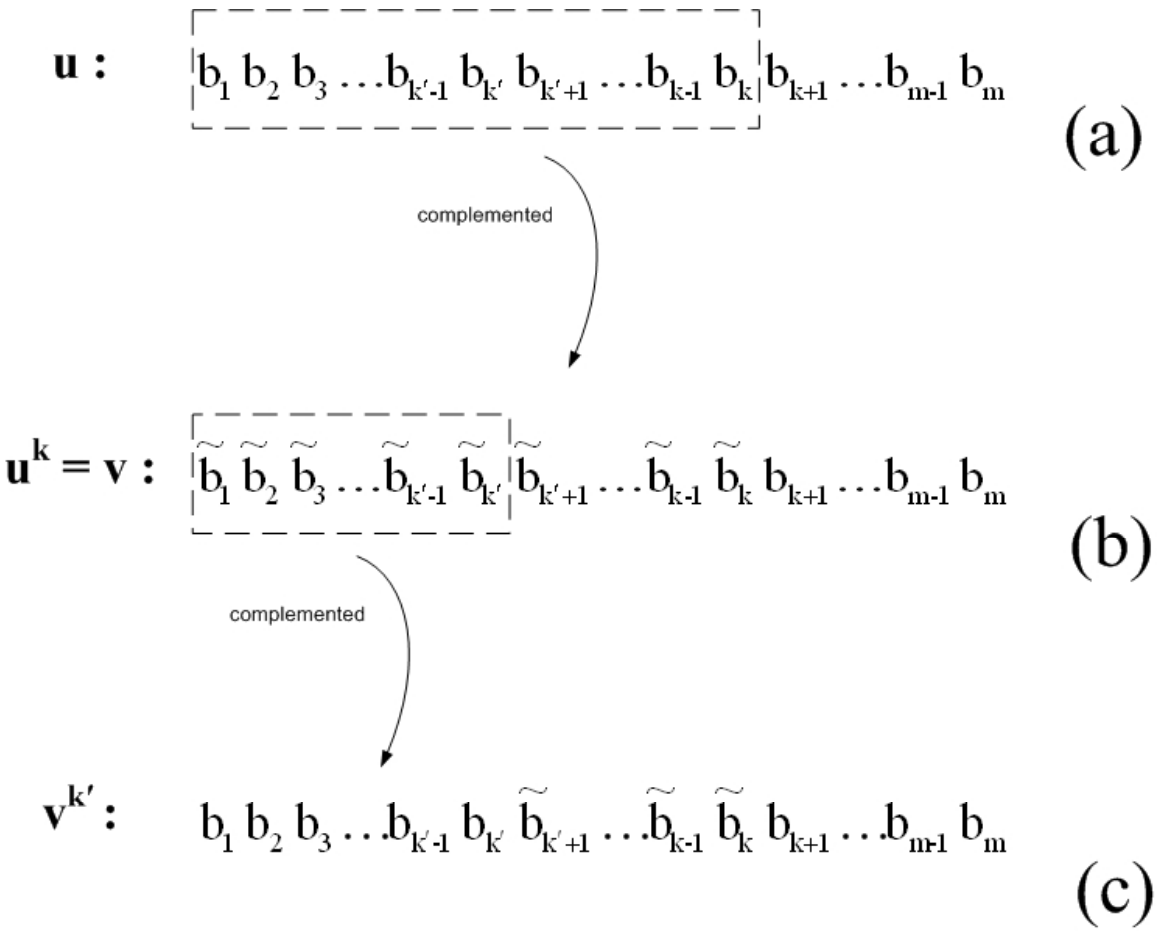


Figure 3.1: An illustration of bit flipping performed when encoding and decoding using Knuth Algorithm. It shows an encoding and an assumed wrong decoding.

index for which the original message could be balanced. Figure 3.1 also illustrates the relation between $\mathbf{u}$ and $\mathbf{v}^{k'}$, as well as the relation between their substrings. (a) is the original word which is the same as the correctly decoded word. (b) is the balanced word generated by flipping the first k bits of the original message word. (c) is the result of an assumed wrong decoding which is equal to $\mathbf{u}$ except for the portion from the $(k'+1)$-th bit to the k-th bit, which is complemented with respect to $\mathbf{u}$.

Knuth used a little bit more than $\log_2(m)$ bits for the auxiliary data. By looking at the total number of balanced words of m bits and the possible user words, we see that the optimum auxiliary data needed is in fact less than $\log_2(m)$ because with m bits, there are 2^m user words, and $\binom{m}{\frac{m}{2}}$ balanced words. Sterling's approximation tells us that

$$\binom{m}{\frac{m}{2}} = \frac{2^m}{\sqrt{\frac{m\cdot\pi}{2}}}, \qquad \text{for } m \gg 1. \tag{3.11}$$

Therefore, the optimum amount of auxiliary data needed is [9]

$$\log_2\left(\frac{2^m}{\frac{2^m}{\sqrt{\frac{m\cdot\pi}{2}}}}\right) = \cdots = \frac{1}{2}\log_2(m) + 0.326. \tag{3.12}$$

3.2 Analysis of the Knuth algorithm when applied to compressed data

Having to deal with only almost balanced input words greatly reduces the cardinality of the input space of the encoder. For this reason the size of the auxiliary data is expected to be reduced. In the case of the bit flipping method described in the previous section, this refers to the bits needed to encode the number of flipped bits, k. However, using the original Knuth method does not necessarily improve the number of auxiliary data bits. To show this, some examples balancing almost balanced sequences with 8 bits are given below:

$$\begin{aligned}
&11100000 \longrightarrow 00011110 \text{ (7 bits flipped)}\\
&11000001 \longrightarrow 00111001 \text{ (5 bits flipped)}\\
&10000011 \longrightarrow 01100011 \text{ (3 bits flipped)}\\
&00000111 \longrightarrow 10000111 \text{ (1 bit flipped).}
\end{aligned}$$

These almost balanced words have weight $8/2 - 1 = 3$. Each of the four words in this example are balanced using Knuth method (bit flipping). The required number

of bit flips (k) is given as well. As can be seen, k can vary from a minimum of 1 to a maximum of 7 (in fact, it can take all odd values)[1]. However, we can use Knuth's method but encode the amount of (original) imbalance as auxiliary data. Note that assuming input words are all equiprobable then encoding either the bit position (original Knuth's) or the weight won't make any difference in the performance (in terms of the number bits that have to be added to balance the word). However, for a set of almost balanced words, as the histograms in Figures 2.5, 2.7, and 2.8 illustrate, it can reduce the number of auxiliary bits.

When $weight(\mathbf{u}) \in [w_{min}, w_{max}]$ where w_{min} and w_{max} are very large numbers, and $w_{max} - w_{min}$ is much smaller (at least a couple orders of magnitude), it is logical to use the form of $weight(u) = B + i$ or "*base + index*". B is known to decoder and i is encoded and sent for each message word. Obviously with a proper choice of B, i is in the range $0 \leq |i| \leq w_{max} - w_{min}$. We can use information on the distribution to come up with a variable-length encoding scheme for the *index* and increase the performance in terms of the amount of auxiliary data sent. However a simpler and less complex fixed-length encoding scheme with much lower delays in the encoder/decoder is enough to show the large improvement that results from applying compression. For instance, by using the histogram of Figure 2.7 as the weight distribution we can do the following straightforward calculations for the auxiliary data. If we choose $B = m/2 = 2^{19}$ as the *base*, corresponding to the 50% point on Figure 2.7, we need

$$\lceil \log_2(2 \times \max\{B - w_{min}, w_{max} - B\}) \rceil = 16 \tag{3.13}$$

bits to encode the index i with a fixed length code.

As an average length (which can be realized with a variable-length scheme) however, we will have

$$\begin{aligned} \lceil E\{B - weight(\mathbf{u})\} \rceil &= \left\lceil \frac{\sum_i \log_2(2|2^{19} - weight(\mathbf{u}_i)|)}{n} \right\rceil \\ &= \left\lceil \sum_{j=w_{min}}^{w_{max}} P_u(j) \log_2(2|2^{19} - j|) \right\rceil \\ &= 14 \end{aligned} \tag{3.14}$$

[1]It is easy to verify that by grouping zeroes and ones next to each other, we can easily make an m-bit word of weight $m/2 + 1$ or $m/2 - 1$ that needs exactly 1 bit flip, $m - 1$ bit flip, or any odd number in between of the two, to be balanced. See appendix A for examples.

where n is the total number of input words (223 in the case of Figure 2.7), the upper sum is over all the input words $\mathbf{u}_i$, and $P_u(j)$ is the probability distribution function of the message word $\mathbf{u}$ having weight equal to j which is obtained from the histogram. This means if a variable length encoding is used on average 14 bits will be needed to encode i.

If we choose the *base* to be the 50% point on the normalized scale, we don't need to send B separately, but if we choose a better value for B we will need fewer bits less to encode i. Using the average weight

$$B = \overline{w} = E\{weight(\mathbf{u})\} \tag{3.15}$$

is a good choice. This leads to

$$\lceil \log_2(2 \times \max\{\overline{w} - w_{min}, w_{max} - \overline{w}\}) \rceil = 16 \tag{3.16}$$

and

$$\left\lceil \sum_{j=w_{min}}^{w_{max}} P_u(j) \log_2(2|\overline{w} - j|) \right\rceil = 12. \tag{3.17}$$

Here (3.16) is the number of bits needed if we encode i with a fixed length code, and (3.17) is the average number of bits needed if a variable length code is used.

Another good choice is to use w_{min} and w_{max}, in which case, $B = w_{min}$ and consequently

$$\lceil \log_2(2 \times (w_{max} - w_{min}) \rceil = 16 \tag{3.18}$$

and

$$\left\lceil \sum_{j=w_{min}}^{w_{max}} P_u(j) \log_2(2|w_{min} - j|) \right\rceil = 14. \tag{3.19}$$

Similar to the previous two cases, (3.18) is the number of bits needed to encode i with a fixed length code, and (3.19) is the average number of bits needed to encode i with a variable length code.

Note that the transmitter and receiver can agree on a proper *base* beforehand based on the statistical characteristics of the data as well as the chosen compression algorithm, or the transmitter can just send it once at the very beginning. Given that the number of input words is assumed to be large (n = 223 in the above example), the overhead of sending this extra auxiliary data is negligible.

The above results show, that after compression the number of auxiliary bits needed for the Knuth balancing method is reduced from about $\log_2(2^{20}) = 20$ to 12 or 14 bits, which is a good improvement.

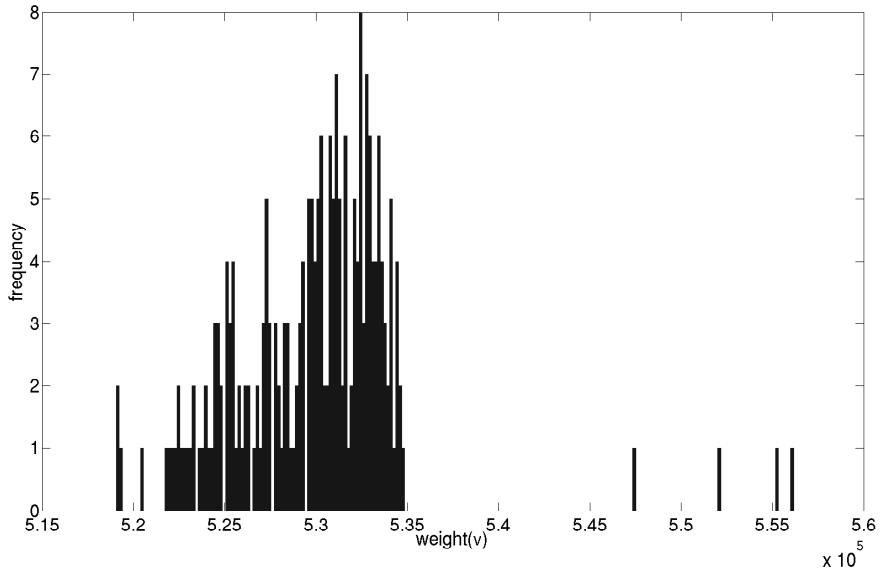


Figure 3.2: The same histogram as is Figure 2.7 but with more bins more bins to better distinguish the weights.

One last thing to be addressed is the tail of weight distribution. As seen in Figures 2.5, 2.7, and 2.8 the histograms have a centralized and focused "bulk" around the average point with a similar shape and consistent general statistical properties but there are a few isolated points located away from the main bulk that comprise the "tail" of the distribution. These points not only reduce, the "similarity" between different test sets which is vital in generalizing the analysis but also in a practical sense force us to use extra auxiliary bits by increasing the $[w_{min}, w_{max}]$ interval. The latter becomes particularly important when we are using fixed length encoding. Denote the minimum and maximum weights of the central bulk of the distribution by w'_{min}, and w'_{max}, respectively, where $w_{min} \leq w'_{min} < w'_{max} \leq w_{max}$. We showed in 3.18 that for the example in Figure 2.7, 16 bits are needed to encode the weight information. However in this example there are 8 points, four on each side, out of the total of 223 points on the histogram (each point corresponds to an input word) that stretch the weight range to twice its size. In other words

$$\lceil \log_2(2 \times (w'_{max} - w'_{min}) \rceil = 15. \tag{3.20}$$

This can be seen in Figure 3.2 which is the same histogram as in Figure 2.7 but with many more bins to better distinguish the weights. The intention is to show the distinction between the central "bulk" which is in the range $[521744, 534801]$ compared to the actual range which is $[519057, 556170]$. There are four point on the right and four points on the left scattered far from the center.

A simple way to resolve this issue is to use a two step encoding scheme. The number of necessary auxiliary bits is chosen based on $[w'_{min}, w'_{max}]$ and a special code word is designated for the very few values of $weight(u)$ that don't fall into the similar interval. This will signal that a special case has occured and the encoder will use more bits to encode the weight information, and so the decoder should expect more bits. Two different fixed lengths will be used for this encoding scheme which is a very small deviation from fixed-length towards variable-length encoding. This helps to provide the benefits (such as simplicity) of fixed-length encoding, reduce the number of auxiliary bits, and most importantly generalize the analysis based on the consistent characteristics of the central bulk of the distribution. Minor variations in the tails of the distribution can then be ignored.

Chapter 4

Applying The Immink-Weber Method to Compressed Data

4.1 An overview of the Immink-Weber method

Immink and Weber in [6] and [13] modified the original Knuth method to achieve optimum performance. They noticed that there are only a limited number of input words that will be mapped to each balanced word using Knuth's algorithm. They used this fact to reduce the number of bits needed for encoding auxiliary data. Since the Knuth algorithm is reversible, for every balanced word the set of user words that will be mapped to it can be generated. Now assuming an (arbitrary) ordering of the words associated with a given balanced word, the only information needed at the decoder is the index of the input word (being sent) in that set. Therefore at the encoder, Knuth's bit flipping is applied until the input word is balanced. Then, the ordered set corresponding to that balanced word is generated. Finally, the encoder locates the index of the input word in that set, encodes this information, and sends it along with the balanced word (encoded input word). At the receiver, a balanced word is received with the auxiliary data. The auxiliary data is decoded to get the index. The inverse of Knuth's method is performed on the balanced word to generate the ordered set. Finally, the user word pointed to by the decoded index is chosen and output.

The method of Immink and Weber is asymptotically optimal in terms of auxiliary data. It is somewhat similar to Knuth's method but with much higher complexity, simply because for every received balanced message the receiver has to generate the

set of possible user words corresponding to it. To avoid using look-up tables, the ordered set can be generated on the fly (for each received word). However, this will be time consuming, especially for large message sizes which result in large sets.

Applying their algorithm to almost balanced input words (in our case applying their algorithm to a compressed source), will certainly change their statistical calculations. This is simply because their assumption that all possible words in the input space (2^m for an input word with m bits) have equal probability of occuring will change. This reduction in the size of the input space of the encoder should result in improved performance similar to that with Knuth's algorithm as shown in the previous chapter.

In order to find exactly how the performance changes, we start by providing some definitions and looking at the Immink and Weber's analytical results. For an m-bit balanced word $\mathbf{v}$, there is a corresponding set, $\sigma_{\mathbf{v}}$, which contains all the m-bit words that can be mapped to $\mathbf{v}$ by a valid Knuth bit flipping. In other words

$$\begin{aligned} &\forall \mathbf{u} \in \sigma_{\mathbf{v}}, \exists k, 1 \leq k \leq m: \\ &\quad \mathbf{u}^k = \mathbf{v} \quad \text{and} \quad \mathbf{v}^k = \mathbf{u} \end{aligned} \tag{4.1}$$

and there is no $k' < k$ such that $\mathbf{u}^{k'}$ is balanced or $\mathbf{v}^{k'} = \mathbf{y}$.

The *running digital sum* (RDS) is defined as

$$z_k = \sum_{i=1}^{k} v_i, \tag{4.2}$$

which is the sum of the first k bits of $\mathbf{v}$, with $1 \leq k \leq m$. Let $Z(\mathbf{v})$ be a set containing all possible (distinct) values of z_j that a balanced word $\mathbf{v}$ might take. Immink and Weber showed that this set has the same size as the corresponding set $\sigma_{\mathbf{v}}$, used for encoding/decoding as explained before. In other words

$$|Z(\mathbf{v})| = |\sigma_{\mathbf{v}}|. \tag{4.3}$$

Since z_j can only change by one as j changes by one, it will take every integer value in the interval $[z_{min}, z_{max}]$ where $z_{min} = \min_{1 \leq j \leq m}\{z_j\}$ and $z_{max} = \max_{1 \leq j \leq m}\{z_j\}$. Therefore we have

$$|\sigma_{\mathbf{v}}| = z_{max} - z_{min} + 1. \tag{4.4}$$

Note that these parameters and expressions are defined for one m-bit *bipolar* word. For simplicity, we will use the "0" character to represent bits with "-1" values.

Obviously we need enough auxiliary data to be able to single out any member of $\sigma_{\mathbf{v}}$ and therefore decode correctly. They derived closed form expression $P(t, m)$ which gives the number of *m-bit* balanced words that have a corresponding set of size $t = z_{max} - z_{min} + 1$. This distribution can be used to find parameters such as the entropy or average number of auxiliary bits needed. Note that the only other parameter needed besides the number of bits in a word is the *span* of the running sum, t.

4.2 Analysis of the Immink-Weber algorithm when applied to compressed data

We need to know how many almost balanced input words can legally be mapped to any given balanced word, $\mathbf{v}$. As a more formal definition, let the input space be restricted to m-bit words with weights close to the balanced weight or

$$w_{min} \leq weight(\mathbf{u}) \leq w_{max}. \tag{4.5}$$

For simplicity, assume the allowed weight interval is symmetric around the balanced weight, i.e.

$$\begin{aligned} w_{min} &= \frac{m}{2} - K \\ w_{max} &= \frac{m}{2} + K \end{aligned} \tag{4.6}$$

or

$$|weight(\mathbf{u}) - \frac{m}{2}| \leq K \tag{4.7}$$

and K is the allowed imbalance, which is a positive integer.

With the above criterion, members of the corresponding ordered sets, $\mathbf{u} \in \sigma_{\mathbf{v}}$, with $|weight(\mathbf{u}) - \frac{m}{2}| > K$, will no longer exist. It is not hard to verify that for any word $\mathbf{u} \in \sigma_{\mathbf{v}}$, we have

$$weight(\mathbf{u}) = z_k + \frac{m}{2} \tag{4.8}$$

which restates the one to one mapping between members of $\sigma_{\mathbf{v}}$ and the possible values

of z_j for $\mathbf{v}$. As a result

$$|z_k - \frac{m}{2}| \leq K, \tag{4.9}$$

and therefore the only members of $\sigma_{\mathbf{v}}$ that remain are those that correspond to such z_k. Then the words corresponding to

$$z_k \in [\max\{z_{min}, \frac{m}{2} - K\}, \min\{z_{max}, \frac{m}{2}\}] \tag{4.10}$$

(and only such words) will remain and the cardinality of the corresponding ordered set is

$$|\sigma'_{\mathbf{v}}| = \min\{z_{max}, \frac{m}{2} + K\} - \max\{z_{min}, \frac{m}{2} - K\} + 1, \tag{4.11}$$

where $\sigma'_{\mathbf{v}}$ is defined the same as $\sigma_{\mathbf{v}}$ but with the restricted weight criterion.

The $P(t, m)$ distribution is insufficient for our purposes here as we need a more detailed distribution based on both z_{min} and z_{max}, or $P'(z_{max}, z_{min}, m)$.

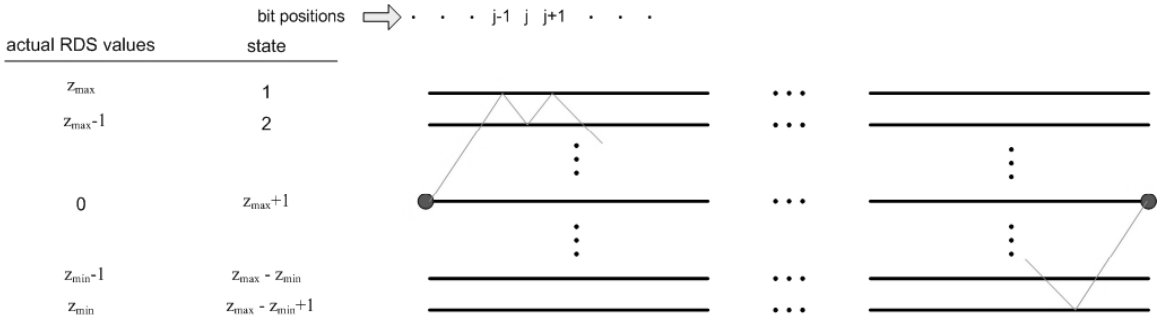


Figure 4.1: Trellis representation of a balanced bit sequence with the RDS as states.

4.3 Deriving the distribution of the size of the corresponding sets

In order to derive expressions for $P'(z_{max}, z_{min}, m)$ the results of [2] are helpful. This gives the number of bipolar sequences of a given length with an RDS within an interval, $[N_1, N_2]$, where N_1 and N_2 are integers. A more general definition for RDS is

$$z_k = z_0 + \sum_{i=1}^{k} u_i, \tag{4.12}$$

where z_0 is the initial value of the RDS. This can be explained by considering an arbitrary m-bit word as a subsequence of a longer sequence rather than a stand-alone sequence. Defining a state for each allowable RDS value, there will be

$$N = N_2 - N_1 + 1 \tag{4.13}$$

states (at any bit position in the sequence, or based on z_j for any $0 \leq j \leq m$), which we call s_1 through s_N. The transition matrix D_N shows the possible transitions between states moving from a given bit position to the next. The matrix dimensions are $N \times N$ and $D_N(i,j) = 1$ if and only if a transition from s_i to s_j is possible, otherwise $D_N(i,j) = 0$. Since from a given bit position the RDS can either increase by one or decrease by one, D_N is an all zero matrix except for the superdiagonal and subdiagonal which are all one. It follows that $D_N^m(i,j) = [D_N(i,j)]^m$ is the number of bit sequences of length m which start at s_i and end at s_j and their RDS remains in an interval (any interval) of size N.

Figure 4.1 shows the transitions between states in the form of a trellis. Solid black lines are constant-state lines for different bit positions. On the left side we can see the relation between the actual RDS values and the state indices. A (valid) path on such a trellis (similar to the one shown in Figure 4.1), corresponds to one specific binary word or sequence and a valid path must have a transition in its state or a jump between the solid horizontal lines after each bit. Note that only a jump to a line immediately above or below the current line is possible per bit. The path shown in Figure 4.1 starts at $s_{z_{max}+1}$ and ends at the same state. This represents an m-bit word ($z_0 = 0$) that has $z_m = 0$, which means it is balanced. Further its RDS remains within the interval $[z_{min}, z_{max}]$, and also at least at one bit position takes the z_{min} value and the z_{max} value at another. This is exactly the type of sequence (path) we are interested in counting and this number is defined as $P'(z_{max}, z_{min}, m)$.

Immink and Weber used D_N^m to derive

$$P(t,m) = \sum_{i=1}^{t} D_t^m(i,i) - 2\sum_{i=1}^{t-1} D_{t-1}^m(i,i) + \sum_{i=1}^{t-2} D_{t-2}^m(i,i) \tag{4.14}$$

For arbitrary z_{max}, z_{min}, and m,

$$D_{z_{max}-z_{min}+1}^m(z_{max}+1, z_{max}+1) \tag{4.15}$$

gives the number of m-bit words that are balanced and their RDS remain in the range $[z_{min}, z_{max}]$ (not necessarily taking z_{max} or z_{min} values) and

$$\begin{aligned} \max_{1\le j\le m}\{z_j\} &\le z_{max} \\ \min_{1\le j\le m}\{z_j\} &\ge z_{min}. \end{aligned} \tag{4.16}$$

In terms of the trellis shown in Figure 4.1, the path remains within s_1 (the bottom most line) and s_N (the top most line) but not necessarily meeting these lines. The number of paths that meet these two bounds (lines) is given by

$$\begin{aligned} &P'(z_{max}, z_{min}, m) \\ &= D^m_{z_{max}-z_{min}+1}(z_{max}+1, z_{max}+1) && (a) \\ &- D^m_{z_{max}-z_{min}}(z_{max}+1, z_{max}+1) && (b) \\ &- D^m_{z_{max}-z_{min}}(z_{max}, z_{max}) && (c) \\ &+ D^m_{z_{max}-z_{min}-1}(z_{max}, z_{max}). && (d) \end{aligned} \tag{4.17}$$

Here, we use the *inclusion-exclusion principle*. The first term, (a), is the same as (4.15). The next two terms, (b) and (c), excludes the paths that meet the top bound but not the bottom one, and the paths that meet the bottom bound but not the top one, respectively. The last term, (d), includes the paths that do not meet either the top or bottom bounds. Such paths are excluded twice by the two preceding negative terms, (b) and (c).

Looking at the definition of $D^m_N(i,j)$ and Figures 4.2-4.4 helps to understand how each term in (4.17) counts the paths described above. (b) is the number of paths that meet the top bound (s_1 is visited at least once or z_j is equal to z_{max} at least for one $1 \le j \le m$), but not the bottom bound. These paths can be represented by a trellis with one state less and the states will be renamed as shown in Figure 4.2. Therefore the starting state and ending state for the sequences will remain $s_{z_{max}+1}$. (c) is the number of paths that meet the bottom bound ($s_{z_{max}-z_{min}+1}$ is visited at least once or z_j is equal to z_{min} at least for one $1 \le j \le m$) but not the top bound. Similar to the previous case we can represent these paths with a trellis with one state less but the changes in the state indices, as shown in Figure 4.3, will cause the starting and ending states to become $s_{z_{max}}$ instead of $s_{z_{max}+1}$. Finally the last term, (d), is the number of paths that do not meet either the top or bottom bounds (neither of

$s_{z_{max}-z_{min}+1}$ and s_1 are visited or z_j remains in the range $[z_{min}+1, z_{max}-1]$ for all $1 \leq j \leq m$). These paths are represented by a trellis with two less states and, as can be seen in Figure 4.4, the change in indices results in the starting and ending state being $s_{z_{max}}$.

Note that the m-bit words in this discussion are balanced, $z_m = 0$, and necessarily $0 \in [z_{min}, z_{max}]$ or $z_{min} \leq 0 \leq z_{max}$. This means z_{min} cannot be larger than 0 and z_{max} cannot be smaller than zero. Therefore for only two cases, namely $z_{min} = 0$ and $z_{max} = 0$ ($z_{min} = z_{max} = 0$ is trivial), (4.17) cannot be used. In those cases, it is straightforward to verify that

$$\begin{aligned} P'(z_{max}, 0, m) = {} & D^m_{z_{max}+1}(z_{max}+1, z_{max}+1) \\ & - D^m_{z_{max}}(z_{max}, z_{max}) \end{aligned} \tag{4.18}$$

and

$$\begin{aligned} P'(0, z_{min}, m) = {} & D^m_{-z_{min}+1}(1,1) \\ & - D^m_{-z_{min}}(1,1). \end{aligned} \tag{4.19}$$

Figure 4.5 will be helpful in that regard.

It is worth noting that $P'(z_{max}, z_{min}, m)$ is a more general distribution than $P(t, m)$ and they are related as

$$\begin{aligned} P(t,m) &= \sum_{\substack{i-j+1=t \\ i,j\in\mathbb{Z} \\ i\geq 0, j\leq 0}} P'(i,j,m) \\ &= \sum_{j=1-t}^{0} P'(t-1+j, j, m). \end{aligned} \tag{4.20}$$

4.4 Using P' to derive the performance

We can use P' to find the average size of the corresponding sets, $|\sigma'_{\mathbf{v}}|$. Remember that $P'(i,j,m)$ is the number of binary sequences of length m, for which the RDS remains between i and j,and the sequences to be balanced are restricted to imbalance of K. Therefore the size of the corresponding set $|\sigma'_{\mathbf{v}}|$ is reduced from that of $|\sigma_{\mathbf{v}}|$. Using

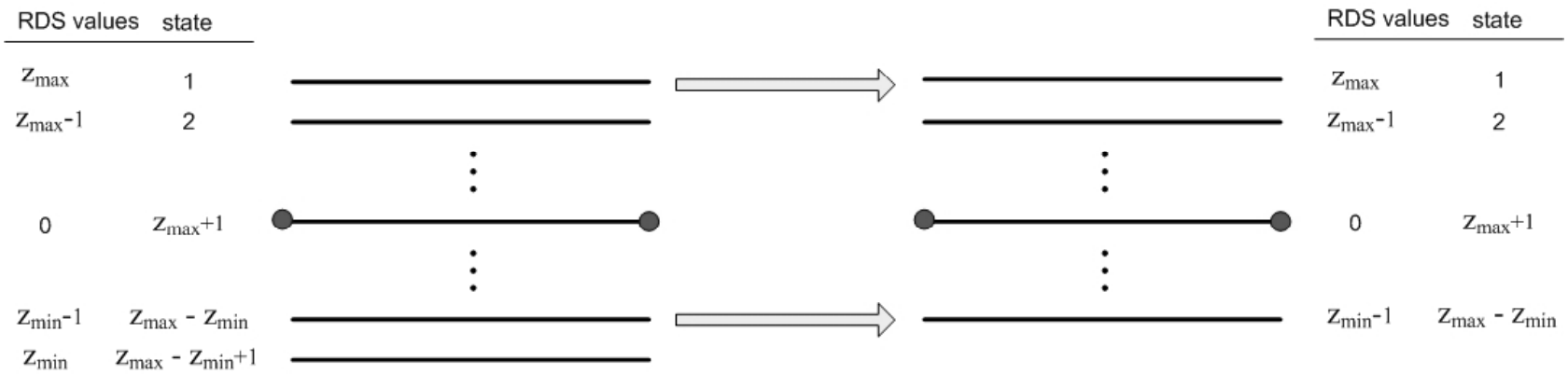


Figure 4.2: This figure illustrates the equivalance between the of number of paths that remain within N states in an $N+1$ state trellis and the number of paths in an N state trellis.

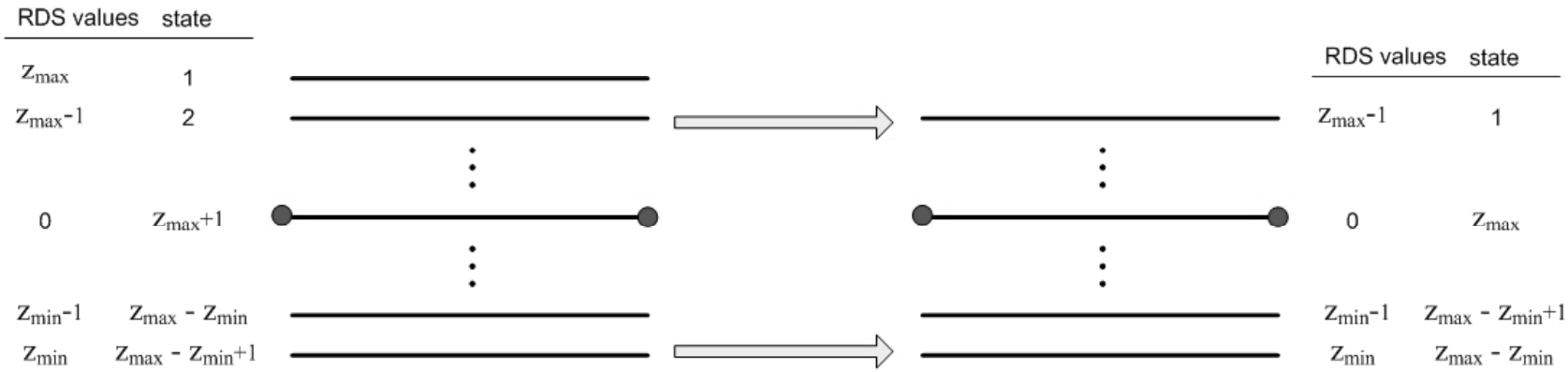


Figure 4.3: Trellis representation of the balanced bit sequence with RDS as states.

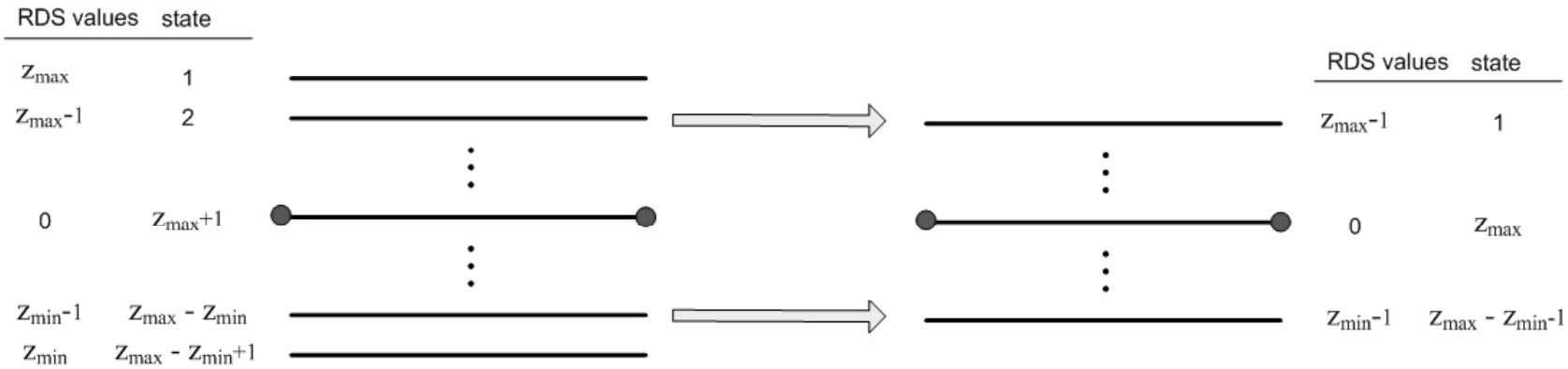


Figure 4.4: Trellis representation of the balanced bit sequence with RDS as states.

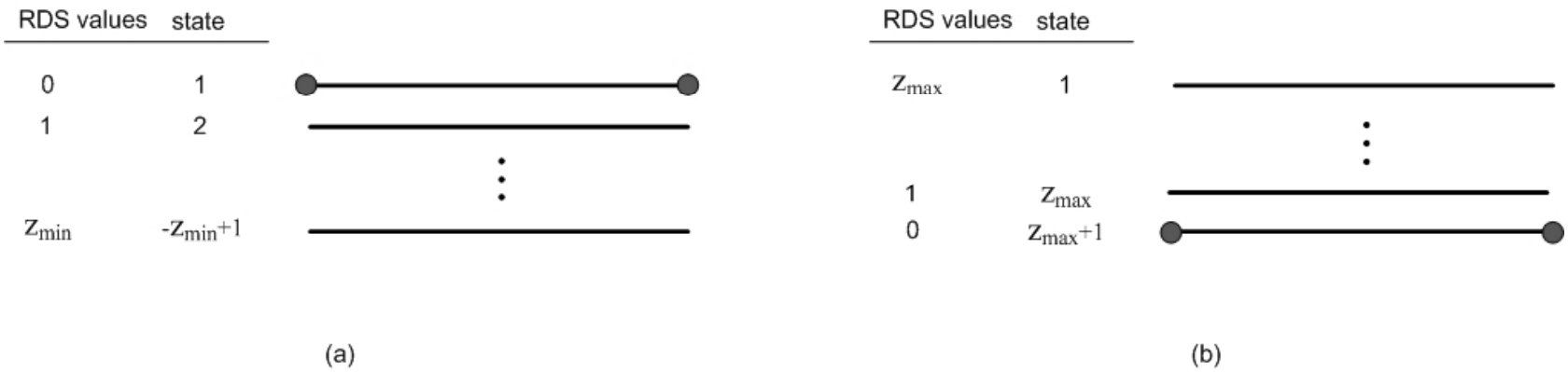


Figure 4.5: Trellis representation for special cases of (a) $z_{max} = 0$ and (b) $z_{min} = 0$.

(4.11), and substituting i and j we have

$$\begin{aligned}|\sigma'_{\mathbf{v}}| &= [\min\{z_{max}, \frac{m}{2}+K\} - \max\{z_{min}, \frac{m}{2}-K\} + 1] \\ &= [\min\{\frac{m}{2}+i, \frac{m}{2}+K\} - \max\{\frac{m}{2}+j, \frac{m}{2}-K\} + 1] \\ &= [\frac{m}{2} + \min\{i, K\} - \frac{m}{2} - \max\{j, -K\} + 1] \\ &= [\min\{i, K\} - \max\{j, -K\} + 1]. \end{aligned} \tag{4.21}$$

To get the expected size, we sum over all RDS spans and for each specific span value (t), we sum over all (i, j) pairs which produce such a span ($i - j + 1 = t$), which gives

$$\begin{aligned}\underset{\mathbf{v}}{E}\{|\sigma'_{\mathbf{v}}|\} &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{\substack{i-j+1=t \\ i,j\in\mathbb{Z} \\ i\geq 0, j\leq 0}} \Big[\min\{K, i\} - \max\{-K, j\} + 1\Big] P'(i, j, m) \\ &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{j=1-t}^{0} \Big[\min\{K, t-1+j\} - \max\{-K, j\} + 1\Big] P'(t-1+j, j, m). \end{aligned} \tag{4.22}$$

In order to compare our results with those in [6] and [9] we need to determine how many bits are needed to encode the auxiliary data. As we mentioned before, we need to identify members of the corresponding ordered set $\sigma'_{\mathbf{v}}$ at the decoder. Let H' denote the average number of bits needed to encode the auxiliary data (average prefix size). In order to simplify the expression for H' we define

$$t'(i, j) = \min\{K, i\} - \max\{-K, j\} + 1. \tag{4.23}$$

Using t' we can write

$$\begin{aligned}H' &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{\substack{i-j+1=t \\ i,j\in\mathbb{Z} \\ i\geq 0, j\leq 0}} u'(i, j) P'(i, j, m) \log_2(t'(i, j)) \\ &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{j=1-t}^{0} t'(t-1+j, j) P'(t-1+j, j, m) \log_2(t'(t-1+j, j)). \end{aligned} \tag{4.24}$$

Here, as in (4.22), t' is the number of input words that are in $\sigma'_{\mathbf{v}}$ where the RDS of $\mathbf{v}$ remains between j, and i. Therefore we need $\log_2(t')$ bits of auxiliary data. There are $P'(i,j,m)$ such balanced words $\mathbf{v}$, and are $t'(i,j)$ input words mapped to each word. Thus a total of $t'(i,j)P'(i,j,m)$ input words need $\log_2(t'(i,j))$ bits. Also similar to (4.22), the sums over possible RDS spans and (i,j) pairs are needed.

H', given in (4.24), is the average prefix size needed to encode the auxiliary data or the average auxiliary data per input word. $\log_2(t'(i,j))$ in H' however neglects the fact that the prefix used itself must be balanced or else the result of concatenating the balanced word ($\mathbf{v}$) and an unbalanced prefix will be unbalanced (although the imbalance will most probably be much much smaller than the original imbalance because the prefix size is very small itself). In practice, to have a completely balanced code word, we need to encode the auxiliary data into a *balanced* prefix. Define the integer function $q = B(p)$ for even positive integers p, as the smallest number of bits to allow a mapping from p different elements into p distinct balanced prefixes

$$\binom{q}{\frac{q}{2}} \geq p.$$

This is because the set $\sigma'_{\mathbf{v}}$ has $p = t'(i,j)$ elements, and each of the p different indices has to be represented by a balanced prefix. Provisioning $q = B(p)$ bits for the prefixes guarantees that a balanced prefix can be found for each index.

Using $B(p)$, we can modify (4.24) to determine the average prefix size needed to encode the auxiliary data using a variable length scheme

$$\begin{aligned}\hat{H}' &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{\substack{i-j+1=t \\ i,j\in\mathbb{Z} \\ i\geq 0, j\leq 0}} t'(i,j)P'(i,j,m)B(t'(i,j)) \\ &= 2^{-m} \sum_{t=2}^{\frac{m}{2}+1} \sum_{j=1-t}^{0} t'(t-1+j,j)P'(t-1+j,j,m)B(t'(t-1+j,j)). \end{aligned} \tag{4.25}$$

Now we can compare the auxiliary data needed for balancing almost balanced words with the auxiliary data needed for balancing sequences generated from an arbitrary source. The following expression gives the average prefix size needed for an arbitrary source

$$H = 2^{-m} \sum_{u=2}^{\frac{m}{2}} tP(t,m)\log_2(t), \tag{4.26}$$

and for the average prefix size when the prefixes are also balanced, we have

$$\hat{H} = 2^{-m} \sum_{t=2}^{\frac{m}{2}} tP(t,m)B(t). \tag{4.27}$$

Note that (4.26) and (4.27) are analogous to (4.24) and (4.25), respectively. t and $P(t,m)$ are used instead of t' and $P'(i,j,m)$ and the sum is over all possible SDR spans. Figure 4.6 shows the results of computing (4.24), (4.25), (4.26), and (4.27) for various input sequence sizes m. These are H', $\hat{H}'$, H, and $\hat{H}$ respectively. The $\log_2(m)$ line is provided as a reference. The imbalanced allowed, K, is set to 5% of m, which means no auxiliary bits are added for $\log_2(m) = 4$ and smaller.

To analyze the performance of balancing methods in terms of auxiliary data added or the (average) number of bits used for prefixes, (4.26), and (4.27) can be used as performance measures for the Immink-Weber construction. On the other hand, (4.24) and (4.25), can be used as performance measures for combining compression and the Immink-Weber construction. In this way, one can consider the performance of the Immink-Weber construction when applied to almost balanced sources, or the combination of techniques as a general approach to balancing arbitrary sources. For this reason, the terms in (4.22), (4.24), and (4.25) are multiplied by 2^{-m}. In other words $P'(i,j,m)/2^m$ is used as the probability of having a corresponding ordered set $(\sigma'_{\mathbf{v}})$ of size $t'(i,j)$. Also, comparing the two sets of curves in Figure 4.6 is a result of considering this combination of techniques as a balanced encoder over the input space of size 2^m.

According to the plot for the combined method, when unbalanced auxiliary data is used, more than two bits improvement is obtained for different word sizes over the original Immink-Weber construction (comparing H and H'). When the auxiliary data is also forced to be balanced, the combined method achieves a three bit improvement for different word sizes over the original Immink-Weber construction (comparing $\hat{H}$ and $\hat{H}'$).

4.7 depicts block diagrams to compare the two discribed approaches to balancing. (a) is a system with an ordinary balanced encoder. (b) is a system in which compression is combined with the original balanced encoder to both reduce the volume of data and improve the code rate of the balanced encoding. Therefore, the two blocks together can be considered as a new balanced encoder

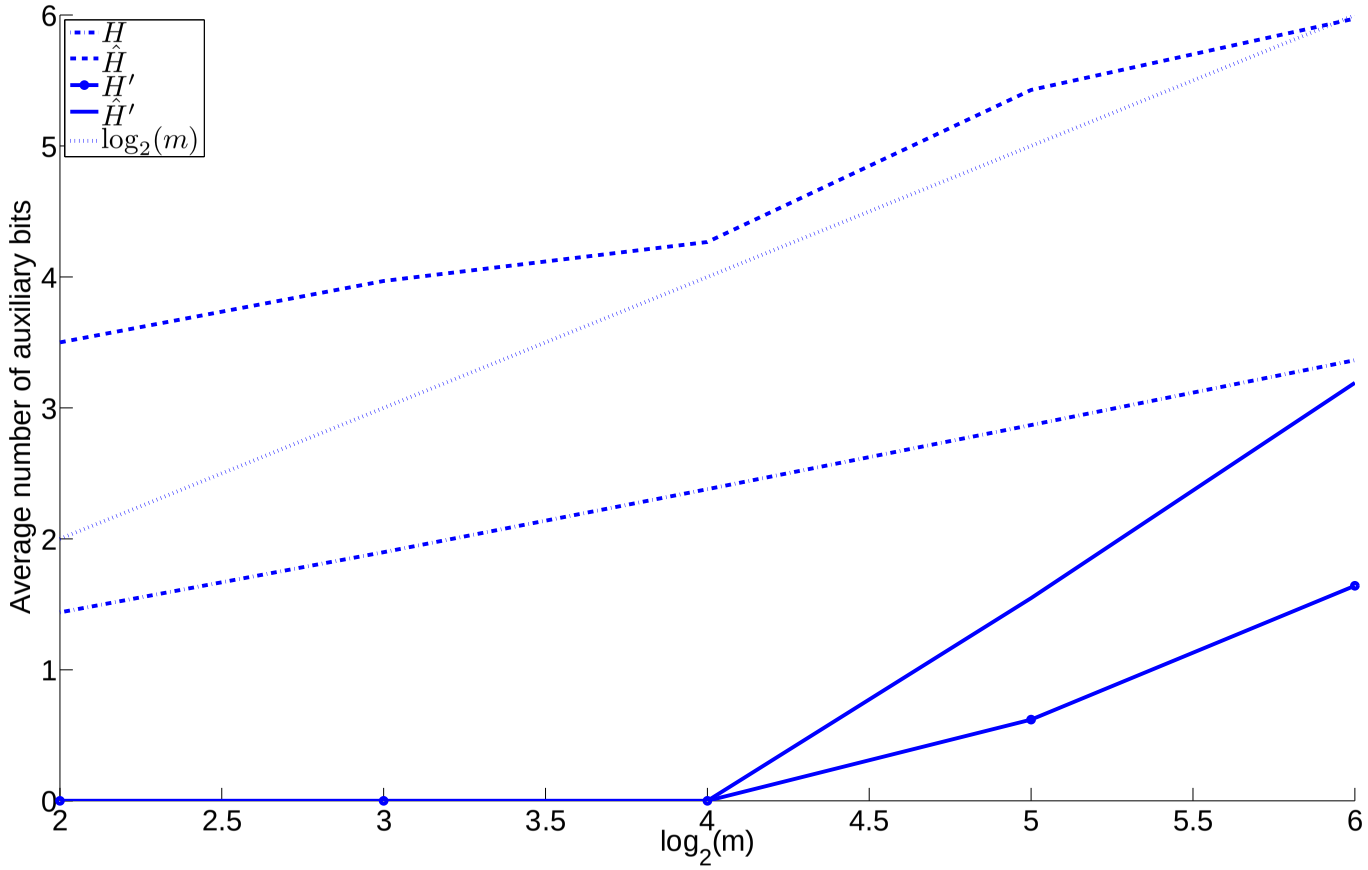


Figure 4.6: Average prefix length as a function of $\log_2(m)$ which can be realized using variable length encoding.

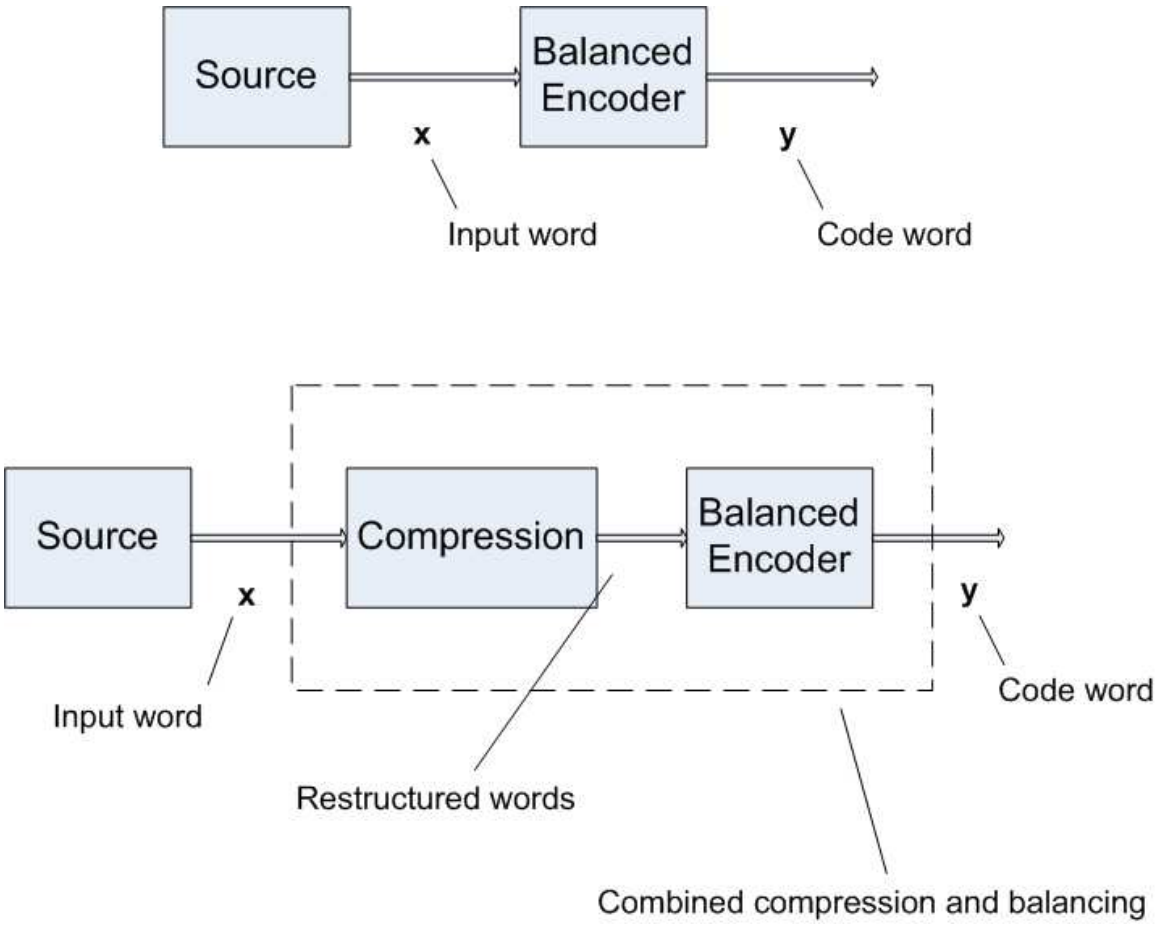


Figure 4.7: Block diagrams of systems with balanced coding, with both ordinary balanced encoder and combined compression and balanced coding.

Chapter 5

Conclusions and Future Work

5.1 Contributions

In this thesis, we focused on balanced coding and the effects of combining source compression and balanced coding (with special interest in holographic storage systems). The contributions of this thesis are summarized as follows. We examined the effect of compression on a generic binary memoryless source in terms of the disparity or imbalance of the output sequence. Weight distribution of standard test data before and after compression using available compression tools, was presented as empirical analysis of how source compression affects the disparity or imbalance of a sequence. It was shown that after compression, the average weight of the output words will be very close to balanced and their deviation about the average will be smaller. We analyzed the Knuth balancing method when applied to compressed data. Expressions were derived for the expected number of auxiliary bits used in balancing and this number was examined based on standard test data. It was shown that the number of auxiliary bits needed decreases with respect to when the algorithm is applied to uncompressed data. The Immink-Weber balancing method was analyzed as a realization of the ideal Knuth method where the number of auxiliary bits are reduced (asymptotically) to the theoretical minimum. Expressions were derived for the performance when applied to almost balanced inputs. Further, plots of the theoretical performance expressions were presented to better visualize the improvements when balancing algorithms are combined with compression.

5.2 Future work

With the promising results observed in this work, it will be interesting to further study joint source coding and balanced coding. A more general approach can be taken to model source coding statistically in terms of the weight distribution of the output sequence (words) or other parameters and source models. A source with memory or correlation between its output symbols, as well as models for the compression process and how it affects the statistics of the output weight, can be examined. These results can be used instead of the generic equiprobable model employed here to analyze the balancing methods and designing codes [9] and [13].

The study of compression algorithms in terms of the compression performance versus output disparity may prove to be very interesting. Modifying known compression algorithms to enhance their output weight distribution (to be more balanced) can also be considered. Avoiding an increase in the expected output length should be possible. For example, a fraction of the expected code length can be sacrificed in return for a lower expected disparity.

Bibliography

[1] J. J. Ashley, M. Blaum, and B. H. Marcus. Report on Coding Techniques for Holographic Storage. Technical report, IBM Research Division, 650 Harry Road, San Jose, CA 95120, USA, March 25 1996.

[2] T. M. Chien. Upper Bound on the Efficiency of DC-constrained Codes. *Bell System Technical Journal*, 49:2267–2287, November 1970.

[3] Sebastian Deorowicz. Silesia compression corpus. `http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia`. last accessed November 3, 2011.

[4] J. F. Heanue, M. C. Bashaw, and L. Hesselink. Channel Codes for Digital Holographic Data Storage. *Journal of Optical Society of America A*, 12(11), November 1995.

[5] L. Hesselink, S. S. Orlov, and M. C. Bashaw. Holographic Data Storage Systems. *Proceeding of the IEEE*, 92(8), August 2004.

[6] K. A. S. Immink and J. H. Weber. Simple Balanced Codes that Approach Capacity. In *ISIT*, Seoul, Korea, June-July 2009.

[7] K. A. Schouhamer Immink. *Codes for Mass Data Storage Systems*. Shannon Foundation Publishers, 1999.

[8] Internet Engineering Task Force (IETF). Request for Comment (RFC) 2616. `http://tools.ietf.org/html/rfc2616`.

[9] D. E. Knuth. Efficient Balanced Codes. *IEEE Transactions on Information Theory*, IT-32(1), January 1986.

[10] Jean loup Gailly and Mark Adler. The gzip homepage. `http://www.gzip.org`, July 2003.

[11] D. Psaltis and G. W. Burr. Holographic data storage. *Computer*, 31(2):52–60, February 1998.

[12] D. Psaltis and F. Mok. Holographic Memories. *Scientific American*, 273(5):70–77, November 1995.

[13] J. H. Weber and K. A. S. Immink. Knuth's Balanced Code Revisited. *IEEE Transactions on Information Theory*, 56(4), April 2010.

Appendix A

Almost balanced words that need certain bit flips to become balanced

This appendix presents a method to construct very close to balanced m-bit words (weight equal $m/2-1$ or $m/2+1$) for any given odd number of bit flips to become balanced.

$weight(\mathbf{u})$	m-bit word	description
$m/2-1$	$\mathbf{u} = \underbrace{00\cdots0}_{m/2+1}\underbrace{11\cdots1}_{m/2-1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k=1$ $weight(\mathbf{u}) = m/2-1$ flip the first bit
	$\mathbf{u} = \underbrace{11\cdots1}_{j}\underbrace{00\cdots0}_{m/2+1}\underbrace{11\cdots1}_{m/2-j-1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k : odd \rightarrow 2j+1$ $weight(\mathbf{u}) = m/2-1$ flip the first k bits
	$\mathbf{u} = \underbrace{11\cdots1}_{m/2-1}\underbrace{00\cdots0}_{m/2+1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k=m-1$ $weight(\mathbf{u}) = m/2-1$ flip every bit except the very last
$m/2+1$	$\mathbf{u} = \underbrace{11\cdots1}_{m/2+1}\underbrace{00\cdots0}_{m/2-1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k=1$ $weight(\mathbf{u}) = m/2+1$ flip the first bit
	$\mathbf{u} = \underbrace{00\cdots0}_{j}\underbrace{11\cdots1}_{m/2+1}\underbrace{00\cdots0}_{m/2-j-1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k : odd \rightarrow 2j+1$ $weight(\mathbf{u}) = m/2+1$ flip the first k bits
	$\mathbf{u} = \underbrace{00\cdots0}_{m/2-1}\underbrace{11\cdots1}_{m/2+1}$	$\mathbf{u} \rightarrow m$-bit binary word, $k=m-1$ $weight(\mathbf{u}) = m/2+1$ flip every bit except the very last

Appendix B

Balancing 6-bit words

An example of a mapping to balance 6-bit words. The tabulated words are divided into two groups. The first consists of almost balanced for which the weights are between $6/2 + 1 = 4$ and $6/2 - 1 = 2$. The second is called very unbalanced words with weights less than 2 or more than 4.

000011 $\longrightarrow$ 100011	111100 $\longrightarrow$ 011100	111000 $\longrightarrow$ 111000	000111 $\longrightarrow$ 000111
000101 $\longrightarrow$ 100101	111010 $\longrightarrow$ 011010	110100 $\longrightarrow$ 110100	001011 $\longrightarrow$ 001011
000110 $\longrightarrow$ 100110	111001 $\longrightarrow$ 011001	110010 $\longrightarrow$ 110010	001101 $\longrightarrow$ 001101
001001 $\longrightarrow$ 101001	110110 $\longrightarrow$ 010110	110001 $\longrightarrow$ 110001	001110 $\longrightarrow$ 001110
001010 $\longrightarrow$ 101010	110101 $\longrightarrow$ 010101	101100 $\longrightarrow$ 101100	010010 $\longrightarrow$ 010010
001100 $\longrightarrow$ 101100	110011 $\longrightarrow$ 010011	101010 $\longrightarrow$ 101010	010101 $\longrightarrow$ 010101
010001 $\longrightarrow$ 110001	101110 $\longrightarrow$ 001110	101001 $\longrightarrow$ 101001	010110 $\longrightarrow$ 010110
010010 $\longrightarrow$ 110010	101101 $\longrightarrow$ 001101	100110 $\longrightarrow$ 100110	011001 $\longrightarrow$ 011001
010100 $\longrightarrow$ 110100	101011 $\longrightarrow$ 001011	100101 $\longrightarrow$ 100101	011010 $\longrightarrow$ 011010
011000 $\longrightarrow$ 111000	100111 $\longrightarrow$ 000111	100011 $\longrightarrow$ 100011	011100 $\longrightarrow$ 011100
100001 $\longrightarrow$ 011001	011110 $\longrightarrow$ 100110		
100010 $\longrightarrow$ 011010	011101 $\longrightarrow$ 100101		
100100 $\longrightarrow$ 011100	011011 $\longrightarrow$ 100011		
101000 $\longrightarrow$ 010110	010111 $\longrightarrow$ 101001		
110000 $\longrightarrow$ 001110	001111 $\longrightarrow$ 110001		

Table B.1: The almost balanced 6-bit words.

$000000 \longrightarrow 111000$	$111111 \longrightarrow 000111$
$100000 \longrightarrow 011100$	$011111 \longrightarrow 100011$
$010000 \longrightarrow 101100$	$101111 \longrightarrow 010011$
$001000 \longrightarrow 111000$	$110111 \longrightarrow 000111$
$000100 \longrightarrow 110100$	$111011 \longrightarrow 001011$
$000010 \longrightarrow 110010$	$111101 \longrightarrow 001101$
$000001 \longrightarrow 110001$	$111110 \longrightarrow 001110$

Table B.2: The very unbalanced 6-bit words.

Appendix C

Immink-Weber method applied to 6-bit words

This appendix presents a table of all 6-bit balanced words, along with their corresponding sets. Members of each set are the user words that are mapped to balanced words using the Knuth (or Immink-Weber) method. The first column of the following table contains the balanced words and their corresponding set. The second column shows the interval in which the SDR of the balanced word is bound. The cardinality of each set is equal to the size of this interval. The next column gives the corresponding $P(t, m)$ term that counts (includes) that balanced word.

The next three columns pertain to balancing almost balanced words. The fourth column in the table is the corresponding $P'(z_{max}, z_{min}, m)$ term which counts (includes) that balanced word. The fifth column has balanced words and their corresponding set. Note that input words in each set is restricted to 6-bit words with weights $6/2 = 3$ (balanced), $6/2 + 1 = 4$ (imbalance of 1), or $6/2 - 1 = 2$ (imbalance of 1). Also, the parameter K showing the allowed weight deviation from the balanced weight, is 1. The last column shows a modified interval for z_k. Note that each possible z_k corresponds to a position by bit flipping up to which the balanced word is mapped to a member of the given set (input word).

balanced words and corresponding ordered sets	$z_k \in I$	c.i.	c.i.	restricting to almost balanced	$z_k \in I \cap [-K, K]$
$111000 \longrightarrow \{011000, 111000, 000000, 001000\}$	$[0, 3]$	$P(4, 6)$	$P'(3, 0, 6)$	$111000 \longrightarrow \{011000, 111000\}$	$[0, 1]$
$110001 \longrightarrow \{010001, 001111, 110001, 000001\}$	$[-1, 2]$	$P(4, 6)$	$P'(2, -1, 6)$	$110001 \longrightarrow \{010001, 001111, 110001\}$	$[-1, 1]$
$011100 \longrightarrow \{111100, 100100, 011100, 100000\}$	$[-1, 2]$	$P(4, 6)$	$P'(2, -1, 6)$	$011100 \longrightarrow \{111100, 100100, 011100\}$	$[-1, 1]$
$100011 \longrightarrow \{000011, 011011, 100011, 011111\}$	$[-2, 1]$	$P(4, 6)$	$P'(1, -2, 6)$	$100011 \longrightarrow \{000011, 011011, 100011\}$	$[-1, 1]$
$001110 \longrightarrow \{101110, 110000, 001110, 111110\}$	$[-2, 1]$	$P(4, 6)$	$P'(1, -2, 6)$	$001110 \longrightarrow \{101110, 110000, 001110\}$	$[-1, 1]$
$000111 \longrightarrow \{100111, 000111, 111111, 110111\}$	$[-3, 0]$	$P(4, 6)$	$P'(0, -3, 6)$	$000111 \longrightarrow \{100111, 000111\}$	$[-1, 0]$
$101100 \longrightarrow \{001100, 101100, 010000\}$	$[0, 2]$	$P(3, 6)$	$P'(2, 0, 6)$	$101100 \longrightarrow \{001100, 101100\}$	$[0, 1]$
$110010 \longrightarrow \{010010, 010010, 000010\}$	$[0, 2]$	$P(3, 6)$	$P'(2, 0, 6)$	$010010 \longrightarrow \{010010, 010010\}$	$[0, 1]$
$110100 \longrightarrow \{010100, 110100, 000100\}$	$[0, 2]$	$P(3, 6)$	$P'(2, 0, 6)$	$110100 \longrightarrow \{010100, 110100\}$	$[0, 1]$
$100101 \longrightarrow \{000101, 011101, 100101\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$100101 \longrightarrow \{000101, 011101, 100101\}$	$[-1, 1]$
$100110 \longrightarrow \{000110, 011110, 100110\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$100110 \longrightarrow \{000110, 011110, 100110\}$	$[-1, 1]$
$101001 \longrightarrow \{001001, 010111, 101001\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$101001 \longrightarrow \{001001, 010111, 101001\}$	$[-1, 1]$
$011010 \longrightarrow \{111010, 100010, 011010\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$011010 \longrightarrow \{111010, 100010, 011010\}$	$[-1, 1]$
$011001 \longrightarrow \{111001, 100001, 011001\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$011001 \longrightarrow \{111001, 100001, 011001\}$	$[-1, 1]$
$010110 \longrightarrow \{110110, 101000, 010110\}$	$[-1, 1]$	$P(3, 6)$	$P'(1, -1, 6)$	$010110 \longrightarrow \{110110, 101000, 010110\}$	$[-1, 1]$
$010011 \longrightarrow \{110011, 010011, 101111\}$	$[-2, 0]$	$P(3, 6)$	$P'(0, -2, 6)$	$010011 \longrightarrow \{110011, 010011\}$	$[-1, 0]$
$001101 \longrightarrow \{101101, 001101, 111101\}$	$[-2, 0]$	$P(3, 6)$	$P'(0, -2, 6)$	$001101 \longrightarrow \{101101, 001101\}$	$[-1, 0]$
$001011 \longrightarrow \{101011, 001011, 111011\}$	$[-2, 0]$	$P(3, 6)$	$P'(0, -2, 6)$	$001011 \longrightarrow \{101011, 001011\}$	$[-1, 0]$
$101010 \longrightarrow \{001010, 101010\}$	$[0, 1]$	$P(2, 6)$	$P'(1, 0, 6)$	$101010 \longrightarrow \{001010, 101010\}$	$[0, 1]$
$010101 \longrightarrow \{110101, 010101\}$	$[-1, 0]$	$P(2, 6)$	$P'(0, -1, 6)$	$010101 \longrightarrow \{110101, 010101\}$	$[-1, 0]$

Table C.1: All 6-bit balanced words with their corresponding set and SDR interval are listed here. Same list is provided when the input is limited to almost balanced words. The table is continued on the next page.

balanced words and corresponding ordered sets	$z_k \in I$	c.i.	c.i.	restricting to almost balanced	$z_k \in I \cap [-K, K]$
$100011 \longrightarrow \{000011, 011011, 100011, 011111\}$	$[-2, 1]$	$P(4, 6)$	$P'(-2, 1, 6)$	$100011 \longrightarrow \{000011, 011011, 100011\}$	$[-1, 1]$
$100101 \longrightarrow \{000101, 011101, 100101\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$100101 \longrightarrow \{000101, 011101, 100101\}$	$[-1, 1]$
$100110 \longrightarrow \{000110, 011110, 100110\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$100110 \longrightarrow \{000110, 011110, 100110\}$	$[-1, 1]$
$101001 \longrightarrow \{001001, 010111, 101001\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$101001 \longrightarrow \{001001, 010111, 101001\}$	$[-1, 1]$
$101010 \longrightarrow \{001010, 101010\}$	$[0, 1]$	$P(2, 6)$	$P'(0, 1, 6)$	$101010 \longrightarrow \{001010, 101010\}$	$[0, 1]$
$101100 \longrightarrow \{001100, 101100, 010000\}$	$[0, 2]$	$P(3, 6)$	$P'(0, 2, 6)$	$101100 \longrightarrow \{001100, 101100\}$	$[0, 1]$
$110001 \longrightarrow \{010001, 001111, 110001, 000001\}$	$[-1, 2]$	$P(4, 6)$	$P'(-1, 2, 6)$	$110001 \longrightarrow \{010001, 001111, 110001\}$	$[-1, 1]$
$110010 \longrightarrow \{010010, 010010, 000010\}$	$[0, 2]$	$P(3, 6)$	$P'(0, 2, 6)$	$010010 \longrightarrow \{010010, 010010\}$	$[0, 1]$
$110100 \longrightarrow \{010100, 110100, 000100\}$	$[0, 2]$	$P(3, 6)$	$P'(0, 2, 6)$	$110100 \longrightarrow \{010100, 110100\}$	$[0, 1]$
$111000 \longrightarrow \{011000, 111000, 000000, 001000\}$	$[0, 3]$	$P(4, 6)$	$P'(0, 3, 6)$	$111000 \longrightarrow \{011000, 111000\}$	$[0, 1]$
$011100 \longrightarrow \{111100, 100100, 011100, 100000\}$	$[-1, 2]$	$P(4, 6)$	$P'(-1, 2, 6)$	$011100 \longrightarrow \{111100, 100100, 011100\}$	$[-1, 1]$
$011010 \longrightarrow \{111010, 100010, 011010\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$011010 \longrightarrow \{111010, 100010, 011010\}$	$[-1, 1]$
$011001 \longrightarrow \{111001, 100001, 011001\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$011001 \longrightarrow \{111001, 100001, 011001\}$	$[-1, 1]$
$010110 \longrightarrow \{110110, 101000, 010110\}$	$[-1, 1]$	$P(3, 6)$	$P'(-1, 1, 6)$	$010110 \longrightarrow \{110110, 101000, 010110\}$	$[-1, 1]$
$010101 \longrightarrow \{110101, 010101\}$	$[-1, 0]$	$P(2, 6)$	$P'(-1, 0, 6)$	$010101 \longrightarrow \{110101, 010101\}$	$[-1, 0]$
$010011 \longrightarrow \{110011, 010011, 101111\}$	$[-2, 0]$	$P(3, 6)$	$P'(-2, 0, 6)$	$010011 \longrightarrow \{110011, 010011\}$	$[-1, 0]$
$001110 \longrightarrow \{101110, 110000, 001110, 111110\}$	$[-2, 1]$	$P(4, 6)$	$P'(-2, 1, 6)$	$001110 \longrightarrow \{101110, 110000, 001110\}$	$[-1, 1]$
$001101 \longrightarrow \{101101, 001101, 111101\}$	$[-2, 0]$	$P(3, 6)$	$P'(-2, 0, 6)$	$001101 \longrightarrow \{101101, 001101\}$	$[-1, 0]$
$001011 \longrightarrow \{101011, 001011, 111011\}$	$[-2, 0]$	$P(3, 6)$	$P'(-2, 0, 6)$	$001011 \longrightarrow \{101011, 001011\}$	$[-1, 0]$
$000111 \longrightarrow \{100111, 000111, 111111, 110111\}$	$[-3, 0]$	$P(4, 6)$	$P'(-3, 0, 6)$	$000111 \longrightarrow \{100111, 000111\}$	$[-1, 0]$

Table C.2: The table is continued from the previous page. All 6-bit balanced words with their corresponding set and SDR interval are listed here. Same list is provided when the input is limited to almost balanced words.

Appendix D

P and P' for 6-bit words

The values of $P(t, m)$ using (4.14) for a sequence of length $m = 6$ and all possible values of t are

$$\begin{aligned} P(2, 6) &= 2 \\ P(3, 6) &= 12 \\ P(4, 6) &= 6. \end{aligned} \tag{D.1}$$

Using (4.17), (4.18), and (4.19), the values of $P'(z_{max}, z_{min}, m)$ for 6-bit sequences and all possible z_{min} and z_{max} are

$$
\begin{aligned}
P'(3,0,6) &= 5-4=1 \\
P'(2,-1,6) &= 13-4-8+1=2 \\
P'(1,-2,6) &= 13-8-4+1=2 \\
P'(0,-3,6) &= 5-4=1 \\
P'(2,0,6) &= 4-1=3 \\
P'(1,-1,6) &= 8-1-1+0=6 \\
P'(0,-2,6) &= 4-1=3 \\
P'(1,0,6) &= 1-0=1 \\
P'(0,-1,6) &= 1-0=1
\end{aligned}
\tag{D.2}
$$

The following verifies (4.20) for 6-bit sequences.

$$
\begin{aligned}
P(2,6) &= 2 = P'(1,0,6)+P'(0,-1,6) = 1+1 \\
P(3,6) &= 12 = P'(2,0,6)+P'(1,-1,6)+P'(0,-2,6) = 3+6+3 \\
P(4,6) &= 6 = P'(3,0,6)+P'(2,-1,6)+P'(1,-2,6)+P'(0,-3,6) = 1+2+2+1
\end{aligned}
\tag{D.3}
$$

The table in Appendix C can be used to verify the above values of P and P'.