UNIVERSITY OF CALIFORNIA, SAN DIEGO

Constrained Coding and Signal Processing for Data Storage Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering
(Communications Theory and Systems)

by

Sharon Aviran

Committee in charge:

Professor Jack K. Wolf, Chair
Professor Paul H. Siegel, Co-Chair
Professor William S. Hodgkiss, Jr.
Professor Lance W. Small
Professor Alexander Vardy

2006

The dissertation of Sharon Aviran is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2006

In loving memory of my grandfather.

Dedicated to my parents and to Hod and Shohum.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I had the unique opportunity of being mentored by two superb advisors who are also leaders in their field - Prof. Paul Siegel and Prof. Jack Wolf. Their patience, high standards, integrity, as well as their respect and dedication to their students have served as a model for me throughout my time at UCSD. I am mostly indebted for their care, encouragement and support during the past four years.

I would like to express my deep gratitude to Jack Wolf for his professional and personal support. His wisdom, advice, and unconventional approach to Ph.D. training were invaluable throughout my work. I am truly grateful to Paul Siegel for his technical insights, encouragement and guidance. I have greatly gained from his comprehensive knowledge and expertise as well as from his strive for perfection.

I would like to thank Prof. Larry Milstein for believing in my potential in my first year at UCSD, and for his assistance in facilitating my work with Paul Siegel and Jack Wolf. I also thank my committee members, Prof. Bill Hodgkiss, Prof. Lance Small, and Prof. Alex Vardy for their time and effort of serving in my committee.

My fellow group members have made my years at UCSD both enjoyable and productive. I would like to thank them for the friendly and cohesive atmosphere. Special thanks to Panu Chaichanavong and Joseph Soriaga for their help with acquiring a technical background, for fruitful discussions, and for their friendly attitude. I am also grateful to Brian Kurkoski. His good will, kindness, and endless assistance in all aspects of my work were very helpful to me in my first two years at the group. I also thank my office mate, Zheng Wu, for the great company and for the many technical discussions over the last two years.

I appreciate the CMRR's dedicated staff which provided a warm and friendly environment in which to do research. Cheryl Hacker, Betty Manoulian, and Iris Villanueva have always been very helpful with all the administrative issues and made it possible for me to concentrate on my research.

I would also like to acknowledge some of the friends I have acquired during my stay at the U.S., who have helped me along the way. Laddan, for her support during difficult times; Sabrina and David, for their friendship and for making us feel at home; and Uriel, for his good will and advice.

I am especially grateful to my family. I thank my parents, whose everlasting love and

care are the source of my strength. Thank you for the enormous help with Shohum, which enabled both Hod and I to have a family and still pursue our professional dreams. Without your help and support, this dissertation would have taken much longer to complete. Finally, I want to express my deepest gratitude to Hod, whose friendship, help, support, patience and valuable advice have enabled me to get to this moment. I will always cherish that.

This research was supported in part by the National Science Foundation (grant CCR-0219582) and by the Center for Magnetic Recording Research (CMRR) at UCSD.

Chapter 3 is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," in *Proc. 2004 IEEE Int. Symp. Inform. Theory*, Chicago, IL, Jun./Jul. 2004, p. 190 and S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2885-2891, Aug. 2005. Chapter 4 is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," to appear in *Proc. 2006 IEEE Int. Symp. Inform. Theory*, Jul. 2006 and S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," submitted to *IEEE Trans. Inform. Theory*, Jan. 2006. Chapter 5 is in part a reprint of the material in the paper: S. Aviran, P. H. Siegel, and J. K. Wolf, "Two-dimensional bit-stuffing schemes with multiple transformers," in *Proc. 2005 IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 1478-1482. Chapter 6 is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," in *Digests IEEE Int. Magnetics Conf. INTERMAG 2005*, Nagoya, Japan, Apr. 2005, pp. 981-982 and S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," *IEEE Trans. Magnetics*, vol. 41, no. 10, pp. 2959-2961, Oct. 2005. The dissertation author was the primary author of all these papers.

VITA

1995	B.A., Statistics and Economics, Hebrew University, Jerusalem, Israel
1999	M.Sc., Operations Research, Technion, Haifa, Israel
2006	Ph.D., Electrical Engineering (Communications Theory and Systems), University of California, San Diego

PUBLICATIONS

S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," to appear in *Proc. 2006 IEEE Int. Symp. Inform. Theory*, Jul. 2006.

S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," submitted to *IEEE Trans. Inform. Theory*, Jan. 2006.

S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," *IEEE Trans. Magnetics*, vol. 41, no. 10, pp. 2959-2961, Oct. 2005.

S. Aviran, P. H. Siegel, and J. K. Wolf, "Two-dimensional bit-stuffing schemes with multiple transformers," in *Proc. 2005 IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 1478-1482.

S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2885-2891, Aug. 2005.

S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," in *Digests IEEE Int. Magnetics Conf. INTERMAG 2005*, Nagoya, Japan, Apr. 2005, pp. 981-982.

S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," in *Proc. 2004 IEEE Int. Symp. Inform. Theory*, Chicago, IL, Jun./Jul. 2004, p. 190.

S. Aviran, N. Lev-Tov, S. Onn, and U. G. Rothblum, "Vertex characterization of partition polytopes of bipartitions and of planar point sets," *Discrete Applied Mathematics*, vol. 124, no. 1-3, pp. 1-15, Dec. 2002.

S. Aviran and S. Onn, "Momentopes, the complexity of vector partitioning, and Davenport-Schinzel sequences," *Discrete and Computational Geometry*, vol. 27, no. 3, pp. 409-417, Jan. 2002.

S. Aviran, "Vertex characterization of shaped partition polytopes," M.Sc. Dissertation, Technion - Israel Institute of Technology, Haifa, Israel, 1999.

ABSTRACT OF THE DISSERTATION

Constrained Coding and Signal Processing for Data Storage Systems

by

Sharon Aviran
Doctor of Philosophy in Electrical Engineering
(Communications Theory and Systems)

University of California San Diego, 2006

Professor Jack K. Wolf, Chair

Professor Paul H. Siegel, Co-Chair

Constrained codes for digital storage systems are studied. A method for improving signal detection in digital magnetic recording systems is also investigated.

The bit stuffing algorithm is a technique for coding constrained sequences by the insertion of bits into an arbitrary data sequence. This approach was previously introduced and applied to the family of (d, k) constraints. Results show that the maximum average rate of the bit stuffing code achieves the Shannon capacity when $k = d + 1$ or $k = \infty$, and fails to achieve capacity for all other (d, k) pairs. A modification to the bit stuffing algorithm is proposed that is based on the addition of controlled bit flipping. It is shown that the modified scheme achieves improved average rates over bit stuffing for most (d, k) constraints. All (d, k) constraints for which this scheme produces codes with an average rate equal to the Shannon capacity are determined.

A general framework for the construction of (d, k)-constrained codes from variable-length source codes is presented. Optimal variable-length codes under the general framework are investigated. The construction of constrained codes from variable-length source codes for encoding unconstrained sequences of independent but biased (as opposed to equiprobable) bits is also considered. It is shown that one can use the Tunstall source coding algorithm to generate optimal codes for a partial class of (d, k) constraints.

Bit-stuffing schemes which encode arbitrary inputs into two-dimensional (2-D) con-

strained arrays are presented. The class of 2-D (d, ∞) constraints as well as the 'no isolated bits' constraint are considered. The proposed schemes are based on interleaving biased bits with multiple biases into a 2-D array, while stuffing extra bits when necessary. The performance of the suggested schemes is studied through simulations.

A method for joint detection and decoding of coded transmission over magnetic recording channels is considered. The standard framework of turbo equalization is modified to account for the colored noise present in high-density magnetic recording systems. The modified scheme incorporates a noise prediction algorithm, which iteratively and selectively whitens the noise, while utilizing the information produced by the turbo equalization scheme. Simulation results demonstrate the performance improvements obtained by the proposed scheme.

1

Introduction

1.1 A Communications Channel View

The notion of a communications channel is commonly associated with a medium through which two objects at separate physical locations can exchange information. A communications system enables the flow of information through the medium by transmitting at one end and receiving at the other. A typical example is a television broadcast via cable or air. A less recognized notion is that the storage of information and its subsequent retrieval also constitute a form of a communications system. We can juxtapose a communications system which transmits information in space, with a data storage system, which transmits information in time.

Consider digital storage systems, also called *digital recorders*, where data is stored and retrieved in the form of binary information or bits. Each stored bit occupies a section of the entire storage medium, and corresponds to a single use of the communications channel, or equivalently, to one transmission. Bit values are stored as one of two possible physical states of the medium. For example, magnetic hard disk drives use a thin layer of magnetic material which can be magnetized entirely in either of two directions. In optical storage devices, such as the Compact Disc (CD), bit values are specified by the presence or absence of minuscule pits on the disc's surface.

The focus of this dissertation is on coding and signal processing techniques that improve the reliability and efficiency of communications systems. Their application to data storage systems can contribute to higher storage capacities and to improved immunity to

errors.

1.2 Reliable Digital Communications

In general, the aim of communications systems is to overcome the various imperfections of communications channels. In any real communications system, the received signal is not a perfect replica of the transmitted signal, but rather a degraded or *noisy* version of it. Noise can arise from imperfections in the transmission and reception devices, as well as from disruptive channel conditions, like scratches on the disc. In digital systems, the noise results in erroneous bit values and in corrupted data. Since most systems can tolerate only a very low incidence of incorrect information, an accurate retrieval of the sent information in the presence of noise is vital. The fields of digital communications and information theory are dedicated to developing efficient and easily-implementable methods as well as theoretical tools for achieving *reliable* transmission through noisy channels.

The foundation for these fields was laid out by Claude Shannon in 1948. In his seminal work, he introduced a mathematical framework which described and quantified the transmission of digital information over generic noisy channels [1]. Using these new concepts, he established that digital systems should be able to communicate reliably over a noisy channel, under a certain limitation. The limitation is given in terms of a maximum rate or speed at which the information can be transmitted through the channel. This fundamental result, known as the *noisy channel theorem*, has triggered a paradigm shift from analog communications to digital communications.

In the proof of the theorem, Shannon argued that reliable digital communication is obtained by means of *channel coding*. In general, channel coding relates to converting the sender's messages into other messages which are the ones sent through the channel. The transmitted messages include redundant information which can be exploited at the receiving end in order to reconstruct the sent messages from the channel outputs. Finally, the sender's original messages are recovered from the reconstructed data. We demonstrate the idea of channel coding with the following well-known example.

Suppose we represent the message to be sent as a sequence of bits. We send these bits over any type of a noisy channel, for example, by burning them on a CD. As a result

of noise, there is a certain likelihood that some of the bits will be read out erroneously, i.e., they will be flipped (from 0 to 1 and vice versa). The probability of such an event serves as a measure of the transmission (or recording) reliability through this channel. Consider now the following coding strategy: an encoder generates five copies of each bit, to be burned on the CD, instead of the original single bit. As before, the retrieval process introduces some errors. The retrieved bits are fed to a decoder, which knows of the five-copies encoding strategy and hence reviews all five bits. Since not all five copies may agree on the same value, the decoder can make an "educated guess" and choose the bit value that is most probable given the five copies. An example of such a guess is by taking a majority vote, which yields the most probable value under certain assumptions. Such a scheme can be seen to reduce the probability of input-bit errors or to improve reliability.

The described coding scheme replaces each bit with a corresponding five-bit block which contains redundant information, i.e., all bits are the same. The amount of added redundancy is usually measured in terms of the resulting *transmission rate*, defined as the average number of information bits that are conveyed in one channel use. The rate of transmission in the example above is $\frac{1}{5}$. The rate reflects the efficiency in which the scheme utilizes the channel. Now, if we would like to further reduce the error probability, we could use the same scheme with longer blocks or more repetitions. However, this comes at the cost of lower transmission rates or decreased efficiency.

From this example, it may seem that this tradeoff is inevitable and that obtaining a vanishingly small error probability requires infinite redundancy. The essence of Shannon's result is that this is not true. More precisely, one can transmit information through the channel with an arbitrarily small error probability at any rate less than the *channel capacity*. The capacity is a unique property of the channel and may be calculated or at least estimated for certain channels.

Although Shannon provided the guiding concept of channel coding, he left the problem of finding practical coding methods unresolved. In the next five decades, considerable research was dedicated to finding practical coding techniques whose rates approach the channel capacity and whose complexity is acceptable. Additional research was devoted to finding the capacity of various channels of interest. The challenge in code design is to provide a prescribed error probability with minimal impact on efficiency. It is also

essential that the scheme will be simple to implement. Various clever and practical coding schemes are available today which obtain both higher rates and better performance than in the example above. Furthermore, some of these schemes can get very close to the capacity of several simple channels. All of these schemes are more complex than the illustrated scheme in terms of the code structure and especially in their decoding strategy. However, they retain the same basic principle of mapping the original messages to longer messages which contain redundant information in the form of certain relations among the transmitted bits. In what follows, we describe how channel coding is accomplished in digital recorders.

1.3 Channel Coding and Signal Processing for Storage

Figure 1.3.1 shows a simplified model of a recording system, with an emphasis on its channel coding and signal processing components. Recording systems typically employ a concatenation of two distinct coding schemes: an *error-correction coding* (ECC) scheme and a *modulation coding* scheme. Each coding scheme realizes a *code*, which is a set of rules for assigning certain output sequences to each of the possible input sequences. An encoder converts the inputs to their assigned outputs, and a matching decoder attempts to recover the inputs from the retrieved outputs. The *rate* of a code is defined as the ratio of the average input length and the average output length. It measures the average number of user bits that are conveyed in each stored bit. The two coding procedures are inherently different in their approach to obtaining improved recording reliability, as we explain next.

As shown in Figure 1.3.1, user information first undergoes encoding by an ECC encoder, which adds extra bits to the input message bits [2]. This transformation is intended to protect the recorded data against multiple random errors that may occur during its retrieval. The encoder systematically generates the extra bits as a function of the input bits by imposing certain mathematical relations on the output bits. The resulting structure of the output allows the decoder to correct and/or detect certain bit errors. The five-copies encoding-decoding strategy we mentioned earlier is an example of the simplest error-correcting code. In Chapter 6, we describe in detail the encoding and decoding of a family of error-correcting codes, called low-density parity check (LDPC) codes [3]. LDPC

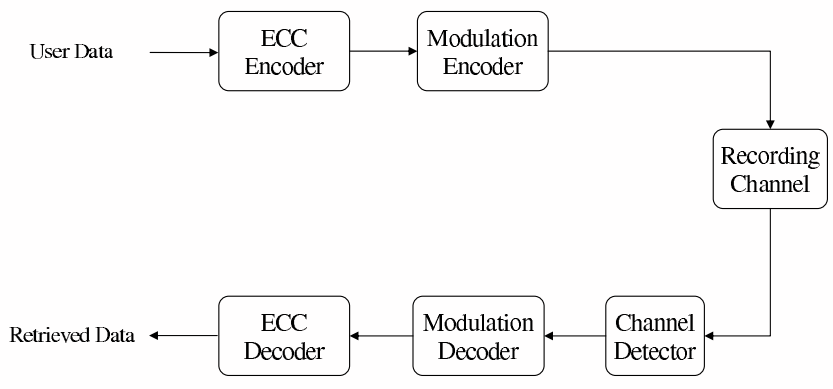


Figure 1.3.1 A simplified model of channel coding and signal processing components in digital recording systems. Channel coding is composed of two concatenated coding schemes employing an error-correcting code and a modulation code.

codes are currently regarded as potential candidates for integration into future-generation recording systems.

The next encoding step involves a *modulation* code. Here, a modulation encoder converts arbitrary input sequences into sequences that satisfy certain predefined restrictions [4]. The output sequences are called *constrained sequences*. Hence the name *constrained code*, which is a more familiar term for a modulation code in the context of digital recording. The constrained sequences are then stored on the medium instead of the original input. This approach is motivated by the observation that particular recorded sequences are more prone to errors during retrieval than others. Therefore, their exclusion from the collection of admissible channel inputs can improve the overall reliability of the system. The role of constrained codes is to avoid certain retrieval failures that are typical of the system. This stands in contrast with error-correcting codes, which attempt to correct those errors that already occurred. Constrained coding for storage systems constitutes the main topic of this dissertation. Chapter 2 provides a comprehensive overview of this topic. Chapters 3, 4, and 5 deal with the design and analysis of constrained codes.

The system model in Figure 1.3.1 illustrates another component - a channel detector. Before we explain its functionality, we assume that the recording channel component incorporates the following operations: converting the binary data to an analog signal, recording the analog signal on the medium, and retrieving it. In this model, channel

detection corresponds to processing the retrieved analog signal in order to accurately determine the binary values of the stored bits. A channel detector can incorporate a variety of signal processing techniques with different functionalities [5]. This includes filtering, sampling, resolving the interference between signals from adjacent stored bits in the presence of noise, and manipulating the noise portion of the signals. Some systems, like optical disc recorders, employ relatively simple detectors, while others, like magnetic hard disk drives, employ more sophisticated devices. Chapter 2 elaborates on the detection techniques used in current disk drives. Chapter 6 studies the adaptation of new detection techniques to improve the reliability of current disk drive technology.

1.4 Coding for Noiseless Channels

In his work, Shannon introduced two powerful concepts which are important in the design and analysis of constrained codes. We briefly review these concepts here, and we refer to them later in Chapters 3-5. Since we deal with digital systems, we limit our discussion to discrete channels, that is, to channels that admit a finite number of symbols as their input.

The first concept is the *input-constrained noiseless channel*, which forms a special case of the *noisy channel* discussed earlier. A channel is input-constrained if its admissible inputs do not include all possible sequences of symbols. In other words, some restrictions are imposed on the sequences that can be transmitted, such as the order of symbols in the sequence. In a noiseless channel, the received messages are an exact replica of the sent messages, so no errors can occur. The maximum possible transmission rate over this channel is the *Shannon capacity* of the channel. It is easy to see that constrained codes in digital recorders serve as a transformation of arbitrary binary messages into binary messages that can be sent over an input-constrained noiseless channel.

A second concept is the *entropy*, which measures the information content of messages generated by an information source. The simplest information source generates symbols that are statistically independent and are drawn according to one specified probability distribution. The basic idea is to establish a link between the predictability of probabilistic events and the amount of information they contain. It is reasonable that the occurrence of

an unlikely event brings more new information than the occurrence of a very likely event. Following this reasoning, Shannon proposed to measure the average information content of each generated symbol by

$$H(p_1, \cdots, p_M) = -\sum_{i=1}^{M} p_i \log_2 p_i,$$

where $p_1, \cdots, p_M$ are the probabilities to observe each of the M different symbols. Here, $\log_2 p_i$ quantifies the amount of information contained in the occurrence of an event whose probability is p_i. The base of the logarithm is 2 since it is convenient to represent the amount of information with binary units such as bits. Shannon extended the definition of entropy to more complex information sources which can still be characterized by a fixed probabilistic behavior.

The relevance of entropy to constrained coding is twofold. First, the conversion of unconstrained sequences to constrained sequences involves a change in the entropy per symbol. In digital recorders, we represent the information as sequences of binary symbols (0 and 1). This means that the entropy per user bit is different than the entropy per stored "constrained" bit. The higher the entropy of a stored bit, the more information it conveys. Therefore, we are interested in designing codes that generate constrained sequences with the highest possible entropy per bit. Second, the maximum possible rate of a constrained code is determined by $\frac{C}{H}$, where C is the Shannon capacity of the constrained channel and H is the entropy per *user* bit. If we model the user data as sequences of independent and equiprobable bits, then $H = 1$ and C is the maximum code rate.

1.5 Dissertation Overview

This dissertation considers two major themes: constrained coding and signal processing, as they apply to recording systems. Chapter 2 provides background on these topics, Chapters 3 - 5 are concerned with design and analysis of constrained codes, and Chapter 6 is concerned with the integration of signal processing methods into new iterative decoding and detection techniques.

The first part of Chapter 2 describes various aspects of constrained coding for storage systems. The rest of the chapter is devoted to a detailed description of the magnetic

recording channel and the detection process in computer hard disk drives.

Chapters 3 and 4 consider constrained coding methods for a commonly used family of constrained channels, called *run-length-limited (RLL)* (d, k) *constraints*. These channels impose limitations on the lengths of runs of consecutive like symbols. The coding schemes we study in these two chapters were motivated by a previously suggested coding scheme, called the *bit stuffing algorithm*. The bit stuffing algorithm generates constrained sequences by inserting extra bits into an arbitrary input stream in a manner that guarantees that the resulting output meets the limitations [6].

In Chapter 3, we propose to modify the bit stuffing algorithm by adding a controlled flipping of unconstrained bits. This modification maintains the simplicity and underlying principles of the bit stuffing technique, and is called the *bit flipping algorithm*. We analyze the bit stuffing and bit flipping algorithms to show that bit flipping achieves improved average rates over bit stuffing for most (d, k) constraints. We further determine all (d, k) constraints for which the bit flipping algorithm produces codes with an average rate equal to the Shannon capacity.

In Chapter 4, we study codes for (d, k) constraints from a *source coding* perspective. In general, source coding, also known as *data compression*, refers to encoding data into a shorter representation in a recoverable manner. The work presented in this chapter was inspired by a recent extension of the bit stuffing and bit flipping algorithms, called the symbol sliding algorithm [7]. We first extend the three algorithms into a general framework for the construction of constrained codes from variable-length source codes. We show that it gives rise to new codes which achieve improved performance over the aforementioned algorithms. We then search for optimal codes under this framework, optimal in the sense of their achievable rates. However, finding such codes appears to be a difficult problem. In an attempt to solve it, we are led to consider the encoding of unconstrained sequences of independent but biased (as opposed to equiprobable) bits. Here, our main result is that one can use the Tunstall source coding algorithm [8] to generate optimal codes for a partial class of (d, k) constraints.

In Chapter 5, we design constrained codes for use in next-generation storage technologies. These technologies give rise to communications channels of a two-dimensional nature, which can be viewed as an extension of the traditional one-dimensional channel.

New coding and detection techniques are required, as many of the existing techniques are not readily applicable to these channels. We present coding schemes that are based on an adaptation of the bit-stuffing algorithm to the encoding of two-dimensional constrained arrays. The proposed schemes can be viewed as an extension of several previously suggested bit stuffing schemes for two-dimensional constrained arrays [9], [10]. We compare the performance of the various schemes through simulations.

Chapter 6 focuses on channel detection in hard disk drives. We propose a method to harness the advantages of novel detection and decoding techniques for dealing with problems that are specific to current disk drive technologies. Specifically, we consider an iterative decoding and detection framework known as *turbo equalization*, which is the state-of-the-art method for channels such as the magnetic recording channel [11]. However, turbo equalization is designed for channels with white noise, a condition which does not apply in magnetic recording systems. In this chapter, we present a modified turbo equalization scheme that accounts for the special characteristics of the noise in magnetic recording systems. It incorporates a *noise prediction* algorithm, which iteratively whitens the noise in a selective manner, while utilizing the information produced by the turbo equalization scheme. Simulation results demonstrate the performance improvements obtained by the proposed scheme.

Bibliography

[1] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pt. I, pp. 379–423, Jul. 1948.

[2] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1977.

[3] R. G. Gallager, *Low-Density Parity-Check Codes*. The M.I.T. Press, Cambridge, MA, USA, 1963.

[4] K. A. S. Immink, *Codes for Mass Data Storage Systems*, Second Edition. Eindhoven, The Netherlands : Shannon Foundation Publishers, 2004.

[5] J. W. M. Bergmans, *Digital Baseband Transmission and Recording*. Boston, Massachusetts : Kluwer Academic Publishers, 1996.

[6] P. E. Bender and J. K. Wolf, "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences," in *Proc. 1993 IEEE Int. Symp. Inform. Theory*, San Antonio, TX, Jan. 1993, p. 6.

[7] Y. Sankarasubramaniam and S. W. McLaughlin, "Symbol sliding: improved codes for the (d, k) constraint," in *Proc. Int. Symp. Inform. Theory and Applic.*, Parma, Italy, Oct. 2004.

[8] B. P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 1967.

[9] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inform. Theory*, vol. 47 , no. 3, pp. 1166–1176, Mar. 2001.

[10] S. Halevy, J. Chen, R. M. Roth, P. H. Siegel, and J. K. Wolf, "Improved bit-stuffing bounds on two-dimensional constraints," *IEEE Trans. Inform. Theory*, vol. 50, no. 5, pp. 824–838, May 2004.

[11] R. Koetter, A. C. Singer, and M. Tüchler, "Turbo equalization," *IEEE Signal Proc. Magazine*, vol. 21, no. 1, pp. 67–80, Jan. 2004.

2

Background on Digital Recording Systems

This chapter provides the background for topics discussed in subsequent chapters. In the following sections, we elaborate on two components of the simplified model of a data storage system, namely, modulation coding and channel detection. Section 2.1 reviews constrained codes and their role in optical and magnetic storage. It is mostly based on material in [1], [2], and [3]. Section 2.2 provides a brief overview of magnetic recording technology for computer hard disk drives. In this section, we concentrate on channel detection techniques. A more detailed description of the reading process in disk drives appears in [3] and [4]. For a recent survey of the state-of-the-art in disk drive technology, we refer the reader to [5] and [6].

2.1 Constrained Coding for Storage

Constrained codes, also known as modulation codes, serve to avoid the recording of those sequences whose retrieval is likely to cause an erroneous read by the system. There are several sources for such retrieval failures, including various deficiencies in the reading technology, such as the timing recovery and detection mechanisms. Impairments of the physical recording channel itself may also severely distort some sequences while only slightly affecting others. Therefore, the characterization of problematic sequences and the definition of effective restrictions to eliminate them is part of the process of system

design. The chosen restrictions are usually prescribed in terms of special properties that an acceptable sequence must have, and are system-dependent.

We motivate and illustrate the use of constrained codes through an example. Consider a magnetic recording device, where the recording process magnetizes portions of a magnetic medium in one of two possible polarities. The bits are consecutively written onto fixed-size spaces, called bit cells, along a strip. Figure 2.1.1 shows a magnetization pattern that arises from the recording of a certain binary sequence. When reading the data, the device can only sense transitions in the direction of magnetization, i.e., it detects the boundary between contiguous bit cells of opposite polarities. The system electronics respond to such a transition with voltage changes, as depicted by the analog voltage waveform in Figure 2.1.1. It can be seen from the waveform that the effect of a transition on the measured voltage is not spatially limited. However, it decreases significantly with increasing distance from the transition point. This means that nearby transitions might significantly contribute to the overall response of the system at a current transition, whereas sufficiently distant transitions will have a more negligible effect. This phenomenon is often observed in communication systems and is commonly referred to as *intersymbol interference* (ISI). If transitions are too close, the system may no longer be able to accurately sense them or distinguish between them.

We would like to store binary information on the described system, while obeying the following writing convention. Each 1 in the input stream dictates a reversal in the direction of magnetization and each 0 stipulates a non-reversal in direction. Figure 2.1.1 shows an input stream that follows this convention and corresponds to the magnetization pattern in the figure. Suppose that our system can detect transitions reliably only if they are at least D microns apart. In the absence of coding, we should account for all possible input patterns, hence we allocate D microns for each bit cell. In this scenario, each recorded bit represents one user information bit. Consider now a coding scheme that can convert arbitrary data into special bit streams with no adjacent 1's, i.e., where there is at least one 0 between any two 1's. Applying the scheme on user input, we guarantee that recorded patterns will contain transitions which are at least two bit cells apart. We can take advantage of the new constellation by decreasing the bit cell size to $\frac{1}{2}D$ microns, thus increasing the *recording density* to $\frac{2}{D}$ bits per micron.

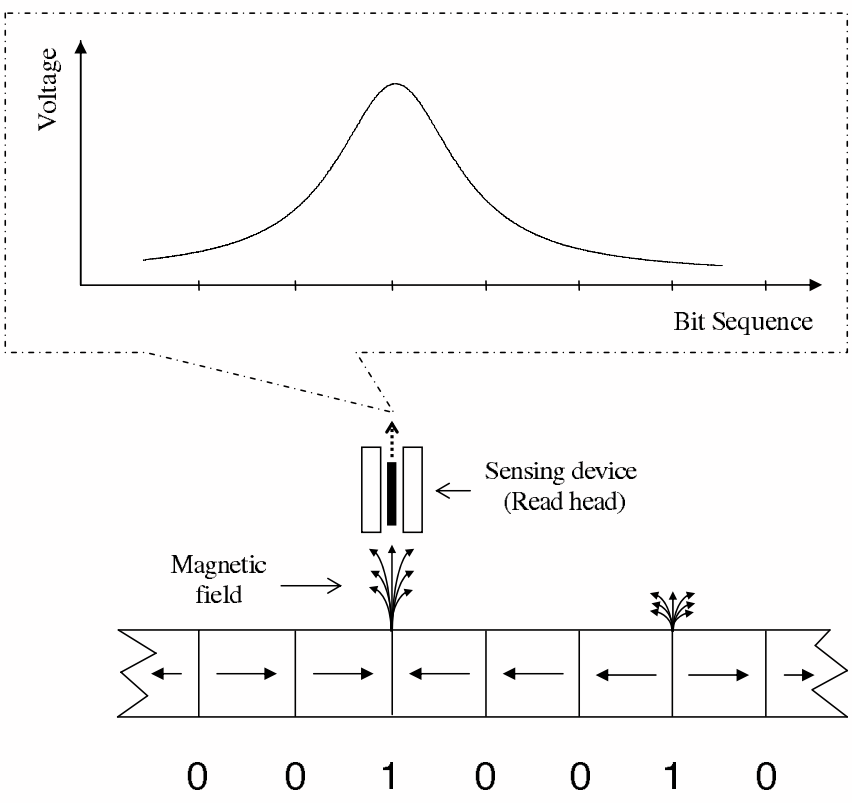


Figure 2.1.1 A simplified model of data retrieval in digital magnetic recording systems.

We are now able to record more bits on a strip of a given length, but we should also consider the overhead incurred by coding. Suppose we use the scheme described in Table 2.1.1, where the encoder simply replaces blocks of three bits at a time with five-bit blocks, as specified by the table. After correction of transmission errors by the ECC scheme, the decoder converts the blocks back using the same table. One can readily observe that the encoded stream indeed contains no adjacent 1's. The code rate is $\frac{3}{5}$, i.e., each recorded bit represents $\frac{3}{5}$ of a user bit. Since two recorded bits occupy D microns, we now write $\frac{6}{5}$ user bits-per D microns, as opposed to one user bit per D microns without coding. In the example above, we used a constrained code to gain a higher storage volume for a given reliability requirement. Alternatively, we could retain the same bit-cell size and use the code to obtain greater physical separation between transitions. In this case, we gain better system performance due to improved detection at the cost of loss in the effective storage capacity. Clearly, there is a tradeoff between these two benefits, therefore

Table 2.1.1 A constrained code for a run-length-limited $(1, \infty)$ constraint, where there must be at least one 0 between any two 1's.

Input Block	**Output Block**
000	00000
001	10000
010	01000
011	00100
100	00010
101	10100
110	10010
111	01010

using coding in conjunction with a smaller bit size may offer some compromise between them.

The example above is a simplification of a real problem that the designers of magnetic recording systems faced until the late 80's. At this time, systems were using a peak detection method, which searches for individual peaks in the output voltage as a means of detecting the recorded bits. A related but slightly different problem still exists in current optical disc recorders. This problem gave rise to a general class of restrictions, known as *run-length-limited* (RLL) (d, k) *constraints*, or simply (d, k) *constraints*. Here, the term *constraint* refers to the set of restrictions that are imposed on the sequences.

The bulk of this dissertation deals with constrained codes for (d, k) constraints and with recent extensions of (d, k) constraints to two-dimensional lattices. A (d, k) constraint requires that successive 1's are separated by at least d 0's and prohibits runs of more than k consecutive zeros, where $k > d$. The parameter k can be set to infinity, in which case only the d restriction applies. It is easy to see that the restriction described in the example (i.e., no adjacent 1's) is in fact a $(1, \infty)$ constraint. The d restriction addresses the above-mentioned problem by ensuring a certain minimum distance between transitions. The k restriction is essential for extracting timing information in order to maintain read-clock accuracy. This can be further explained as follows. The reading pro-

cess constantly adjusts the phase and frequency of a read-clock such that the clock keeps track of the estimated locations of the bit cell boundaries. This is useful for accurate detection as well as for avoiding spurious or missing bits. When the device responds to a reversal in magnetization, the timing recovery mechanism uses the resulting analog signal to derive positioning information. Since only transitions produce nonzero output signal, it is desirable to ensure that transitions are frequent enough for adequate clock synchronization. We meet this goal by limiting the number of consecutive 0's in the recorded data.

The family of (d, k) constraints has been found to be useful both in magnetic and optical recording applications. It therefore received much attention and is well understood. Various codes for (d, k) constraints have become part of virtually all magnetic and optical recording systems over the last four decades. These codes were initially adopted by the magnetic recording industry and were integrated into various disk and tape-based products. They have greatly contributed to the tremendous growth in storage densities that this industry has achieved. Nowadays, (d, k) constraints are mostly found in consumer-electronics products that are based on optical storage technology. For example, the CD and the DVD employ constrained codes for a $(2, 10)$ constraint. In magnetic recording, however, these constraints have become somewhat obsolete following the abandonment of the peak detection method in the early 90's. A new technology, called partial-response maximum-likelihood (PRML) detection, was adopted. PRML detection has circumvented the problem of detecting close transitions, and hence (d, k) constraints were no longer suitable for it. Present-day disk drives for computers are using PRML technology together with another type of run-length-limited constraint called a $(0, G/I)$ constraint. Nevertheless, (d, k) constraints are used in less prominent magnetic storage products such as magnetic tapes, floppy disks, and the new DVR.

A variety of other constraints have been found useful in digital recording. The restriction to *DC-free* sequences is notably the most common of these constraints. DC-free sequences have a spectral null at zero frequency, i.e., they have no zero-frequency content. The enforcement of a spectral null also results in the suppression of low-frequency components near the zero frequency. This is advantageous in systems which intentionally cut-off low frequencies in order to avoid the adverse effects they have on certain mech-

anisms. Since the cut-off will additionally affect sequences containing low-frequency components, it is desirable to avoid them. The actual construction of DC-free sequences relies on one of their characteristics in time rather than in the frequency domain. In essence, one needs to keep track of the sequence and constantly ensure that the total number of observed 0's does not deviate much from the total number of observed 1's. Since this criterion does not conflict with RLL limitations, many systems, such as optical disc recorders, impose a combination of DC-free and RLL constraints.

Another popular constraint is the run-length-limited $(0, G/I)$ constraint, which we mentioned earlier in the context of computer hard disk drives. It limits the length of runs of consecutive 0's between 1's, where G specifies the overall permissible maximum, and I specifies the maximum in each of the even and odd interleaves. The constraint addresses two problems arising in PRML technology. First, the timing recovery techniques employed are based on the sampled signal and do not operate properly in the presence of long runs of 0's. The G restriction solves this problem and is identical to the k constraint used by old disk drives. The I constraint is used to limit the memory and delay involved in the PRML detection process.

The introduction of PRML technology has also given rise to other proposed constraints that aim at improving its detector performance. This approach is based on the fact that the detector cannot distinguish very well between certain pairs of recorded sequences and commonly replaces one with the other. Since this is the cause of the most common detection errors, a suitable constraint can obtain improved detection by eliminating either one or both sequences of each pair. Constraints in this category include matched-spectral-null (MSN) constraints, defined in the frequency domain, and the RLL-type maximum-transition-run (MTR) constraints. In practice, hard disk drives use another approach to improve detection performance, called post-processing. We will refer to this technique in the next section.

The practical need for constrained codes has stimulated extensive research, mostly aimed at devising high-rate yet simple codes. The code rate is evaluated with respect to the *Shannon capacity* of the constraint, defined as

$$cap(\mathcal{C}) = \lim_{n\to\infty} \frac{\log_2 N_{\mathcal{C}}(n)}{n},$$

where $N_{\mathcal{C}}(n)$ is the number of sequences of length n that meet the constraint $\mathcal{C}$ [7]. The

capacity represents the amount of information that can be carried by the constrained sequences. It therefore plays an important role in code design. The common assumption in constrained-code design is that the unconstrained user-data is a stream of independent and equiprobable bits. Under this assumption, the Shannon capacity forms an upper bound on the rate achievable by any coding scheme that encodes the user-data into constrained data. The capacity of the various constraints discussed above can be calculated using a general method that Shannon prescribed. The method applies for any constraint that can be described by a finite labeled directed graph, as long as the labels of the outgoing edges at each state are distinct.

When considering constrained coding schemes, it is also important to bear in mind that in current storage systems, the decoder operates on the binary bit estimates generated by the channel detector. Since erroneous detector estimates are inevitable, we might observe scenarios where the decoding of a sequence with few errors results in significantly more errors at the recovered input. In this case, the ECC scheme may not suffice to protect against such a volume of errors. For this reason, avoiding this problem, called error propagation, is another major concern in code design. A comprehensive survey of the many available coding schemes exceeds the scope of this dissertation and can be found, for example, in [1]. In what follows, we briefly mention some widely-used approaches to constrained-code design, as well as key characteristics of constrained codes.

Block codes are probably the most prominent and broad class of constrained-code constructions. A block coding scheme repeatedly maps arbitrary input blocks of length m, called *source words*, into selected output blocks of length n, called *codewords*. Encoding involves the partitioning of the input stream and the replacement of input blocks with output blocks. Decoding is done in a similar fashion. There are various possible mappings and different ways in which the encoder and decoder realize them. The code we described earlier (see Table 2.1.1) is an example of the simplest block code in terms of encoding and decoding. Since the output blocks in Table 2.1.1 can be freely concatenated without violating the constraint, the encoder can instantaneously replace each source word without accounting for previous or future codewords. The same principle applies at the decoder.

More sophisticated block coding schemes involve encoders and/or decoders that must consider other neighboring blocks in order to determine their current output. In particu-

lar, block codes with *state-dependent* encoders and *sliding-block* decoders are of major importance in digital recording. A state-dependent encoder introduces memory to the encoding process by keeping track of part of its prior output. Each state of the encoder corresponds to a set of possible past outputs. The next output is then a function of both the current input and the current state, and is usually specified by a lookup table that is associated with each state. A sliding-block decoder can determine the current m-bit source word by looking at the current n-bit codeword as well as at several past and/or future neighboring n-bit codewords.

The importance of these codes is derived from two sources. First, they obtain high rates while using relatively small lookup tables and ensuring limited error-propagation. Second, there is a systematic design procedure for such codes, known as the *state-splitting algorithm*. The algorithm, introduced by Adler, Coppersmith and Hassner in 1983, has made notable progress in block code design. It was the first algorithm that provided a general method for constructing block codes of any given feasible rate with the above-noted properties. Furthermore, it is applicable to a broad class of practical constraints. It is worth noting that the above-mentioned codes fall short when large block lengths are desired, as the search in large lookup tables becomes infeasible. Block codes that are based on enumerative coding avoid this problem, but on the other hand, are susceptible to error propagation.

Block codes are prominent in digital recorders mainly due to various system considerations that dictate usage of fixed-length blocks. Nonetheless, non-block codes were studied as well. A *variable-length* (VL) code permits input source words and/or output codewords of variable size. If the ratio of lengths of a source word and its corresponding codeword is fixed, we say that the code has a *fixed rate*. With fixed-rate VL codes, it is still possible to compute the output length that corresponds to a large input block, regardless of the input content. This property is important in real systems and makes these codes a possible alternative to block codes. Numerous fixed-rate VL codes which are based on small lookup tables have been proposed.

We are left to consider the class of *variable-rate* codes, where the length of the output can vary depending on the actual content of the input and not only on its size. These codes were less thoroughly investigated compared with the former classes due to the reasons

mentioned earlier. However, they can offer very simple coding schemes with very high rates. Chapters 3 and 4 study a number of variable-rate coding schemes that are based on a previously-suggested variable-rate coding technique called *bit stuffing*. The bit stuffing technique obtains near-capacity rates using very simple encoding-decoding principles. It is also applicable to a wide range of constraints. Recently, attempts have been made to adapt the bit stuffing approach to the practical needs of industry. In particular, fixed-rate codes that are based on the simple principles of bit stuffing were proposed for $(0, k)$ and $(0, G/I)$ constraints [8], [9].

We conclude by mentioning recent progress in storage technology which presents new challenges for constrained coding. Newly emerging techniques, such as holographic data storage [10] and multi-track optical recording [11], give rise to two-dimensi-onal (2-D) models of the stored data. These technologies store the binary information either along contiguous tracks or as pages that are projected onto a holographic medium. For example, Figure 2.1.2 shows a simplified model of a holographic storage system. The data are first organized in the form of a page, using a pixelated device called a spatial light modulator. The information page is then carried by an object beam that goes through the device. The object beam intersects another beam (a reference beam) within a photo-sensitive storage material, and the resulting optical interference pattern changes certain properties of the material. Reading is performed by illuminating the material with the reference beam. This reconstructs the object beam, which is then sensed by an array of detectors.

From a communications standpoint, these systems can be viewed as a 2-D channel that introduces noise and ISI. Similarly to the one-dimensional (1-D) case, imposing a constraint on a two-dimensional data array can be an effective tool for improving detection performance. Two-dimensional constraints that eliminate certain problematic patterns were proposed in [10], [11]. This reasoning motivates the new field of 2-D constrained coding, which turns out to pose a collection of intriguing and difficult problems. In particular, many conventional 1-D constrained coding techniques are not readily applicable to 2-D. This is primarily due to lack of a finite-state graph description of many 2-D constraints of interest. Moreover, the fundamental limit on the achievable code rates, i.e., the Shannon capacity, is currently unknown for many constraints and is deemed to be more difficult to compute than its 1-D counterpart.

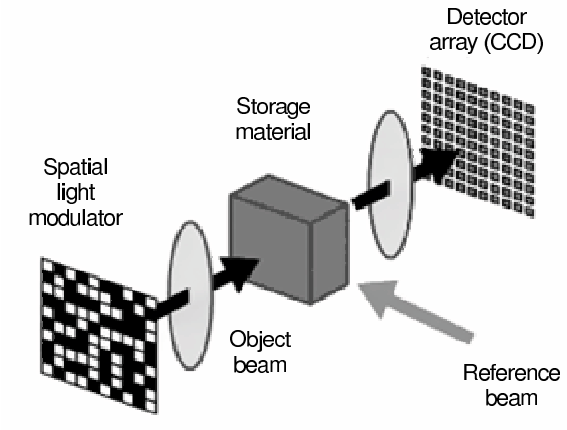


Figure 2.1.2 The basic components of a holographic data storage system. Taken from [10].

Two-dimensional constraints are the topic of Chapter 5. In this chapter, we consider coding schemes that are based on a generalization of the 1-D bit stuffing technique. As we will show, the bit stuffing approach can be easily extended to a variety of 2-D constraints, and is independent of a graph-based representation. The extension to 2-D maintains the simplicity of the original technique and often achieves high rates. Furthermore, in cases where it is amenable to analysis, it serves as a tool for deriving analytical bounds on the unknown capacity of the constraint.

2.2 Magnetic Recording in Hard Disk Drives

Hard disk drives store data along concentric circular tracks on the surface of a disk coated with a magnetic medium. During the writing process, the disk rotates while a device called a write head induces a magnetic field on the medium. The applied magnetic field is strong enough such that the medium remains magnetized after the head has progressed along the track. In *longitudinal magnetic recording*, the write head can magnetize the medium either along the direction of the disk motion or against it, as shown in Figure 2.1.1. Each cell along the track is magnetized entirely in one of two directions, hence it can represent the value of a bit. The induced magnetic field is generated by an electrical current flowing into the write head, where the direction of the current determines the magnetization direction. Hence, the binary bits translate into a corresponding current,

which can change its direction only at the bit cell boundaries. Reading is performed by a different device called a read head. It is constructed of a material that reacts to changes in the direction of magnetization by changing its electrical resistance. These changes are translated by the reading circuitry into registered voltage changes.

2.2.1 Channel Model

The measured voltage is the analog signal that is generated by the read head and processed by the system. The response of the read head to an isolated transition in the direction of magnetization is usually approximated by the following function:

$$h(t) = \frac{A}{1 + \left(\frac{2t}{PW50}\right)^2},$$

where t represents time, A is the peak amplitude, and PW50 is the width of the function at 50% of its peak amplitude. A transition will produce either the pulse $h(t)$ or the pulse $-h(t)$, depending on its type (i.e., $\longrightarrow | \longleftarrow$ or $\longleftarrow | \longrightarrow$). The function $h(t)$ is well-known as the *Lorentzian pulse* and is illustrated in Figure 2.2.1 for $A = 1$ and $PW50 = 1$. One can observe the extent of ISI for a given recording density from the pulse shape. Specifically, let T_b denote the time required for the read head to move over a bit cell, i.e., the *bit duration*. The *linear recording density* is defined as $D = PW50/T_b$, measuring how many bits are packed into the center of the pulse (see Figure 2.2.1 for an example when $D = 2$). It can be seen that for a given $PW50$, decreasing the bit size does not change the generated pulses. However, the transitions get closer together and therefore there is stronger interference between them.

The model of the overall response of the system to the entire recorded pattern is based on the assumption that the recording channel is linear. Additionally, it is convenient to model the recorded information as a stream of bipolar symbols $\{x_i\}$, $x_i \in \{-1, +1\}$, which are modulated to form a rectangular current waveform with amplitude $+1$ or -1. The waveform amplitude corresponds to the direction of the current that flows into the write head. The binary user input $\{a_i\}$ is mapped into bipolar symbols before the actual writing takes place. Using these conventions and assumptions, it can be shown that the

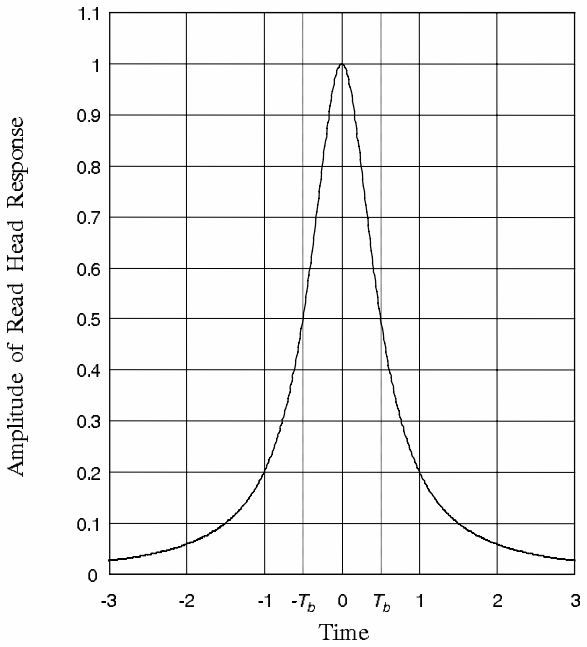


Figure 2.2.1 Lorentzian head response to an isolated transition for $A = 1$ and $PW50 = 1$. For a recording density $D = PW50/T_b = 2$, the bit duration T_b is equal to $\frac{1}{2}$.

read-back signal takes the form

$$y(t) = \sum_i x_i g(t - iT_b) + n(t), \tag{2.2.1}$$

where $n(t)$ describes the noise due to the readout electronics and

$$g(t) = \frac{1}{2}\left(h(t) - h(t - T_b)\right). \tag{2.2.2}$$

Here, $g(t)$ represents the response of the channel to a single isolated bit, such as a single 1 preceded and followed by an infinite number of 0's. The formulation in (2.2.1) expresses the ISI between adjacent bits rather than between transitions. The noise $n(t)$ is typically modeled as white and Gaussian. It is important to note that at the high recording densities of state-of-the-art disk drives, material granularity is becoming a significant source of noise. This noise, called *media noise*, is not white and depends on the recorded patterns. Although more accurate models that accommodate media noise exist, we shall work with the simpler white Gaussian model, which still provides a good approximation of present systems.

2.2.2 Partial-Response Maximum-Likelihood Detection

We have seen that the read and write processes introduce intersymbol interference and noise. The task of the receiver is then to estimate or detect the actual recorded bit values from the read-back signal, which takes the form of (2.2.1). The old approach to channel detection in disk drives is called *peak detection*. In essence, it scans the analog signal and identifies large-enough peaks, which correspond to transitions. It then reconstructs the recorded sequence from the peak locations. However, this method is limited in its ability to resolve ISI. At certain recording densities, the arising ISI results in missed and shifted peaks, leading to poor detection.

Further increases in recording densities were facilitated by the introduction of the partial-response maximum-likelihood (PRML) approach to detection. As opposed to peak detection, PRML accounts for the existing ISI and hence detects sequences of many bits together, rather than each bit separately. At the basis of this approach is a well-known method for optimal sequence detection of signals in ISI channels with additive white Gaussian noise [12]. According to this method, the analog signal is first filtered and then sampled, such that each sample corresponds to one bit. Subsequently, a sequence of samples is fed to a *maximum-likelihood* (ML) sequence detector, based on the Viterbi algorithm [13]. The detector's estimate is the sequence which maximizes the likelihood of receiving the observed samples over all possible sequences. The estimate is derived from a known interference model of the channel, i.e., from the channel response.

Unfortunately, the complexity of the Viterbi algorithm grows drastically with the duration of the channel response, which renders this approach impractical for the Lorentzian channel. The PRML approach circumvents this problem by taking the following two steps:

- Equalizing the read-back signal to a target signal that has limited ISI or shorter response.
- Performing maximum-likelihood detection that is matched to the target channel response.

The basic idea is to allow for limited interference from nearby bits but also to take this interference into account during detection. This is often referred to as controlled ISI.

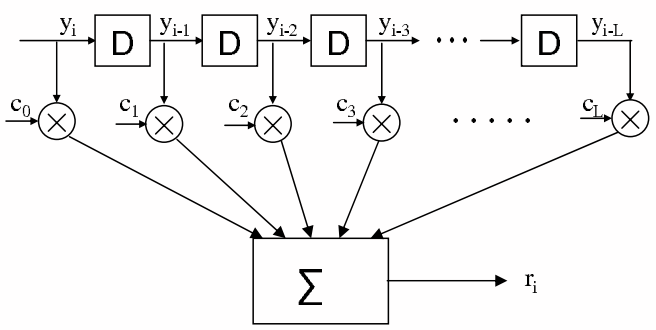


Figure 2.2.2 A tapped delay line finite impulse response (FIR) filter used for equalization.

Equalization takes place after filtering and sampling. It shapes the samples of the original channel response into another set of samples of a much shorter duration. For example, consider the target response whose samples are represented by the polynomial $1 - D^2$, where D is a delay operator. This means that after shaping, the noiseless sample values can be expressed as $s_i = x_i - x_{i-2}$, where $\{x_i\}$ is the bipolar input to the channel. Therefore, the shaping has eliminated the interference caused by all bits except for the next-to-nearest neighboring bit. Present systems implement the shaping by using a finite impulse response (FIR) linear filter, as modeled by the tapped delay line in Figure 2.2.2. As noted earlier, the Viterbi detector derives its estimates using the known channel structure. Since the Lorentzian channel has been transformed to the target channel, the detector operates according to the new target channel model.

Reducing ISI by means of equalization comes at a cost, namely, enhancement of the electronics noise and the introduction of correlation into its samples. This is referred to as *noise enhancement and coloration*. Such effects are undesirable as they degrade the performance of the Viterbi detector, which is optimal for white noise but not for correlated noise. In general, we can say that the severity of these effects is a function of the differences between the original and the target shapes. More precisely, it depends on the differences between their spectral characteristics. For this reason, it is important to choose a target which provides a good approximation to the real channel response. On the other hand, one should keep in mind that a practical target must also be sufficiently short. Since increased recording densities introduce more ISI, the chosen target also depends on the density. Usually, longer targets provide a better fit when densities increase.

PRML systems use several targets from a general class called *partial-response* (PR). This class includes targets of the general polynomial form $(1-D)(1+D)^N$, $N \geq 1$. The primary reason for this choice is that PR targets provide a good match to the actual channel response. Furthermore, the use of small integer coefficients reduces the complexity of the Viterbi detector. The most frequently used targets are called PR4 and EPR4, and correspond to the polynomials $(1-D)(1+D) = 1-D^2$ and $(1-D)(1+D)^2$, respectively. Their short duration facilitates acceptable detection complexity. However, the steady increase in recording densities has created a need for more suitable higher-order targets. Today, disk drives employ targets with longer duration and non-integer coefficients, as will be discussed in the next subsection and in Chapter 6.

2.2.3 System Overview

In this subsection, we briefly review the signal processing and coding modules that constitute a PRML system. Before we start, it is important to note that PRML systems have evolved since their introduction in 1992 [3], mainly through the addition of several signal processing techniques [5], [6]. Here, we will refer to the original PRML architecture, and only briefly mention the more recent additions. In Chapter 6, we will elaborate more on one of these techniques, called NPML.

Figure 2.2.3 shows a block diagram of a PRML system. The data is processed in blocks of 512 bytes, called sectors. It is first fed to an ECC encoder, which adds extra bytes to the data in order to correct possible random errors that occur due to noise and due to failures of other components in the system. The code used is Reed-Solomon, which is also common in a wide range of other communication systems [11]. The code is designed such that it is capable of correcting up to a specified number of erroneous bytes. Additionally, Reed-Solomon codes can cope well with bursts of errors. This renders them especially suitable for disk drives, where error-propagation caused by the modulation decoder as well as defects on the medium might result in error bursts. To further combat error bursts, the sector data is partitioned into several blocks, where each block is encoded separately. The output blocks are then interleaved before being passed to the modulation encoder.

The modulation encoder imposes a $(0, G/I)$ constraint on the data. At the output of

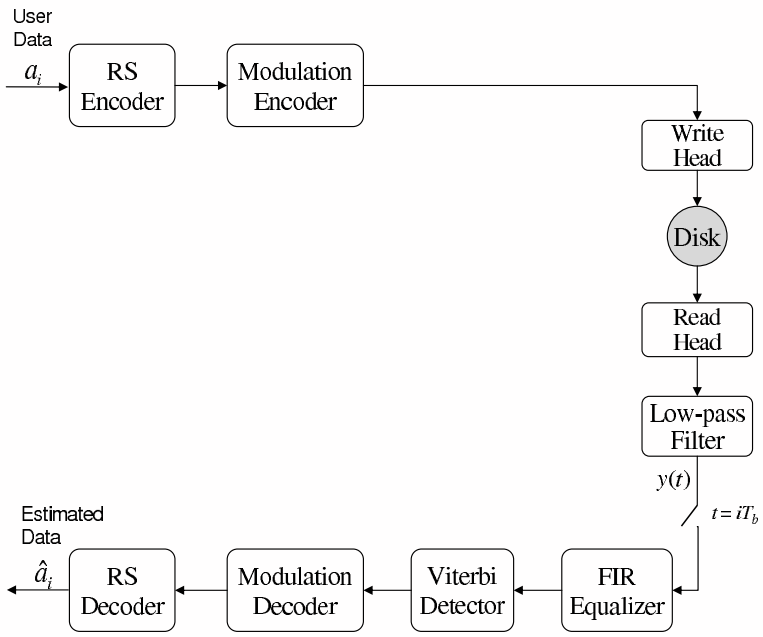


Figure 2.2.3 Block diagram of a partial-response maximum-likelihood (PRML) system.

the encoder, an additional operation called *precoding* is required. The reason for precoding can be explained as follows. Recall that in constrained coding, we use a convention where a 1 indicates a transition and a 0 indicates no transition. However, the write head is controlled by a current which flows in one of two directions, according to input values of 0 and 1. Consequently, the 1's and 0's are interpreted as different directions of magnetization and not as transitions/no-transitions. The transformation between these two conventions is performed by the precoder. The inverse transformation takes place prior to modulation decoding. We consider precoding as part of the modulation component and do not explicitly specify it in Figure 2.2.3.

During a read operation, the read head generates a continuous signal that is filtered by a low-pass filter and sampled at the symbol rate ($1/T_b$). An FIR equalizer shapes the noisy samples of the read-back signal into samples consisting of a sampled PR signal and additive total distortion component, consisting mainly of correlated Gaussian noise. The outputs of the equalizer are fed to the Viterbi detector, which outputs bit estimates, or *bit decisions*, for the entire sector data. These decisions undergo modulation decoding, which provides estimates of the input to the modulation encoder. These estimates are

de-interleaved and processed by the RS decoder, which attempts to correct any existing bit errors and to reproduce the user input.

In the last decade, two additional techniques have improved the performance of PRML systems. One technique combats the effects of noise enhancement and coloration. It simply whitens the noise component before the noisy samples are fed into the Viterbi detector. This involves the addition of an FIR filter at the output of the PR shaping equalizer. The filter manipulates the correlated noise component as well as the signal component. It results in noise that is approximately white, but also in a more complex signal or channel response. The resulting channel model has longer duration and non-integer coefficients. Consequently, the Viterbi detector is modified to accommodate the new channel response, at the cost of increased complexity. The new architecture is called noise-predictive maximum-likelihood (NPML) detection and is the topic of Chapter 6. In that chapter, we adapt the ideas that serve as the basis of this method to new and promising next-generation signal processing and coding techniques.

A second addition to PRML architecture is a *post-processing* unit. It aims to improve the performance of the Viterbi detector by detecting and correcting some of the errors made by the Viterbi detector, before its estimates are sent to the modulation decoder. The post-processor utilizes both the estimates of the detector and the noisy outputs of the PR equalizer. Based on these inputs and on the channel model, the unit infers whether one of several most likely detection errors has occurred. The addition of a small number of extra bits to the outputs of the modulation encoder aids the post-processor in evaluating the actual error events that have occurred.

Bibliography

[1] K. A. S. Immink, *Codes for Mass Data Storage Systems*, Second Edition. Eindhoven, The Netherlands : Shannon Foundation Publishers, 2004.

[2] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, “Codes for digital recorders,” *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[3] R. D. Cideciyan, F. Dolive, R. Hermann, W. Hirt, and W. Schott, “A PRML system for digital magnetic recording,” *IEEE J. Select. Areas Commun.*, vol. 10, no. 1, pp. 38–56, Jan. 1992.

[4] J. W. M. Bergmans, *Digital Baseband Transmission and Recording*. Boston, Massachusetts : Kluwer Academic Publishers, 1996.

[5] A. Dholakia, E. Eleftheriou, T. Mittelholzer, and M. P. C. Fossorier, "Capacity-approaching codes: can they be applied to the magnetic recording channel?," *IEEE Commun. Magazine*, vol. 42, no. 2, pp. 122–130, Feb. 2004.

[6] R. D. Cideciyan, J. D. Coker, E. Eleftheriou, and R. L. Galbraith, "Noise predictive maximum likelihood detection combined with parity-based post-processing," *IEEE Trans. Magnetics*, vol. 37, no. 2, pp. 714–720, Mar. 2001.

[7] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pt. I, pp. 379–423, Jul. 1948.

[8] Y. Sankarasubramaniam and S. W. McLaughlin, "A fixed-rate bit stuffing approach for high efficiency k-constrained codes," in *Proc. 2005 IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 543–547.

[9] Y. Sankarasubramaniam and S. W. McLaughlin, "Bit stuff encoding for $(0, G/I)$ constraints," submitted to *2006 IEEE Int. Symp. Inform. Theory*, Seattle, WA, Jul. 2006.

[10] J. Ashley, M. P. Bernal, G. W. Burr, H. Coufal, H. Guenther, J. A. Hoffnagle, C. M. Jefferson, B. Marcus, R. M. Macfarlane, R. M. Shelby, and G. T. Sincerbox, "Holographic data storage," *IBM J. Res. Develop.*, vol. 44, no. 3, pp. 341–368, May 2000.

[11] A. H. J. Immink, W. M. J. Coene, A. M. van der Lee, C. Busch, A. P. Hekstra, J. W. M. Bergmans, J. Riani, S. J. L. V. Benden, and T. Conway, "Signal processing and coding for two-dimensional optical storage," in *Proc. IEEE Globecom 2003*, San Fanscisco, CA, Dec. 2003, pp. 3904–3908.

[12] G. D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. 18, no. 3, pp. 363–378, May 1972.

[13] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[14] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1977.

3

The Bit Flipping Algorithm for Run-Length-Limited Constrained Coding

3.1 Introduction

Digital recording systems commonly use a constrained modulation code to improve detection reliability. Such a code applies an invertible mapping from arbitrary user-data sequences into a set of binary sequences that have special properties. The set of permissible target sequences is defined in terms of a *constraint*. One constraint, which has found widespread use in magnetic and optical recording applications, is the *run-length-limited* (d, k) *constraint* [1], [2]. A binary sequence is a (d, k)*-sequence* if it has the following two properties: successive ones are separated by at least d zeros and the number of consecutive zeros does not exceed k. The d restriction serves to alleviate intersymbol interference and the k restriction assists in timing recovery. Relevant (d, k) pairs range over all integers d, k, such that $0 \leq d < k \leq \infty$. One can use a labeled directed graph to generate all possible (d, k)-sequences by reading off the labels along paths in the graph. This graph is referred to as a (d, k) *constraint graph*. A graph that produces these sequences for $k < \infty$ is shown in Figure 3.1.1.

Let $N_{d,k}(n)$ be the number of distinct (d, k)-sequences of length n. The *Shannon*

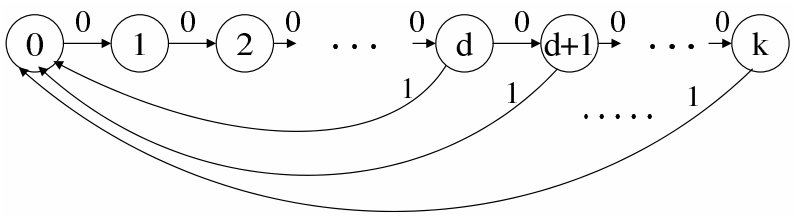


Figure 3.1.1 Constraint graph for the (d, k) constraint with $d > 0$ and k finite.

capacity of a (d, k) constraint is defined as

$$C(d,k) = \lim_{n\to\infty} \frac{\log_2 N_{d,k}(n)}{n}.$$

The capacity can be computed by applying a more general result derived by Shannon [3]. It was shown (see e.g. [1]) that

$$C(d,k) = \log_2 \lambda_{d,k},$$

where $\lambda_{d,k}$ is the largest real eigenvalue of the adjacency matrix of the constraint graph. Therefore, $\lambda_{d,k}$ is the largest real root of the characteristic polynomial of the matrix $P_{d,k}(z)$, which takes the form

$$P_{d,k}(z) = \begin{cases} z^{k+1} - \sum_{j=0}^{k-d} z^j, & k \text{ is finite} \\ z^{d+1} - z^d - 1, & k = \infty. \end{cases}$$

It was further shown that for all values of d and k the capacity exists and that $\lambda_{d,k} \in (1, 2)$ for all (d, k) pairs such that $(d, k) \neq (0, \infty)$.

The idea of constrained coding by insertion of extra bits into an uncoded data stream was introduced by Lee [4]. Bender and Wolf [5] proposed a modification to Lee's algorithm which is intended for encoding (d, k)-sequences. Their technique is known as the *bit stuffing algorithm*. The bit stuffing algorithm first converts the input sequence into a sequence having different statistical properties. It then inserts additional bits in a manner that guarantees that the resulting sequences satisfy the (d, k) constraint. Both operations are invertible so that the input sequence can be reproduced.

Bit stuffing has been used in various applications, such as in the X.25 protocol, where it has been used to ensure that the bit pattern of the frame delimiter flag will not appear

in the data sequence [9]. The emphasis of this work is on finding the most efficient bit stuffing algorithm in terms of the asymptotical rate that can be achieved.

It is well known that the maximum possible rate of a constrained code equals the capacity of the constraint [3]. The most efficient code is thus a code whose rate equals the capacity. We say that such a code *achieves capacity* or is *capacity-achieving*. Bender and Wolf [5] showed that the bit stuffing algorithm achieves capacity for all $(d, d+1)$ and (d, ∞) constraints and fails to achieve capacity for all other cases. This leaves room for improvement whenever $d + 2 \leq k < \infty$.

In this chapter, we modify the bit stuffing algorithm by flipping certain bits from the converted input sequence while the logic of insertion of extra bits remains unchanged. We name the proposed modification the *bit flipping algorithm*. We analyze the performance of both algorithms to obtain the following main results of this chapter (see Section 3.2.4 for the precise statement of Theorem 3.2.6):

Theorem 3.2.6: Let $d \geq 1$ and $d + 2 \leq k < \infty$. Then the bit flipping algorithm achieves a greater maximum average rate than the bit stuffing algorithm.

Theorem 3.3.1: Let $d \geq 0$ and $d + 2 \leq k < \infty$. Then the bit flipping algorithm achieves (d, k) capacity if and only if $d = 2$ and $k = 4$.

In Section 3.2, we give an example that motivated the idea of flipping. In this section, we study in detail both the bit stuffing and the bit flipping algorithms. We devote the rest of Section 3.2 to establishing a sequence of lemmas needed to prove the performance improvement (Theorem 3.2.6). In Section 3.3, we characterize all (d, k) constraints for which the bit flipping algorithm achieves capacity (Theorem 3.3.1).

3.2 Improving the Performance of Bit Stuffing by Bit Flipping

In this section, we introduce the bit stuffing algorithm and propose a modification to it - the bit flipping algorithm. We derive explicit expressions for the asymptotic average rates of both algorithms. We use these expressions to show that the proposed algorithm yields a higher average rate than the bit stuffing algorithm for all $d \geq 1$ and $d + 2 \leq k < \infty$.

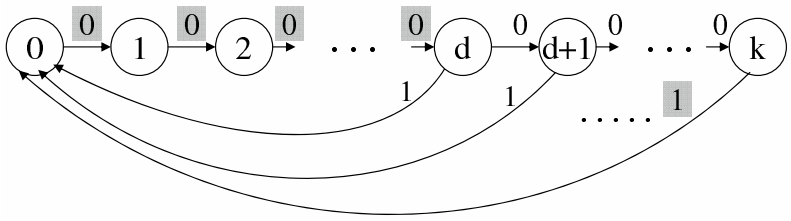
3.2.1 The Bit Stuffing Algorithm

We begin by describing the bit stuffing encoder, which encodes arbitrary data sequences into (d, k)-constrained sequences. The encoder consists of the following two components:

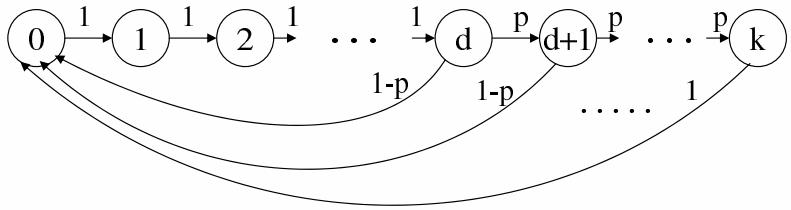
- A binary distribution transformer,
- A constrained encoder.

Assume that the input is a sequence of independent and identically distributed unbiased (i.e., Bernoulli with probability $\frac{1}{2}$) random bits. The *binary distribution transformer* (DT) converts the unbiased sequence into a sequence of independent bits, whose probability of a 0 is some $p \in [0, 1]$ (i.e., Bernoulli with probability p). We say that the output sequence is *p-biased* and refer to it as the *biased sequence*. We also refer to p as the *bias*. This conversion can be implemented in a one-to-one manner. Hence we can apply the reverse transformation to recover the unbiased data. The asymptotic expected rate of such a scheme is $h(p)$, where $h(p) = -p \log_2(p) - (1 - p) \log_2(1 - p)$ is the binary entropy function. A possible method of conversion would be to use the Elias code [6, pp. 61-62]. However, this approach was designed for infinite input sequences. One modification to this idea that applies to finite sequences and can be implemented using a finite precision arithmetic appears in [7]. A brief overview of other applicable methods appears in the introduction of Chapter 4 in the context of general (i.e., non-binary) DT's.

The *constrained encoder*, also referred to as the *bit stuffer*, inserts extra bits into the biased sequence in order to avoid possible violations of the (d, k) constraint. It writes the biased sequence while keeping track of the number of consecutive zeros in the sequence, called the *run length*. Once the run length equals k, the bit stuffer inserts a 1 followed by d 0's. This guarantees that both the d and k restrictions are satisfied. Whenever encountering a biased 1, the bit stuffer inserts d 0's so as to satisfy the d limitation. The inserted bits are also called *stuffed bits*. The graph in Figure 3.2.1(a) describes the bit stuffer, where the edge labels are the output symbols. The stuffed bits are highlighted. Note that for $d > 0$ and finite k the bit stuffer is in fact a realization of the graph in Figure 3.2.1(b), which is similar to the constraint graph in Figure 3.1.1 except for the edge labels. Here the edge labels represent the probabilities that the next bit will assume



(a)



(b)

Figure 3.2.1 Graph description of a bit stuffing encoder for the (d, k) constraint with $d > 0$ and k finite.

the value 0 or 1. If the bit assumes a value of 0, then we move to the next state to the right. If it is a 1, then we return to state 0. We will find this graph representation of the bit stuffer useful in Sections 3.2.2 and 3.2.3.

The decoder is comprised of the corresponding two components, arranged in a reverse order. The constrained decoder reads the encoded constrained sequence and keeps track of the run length. Whenever reaching a run length of k, it deletes the $d + 1$ stuffed bits that follow it. If it encounters a 1, then it removes the next d stuffed 0's. This results in the encoded p-biased sequence, which is then fed into the inverse distribution transformer, so as to obtain the original unbiased data.

The proposed scheme produces a variable rate code. Its expected rate is the product of the expected rates of the two components, the first being $h(p)$ when the code length goes to infinity. Note that we could directly apply just the bit stuffer component to the unbiased input data to obtain a (d, k) constrained sequence. However, adding the DT in fact results in an improved overall average rate. This sheds some light on the role of the transformer,

namely to better fit the data to the constraint. To further explain, observe that each 1 in the biased sequence results in d stuffed 0's. On the other hand, $k-d$ consecutive biased 0's will result in the stuffing of a single 1 followed by d 0's. Thus, as $k-d$ increases we would expect that fewer 1's in the input sequence will result in fewer stuffed bits. Such sequences will yield a higher rate in the bit stuffing encoding process. The distribution transformed sequences have this desired property on average. On the other hand, as we increase the probability of a 0, the rate of the first component $h(p)$ decreases. Thus, we need to optimize p in order to maximize the average overall rate. Optimization is done numerically due to the complexity of the rate expression. We shall show in Section 3.2.4 that, as expected, having more 0's than 1's indeed yields a better overall rate for most cases where $d > 0$, though this is not always true.

Bender and Wolf [5], [8] analyzed the performance of the bit stuffing algorithm by deriving an expression for its average asymptotic rate. We limit our discussion to a finite k and follow the methods of their derivation. We will later show that an infinite k is not of interest to our work due to the fact that bit stuffing achieves capacity in this case. We start by modeling the constrained (d,k)-sequences by a one-state constraint graph, depicted in Figure 3.2.2(a). The edges in our graph represent the allowable runs of consecutive 0's followed by a 1. In order to get a description of the corresponding biased input data sequences we remove the stuffed 0's and 1's and obtain the graph in Figure 3.2.2(b).

We can now calculate the average rate of the constrained encoder component. Having assumed that the biased sequence is i.i.d. Ber(p), the average input length is

$$L_{in} = \sum_{j=0}^{k-d-1} (j+1)p^j(1-p) + (k-d)p^{k-d} = \frac{1-p^{k-d}}{1-p}$$

for all p such that $0 \leq p < 1$, and the average output length is

$$L_{out} = L_{in} + d + p^{k-d}.$$

Therefore, the asymptotic average information rate of the algorithm is

$$I(p,d,k) = \frac{L_{in}}{L_{out}} \times h(p) = \frac{(1-p^{k-d})h(p)}{1-p^{k-d+1}+d(1-p)}$$

for all p such that $0 \leq p < 1$ and $I(p,d,k) = 0$ for $p = 1$.

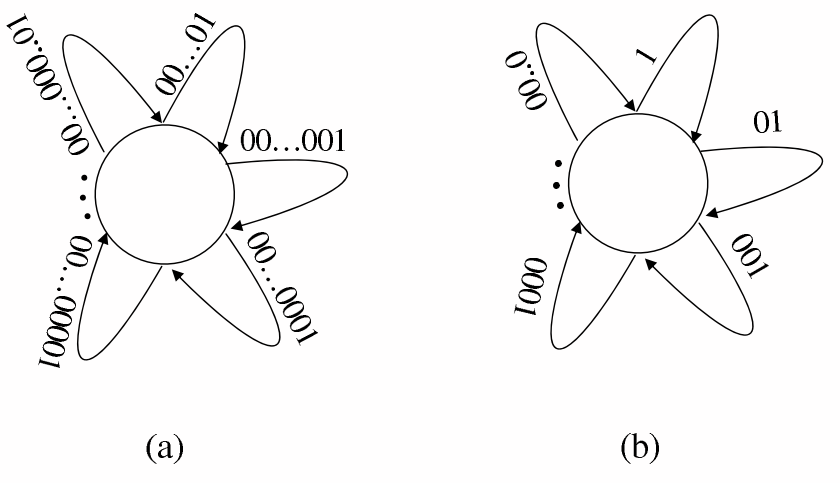


Figure 3.2.2 Graph description of a (d, k) constraint where the edge labels are the allowable runs.

Bender and Wolf [5], [8] used this expression to obtain a characterization of all cases where the bit stuffing algorithm is capacity-achieving, i.e., its maximum average rate equals the capacity of the (d, k) constraint. Their results are summarized in the following proposition.

Proposition 3.2.1. *The bit stuffing algorithm for (d, k) constraints achieves (d, k) capacity for the following cases:*

- $k = d + 1$ *for all* $d \geq 0$
- $k = \infty$ *for all* $d \geq 0$.

It does not achieve capacity for all other values of d and k.

For the remaining *non* capacity-achieving cases, numerical optimization of the average rate shows that bit stuffing codes achieve rates that are very close to capacity. Thus, the algorithm is said to be *near-capacity achieving* for these cases. For detailed results see [8].

3.2.2 Motivating Example - Maxentropic Measure for the $(2, 4)$ Case

In this subsection we demonstrate an example that triggered the development of the bit flipping algorithm. Before we go through the example we need to introduce the notion

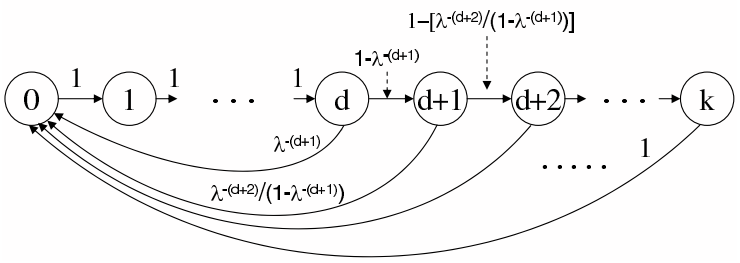


Figure 3.2.3 Edge probabilities for maxentropic (d, k)-sequences.

of the *maxentropic measure*.

Consider a constraint graph where we assign nonzero probabilities to the labeled edges leaving each state, thus producing an information source. A result by Shannon [3] states that for any such graph one can always assign certain probabilities to the edges at each state, such that the resulting constrained sequences have maximum entropy. This set of probabilities can be computed by formulas that Shannon prescribed and is called the *maxentropic measure*. Shannon further showed that this maximum entropy is equal to the capacity of the constrained system. An encoding scheme is thus capacity-achieving if it induces a maxentropic measure on its generated constrained output. Applying Shannon's result to (d, k) constraints (see [1] for a complete derivation) yields the probabilities shown in Figure 3.2.3, where $\lambda_{d,k} = 2^{C(d,k)}$.

Recently, Wolf [9] proposed a modification to bit stuffing based on Shannon's result. He showed that the modified scheme achieves rates that are equal to capacity for all values of d and k. The idea is to let the bit stuffer realize the maxentropic measure by feeding it with several distinct biased streams. For example, when k is finite we have states $d, d+1, \cdots, k-1$ with two edges exiting from each state, while the other states have only one exiting edge. Each pair of emanating edges corresponds to a random bit with a certain bias. The single edges correspond to stuffed bits. Denote the maxentropic probability when moving from state i to state $i+1$ by p_i, then $p_i = 1$ for $i = 0, 1, \cdots, d-1, k$. We first "break" the unbiased data into $k-d$ distinct streams, denoted $S_d, S_{d+1}, \cdots, S_{k-1}$, and input the streams into $k-d$ different DT's with biases $p_d, p_{d+1}, \cdots, p_{k-1}$. Note that each stream is fed into exactly one of the transformers. This results in $k-d$ biased streams, $S_d^*, S_{d+1}^*, \cdots, S_{k-1}^*$, with biases $p_d, p_{d+1}, \cdots, p_{k-1}$ respectively. Having multiple biased streams, the bit stuffer takes a bit from stream S_i^* when in state i. Clearly, the encoded

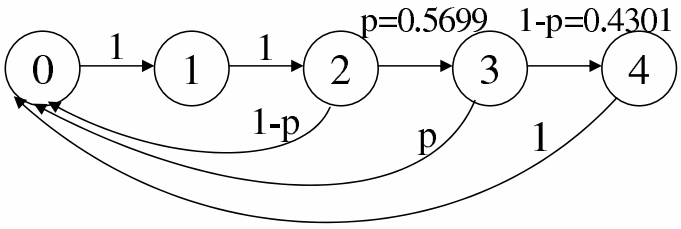


Figure 3.2.4 Edge probabilities for the $(2, 4)$ maxentropic measure.

sequences have maximum entropy.

Let us look at the maxentropic measure for the $(2, 4)$ case, shown in Figure 3.2.4. Denote the probability of moving from state 2 to state 3 by p^*. It turns out (as will be confirmed in Section 3.3) that the probability of moving from state 3 to state 4 equals $1-p^*$, where $p^* \approx 0.5699$. This special property suggests that in this case we do not need two DT's in order to achieve capacity. We can use a single transformer with $\mathsf{Pr}(0) = p^*$ and modify the bit stuffing algorithm to obey the following rule:

- If the current run length equals 2 then write the next biased bit.
- If the current run length equals 3 then write the complement of the next biased bit, i.e., *flip* the next biased bit before writing.

In other words, flip the biased bit only when in state 3.

Recall that bit stuffing does not achieve capacity in the $(2, 4)$ case. Yet, the addition of bit flipping resulted in a capacity-achieving algorithm in this case. Thus, at least in this case, a conditional bit flipping improves the performance of bit stuffing with only a single transformer. This observation motivated us to examine whether we could do better by flipping in the general case. We generalize the flipping idea and analyze the resulting algorithm in the subsection to follow.

3.2.3 The Bit Flipping Algorithm

Consider the case where k is finite and $k \geq d + 2$ and let l be an integer such that $d + 1 \leq l \leq k - 1$. Suppose we run the bit stuffing algorithm using a single DT. We modify the logic of the bit stuffer in the following manner:

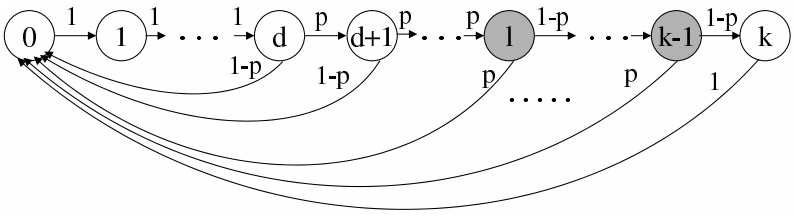


Figure 3.2.5 Graph description of a possible bit flipping encoder for the (d, k) constraint.

- If the current run length is smaller than l then write the next biased bit
- If the current run length is greater or equal to l then flip the next biased bit before writing.

In other words, flip the biased bit starting from state l. *The bit flipping algorithm* is illustrated by the constraint graph in Figure 3.2.5.

Four interrelated questions arise. What is the optimal flipping position? For which constraints can we improve bit stuffing rate by flipping? Can we achieve capacity for more constraints using flipping? If not, how far from capacity are we? In the rest of this chapter we settle these four questions. This subsection and Section 3.2.4 deal with the first two questions. Section 3.3 addresses the latter two.

We first derive an expression for the average rate. When flipping a bit starting from state l, the average biased input length is

$$\begin{aligned} L_{in} = & \sum_{j=0}^{l-d-1} (j+1)p^j(1-p) \\ & + \sum_{j=0}^{k-l-1} (l-d+j+1)p^{l-d}(1-p)^j p \\ & + (k-d)p^{l-d}(1-p)^{k-l} \\ = & \frac{1-p^{l-d}}{1-p} + p^{l-d-1}(1-(1-p)^{k-l}) \end{aligned}$$

and the average output length is

$$L_{out} = L_{in} + d + p^{l-d}(1-p)^{k-l}.$$

The asymptotic overall average rate $R(p,l,d,k)$ is given by:

$$\begin{aligned} R(p,l,d,k) &= \frac{L_{in}}{L_{out}} \times h(p) \\ &= \frac{h(p)}{1+\frac{d+p^{l-d}(1-p)^{k-l}}{L_{in}}} \\ &= \frac{[p^{l-d-1}(1-2p-(1-p)^{k-l+1})+1]h(p)}{p^{l-d-1}(1-2p-(1-p)^{k-l+2})+1+d(1-p)} \end{aligned}$$

for all p such that $0 \leq p < 1$ and for all l such that $d+1 \leq l \leq k$, and by $R(p,l,d,k) = 0$ for $p = 1$. Note that the rate of the bit stuffing algorithm, where no flipping occurs, is a special case of this expression with $l = k$, i.e., $R(p,k,d,k) = I(p,d,k)$.

Our main goal is to show that under certain conditions the proposed algorithm can achieve a better average rate than bit stuffing. However, the next lemma suggests a more extensive result. It states that when using a transformer with a bias greater than 0.5, state $k-1$, i.e., one state before the last, is always the optimal state for flipping. A special case of this result is that $R(p,k,d,k) < R(p,k-1,d,k)$ when $0.5 < p < 1$. In other words, when l is set to $k-1$, the bit flipping algorithm performs better than bit stuffing for the constraints given in Lemma 3.2.2. As one can observe in the proof, it is actually straightforward to prove this special case. Nonetheless, we are looking for the best possible performance. Moreover, finding that the optimal flipping position is always $k-1$ provides a simple and general formulation of the algorithm, which is independent of d and k for the considered cases.

Lemma 3.2.2. *Let $d \geq 1$, $d+2 \leq k < \infty$ and $0.5 < p < 1$. Then*

$$R(p,l,d,k) < R(p,k-1,d,k)$$

for all l such that $d+1 \leq l \leq k-2$ or $l = k$.

Proof. Define

$$\begin{aligned} A_l &= \frac{d+p^{l-d}(1-p)^{k-l}}{L_{in}} \\ &= \frac{d+p^{l-d}(1-p)^{k-l}}{1-p^{l-d}+(1-p)p^{l-d-1}(1-(1-p)^{k-l})}(1-p). \end{aligned}$$

Then we can write

$$R(p,l,d,k) = \frac{h(p)}{1+A_l}. \tag{3.2.1}$$

Clearly, $A_l \geq 0$ for all $d+1 \leq l \leq k$. Therefore, $A_i > A_j$ if and only if $R(p,i,d,k) < R(p,j,d,k)$. We now show that A_{k-1} is strictly smaller than any other A_l. First observe that $A_{k-1} < A_k$ for any $0.5 < p < 1$, as can be seen directly from the simplified forms

$$A_k = \frac{d + p^{k-d}}{1 - p^{k-d}}(1-p)$$

and

$$A_{k-1} = \frac{d + p^{k-d-1}(1-p)}{1 - p^{k-d}}(1-p).$$

In order to prove that $A_{k-1} < A_l$ for any $d+1 \leq l < k-1$ and $0.5 < p < 1$ it suffices to show that $A_l < A_{l-1}$ for any $d+2 \leq l \leq k-1$.

It is easy to verify that the denominators of

$$A_l = \frac{[d + p^{l-d}(1-p)^{k-l}](1-p)}{1 - p^{l-d} + (1-p)p^{l-d-1}(1-(1-p)^{k-l})}$$

and

$$A_{l-1} = \frac{[d + p^{l-d-1}(1-p)^{k-l+1}](1-p)}{1 - p^{l-d-1} + (1-p)p^{l-d-2}(1-(1-p)^{k-l+1})}$$

are both positive for any $0 \leq p < 1$ and any $d+1 \leq l \leq k$. Therefore, multiplying the inequality $A_l < A_{l-1}$ by the product of the two denominators and dividing by $(1-p)$ we obtain the following equivalent inequality:

$$\begin{aligned}
&(d + p^{l-d}(1-p)^{k-l}) \\
&\quad \cdot(1 - p^{l-d-1} + (1-p)p^{l-d-2}(1-(1-p)^{k-l+1})) \\
&< (d + p^{l-d-1}(1-p)^{k-l+1}) \\
&\quad \cdot(1 - p^{l-d} + (1-p)p^{l-d-1}(1-(1-p)^{k-l})).
\end{aligned} \tag{3.2.2}$$

For any $0.5 < p < 1$, inequality (3.2.2) reduces to

$$(1-p)^{k-l-1}[(1-p)d + p(1-p^{l-d})] < d. \tag{3.2.3}$$

Now, observe that for any $0.5 < p < 1$ the expression in the square brackets is a convex combination of d and $1 - p^{l-d}$. Also, note that $1 - p^{l-d} < 1 \leq d$. Hence,

$$(1-p)d + p(1-p^{l-d}) < d.$$

Since $l \leq k-1$ and $0.5 < p < 1$, we have

$$0 \leq (1-p)^{k-l-1} \leq 1$$

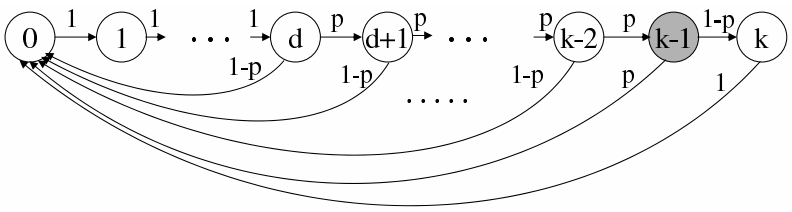


Figure 3.2.6 Graph description of an optimal bit flipping encoder for the (d, k) constraint.

implying that inequality (3.2.3) holds for any $d \geq 1$, $l \leq k-1$, and $0.5 < p < 1$, as desired. □

Looking at the graph in Figure 3.2.6 we can interpret this result as follows. At each of the states $d, d+1, \cdots, k-2$, we would rather have a 0 and move to the right than have a 1 and move back to state 0, where we would have to stuff $d > 0$ bits. However, this is no longer the case when at state $k-1$. Having a 0 will result in $d+1$ stuffed bits as opposed to d stuffed bits due to a 1. In this case, we prefer to go back to state 0 rather than move to the right, a preference reflected in the bit flipping.

Also, note that this result holds only for $p > 0.5$ and in fact is not true for $p < 0.5$. However, the former suffices for our purposes, as we will show later that the optimal bias for bit stuffing is greater than 0.5 for all cases considered but one.

3.2.4 Performance Improvement

Recall that the bit stuffing algorithm achieves capacity for the (d, ∞) and $(d, d+1)$ constraints for any d and does not achieve capacity for all other cases. Therefore, there is room for performance improvement for all (d, k) constraints such that $d + 2 \leq k < \infty$ and $d \geq 0$. In this subsection, we prove that the bit flipping algorithm achieves a higher rate than bit stuffing for most of these constraints.

As mentioned earlier, the result of Lemma 3.2.2 is limited to transformers with a bias greater than 0.5. Since the optimal bias for bit stuffing may be smaller than 0.5, Lemma 3.2.2 does not guarantee that flipping at state $k-1$ is superior to bit stuffing. Because of the complexity of the bit stuffing rate derivative, we cannot find an explicit form for the optimal bias. Instead, in the next sequence of lemmas we examine the rate derivative to show that the optimal bias for bit stuffing is indeed greater than 0.5 for the

following (d, k) pairs:

- $d + 3 \leq k < \infty$ and $d \geq 1$
- $k = d + 2$ and $d \geq 2$.

This result together with Lemma 3.2.2 guarantees the performance improvement in these cases.

Lemma 3.2.3. *Let* $0 < p \leq 0.5$, $d \geq 1$ *and* $d + 3 \leq k < \infty$, *then*

$$\frac{dI(p, d, k)}{dp} > 0.$$

Proof. Consider the bit stuffing rate derivative

$$\frac{dI(p, d, k)}{dp} = \frac{\frac{d\Big((1-p^{k-d})h(p)\Big)}{dp} f(p) - \frac{d\Big(f(p)\Big)}{dp}[(1 - p^{k-d})h(p)]}{(f(p))^2}$$

where $f(p) = 1 - p^{k-d+1} + d(1 - p)$. We denote the derivative's numerator by $I'_{num}(p)$ and shall show that $I'_{num}(p) > 0$ for all $0 < p \leq 0.5$, $d \geq 1$ and $k - d \geq 3$. By rearranging terms we can rewrite

$$\begin{aligned} I'_{num}(p) = \; & h(p) \cdot \Big[\; -(k-d)p^{k-d-1}[1 - p^{k-d+1} + d(1-p)] \\ & \qquad + [(k-d+1)p^{k-d} + d](1 - p^{k-d}) \; \Big] \\ & + \tfrac{dh(p)}{dp} \cdot \Big[\, 1 - p^{k-d+1} + d(1-p) \, \Big](1 - p^{k-d}). \end{aligned}$$

Defining $A = 1 - p^{k-d+1} + d(1 - p)$, $B = (k - d + 1)p^{k-d} + d$, and $C = 1 - p^{k-d}$, we see that $A, B, C > 0$ for all $0 < p \leq 0.5,\ d \geq 1,\ k - d \geq 3$, and

$$I'_{num}(p) = h(p)\Big[- (k-d)p^{k-d-1}A + BC\Big] + \frac{dh(p)}{dp}AC.$$

Now, $\frac{dh(p)}{dp} \geq 0 \ \ \forall\, 0 < p \leq 0.5$, implying that $\frac{dh(p)}{dp}AC \geq 0$. Hence, since $h(p) > 0$ we need only show that

$$[-(k-d)p^{k-d-1}A + BC] > 0 \tag{3.2.4}$$

for the given values of p, k, and d. Writing the latter expression explicitly and rearranging terms yields

$$\begin{aligned}[-(k-d)p^{k-d-1}A + BC] &= p^{k-d}\left[(k-d-1)(d+1)+2-p^{k-d}\right] \\ &\quad + d - p^{k-d-1}(k-d)(d+1) \\ &> 0.\end{aligned}$$

We distinguish between two cases: $k-d \geq 4$ and $k-d=3$.

In the first case we use the fact that

$$(k-d-1)(d+1)+2-p^{k-d} \geq 3\times 2+2-\frac{1}{16} > 0$$

hence,

$$p^{k-d}[(k-d-1)(d+1)+2-p^{k-d}] > 0 \qquad \forall \;\; 0<p\leq 0.5,\;\; d\geq 1,\;\; k-d\geq 4.$$

It follows that showing that

$$d - p^{k-d-1}(k-d)(d+1) \geq 0$$

will guarantee that inequality (3.2.4) holds. Define $l = k-d$; then, we want to prove that $lp^{l-1}(d+1) \leq d$ or

$$lp^{l-1} \leq \frac{d}{d+1}, \qquad \forall \;\; l \geq 4. \tag{3.2.5}$$

We first consider the derivative of the left-hand side of inequality (3.2.5)

$$\frac{d(lp^{l-1})}{dl} = p^{l-1} + lp^{l-1}\ln(p) = p^{l-1}\left(1 - l\ln(\frac{1}{p})\right).$$

For any $0<p\leq 0.5$ we have $0.693 \approx \ln(2) \leq \ln(\frac{1}{p}) < \infty$, which implies that $l\ln(\frac{1}{p}) > 1$ and $\frac{d(lp^{l-1})}{dl} < 0$. Consequently, for any $0<p\leq 0.5$ and any $l \geq 4$ we have

$$lp^{l-1} \leq 4p^3 \leq 4p^3\Big|_{p=\frac{1}{2}} = \frac{1}{2}.$$

The right-hand side of inequality (3.2.5) is lower bounded by $\frac{1}{2}$ for all $d \geq 1$, yielding

$$lp^{l-1} \leq \frac{1}{2} \leq \frac{d}{d+1}, \qquad \forall\, d\geq 1,\;\; l\geq 4,\;\; 0<p\leq 0.5.$$

In the second case, we assign $k-d=3$ in the left-hand side of inequality (3.2.4) and get

$$[-(k-d)p^{k-d-1}A+BC]=d[1+2p^3-3p^2]+4p^3-p^6-3p^2.$$

Since $1+2p^3-3p^2$ is positive for all $0<p\leq 0.5$, then inequality (3.2.4) holds if and only if $\frac{3p^2+p^6-4p^3}{1+2p^3-3p^2}<d$. Instead, we show that

$$3p^2+p^6-4p^3<1+2p^3-3p^2 \tag{3.2.6}$$

resulting in

$$\frac{3p^2+p^6-4p^3}{1+2p^3-3p^2}<1\leq d.$$

Differentiating both sides of inequality (3.2.6) we observe that for $0<p\leq 0.5$, the left-hand side is an increasing function of p and the right-hand side is a decreasing function of p. Therefore,

$$\begin{aligned}3p^2+p^6-4p^3\leq [3p^2+p^6-4p^3]\Big|_{p=\frac{1}{2}} &= \frac{17}{64}\\ &< \frac{1}{2}=[1+2p^3-3p^2]\Big|_{p=\frac{1}{2}}\\ &\leq 1+2p^3-3p^2\end{aligned}$$

confirming inequality (3.2.4). □

Lemma 3.2.4. *Let* $0<p\leq 0.5$, $d\geq 2$, *and* $k-d=2$, *then*

$$\frac{dI(p,d,k)}{dp}>0.$$

Proof. We proceed along the lines of the preceding proof and want to show that

$$[-(k-d)p^{k-d-1}A+BC]>0$$

for the given values of p,k, and d. For $k-d=2$ we need to show that $d(1-p)^2+3p^2-2p-p^4>0$. Now, for $0<p\leq 0.5$ and $d\geq 2$ we have the following inequalities: $d(1-p)^2\geq \frac{d}{4}\geq \frac{1}{2}$, $-\frac{1}{3}\leq 3p^2-2p<0$ and $-\frac{1}{16}\leq -p^4<0$. Combining the three inequalities we conclude that

$$d(1-p)^2+3p^2-2p-p^4\geq \frac{1}{2}-\frac{1}{3}-\frac{1}{16}>0.$$

□

We are now ready to conclude that the optimal bit stuffing bias is strictly greater than 0.5 for the above mentioned (d, k) pairs.

Lemma 3.2.5. *Let (d, k) satisfy one of the following conditions:*

1. $d + 3 \leq k < \infty$ *and* $d \geq 1$
2. $k = d + 2$ *and* $d \geq 2$.

Then

$$\max_{0 \leq p \leq 1} I(p, d, k) = I(p^*, d, k)$$

for some $p^* \in (0.5, 1)$.

Proof. It is easy to verify that the bit stuffing rate function $I(p, d, k)$ is continuous in p on the compact set $[0, 1]$. Thus, it attains a maximum somewhere in that set. The maximum must be attained for some $p^* \in (0, 1)$ since $I(p, d, k) = 0$ for $p \in \{0, 1\}$ and is strictly positive for any $p \in (0, 1)$. The rate derivative exists in the set $(0, 1)$, hence, a necessary condition for a maximum at $p^* \in (0, 1)$ is that $\frac{dI(p,d,k)}{dp}(p^*) = 0$. Lemmas 3.2.3 and 3.2.4 show that $\frac{dI(p,d,k)}{dp} > 0$ for any $0 < p \leq 0.5$, thus implying that the maximum is attained for some $p^* > 0.5$. □

This result brings us back to our discussion in Section 3.2.1. As said, each 1 we encounter results in d stuffed 0's, while only $k - d$ consecutive 0's result in $d + 1$ stuffed bits, or $\frac{d+1}{k-d}$ stuffed bits per biased 0. It seems that the asymmetry between $\frac{d+1}{k-d}$ and d determines the best bias. We would expect that whenever $\frac{d+1}{k-d} < d$ then inputting fewer 1's will result in fewer stuffed bits and in a better overall rate. Indeed, $\frac{d+1}{k-d} \leq d$ if and only if $d > 0$ and $k - d > 1$, with equality only when $d = 1$ and $k - d = 2$. For all other cases $\frac{d+1}{k-d} > d$. Thus, a transformer that biases the data towards more 0's ($p > 0.5$) would perform better and the optimum is achieved for $p > 0.5$. The case $(1, 3)$ is not covered by these arguments but can be analyzed explicitly. Also, note that the optimal bit flipping bias may differ from the optimal bit stuffing bias. Nonetheless, once bit flipping performs better than bit stuffing's best performance, then optimizing the bit flipping rate may even further improve its performance.

We are finally in a position to state the main result of this section. We show that for all $d \geq 1$ and $d + 2 \leq k < \infty$, flipping the biased bit at state $k - 1$ strictly improves upon bit stuffing.

Theorem 3.2.6. *Let $p^*, p^{**} \in [0,1]$ be the optimal biases for the bit stuffing and the bit flipping algorithms, respectively. Then for all $d \geq 1$ and $d+2 \leq k < \infty$ the following holds:*

$$I(p^*, d, k) < R(p^{**}, k-1, d, k).$$

Proof. Combining Lemmas 3.2.2 and 3.2.5 we obtain that

$$I(p^*, d, k) = R(p^*, k, d, k) < R(p^*, k-1, d, k) \leq R(p^{**}, k-1, d, k)$$

for all $d+3 \leq k < \infty$ and $d \geq 1$ and for $k = d+2$ and $d \geq 2$. It is left to examine the case of the $(1,3)$ constraint. In this case, we numerically optimize both algorithms' rate functions and obtain the following optimal biases and optimal rates: $p^* = 0.4906$ and $I(p^*, d, k) = 0.5456$ versus $p^{**} = 0.5557$ and $R(p^{**}, k-1, d, k) = 0.5501$. □

The result of Theorem 3.2.6 is reasonable since the asymmetry between $\frac{d+1}{k-d}$ and d still dictates a biasing of the input data towards more 0's. However, the option of flipping allows for more flexibility when fitting the data to the constraint. It enables us to change our preference at a certain state. Indeed, when reaching a run length of $k-1$ we can save a single stuffed bit by having a 1 versus a 0. Consequently, this changes our preferences in favor of 1's at this state and the opportunity to do so results in an improved rate.

We would like to point out that the case where $d = 0$ was not dealt with in Theorem 3.2.6. In this case, it is easy to show that bit stuffing is optimal for $p^* \in (0, 0.5)$ and that there is no bias for which the bit flipping algorithm can improve on the bit stuffing optimum. This observation agrees with the intuitive reasoning that was given for Lemma 3.2.5 and Theorem 3.2.6.

3.3 When Does the Bit Flipping Algorithm Achieve Capacity?

In this section, we characterize the constraints for which the bit flipping algorithm is capacity-achieving. We shall prove that, as numerical evidence suggests, bit flipping is capacity-achieving for the $(2,4)$ constraint. Moreover, we shall find that this is the only

capacity-achieving case. We conclude with performance results for the $(2, 4)$ case and for some selected other constraints.

Recall that an encoder for a (d, k) constraint is capacity-achieving if it induces a maxentropic probability measure on the generated (d, k)-sequences. A well-known property of these *maxentropic* (d, k)*-sequences* is that the probability of a run of i 0's followed by a 1 is equal to $\lambda_{d,k}^{-(i+1)}$. This property follows from results of Shannon [3] and of Zehavi and Wolf [10]. We use it in the proof of the next theorem, which outlines a complete characterization of capacity-achieving cases.

Theorem 3.3.1. *Let $d \geq 0$ and $d + 2 \leq k < \infty$. Then the bit flipping algorithm achieves (d, k) capacity if and only if $d = 2$ and $k = 4$.*

Proof. Let us parse the encoded (d, k)-sequence into a concatenation of (possibly empty) runs of 0's followed by a single 1. Let X_i be a random variable denoting the length of the i'th phrase in the parsed sequence. As mentioned earlier, the bit flipping algorithm achieves capacity if and only if it generates maxentropic (d, k)-sequences. These sequences must satisfy the following properties [1], [3], [10].

1. The X_i's are i.i.d.

2. $\mathsf{Pr}(X = i) = \lambda_{d,k}^{-i}$, where $\lambda_{d,k} = 2^{C(d,k)}$.

We start with the case where $d + 2 < k < \infty$ and then proceed to deal with $k = d + 2$.

We now use the bit flipping graph description in Figure 3.2.6 for $d + 2 < k < \infty$ in order to translate these optimality properties to the following set of $k - d + 1$ equations

$$\begin{cases} \mathsf{Pr}(X = i) = p^{i-d-1} \times (1 - p) = \lambda_{d,k}^{-i} \\ \mathsf{Pr}(X = k) = p^{k-d-1} \times p = \lambda_{d,k}^{-k} \\ \mathsf{Pr}(X = k + 1) = p^{k-d-1} \times (1 - p) = \lambda_{d,k}^{-(k+1)} \end{cases}$$

where $d + 1 \leq i \leq k - 1$. The first $k - d - 1$ equations yield

$$\frac{\mathsf{Pr}(X = i + 1)}{\mathsf{Pr}(X = i)} = p = \frac{1}{\lambda_{d,k}} \quad \forall\, d + 1 \leq i \leq k - 2. \tag{3.3.1}$$

Dividing the equation for $i = k$ by the equation for $i = k - 1$ yields

$$\frac{\mathsf{Pr}(X = k)}{\mathsf{Pr}(X = k - 1)} = \frac{p^{k-d}}{p^{k-d-2}(1 - p)} = \frac{p^2}{1 - p} = \frac{1}{\lambda_{d,k}}. \tag{3.3.2}$$

Combining (3.3.1) and (3.3.2) we have

$$\begin{aligned} & \frac{p^2}{1-p} = \frac{1}{\lambda_{d,k}} = p \\ \Leftrightarrow \quad & p(2p-1) = 0 \\ \Leftrightarrow \quad & p = 0 \quad \text{or} \quad p = \frac{1}{2}. \end{aligned}$$

Since $p = 0$ results in zero entropy and zero rate then we are left with $p = \frac{1}{2}$. However, $p = \frac{1}{2}$ implies $\lambda_{d,k} = 2$, which contradicts k being finite. Consequently, the bit flipping algorithm can never achieve capacity when $d + 2 < k < \infty$.

We now refer to the graph in Figure 3.2.6 as it appears for the special case of $k = d+2$. An argument similar to that at the beginning of the proof shows that the bit flipping algorithm achieves capacity if and only if the following three equations hold:

$$\begin{cases} \mathsf{Pr}(X = d+1) = 1 - p = \lambda_{d,k}^{-(d+1)} \\ \mathsf{Pr}(X = d+2) = p \times p = \lambda_{d,k}^{-(d+2)} \\ \mathsf{Pr}(X = d+3) = p \times (1-p) = \lambda_{d,k}^{-(d+3)}. \end{cases}$$

The first two equations reduce to

$$\begin{aligned} & \begin{cases} p = 1 - \lambda_{d,k}^{-(d+1)} \\ p = \lambda_{d,k}^{-\frac{(d+2)}{2}} \end{cases} \\ \Leftrightarrow \quad & 1 - \lambda_{d,k}^{-(d+1)} = \lambda_{d,k}^{-\frac{(d+2)}{2}} \\ \Leftrightarrow \quad & \lambda_{d,k}^{d+1} - \lambda_{d,k}^{\frac{d}{2}} - 1 = 0. \end{aligned}$$

We now substitute the two expressions we have for p into the third equation and get

$$\begin{aligned} & \lambda_{d,k}^{-\frac{(d+2)}{2}} \times \lambda_{d,k}^{-(d+1)} = \lambda_{d,k}^{-(d+3)} \\ \Leftrightarrow \quad & \lambda_{d,k}^{-(d+3)} \times [1 - \lambda_{d,k}^{-\frac{d}{2}+1}] = 0 \\ \Leftrightarrow \quad & \lambda_{d,k} = 0 \quad \text{or} \quad \lambda_{d,k}^{-\frac{d}{2}+1} = 1. \end{aligned}$$

Since $\lambda_{d,k} \in (1, 2)$ we are left with $\lambda_{d,k}^{-\frac{d}{2}+1} = 1$, which requires $-\frac{d}{2} + 1 = 0$ or $d = 2$. Consequently, the bit flipping algorithm produces a capacity-achieving code if and only if $d = 2$, $k = d + 2 = 4$ and $\lambda_{d,k}$ is a root of $z^{d+1} - z^{\frac{d}{2}} - 1$.

Table 3.3.1 Simulation results for optimal performance of bit stuffing versus bit flipping for some (d, k) constraints.

Constraint	Bit Stuff Avg. Rate	Bit Stuff Optimal Bias	Bit Flip Avg. Rate	Bit Flip Optimal Bias	Capacity	Bit Stuff $\frac{\text{Avg. Rate}}{\text{Capacity}}$	Bit Flip $\frac{\text{Avg. Rate}}{\text{Capacity}}$
(1,3)	0.5456	0.4906	0.5501	0.5557	0.5515	98.94%	99.76%
(1,4)	0.6103	0.5275	0.6157	0.5628	0.6175	98.83%	99.71%
(1,7)	0.6754	0.5831	0.6779	0.5928	0.6792	99.44%	99.81%
(2,4)	0.4006	0.5206	0.4057	0.5699	0.4057	98.74%	**100.00**%
(2,5)	0.4579	0.5634	0.4638	0.5930	0.4650	98.47%	99.74%
(3,6)	0.3680	0.5845	0.3730	0.6097	0.3746	98.24%	99.57%
(4,8)	0.3364	0.6320	0.3403	0.6480	0.3432	98.02%	99.16%
(5,9)	0.2914	0.6434	0.2946	0.6577	0.2978	97.85%	98.93%

It remains to show that $\lambda_{d,k}^{d+1} - \lambda_{d,k}^{\frac{d}{2}} - 1 = 0$ in the $(2, 4)$ case, i.e., $\lambda_{2,4}^3 - \lambda_{2,4} - 1 = 0$. Recall that $\lambda_{d,k}$ is the largest real root of the characteristic polynomial $P_{d,k}(z)$, which for a finite k takes the form

$$P_{d,k}(z) = z^{k+1} - \sum_{j=0}^{k-d} z^j.$$

When $d = 2$ and $k = 4$ we can factor $P_{2,4}(z)$ and write it as

$$P_{2,4}(z) = z^5 - z^2 - z - 1 = (z^3 - z - 1) \times (z^2 + 1).$$

Since $\lambda_{d,k}$ is real, then it must be a root of $z^3 - z - 1$, which completes the proof. □

For the remaining non capacity-achieving cases we can numerically optimize the rates of both algorithms. Table 3.3.1 shows optimal average rates for the bit stuffing and the bit flipping algorithms for a number of constraints. Also shown are the corresponding optimal biases (i.e., the probability of a 0), the capacity of each constraint and the relative performance of the algorithms.

Acknowledgment. This chapter is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," in *Proc.*

2004 IEEE Int. Symp. Inform. Theory, Chicago, IL, Jun./Jul. 2004, p. 190 and S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2885-2891, Aug. 2005.

Bibliography

[1] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, "Codes for digital recorders," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[2] B. H. Marcus, P. H. Siegel, and J. K. Wolf, "Finite-state modulation codes for data storage," *IEEE J. Select. Areas Commun.*, vol. 10, no. 1, pp. 5–37, Jan. 1992.

[3] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pt. I, pp. 379–423, Jul. 1948.

[4] P. Lee, "Combined error-correcting/modulation recording codes," Ph.D. dissertation, Univ. California, San Diego, La Jolla, 1988.

[5] P. E. Bender and J. K. Wolf, "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences," in *Proc. 1993 IEEE Int. Symp. Inform. Theory*, San Antonio, TX, Jan. 1993, p. 6.

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

[7] C. B. Jones, "An efficient coding system for long source sequences," *IEEE Trans. Inform. Theory*, vol. 27, no. 3, pp. 280–291, May 1981.

[8] P. E. Bender, "Redundancy re-organization for the magnetic channel," Ph.D. dissertation, Univ. California, San Diego, La Jolla, 1992.

[9] J. K. Wolf, "An information theoretic approach to bit stuffing for network protocols," in *Proc. 3rd Asia-Europe Workshop on Information theory*, Kamogawa, Japan, Jun. 2003, pp. 18–21. Also presented at DIMACS Workshop on Network Information Theory, New Jersey, USA, 2003.

[10] E. Zehavi and J. K. Wolf, "On runlength codes," *IEEE Trans. Inform. Theory*, vol. 34, no. 1, pp. 45–54, Jan. 1988.

4

Optimal Parsing Trees for Run-Length Coding of Biased Data

4.1 Introduction

In constrained-code design, one typically models the unconstrained user-data as a stream of independent and equiprobable bits (i.e., Bernoulli random bits with $\mathsf{Pr}(0) = \mathsf{Pr}(1) = \frac{1}{2}$). An important, but not the only, design goal is converting such inputs into constrained sequences with high efficiency. As in the previous chapter, efficiency relates to asymptotic encoding rates and the emphasis of our work is on designing efficient (d, k)-codes.

The central theme of this chapter is twofold: a study of prior (d, k)-code constructions from a source coding perspective and the construction of new (d, k)-codes based on variable input-length source codes. The idea of constructing (d, k)-codes from source codes is not new. Numerous previous constructions have arisen from the following duality between constrained coding and source coding. One first models the constrained stream as a structured source from which redundancy can be removed to form unconstrained and nearly Bernoulli$(1/2)$-distributed output. By reversing a source encoder-decoder pair, the decoder of a suitable source code is used to encode unconstrained Bernoulli$(1/2)$-distributed input into constrained sequences, in a recoverable manner. Applications to (d, k)-code design include the adaptation of arithmetic coding techniques [1], [2], [3],

pioneered by Martin, Langdon, and Todd. An interesting work by Kerpez [4] derives three (d, k)-codes from a Huffman code [5], a Tunstall code [5], [6], and an enumerative code [7]. The rates of the four above-mentioned constructions were shown to converge to the (d, k) capacity with increasing block length [4]. A principle common to all methods is that the choice of source code is guided by the special properties of *maxentropic* (d, k)-sequences. As mentioned before, such sequences are desirable as they correspond to maximizing the constrained-code rate [8]. It is well-known [8], [9] (see also Chapter 3) that they can be parsed into a concatenation of binary strings from a predefined set of size $M = k - d + 1$, where the strings are statistically independent and identically distributed (i.i.d.). The source code then serves as a *distribution transformer* between an i.i.d. M-ary maxentropic source and an i.i.d. Bernoulli$(1/2)$ source. The corresponding (d, k)-code simply applies the inverse transformation so as to induce a maxentropic distribution on the output.

An alternative design approach emerges from the literature on lossless coding of i.i.d. sources for transmission over noiseless, memoryless channels with unequal symbol-transmission costs. One can accommodate (d, k)-codes into this framework by modeling (d, k)-sequences as the outputs of a special memoryless channel [10]. This approach is closely related to our work and is much less investigated. Existing literature is mainly concerned with two types of source codes: fixed-to-variable length and variable-to-fixed length, the latter being sparsely studied [5]. The most relevant work of the first type is a recent algorithm by Golin and Rote [11], which efficiently finds a prefix-code of minimum average transmission cost per source symbol when the costs are integers. An application to (d, k)-codes is straightforward and appears in the paper. As for the second type, Lempel, Even, and Cohn [12] derived an algorithm for constructing a prefix-free code of minimum average transmission cost per source symbol when the source symbols are equiprobable. Section 4.3 in this chapter provides more details on this method, in the context of (d, k) constraints. It is interesting to note that both papers do not view the problem from an information-theoretic standpoint, but rather treat it from a combinatorial optimization perspective. As such, they are not concerned with maxentropic distributions. Our work relates to (d, k)-codes of the second type, but relaxes the equiprobable source assumption.

This work builds upon three prior (d, k)-code constructions: the *bit stuffing*, *bit flipping*, and *symbol sliding* algorithms. Recall that in Chapter 3, we showed that a simple modification to the bit stuffing algorithm can achieve improved average rates for most (d, k) constraints. In a following work, Sankarasubramaniam and McLaughlin [13], [14] generalized both bit stuffing and bit flipping into an improved code construction, called the *symbol sliding algorithm*. One of their key insights in [13] was an interpretation of bit stuffing and bit flipping as applying bijective mappings between two distinct predetermined sets of binary strings. Symbol sliding is essentially an adjustment of the mapping to obtain further improved rates. Indeed, they demonstrate rate gains over bit flipping for several constraints and prove that symbol sliding additionally achieves capacity for all $(d, 2d + 1)$ constraints.

The constructions we consider here have a distinctive characteristic - a binary DT as a first encoding step. Although one can directly (d, k)-encode the standard equiprobable input, it turns out that the introduction of a bias into the data is key to achieving improved asymptotic rates (see Chapter 3 and [13]). Intuitively speaking, this transformation better conforms the data to the characteristics of maxentropic (d, k)-sequences [13]. This in turn leads to improved rates at the constrained encoding step and to improved *overall* rates. It is important to note that the binary DT is a special case of general distribution transformers, such as the ones introduced by Kerpez [4] and others [1]. In fact, some of the above-noted methods can be readily applied to the binary case, where a Bernoulli(p) distribution replaces the M-ary maxentropic one. Hence, a direct transformation to a maxentropic distribution has a similar implementation to a scheme that uses a binary DT. Still, the schemes presented here provide alternative methods of approximating a maxentropic distribution. Since they perform the actual constrained coding on a biased source, the challenge is to approximate the target distribution with this non-conventional source, while using simple techniques. This is, in a sense, joint source and constrained coding.

In this work, we first examine bit stuffing, bit flipping and symbol sliding from a source coding viewpoint. This new perspective has motivated us to extend them into a general framework for constructing (d, k)-codes from variable-length source codes. We first put the framework in the context of relevant source coding literature. We next demon-

strate that it gives rise to new code constructions which further improve upon the three aforementioned algorithms. This prompts us to search for optimal codes under the general framework, optimal in the sense of maximal achievable asymptotic rates. Nevertheless, finding such codes is a complex and difficult problem. We therefore resort to studying a simplified related problem, where we seek an optimal (d, k)-code for an i.i.d. Bernoulli(p)-distributed source. In this case, some interesting properties of optimal (d, k)-codes arise, leading to a simplified solution for a partial class of (d, k) constraints. The solution makes use of the Tunstall algorithm [15], which was originally developed to generate optimal variable-to-fixed length source codes.

The rest of the chapter is organized as follows. The symbol sliding algorithm is reviewed in Section 4.2. In Section 4.3, we outline a framework, taken from the source coding literature, for variable-length codes for noiseless memoryless channels with arbitrary transmission costs. Here, we introduce notations and basic source coding concepts as well as survey relevant algorithms and results. Section 4.4, which is the essence of the chapter, is devoted to studying (d, k)-codes that are based on variable-length source codes. We conclude in Section 4.5 with some related open problems and with a discussion of the core differences between the various constructions.

4.2 Background: the Symbol Sliding Algorithm

The work presented in this chapter was inspired by the recently suggested interpretation of bit stuffing and bit flipping and by their generalization to the symbol sliding algorithm [13]. In this section, we describe the above-mentioned interpretation and review the symbol sliding algorithm.

Recall that for a finite k, any (d, k)-sequence can be viewed as a unique concatenation of strings, each string corresponding to an allowable run of consecutive 0's followed by a 1. Let $\Gamma_{d,k} = \{0^d1, 0^{d+1}1, \cdots, 0^{k-1}1, 0^k1\}$ be the set of pertinent strings, where 0^t stands for a run of t consecutive 0's. Throughout the chapter, we refer to the strings in $\Gamma_{d,k}$ as *constrained phrases*. Note that when $k = \infty$, there exists an equivalent description that uses strings from $\Gamma_{d,\infty} = \{0, 0^d1\}$.

Let us start by taking a closer look at the bit stuffer. Consider its graphical description

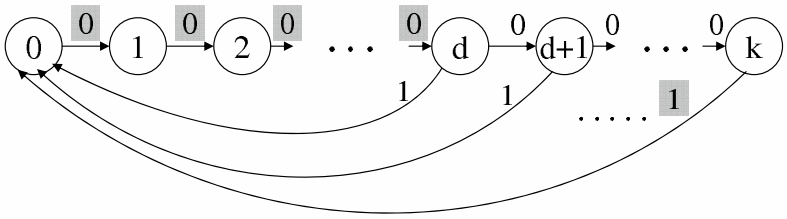


Figure 4.2.1 Graph description of a bit stuffer for a (d, k) constraint with $d > 0$ and k finite.

in Figure 4.2.1, and suppose that we start at state zero. We next stuff d 0's and progress to state d, from which we continue walking on the graph by reading the p-biased bits. At some point, after reading at least one and at most $k - d$ bits, we return to the zero state, thus completing a cycle on the graph. Encoding continues while repeatedly completing cycles at the zero state. By observing that the output generated at each cycle corresponds to one of the constrained phrases in $\Gamma_{d,k}$, it is possible to associate segments of the p-biased input stream with each of these phrases. By inspecting the graph, one can see that the bit stuffer associates the $k - d + 1$ input strings that are listed in Figure 4.2.2 with the various constrained phrases, according to the mapping that is specified by the arrows in the figure. To further illustrate these input-output relations, we distinguished the biased input bits from the stuffed bits by highlighting the latter. Figure 4.2.2 also lists the probability of occurrence of each of the input strings as a function of the bias p. The generated constrained phrases are statistically independent and obey the same distribution

$$P_{BS}^{d,k}(p) = \left\{1-p, p(1-p), \cdots, p^{(k-d-1)}(1-p), p^{(k-d)}\right\}, \tag{4.2.1}$$

where position i ($0 \leq i \leq k - d$) represents the probability of the phrase $0^{d+i}1$. At this point, we note that these relations were demonstrated earlier in Chapter 3 by the one-state constraint graph in Figure 3.2.2.

A similar interpretation of the bit flipping algorithm is demonstrated in Figure 4.2.3. First observe that bit stuffing and bit flipping operate identically on the top $k - d - 1$ input strings. The algorithms differ in the constrained phrases that they associate with each of the two input strings on the bottom. The flipping of the bit at state $k-1$ translates into a switching between the two associated phrases. More precisely, the longest phrase 0^k1 is now associated with the input $0^{k-d-1}1$, whereas the second longest phrase $0^{k-1}1$

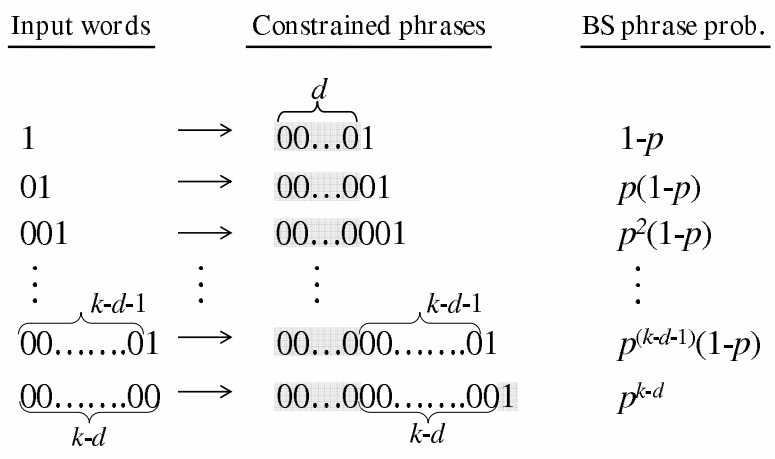


Figure 4.2.2 The bit stuffer induces a mapping between a set of input words and the set of (d, k)-constrained phrases.

is associated with the input 0^{k-d}. The algorithm is specified by the modified associations between the two leftmost lists in Figure 4.2.3. This modification has only one effect on the generated (d, k)-sequences - it changes the distribution that is induced on the constrained phrases. The two longest phrases exchange their probability of occurrence, as illustrated by the right-hand side of Figure 4.2.3.

With this interpretation in mind, we now try to explain the rate improvements gained by bit flipping, as stated in Theorem 3.2.6. We first remind the reader of the statistical characteristics of capacity-achieving (d, k)-encoders, which produce maxentropic (d, k)-sequences. As we will soon show, analysis of such encoders can guide us in improving the performance of existing encoding algorithms, such as bit stuffing. As we have noted in Chapter 3 (see Section 3.3), the output of capacity-achieving encoders has the following properties:

1. The constrained phrases are statistically independent and identically distributed.

2. The probability of a constrained phrase of length i is equal to $2^{-iC(d,k)}$, or equivalently, to $\lambda_{d,k}^{-i}$.

Following [13], we denote the maxentropic distribution of the constrained phrases as the vector

$$\Lambda_{d,k} = \left(\lambda_{d,k}^{-(d+1)}, \lambda_{d,k}^{-(d+2)}, \cdots, \lambda_{d,k}^{-(k)}, \lambda_{d,k}^{-(k+1)}\right). \tag{4.2.2}$$

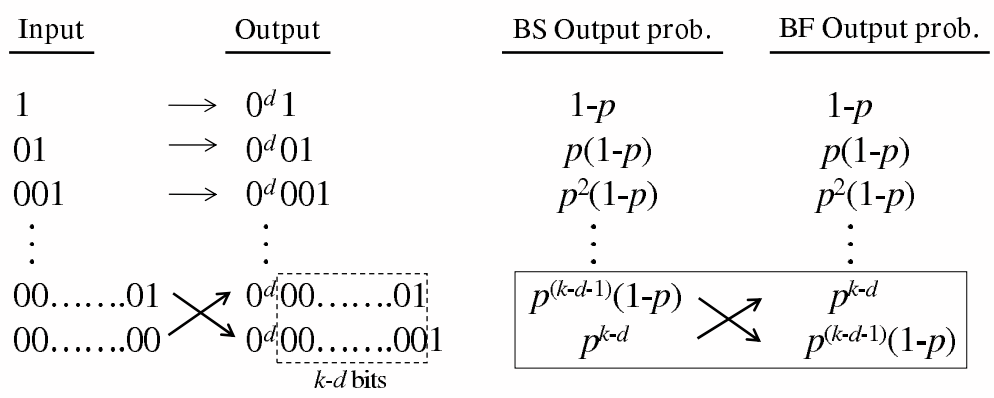


Figure 4.2.3 Bit flipping corresponds to switching between the probabilities of the two longest constrained phrases.

An apparent property of $\Lambda_{d,k}$ is the positive correlation between a phrase's length and its scarcity. Next, we find whether this property also holds for the bit-stuffing induced distribution $P_{d,k}^{BS}(p)$.

Suppose we run the BS algorithm with a bias p that is greater than 0.5. This is a reasonable assumption given that the optimum is often attained in the range $(0.5, 1)$ (see Lemma 3.2.5). Comparing the bit-stuffing phrase probabilities with the maxentropic probabilities, we see that the first $k-d$ bit-stuffing probabilities mimic the above-described behavior of the maxentropic probabilities, that is, longer phrases are less frequent. The only exception is the last probability, which satisfies $p^{(k-d)} > p^{(k-d-1)}(1-p)$ with respect to the probability preceding it, as opposed to $\lambda_{d,k}^{-(k+1)} < \lambda_{d,k}^{-(k)}$. In their paper [13], Sankarasubramaniam and McLaughlin notice that bit flipping is essentially switching between the positions of the last two probabilities. This leads them to view bit flipping as altering the bit-stuffing induced distribution so as to alleviate the discrepancy between the generated and the maxentropic phrase distributions. They argue that the modified distribution provides a better match to $\Lambda_{d,k}$ than the initial distribution, and attribute the rate gains when $p > 0.5$ (see the special case of Lemma 3.2.2 when $l = k$) to the improved match.

Is it possible to extend the idea of phrase-probability switching to obtain further improved rates? Sankarasubramaniam and McLaughlin raise this question and suggest the *symbol sliding* algorithm as a construction that results in improved rates for many con-

straints. The guiding principle of construction is to improve the matching to the maxentropic vector $\Lambda_{d,k}$. The algorithm obtains this goal by altering the associations between the constrained phrases and the bit-stuffing input words, resulting in the switching of certain phrase probabilities. *Symbol sliding with index* j, denoted by $SS(j)$, corresponds to sliding $p^{(k-d)}$ up by j positions, while pushing each of $p^{(k-d-j)}(1-p), p^{(k-d-j+1)}(1-p), \cdots, p^{(k-d-1)}(1-p)$ down by one position. Figure 4.2.4 illustrates the modified associations as well as the new induced distribution that are involved in $SS(j)$. It easily follows from the preceding discussion that BS and BF are special cases of symbol sliding with indices $j = 0$ and $j = 1$, respectively.

In practice, one can implement a symbol sliding encoder with index j by replacing the bit stuffer with a component that performs the following procedure on the biased input stream.

- If $j = 0$, then run the *bit stuffing* algorithm.
- If $j = 1$ and $d + 2 \leq k < \infty$, then run the *bit flipping* algorithm.
- If $2 \leq j \leq k - d$ and $d + 2 \leq k < \infty$, then initialize by writing d consecutive 0's. Continue by writing the biased stream while keeping track of the run length of 0's in the **encoded stream**. In parallel, perform the following operations on the encoded stream:
  1. When encountering a biased 1, insert d 0's.
  2. Once the run length equals $k - j$, insert a 0.
  3. Once the run length equals $k + 1$, replace the $k + 1$ consecutive 0's (including the inserted 0 at a run length of $k - j$) with the string $0^{k-j}10^d$.

At the decoder, the biased stream is recovered by inverting the operations of the encoder. When $j = 0, 1$, the decoder is simply the BS or BF decoder, respectively. Otherwise, we treat the case where $2 \leq j \leq k - d - 1$ and the case where $j = k - d$ differently, as described below.

- If $2 \leq j \leq k - d - 1$, discard the first d 0's in the constrained stream and set the run length to 0. Scan the rest of the stream while keeping track of the run length and applying the following two operations.

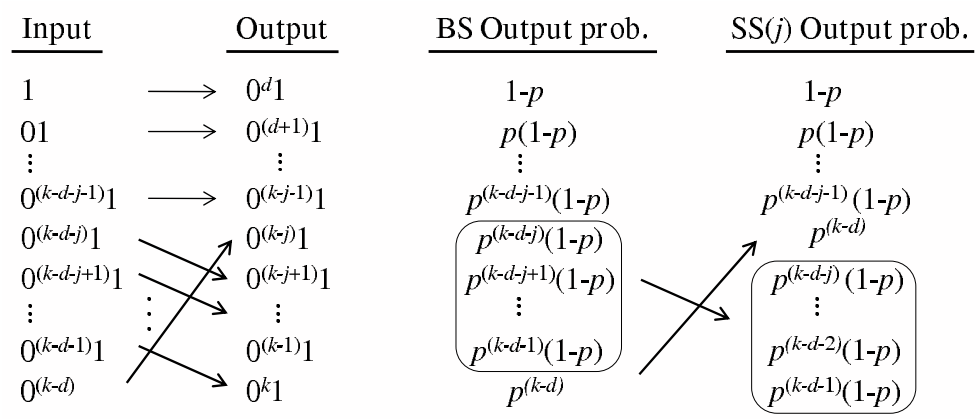


Figure 4.2.4 Symbol sliding with index j corresponds to sliding $p^{(k-d)}$ up by j positions, while pushing each of $p^{(k-d-j)}(1-p)$, $p^{(k-d-j+1)}(1-p)$, $\cdots$, $p^{(k-d-1)}(1-p)$ down by one position.

1. When encountering a 1, remove the next d 0's and set the run length to 0.
2. If the run length equals $k-d-j$ and the next bit is 0, discard the 0 bit. If the run length equals $k-d-j$ and the next bit is 1, replace the string $0^{k-d-j}10^d$ with the string 0^{k-d} and set the run length to 0.

- If $j = k-d$, discard the first d 0's in the constrained stream and check the value of the next bit. If it is 0, discard it. Otherwise, insert $k-d$ consecutive 0's *before* it. Perform the following two scans consecutively.

  Scan 1 When encountering a 1, remove the next d 0's and check the value of the next bit. If it is 0, discard it and resume scanning, starting at the next bit. If it is 1, insert $k-d$ consecutive 0's *before* it and resume scanning, starting at the current 1 (i.e., move the scanner pointer to this bit and repeat **Scan 1**).

  Scan 2 Scan the output of **Scan 1**. Whenever encountering a 1 after $k-d$ consecutive 0's, remove the 1.

We have seen that both BS and BF are special cases of symbol sliding. It follows from Proposition 3.2.1 and Theorem 3.3.1 that symbol sliding achieves capacity for all $(d, d+1)$ and (d, ∞) constraints, as well as for the $(2, 4)$ constraint. The following proposition extends the capacity-achieving property of symbol sliding to an additional

class of (d, k) constraints, namely, the class of $(d, 2d + 1)$ constraints [13].

Proposition 4.2.1. *Let $0 \leq d + 2 \leq k < \infty$ and $2 \leq j \leq k - d$. The symbol sliding algorithm with index j achieves (d, k) capacity if and only if $k = 2d+1$ and $j = k-d = d + 1$.* □

Proposition 4.2.1 further states that there are no other constraints for which symbol sliding achieves capacity. Nevertheless, for all remaining constraints, symbol sliding introduces the sliding index j as an additional parameter to optimize. This provides more flexibility in fitting the resulting distribution to the maxentropic target distribution, compared to the rate optimization of BS and BF. Numerical optimization results that are reported in [13] demonstrate that symbol sliding indeed improves over BS and BF for some constraints, yet not for all. Since optimization is essentially performed jointly over p and j, it is important to note the following proposition, which identifies certain relations between the bias and the sliding index [13].

Proposition 4.2.2. *Let $0 \leq d < k < \infty$. Then for $0 < j \leq k - d$, the average rate of $SS(j)$ is greater than the average rate of $SS(j - 1)$ if and only if $p^j + p > 1$.* □

Unfortunately, Proposition 4.2.2 does not suffice to deduce the superiority of a certain sliding index for a given (d, k) constraint, as is established by Theorem 3.2.6 for the bit flipping algorithm (i.e., for $j = 1$). It is only possible to infer that if $SS(j - 1)$ is optimal for a bias p such that $p^j + p > 1$, then the optimized $SS(j)$ will outperform the optimized $SS(j - 1)$. The difficulty thus is in determining the range in which the optimal bias of $SS(j - 1)$ falls. Proposition 4.2.2 does, however, provide some insight into the jointly optimal values by implying the optimal sliding index for p. Specifically, it follows from the proposition that $SS(j^\star)$ maximizes the rate at a bias p if and only if $1 - p \leq p^{j^\star} \leq (1-p)/p$. Equivalently, the optimal index $j^\star$ must satisfy the condition $p^{(k-d-j^\star)}(1-p) \leq p^{(k-d)} \leq p^{(k-d-j^\star-1)}(1 - p)$, which is indicative of the relations between the optimally-shuffled constrained-phrase probabilities. In other words, the best performance is attained when positioning p^{k-d} in a location such that the induced probabilities form a decreasing series, as illustrated in Figure 4.2.5. Recall that this is also the case with the maxentropic probabilities. The reason for pointing this out at this stage is that it will prove relevant to the different perspective on symbol sliding we present in Section 4.4.1.

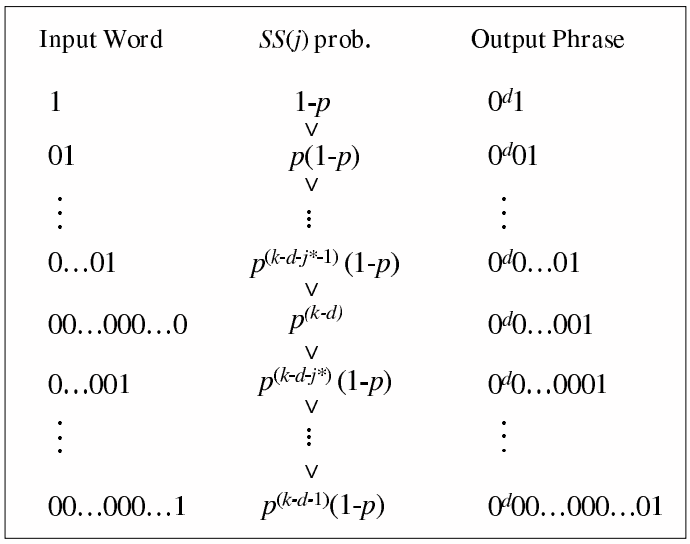


Figure 4.2.5 The optimal sliding index, $j^\star$, depends on p and satisfies $p^{(k-d-j^\star)}(1-p) < p^{(k-d)} < p^{(k-d-j^\star-1)}(1-p)$.

4.3 Preliminaries: Variable-Length Source Codes for Noiseless and Memoryless Channels

Consider the system depicted in Figure 4.3.1. A memoryless binary information source produces p-biased sequences for a given bias p. The sequences are to be transmitted over a memoryless, noiseless channel which admits an alphabet of K symbols $\Sigma = \{\alpha_1, \alpha_2, \ldots, \alpha_K\}$. Each channel symbol α_i has an associated transmission cost c_i. For notational convenience, we assume that the channel symbols are given ordered by nondecreasing cost, i.e., $c_1 \leq c_2 \leq \cdots \leq c_K$. An encoder converts the p-biased binary sequences into K-ary channel-admissible sequences by parsing the input stream into binary strings, hereafter called *source words*, and by replacing each source word with a channel symbol. A coding scheme of this type uses a *code*, which can be thought of as a dictionary with K entries that correspond to each of the source words in a predetermined set $W = \{w_1, w_2, \cdots, w_K\}$. The parsing step partitions the input stream into a concatenation of words from the dictionary W. The code additionally specifies a bijective assignment of the K channel symbols to the words in W. The replacement step replaces each dictionary string with its assigned channel symbol. Thus, a code has two parameters: a set of source

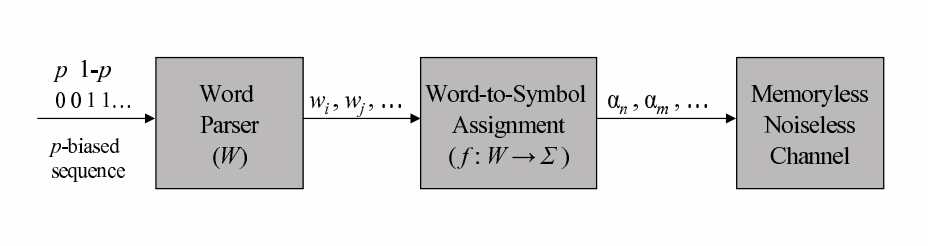


Figure 4.3.1 Block diagram of a system for variable-length encoding of p-biased sequences for transmission over a memoryless, noiseless channel.

words W and an assignment $f : W \rightarrow \Sigma$ of the output channel symbols to the input source words. Throughout the remainder of this chapter, we restrict our attention to codes that use exhaustive and prefix-free source-word sets. Whereas exhaustivity is required to guarantee the parsing of any input stream, the prefix property, which leads to unique parsing and encoding without delay, is not necessary for the assurance of unique decodability of the input. Word-sets that are exhaustive and prefix-free are said to be *complete*.

When studying prefix-free word sets, it is convenient to work with their tree representations. Specifically, there is a well-known bijection between complete word sets and *complete labeled trees*. Such trees are also called *parsing trees* in the source-coding literature [5]. In the sequel, we will deal mostly with binary parsing trees, in which each internal node has exactly two children. We shall label the left branch with a 0 and the right branch with a 1. Each leaf node corresponds to the binary string that is read off the labels along the path from the root to the leaf. If a tree T represents a word set $W = \{w_1, w_2, \cdots, w_K\}$, then we shall use the notation $T = \{w_1, w_2, \cdots, w_K\}$. To describe a code $f : W \rightarrow \Sigma$ on the tree description T of the word set W, we simply label the K leaves according to the assignment specified by f. In particular, we list the codewords $w_1, \cdots, w_K$ in a vector $V = (v_1, v_2, \cdots, v_K)$, where v_i corresponds to the word w_j such that $f(w_j) = \alpha_i$. In light of the assumption that $c_1 \leq c_2 \leq \cdots \leq c_K$, the labeling V lists the source words in a nondecreasing order of their associated transmission costs. Hereafter, we shall refer to a code $f : W \rightarrow \Sigma$ by specifying the tree-representation parameter pair (T, V). Figure 4.3.2 illustrates five trees which represent all complete word sets of size 4. It also shows leaf labels that correspond to different codes defined on these trees.

Suppose that we are parsing p-biased sequences into words from a given parsing

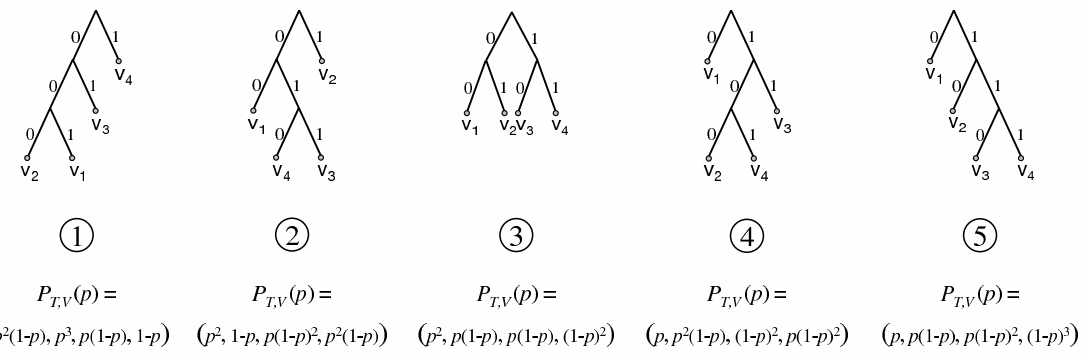


Figure 4.3.2 Tree representations of all five complete word sets of size 4, together with leaf labelings that correspond to codes defined on these word sets, and with the code-induced distributions.

tree $T = \{w_1, w_2, \cdots, w_K\}$, and that we are encoding the words using a code $V = (v_1, v_2, \cdots, v_K)$. In this case, we can compute the asymptotic probability distribution that is induced on the source-word sequences as well as on the channel-symbol sequences. The zero-memory and stationarity of the binary input extends to both sequences. Hence, one can fully characterize their statistics by specifying a probability distribution on the set of source words and on the set of channel symbols. It is easy to see that if a word v_i consists of l_i 0's and r_i 1's, then it occurs with probability $\mathsf{Pr}(v_i, p) = p^{l_i}(1-p)^{r_i}$. Clearly, the associated channel symbol α_i occurs with the same probability. We call the arising distribution the *code-induced distribution* and denote it by $P_{T,V}(p) = (P(v_1), P(v_2), \ldots, P(v_K))$, where $P(v_i)$ represents $\mathsf{Pr}(v_i, p)$. The distributions that are induced by the codes in Figure 4.3.2 appear below each tree.

Throughout this work, the parameter of interest is the *asymptotic average information rate* of a code (T, V), defined as the asymptotic expected input-message length per unit of transmission cost. In our setting, we can formulate the asymptotic expected input length as

$$L_{T,V}(p) = \sum_{v_i \in V} P(v_i) \cdot L(v_i),$$

where $L(v_i)$ stands for the length of v_i. The corresponding expected transmission cost can be expressed as

$$C_{T,V}(p) = \sum_{v_i \in V} P(v_i) \cdot c_i \tag{4.3.1}$$

and the code rate is given by

$$R_{T,V}(p) = \frac{L_{T,V}(p)}{C_{T,V}(p)} = \frac{\sum_{v_i \in V} P(v_i) \cdot L(v_i)}{\sum_{v_i \in V} P(v_i) \cdot c_i}. \tag{4.3.2}$$

A natural problem of interest is finding a code that maximizes the rate for a given fixed p. Such a code is said to be *optimal*. Although this problem is mentioned in the literature (see [5] and [12]), to the best of our knowledge, it has not been treated in its general form. Nonetheless, we point out two special cases of the problem that were previously addressed and solved. In both cases, the solution takes the form of an algorithm that constructs an optimal tree.

Lempel, Even, and Cohn [12] studied the case where the input is restricted to unbiased sequences, and thus the code-induced distribution is dyadic. Their algorithm is based on an adaptation of the principle underlying the well-known Huffman algorithm. More precisely, an optimum code exists in which the two most costly symbols are assigned to two source words which are of maximal (and identical) length and differ only in the last bit. A tree is then constructed from the bottom up by successively merging the corresponding sibling leaves into a leaf that represents a new channel symbol, whose cost is a function of the merged-symbols costs. However, unlike the Huffman technique, such a construction does not necessarily result in an optimal tree. Instead, one needs to iterate over a sequence of tree constructions, while improving the merging cost function between iterations. The resulting tree sequence is guaranteed to converge to an optimal tree. Unfortunately, this algorithm relies on key properties of optimal trees that do not hold in the general case of p-biased input. Hence, a straightforward adaptation does not seem to solve the general problem. We are unaware of any other reported attempts to tackle either the general problem or this special case.

The second special case relates to the minimization of the compression ratio of variable-to-fixed length codes, and is more extensively documented in the source coding literature. A variable-to-fixed length (VFL) code partitions an M-ary source sequence into a concatenation of variable-length M-ary source words that are encoded into uniform-length codewords, possibly defined on a different D-ary alphabet. Under the assumption of a memoryless and stationary source, which is ruled by a probability distribution P, the

compression ratio for this class of codes takes the form

$$R_T(P) = \frac{m}{\sum_{w_i \in T} P(w_i) \cdot L(w_i)},$$

where m is the common length of all output codewords and T is the parsing tree used by the code. Note that the compression ratio is independent of the specific codeword assignment that the VFL code applies. The optimization problem thus reduces to finding the parsing tree that maximizes the expected parse-string length

$$L_T(P) = \sum_{w_i \in T} P(w_i) \cdot L(w_i).$$

One can now easily derive the binary (i.e., $M = 2$) version of this problem from the general problem by assigning a constant cost to all channel symbols ($c_i = c$ for all $1 \leq i \leq K$).

In [15], Tunstall provided a simple procedure to construct an M-ary parsing tree which maximizes $L_T(P)$ for a memoryless source with a given probability distribution P and for any valid parsing tree size K. The idea is to grow the tree from the top down by successively extending it along the leaf of largest probability. More formally, let us denote the source alphabet by $S = \{s_1, s_2, \cdots, s_M\}$ and its letter probabilities by $P = \{p_1, \cdots, p_M\}$; then, the following algorithm produces an optimal parsing tree.

Tunstall's Algorithm

1. Initialize: let $T = S = \{s_1, s_2, \cdots, s_M\}$ be the tree containing the root and its M children. The leaves of T correspond to each of the M source letters and their respective probabilities are listed in P.

2. If T has K leaves then stop, else perform the following operations on T:

   - Select a leaf $w_i \in T$ with maximal probability and add its M children to the tree (equivalently, replace w_i with its M single-letter extensions $w_i s_1$, $w_i s_2$,$\cdots$,$w_i s_M$),
   - Compute the leaf probabilities for the extended tree.

   Go to step 2.

Numerous papers investigated the performance and properties of Tunstall codes under the source model assumptions made above (a comprehensive survey and a list of references appear in [5]). A particularly interesting perspective on these codes follows from a result by Jelinek and Schneider [6]. They show that

$$H(T) = H(P) \cdot L_T(P), \tag{4.3.3}$$

where $H(P) = -\sum_{1 \le i \le M} p_i \log_2 p_i$ is the source entropy and

$$H(T) = -\sum_{w_i \in T} P(w_i) \log_2 P(w_i)$$

is the entropy of the tree-induced distribution. As pointed out by Abrahams [5], this implies that the Tunstall tree maximizes the entropy of the latter distribution among all parsing trees. Additionally, it minimizes the measure

$$\sum_{w_i \in T} P(w_i) \log_2 \left(\frac{P(w_i)}{q_i} \right)$$

(known as the Kullback-Leibler distance), with respect to a uniform distribution $Q = \{q_1, \cdots, q_K\} = \{\frac{1}{K}\}_{i=1}^{K}$ [5]. Thus, we can think of this technique as attempting to generate fixed-length codewords which are fairly close to being equiprobable [5], [16].

In the general case, one can analogously show that the maximum-rate code also minimizes a "distance" measure defined by

$$D(T, Q) = \frac{\sum_{w_i \in T} P(w_i) \log_2 q_i}{\sum_{w_i \in T} P(w_i) \log_2 P(w_i)}, \tag{4.3.4}$$

with respect to a given distribution $Q = \{q_1, \cdots, q_K\} = \{2^{-C \cdot c_i}\}_{i=1}^{K}$ [5]. Here, C stands for the Shannon capacity of the memoryless channel [8] and is given by $C = \log_2 X$, where X is the largest real root of the channel's characteristic equation $X^{-c_1} + X^{-c_2} + \cdots + X^{-c_K} = 1$. The distribution Q is the familiar maxentropic distribution of the channel [8]. We also note that the reduction of the problem from maximizing (4.3.2) to minimizing (4.3.4) follows from (4.3.3) and from a division of (4.3.2) by $-C$. Intuitively, we seek to best approximate the channel's capacity-achieving distribution with a leaf-distribution of a parsing tree. The criterion for proximity is (4.3.4).

Finally, in the next section, we focus on another related problem which arises in the context of the bit-stuffing approach. We remind the reader that the bit stuffing and the

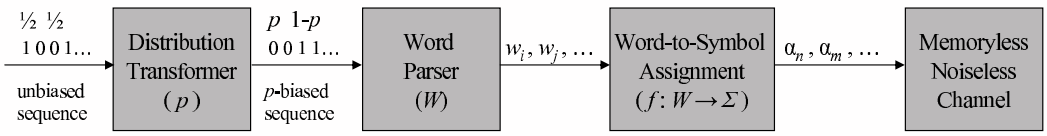


Figure 4.3.3 Block diagram of a variable-length encoding system with a binary distribution transformer for memoryless, noiseless channels.

other algorithms presented here operate on unbiased sequences and rely on the availability of a DT, with the flexibility of properly choosing the bias. The DT introduces an additional parameter to optimize, with the potential benefit of better fitting the maxentropic measure. One can apply this concept to coding for a general memoryless channel by assuming unbiased sequences as input and by adding a DT to the system in Figure 4.3.1, yielding the system in Figure 4.3.3. The problem, then, is to find a pair $(p, (T, V))$ of a bias and a code that jointly maximize the *asymptotic average overall rate* of the system, given by

$$I_{T,V}(p) = R_{T,V}(p) \cdot h(p).$$

Although this problem is most interesting to us, it is not a conventional source coding problem and has not been reported in that literature before.

4.4 A Source Coding Perspective on (d, k) Codes

We devote this section to studying binary-transformer (d, k)-codes from a source coding point of view. In Section 4.4.1, we present a general framework for the construction of variable-length (d, k)-codes from source codes. We derive this framework as a special case of the general setting described in Section 4.3. We then revisit the bit stuffing, bit flipping, and symbol sliding algorithms, and formulate them as special cases of source codes under this framework. The proposed perspective on the three algorithms has motivated us to examine the performance of other code constructions. In the latter two subsections, we consider two problems which concern optimal variable-length (d, k)-codes. Section 4.4.2 considers the problem of jointly optimizing the code and the bias. Section 4.4.3 addresses an optimization problem in which the bias is fixed.

4.4.1 Binary-Transformer Algorithms Revisited

Recall the alternate description of binary (d,k)-sequences as a free concatenation of strings from the set of constrained phrases $\Gamma_{d,k} = \{0^d1, 0^{d+1}1, \ldots, 0^k1\}$. We now define a super-alphabet $\Sigma = \{\alpha_1, \alpha_2, \ldots, \alpha_{k-d+1}\}$, such that the symbol α_i represents the string $0^{i-1+d}1$ for all i; hence Σ represents $\Gamma_{d,k}$. This allows one to view the encoding of p-biased sequences into (d,k)-sequences as equivalent to the encoding of such sequences for transmission over a memoryless noiseless channel, which admits the super-alphabet Σ [10], [1]. Hereafter, we assume we use the system and channel model of Section 4.3, and we set the symbol transmission costs to equal the length of the represented strings, i.e., $c_i = L(0^{i-1+d}1) = i+d$ for all i. Under this framework, a (d,k)-code (T,V) replaces the parsed input words with constrained phrases from $\Gamma_{d,k}$, while inducing a probability distribution on the statistically independent constrained phrases. The *asymptotic average information rate of the* (d,k)*-code* is

$$R_{T,V}^{d,k}(p) = \frac{L_{T,V}(p)}{L_{T,V}^{out}(p)} = \frac{\sum_{v_i \in V} P(v_i) \cdot L(v_i)}{L_{T,V}^{out}(p)}, \tag{4.4.1}$$

where $L_{T,V}^{out}(p)$ is the *expected binary output length* of the code, and is given by

$$L_{T,V}^{out}(p) = \sum_{v_i \in V} P(v_i) \cdot L(\alpha_i). \tag{4.4.2}$$

Here, $L(\alpha_i)$ stands for the length of the string that α_i represents, i.e., for $L(0^{i-1+d}1) = i+d$. It is through the definition of costs $c_i = L(0^{i-1+d}1)$ that the expected transmission cost of the code $C_{T,V}(p)$ becomes its expected binary output length $L_{T,V}^{out}(p)$. Consequently, the average (d,k)-code rate $R_{T,V}^{d,k}(p)$ equals the average transmission-code rate $R_{T,V}(p)$. An optimal parsing-tree code for the above defined channel will therefore yield a maximal rate parsing-tree (d,k)-code. Similarly, we can derive (d,k)-codes for systems that operate on unbiased data and utilize a DT to introduce a bias into the data. The overall rate of the (d,k)-code is

$$I_{T,V}^{d,k}(p) = R_{T,V}^{d,k}(p) \cdot h(p). \tag{4.4.3}$$

Before we proceed to examine parsing trees in more detail, it is useful to introduce the following lemma, which states an important property of optimal parsing-tree codes for transmission over an arbitrary memoryless channel.

Lemma 4.4.1. *Assume a channel admitting an alphabet of K symbols with associated costs $c_1 \leq c_2 \leq \cdots \leq c_K$. Let T be a parsing tree with K leaves and p be the bias of a memoryless information source. If $V = (v_1, v_2, \cdots, v_K)$ is a labeling of the leaves of T such that*

$$\mathsf{Pr}(v_1, p) \geq \mathsf{Pr}(v_2, p) \geq \cdots \geq \mathsf{Pr}(v_K, p) \tag{4.4.4}$$

then $R_{T,V}(p)$ is maximum over all possible labelings of the leaves of T.

Proof. We wish to find a labeling of the leaves of T which maximizes the resulting rate $R_{T,V}(p) = \frac{L_{T,V}(p)}{C_{T,V}(p)}$. A key observation is that the asymptotic expected input length $L_{T,V}(p)$ is independent of the labeling, but rather is a function of the tree and bias only. We can then write

$$L_{T,V}(p) = \sum_{v_i \in V} \mathsf{Pr}(v_i, p) \cdot L(v_i) = \sum_{w_i \in T} \mathsf{Pr}(w_i, p) \cdot L(w_i) = L_T(p)$$

for all $V(T)$. Thus, we need only find a labeling that minimizes the expected transmission cost

$$C_{T,V}(p) = \sum_{i=1}^{K} \mathsf{Pr}(v_i, p) \cdot c_i.$$

It is easy to see that the minimum is attained when sorting the leaves in order of non-increasing probability and by assigning the i'th leaf (in the sorted list) to the i'th channel symbol. This is reflected in the labeling proposed in (4.4.4). □

Lemma 4.4.1 implies that a search for a maximal rate code needs only account for the one code that optimizes the assignment for each of the candidate parsing trees. It not only simplifies the search but also provides a simple means of obtaining the best code for a given tree. Once a tree is chosen, the optimal code simply assigns the least costly symbol α_1 to the most probable leaf, the next least costly symbol α_2 to the second most probable leaf, and so on. We can thus omit the labeling V when referring to a code (T, V). The lemma will also prove relevant in understanding the algorithms discussed in Section 4.2, as explained below.

Thus far, we have outlined a general framework for (d, k)-codes that are based on variable-length source codes for a memoryless channel. A code is specified by a parsing tree in combination with the most efficient assignment of constrained phrases to the words

of the tree. In Section 4.2, we interpreted bit stuffing as a particular mapping between the $k-d+1$ strings in

$$\{1, 01, 0^2 1, \cdots, 0^{k-d-1}1, 0^{k-d}\} \tag{4.4.5}$$

and the $k-d+1$ constrained phrases in $\Gamma_{d,k}$. Observing that the strings in (4.4.5) constitute the leaves of a complete tree, we can view bit stuffing, bit flipping and symbol sliding as special cases of source codes under the general framework. For example, when $k-d=3$ we consider all possible parsing trees of size 4, shown in Figure 4.3.2. The leftmost tree corresponds to the word set $\{1, 01, 0^2 1, 0^3\}$ and is used to generate the input words to the three algorithms. Operating on a p-biased stream, it gives rise to the distribution $\{1-p, p(1-p), p^2(1-p), p^3\}$ on the input words. The sole difference between the algorithms is their specified assignments of the constrained phrases $\{0^d 1, 0^{d+1}1, 0^{d+2}1, 0^{d+3}1\}$ to the input words. This, in fact, determines the constrained phrase probabilities. Figure 4.4.1 demonstrates the four assignments that bit stuffing, bit flipping, $SS(3)$, and $SS(4)$ apply, as different labelings of the same tree. Each assignment amounts to a different constrained-phrase probability vector, which appears underneath each tree. For a general value of $k-d$, the tree representation of (4.4.5) has the form depicted in Figure 4.4.2. From now on, we shall refer to (4.4.5) as the *bit-stuffing tree* and denote it by T_{BS}.

Consider now an optimal assignment for the bit-stuffing tree, as indicated by Lemma 4.4.1. It is attained by labeling the leaves in order of non-increasing probability, where the leaf probabilities are

$$\{1-p, p(1-p), p^2(1-p), \cdots, p^{(k-d-1)}(1-p), p^{(k-d)}\}. \tag{4.4.6}$$

However, we have seen in Section 4.2 that the ordering varies with p, and consequently, so does the optimal labeling. A search for such an optimal assignment is implicitly performed by the symbol sliding algorithm. The sliding of $p^{(k-d)}$ up to any index $j>0$ (see Figure 4.2.5) attempts to rearrange the induced probabilities in decreasing order and thus apply the labeling of Lemma 4.4.1. To obtain an ordered list of (4.4.6), it is sufficient to slide $p^{(k-d)}$ up to an index $j^\star$ such that p and $j^\star$ satisfy

$$\begin{cases} p^{(k-d-j^\star)}(1-p) \le p^{(k-d)} < p^{(k-d-j^\star-1)}(1-p), & \text{if } 0 \le j^\star \le k-d-1 \\ 1-p \le p^{(k-d)}, & \text{if } j^\star = k-d. \end{cases} \tag{4.4.7}$$

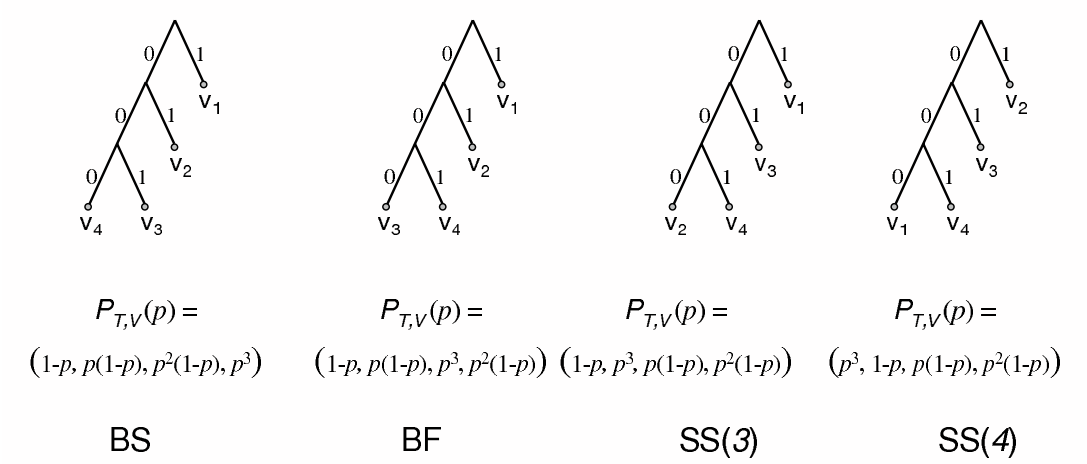


Figure 4.4.1 Four labelings of the bit-stuffing tree and their induced constrained-phrase probabilities, which correspond to (from left to right) bit stuffing, bit flipping, symbol sliding with index 3 and symbol sliding with index 4.

When optimizing for j, the algorithm slides $p^{(k-d)}$ up to its proper position in the ordered set, as implied by Proposition 4.2.2. In summary, bit stuffing and bit flipping apply fixed assignments irrespectively of the bias. In contrast, the extension to symbol sliding results in optimized assignment per given bias, and can thus potentially achieve improved rates over the former two algorithms.

Having optimized both the assignment and the bias for the bit-stuffing tree, we wish to examine the achievable rates associated with other parsing trees. For a given bias, each parsing tree of size $k-d+1$ may be considered in conjunction with its optimal assignment. We now remind the reader that the symbol sliding algorithm was motivated by the idea that a judicious shuffling of the probabilities in (4.4.6) could result in an improved match to the maxentropic vector $\Lambda_{d,k}$ [13]. Nevertheless, can other trees induce probabilities that provide an even better match, better in the sense of a smaller distance $D(T, \Lambda_{d,k})$, as defined in (4.3.4)? In what follows, we deal with two problems that were raised in the general discussion in Section 4.3, as they apply to (d, k)-codes. In particular, we are initially interested in finding the tree and bias that jointly maximize the overall rate of a scheme which includes a DT. We then address the somewhat simpler problem of finding the optimal tree when the bias p is given. Although the first problem is more interesting in the context of bit stuffing, an efficient solution to the second problem may simplify the

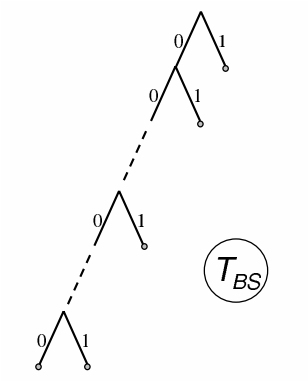


Figure 4.4.2 The general form of the bit-stuffing tree.

solution and analysis of the first.

4.4.2 Jointly Optimal Bias and Parsing Tree Code

We begin by describing our setup for the joint optimization of the tree and the bias. We evaluated the rate of each tree $R_T^{d,k}(p)$ by explicitly expressing $L_T(p)$ as a function of p and $L_T^{out}(p)$ as a piecewise function of p. The latter function corresponds to the best assignment and hence varies with the changes in probability ordering. Multiplying $R_T^{d,k}(p)$ by $h(p)$, we obtained the overall rate per bias. Although one can obtain a closed-form expression for the average rate associated with each tree and each bias, the complexity of the rate expressions makes analysis intractable. For that reason, numerical optimization was carried out, considering all possible parsing trees of size $k - d + 1$ and all biases p such that $0 < p < 1$. In addition, the fast-growing number of candidate trees confined the search to small values of $k - d$. For each such value, a broad range of (d, k) pairs was considered.

Table 4.4.1 shows optimal trees together with their corresponding optimal average rates for numerous (d, k) constraints, where $k - d = 3$. We refer to the trees by their numeric labels, as they appear in Figure 4.3.2, with the only exception being the reference to the bit-stuffing tree. For this tree, we provide an acronym for the most efficient of the bit stuffing (BS), bit flipping (BF), and symbol sliding (SS) algorithms. This merely indicates whether the optimal sliding index at the optimal bias equals 0, 1, or is greater than 1. Also shown are the optimal bias (where $p = \mathsf{Pr}(0)$), the capacity of each constraint,

and the efficiency of the optimal code. For comparison purposes, the optimized average rate and efficiency of the symbol sliding algorithm are given as well. We note that for the purpose of brevity, the table provides details of only the best tree for a range of d's, although optimization was carried out for all $(d, d+3)$ pairs such that $0 \leq d \leq 30$. It is also worth noting that it suffices to restrict optimization to a subset of all parsing trees. This can be done due to symmetries between the probability sets that certain trees induce, and also because of the symmetry of the binary entropy function around $p = 0.5$. Specifically, if there exist trees T and S such that $\{\mathsf{Pr}(w_i, p)\}_{w_i \in T} = \{\mathsf{Pr}(u_i, 1-p)\}_{u_i \in S}$ for all $p \in (0,1)$, then we have $I_T^{d,k}(p) = I_S^{d,k}(1-p)$ for all $p \in (0,1)$. We can thus disregard one of these trees when optimizing over the interval $(0,1)$. We also omit trees which represent different word sets but induce exactly the same leaf-distribution as a function of p. Such trees exhibit the same performance. In the case where $k - d = 3$, the mentioned symmetries allow one to limit the search to the three leftmost trees in Figure 4.3.2.

Similar details for numerous $(d, d+4)$ and $(d, d+5)$ constraints appear in Tables 4.4.2 and 4.4.3, respectively. After eliminating redundant trees, we were left to consider 7 trees for $k-d=4$ and 19 trees for $k-d=5$. Figure 4.4.3 and Figure 4.4.4 depict the optimal trees that are listed in Tables 4.4.2 and 4.4.3, respectively. In the case where $k-d=6$, we did not construct all possible trees but accounted only for 17 trees. One of these trees is the bit-stuffing tree of size 7, while the others were generated by extending each of the leaves of some of the trees of size 6. Among the extended trees are trees number 3, 4 and 5 in Figure 4.4.4, which are optimal for many $(d, d+5)$ constraints. In this chapter, we did not include detailed results of the optimization for $k-d=6$. We shall, however, refer briefly to these results in the following discussion.

It can be seen from Tables 4.4.1 - 4.4.3 that for many constraints, there exists a code construction which outperforms symbol sliding. Similar outcomes were also observed in the $k-d=6$ case. This means that certain trees give rise to probability distributions which provide a better match to the maxentropic vector $\Lambda_{d,k}$ than the bit-stuffing tree's induced distribution. For example, when $k-d=3$ and $5 \leq d \leq 30$, the distribution $P_3(p) = \{(1-p)^2, p(1-p), p(1-p), p^2\}$ leads to improved performance over the bit-stuffing distribution $P_{BS}^{d,d+3}(p) = \{1-p, p(1-p), p^2(1-p), p^3\}$. This is surprising since

Table 4.4.1 Numerical results for optimal performance of parsing-tree codes for various $(d, d+3)$ constraints.

Constraint	Best Tree	Best Tree Avg. Rate	Best Tree Optimal Bias	Best Tree Avg. Rate / Capacity	Symbol Sliding Avg. Rate	Symbol Sliding Avg. Rate / Capacity	Capacity
(0,3)	2	0.94500	0.716	99.81%	0.94089	99.38%	0.94678
(1,4)	SS	0.61577	0.746	99.73%	0.61577	99.73%	0.61745
(2,5)	SS	0.46496	0.724	100%	0.46496	100%	0.46496
(3,6)	SS	0.37421	0.651	99.90%	0.37421	99.90%	0.37459
(4,7)	SS	0.31361	0.654	99.80%	0.31361	99.80%	0.31423
(5,8)	3	0.27046	0.570	99.84%	0.26991	99.64%	0.27088
(6,9)	3	0.23788	0.562	99.88%	0.23691	99.47%	0.23817
(7,10)	3	0.21237	0.555	99.90%	0.21110	99.30%	0.21258
(8,11)	3	0.19184	0.550	99.92%	0.19037	99.15%	0.19199
(9,12)	3	0.17495	0.545	99.93%	0.17334	99.02%	0.17507
(10,13)	3	0.16081	0.542	99.94%	0.15911	98.89%	0.16090
$10 < d < 20$	3						
(20,23)	3	0.08903	0.523	99.98%	0.08738	98.14%	0.08904
$20 < d < 30$	3						
(30,33)	3	0.06158	0.516	99.99%	0.06023	97.80%	0.06159

the structure of the bit-stuffing distribution resembles the finite geometric series that the maxentropic phrase probabilities form. To further explain, observe that the bit-stuffing probabilities, except for $p^{(k-d)}$, have the form $p^i(1-p)$ for $k-d$ consecutive i's. Thus, they nearly form a geometric series, whereas it is clear that the maxentropic probabilities do form such a series. On the other hand, not only does $P_3(p)$ have a different structure, it actually contains two identical probabilities. This property recurs for trees number 3 and 5 of Tables 4.4.2 and 4.4.3, respectively. When $k-d=6$, the property persists in the trees that perform the best for all $7 \leq d \leq 30$.

Another interesting effect is a convergence towards a specific tree which obtains the

Table 4.4.2 Numerical results for optimal performance of parsing-tree codes for various $(d, d+4)$ constraints.

Constraint	Best Tree	Best Tree Avg. Rate	Best Tree Optimal Bias	Best Tree Avg. Rate / Capacity	Symbol Sliding Avg. Rate	Symbol Sliding Avg. Rate / Capacity	Capacity
(0,4)	BS	0.97101	0.468	99.57%	0.97101	99.57%	0.97523
(1,5)	BF	0.64901	0.574	99.71%	0.64901	99.71%	0.65090
(2,6)	2	0.49722	0.588	99.86%	0.49706	99.83%	0.49791
(3,7)	SS	0.40569	0.755	100%	0.40569	100%	0.40569
(4,8)	2	0.34300	0.558	99.93%	0.34294	99.92%	0.34323
(5,9)	2	0.29725	0.549	99.80%	0.29717	99.77%	0.29786
(6,10)	3	0.26286	0.550	99.84%	0.26228	99.61%	0.26330
(7,11)	3	0.23576	0.554	99.88%	0.23472	99.44%	0.23603
(8,12)	3	0.21376	0.584	99.91%	0.21240	99.27%	0.21396
(9,13)	3	0.19553	0.584	99.92%	0.19396	99.11%	0.19570
(10,14)	3	0.18017	0.584	99.91%	0.17847	98.96%	0.18034
$10 < d < 20$	3						
(20,23)	3	0.10090	0.584	99.64%	0.09922	97.98%	0.10126
$20 < d < 30$	3						
(30,33)	3	0.07007	0.584	99.43%	0.06871	97.51%	0.07047

best rate, starting from a certain d. For example, tree number 3 seems to asymptotically yield the best code when $k-d=3$. We encountered this effect for all examined values of $k-d$. Although we have not proved that this indeed holds for d's larger than 30, from now on, we shall call these trees the *asymptotically optimal trees*. We outline the difference between these trees and the bit-stuffing tree using the following definitions. We say that a binary tree is a *skew tree* if it is obtained by either consistently extending its rightmost leaf or by consistently extending its leftmost leaf. In contrast, a *balanced tree* of size K is a binary tree where each subtree of the root is of the same height if $K = 2^D$ for some D, or where the two subtrees differ in height by at most 1 and are balanced as well, if

Table 4.4.3 Numerical results for optimal performance of parsing-tree codes for various $(d, d+5)$ constraints.

Constraint	**Best Tree**	**Best Tree Avg. Rate**	**Best Tree Optimal Bias**	**Best Tree Avg. Rate / Capacity**	**Symbol Sliding Avg. Rate**	**Symbol Sliding Avg. Rate / Capacity**	**Capacity**
(0,5)	BS	0.98545	0.48	99.73%	0.98545	99.73%	0.98811
(1,6)	2	0.66805	0.733	99.85%	0.66730	99.74%	0.66903
(2,7)	3	0.51696	0.417	99.92%	0.51643	99.82%	0.51737
(3,8)	SS	0.42457	0.785	99.88%	0.42457	99.88%	0.42507
(4,9)	SS	0.36199	0.778	100%	0.36199	100%	0.36199
(5,10)	SS	0.31560	0.773	99.93%	0.31560	99.93%	0.31580
(6,11)	4	0.27996	0.649	99.85%	0.27979	99.79%	0.28037
(7,12)	5	0.25203	0.554	99.91%	0.25131	99.63%	0.25226
(8,13)	5	0.22922	0.555	99.94%	0.22815	99.47%	0.22937
$8 < d < 30$	5						
(30,33)	5	0.10989	0.561	99.43%	0.10832	98.01%	0.11052

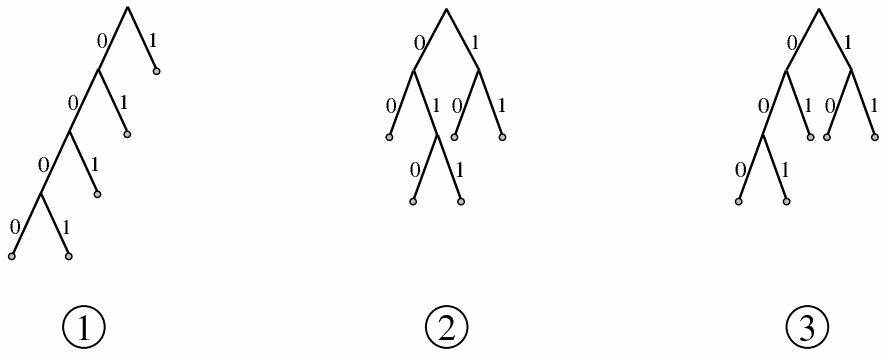


Figure 4.4.3 Tree representations of the three codes which are optimal for various $(d, d+4)$ constraints in Table 4.4.2. The leftmost tree is the bit-stuffing tree.

$K \neq 2^D$. Clearly, the bit-stuffing tree is a skew tree, while the asymptotically optimal trees are all balanced.

A few questions arise from the preceding discussion. Do the asymptotically optimal trees which we found maintain their optimality as d approaches infinity? If so, is there an asymptotically optimal tree for any given $k - d$? Furthermore, can we characterize these trees, for example, by their skewness? Lastly, how can one efficiently find these trees? These questions are difficult to address without a good insight into the joint optimization problem. In the following subsection we try to pursue a better understanding of the problem by studying another related problem.

4.4.3 Optimal Parsing Tree Codes for a Given Bias

The complexity of the joint optimization problem has led us to decompose it into simpler problems. We next tackle the problem of finding the optimal tree when the bias is given; that is, we consider a system which does not include a DT. With a solution at hand, we can approach the original problem by a two-stage optimization. First, one obtains the optimal tree per bias and afterwards, a DT is added to the system and the overall rate is optimized. However, as pointed out in Section 4.3, the problem of finding optimal trees for a fixed arbitrary bias is still hard and has not been addressed in prior literature, neither for a general channel nor for the special (d, k)-channel. The only case

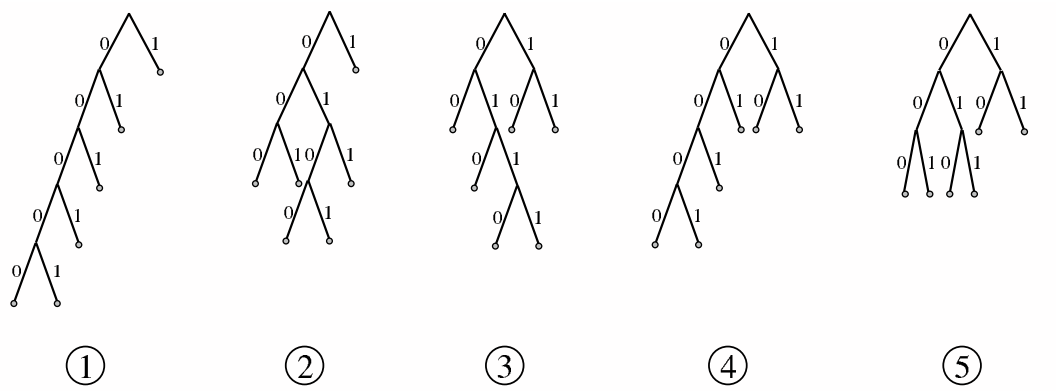


Figure 4.4.4 Tree representations of the five codes which are optimal for various $(d, d+5)$ constraints in Table 4.4.3. The leftmost tree is the bit-stuffing tree.

that was solved algorithmically is when $p = 0.5$ [12]. Therefore, for general p values, we resorted to an exhaustive search over all possible parsing trees of size $k - d + 1$. The search was performed for the same values of $k - d$ and in the same range of (d, k) constraints as those considered in the preceding subsection. Unlike before, we do not eliminate symmetric trees since we examine the rates $R^{d,k}_{T,V}(p)$ for any given p.

Now, suppose we fix $k - d$, and we inspect the various rates as functions of p, while gradually increasing d, starting from $d = 0$. When examining the optimal (d, k)-code per bias, we noticed that a fixed pattern emerges once a certain d value is crossed. Specifically, past this point (and up to $d = 30$), it appears that the optimal tree per bias is fixed, and that the range of biases ($0 < p < 1$) is divided into continuous subintervals, each corresponding to a certain optimal tree. Figure 4.4.5 illustrates this effect with an example of the rate functions of several $(6, 9)$-codes. The five curves correspond to the five parsing trees of size $k - d + 1 = 4$, shown in Figure 4.3.2. We found that the d thresholds for $k - d = 3, 4$ and 5 are $3, 6$ and 6, respectively. Yet, our most interesting finding is that the fixed optimal tree for each considered bias is, in fact, the Tunstall tree for that bias. We can intuitively explain it as follows. A code maps the input words into phrases of various lengths ranging from $d + 1$ to $k + 1$. When d is considerably larger than the fixed $k - d$, the variation in phrase lengths is negligible, and they are approximately equal. That said, it only seems reasonable that the optimal VFL coding scheme will prove to be an efficient scheme in those cases as well. Still, there seems to be more to these observations than the

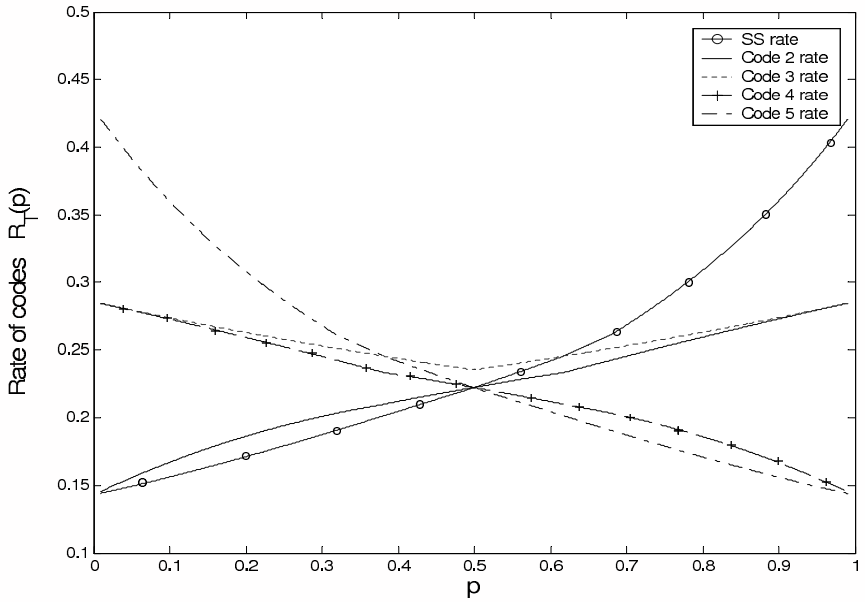


Figure 4.4.5 The achievable rates of five $(6, 9)$-codes as functions of p.

given interpretation, as the observed d thresholds are comparable to the $(k - d)$'s. It is an interesting question whether the revealed properties indeed apply to arbitrarily large d's and $(k - d)$'s. In the rest of this subsection we settle this question. We further discuss its implications on the solutions to the two optimization problems that are discussed in this chapter.

At this point, it is helpful to make a number of undemanding observations in order to elucidate two issues: the non-uniqueness of the Tunstall tree and the sub-optimality of non-Tunstall trees. First, consider a situation where at a certain iteration of the algorithm, there exist $j > 1$ leaves with the same maximal probability. Then, in each of the next j iterations, the algorithm will choose arbitrarily one of these leaves since they are the most probable. After j iterations, one ends up with the same tree, regardless of the arbitrary ordering. Now, suppose we have reached the desired tree size and have stopped before the j'th iteration. In this case, some of the different orderings will extend different subsets of the j leaves, leading to different Tunstall trees. Nonetheless, it can be easily shown that these trees result in the same average input length and in the same induced leaf probabilities. Therefore, in our context, the (d, k)-codes that correspond to such trees achieve the same rate, which allows us to treat them as indistinguishable and refer to any of them as

the Tunstall code. Finally, if the algorithm does not encounter leaves with equal maximal probability, then the Tunstall tree is unique.

When considering VFL codes other than Tunstall codes, it is important to stress that such codes are suboptimal with respect to maximizing $L_T(P)$. In other words, if there exist multiple optimal parsing trees, then they are exactly the indistinguishable Tunstall trees we described above, and no others. In this sense, we can say that the Tunstall tree is uniquely optimal. We formally state this observation in Lemma 4.4.4, as we shall later make use of it. Before proving the lemma it is useful to introduce the following definition and simple characterization of a Tunstall tree, given in [17]. For completeness we provide a short proof of the proposed characterization.

Definition 4.4.2. A complete M-ary tree verifies the *ordering property* if one can list its nodes in order of non-increasing probability so that the obtained list can be divided into two parts, the upper containing all internal nodes in the correct order of extension, and the lower containing all leaves.

Proposition 4.4.3. *An M-ary parsing tree is a Tunstall tree if and only if it verifies the ordering property.*

Proof. Suppose a Tunstall tree contains an internal node α and a leaf β such that $P(\alpha) < P(\beta)$. Then, at one of the preceding extensions, the algorithm should have chosen the leaf β instead of α, which contradicts β being a leaf while α being already extended. Conversely, if a tree cannot be obtained by the algorithm, then one can always construct the tree by a series of successive leaf extensions. However, at a certain extension, the chosen leaf is not a leaf of greatest probability. Immediately following this extension, we have a leaf $\beta^\star$ whose probability is greater than the probability of an internal node. This may change only if one of the subsequent steps extends the leaf $\beta^\star$, in which case, the ordered list of internal nodes does not appear in the correct order of extension. □

Note that for our purposes, we make a distinction only between trees that can be generated by the Tunstall algorithm and trees that cannot. In other words, we are not concerned with different orderings of extensions during the construction of two given trees, as long as the final forms of the trees are identical. For that reason, the criterion for a tree that *can be* obtained by the algorithm reduces to the probability of any internal node

being greater than or equal to the probability of any leaf. We next exploit this criterion to prove that any optimal VFL code corresponds to a Tunstall tree.

Lemma 4.4.4. *Let P be a probability distribution of a memoryless source, and T be a tree that maximizes $L_T(P)$. Then, T can be obtained by the Tunstall algorithm.*

Proof. Assume that a tree T is optimal but can not be produced by the algorithm. From Proposition 4.4.3 and the preceding discussion we know that T does not verify the ordering property, thus there exist an internal node α and a leaf β in T such that $P(\alpha) < P(\beta)$. We now construct a new parsing tree T' from T by pruning the subtree T^{sub} that descends from node α (i.e., all children and all other descendants of α farther down in T) and reproducing the subtree under the leaf β. Now, it is well known that for a complete tree, the summation of probabilities of all internal nodes including the root is equal to the average input length. More formally, letting I represent the set of internal nodes of T excluding the root, one can verify that

$$L_T(P) = 1 + \sum_{a \in I} P(a). \tag{4.4.8}$$

Using (4.4.8) to compare $L_T(P)$ to $L_{T'}(P)$, it can be seen that the replacement of T^{sub} from α to β results in an increased average input length, that is, $L_{T'}(P) > L_T(P)$. This contradicts the optimality of T, and hence proves the lemma. □

Let us now return to optimal variable-length codes for the (d, k)-channel. The aforementioned observations suggest that for certain (d, k) pairs, the optimal variable-length (d, k)-code is a Tunstall code for any given p. Our findings further indicate a threshold behavior of the applicable (d, k) pairs, namely, that given $m = k - d$, there exists some d_m, such that all $(d, d + m)$ constraints with $d \geq d_m$ have this property. The next lemma asserts these conjectured properties.

Lemma 4.4.5. *Let $m > 0$ be an integer and $T_{Tun}(p)$ be the Tunstall tree of size $m + 1$ that corresponds to a p-biased binary memoryless source. Then, there exists an integer d_m, such that for any $(d, d + m)$ constraint with $d \geq d_m$ the following holds:*

$$R^{d,d+m}_{T_{Tun}(p)}(p) \geq R^{d,d+m}_{T}(p) \quad \forall \ 0 < p < 1 \tag{4.4.9}$$

for any parsing tree T of size $m + 1$.

Proof. Suppose p is fixed and let T be a tree description of a $(d, d+m)$-code. As implied by Lemma 4.4.1, we need only consider codes which apply an optimal assignment V.

Observe first that the expected binary output length of the code can be expressed as the following sum of two terms:

$$L_T^{out}(p) = \sum_{v_i \in V} P(v_i) \cdot (d+i) = d + \sum_{i=1}^{m+1} P(v_i) \cdot i = d + L_T^{sub}(p),$$

where the second term $L_T^{sub}(p)$ depends on the code, on the bias, and on $m = k - d$, but is independent of d. We now rewrite the average information rate of the code as

$$R_T^{d,d+m}(p) = \frac{L_T(p)}{d + L_T^{sub}(p)}, \tag{4.4.10}$$

and we note that $L_T(p)$ is independent of d as well.

Next, consider the $(d, d+m)$-code that corresponds to the Tunstall tree $T_{Tun}(p)$. This code attains the maximum rate for any given bias if and only if (4.4.9) holds for any parsing tree T of size $m+1$. Substituting (4.4.10) into (4.4.9) and rearranging terms, we obtain the equivalent condition

$$d \times \left(\frac{1}{L_T(p)} - \frac{1}{L_{T_{Tun}(p)}(p)} \right) \geq \frac{L^{sub}_{T_{Tun}(p)}(p)}{L_{T_{Tun}(p)}(p)} - \frac{L_T^{sub}(p)}{L_T(p)} \quad \forall\ 0 < p < 1 \tag{4.4.11}$$

for any of the considered trees. As both $L_T(p)$ and $L_T^{sub}(p)$ are independent of d, the right-hand side of (4.4.11) as well as the expression in parenthesis on the left-hand side are independent of d. Since a Tunstall tree maximizes the average input length $L_T(p)$ over all parsing trees, we have

$$\frac{1}{L_T(p)} - \frac{1}{L_{T_{Tun}(p)}(p)} \geq 0 \quad \forall\ 0 < p < 1. \tag{4.4.12}$$

Furthermore, it follows from Lemma 4.4.4 that inequality (4.4.12) is strict whenever T is not a Tunstall tree. Hence, for a large enough d, the left-hand side of inequality (4.4.11) will be greater than its right-hand side for all $0 < p < 1$. In case T is a Tunstall tree, then it induces the same leaf probabilities as $T_{Tun}(p)$, and thus achieves the same rate. We now complete the proof by setting d_m to the smallest d for which the condition in (4.4.11) holds for all parsing trees. □

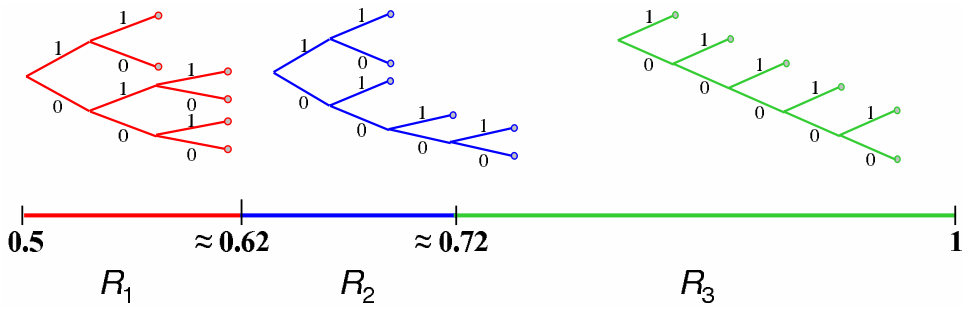


Figure 4.4.6 Tunstall regions with their corresponding trees for $K = 6$. After [17].

In light of Lemma 4.4.5, we proceed to examine additional characteristics of Tunstall trees. Here we present results by Fabris, Sgarro, and Pauletti [17], pertaining to the relationship between the bias of the source and the structure of the Tunstall tree. They define a *Tunstall region* to be the set of all source probability distributions that are optimally encoded by the same Tunstall code. An analysis of the binary case results in a full characterization of these regions and in a simple procedure for computing them. Representing the source distribution by its bias p, it is proved that the Tunstall regions have the form of continuous subintervals of the unit interval $(0, 1)$. As noted earlier, there exist biases for which the Tunstall tree is not unique, thus implying that the subintervals are not necessarily disjoint. We have seen, however, that the performances of the multiple Tunstall codes are the same, hence one can choose a single representative tree per bias. To resolve this ambiguity, Fabris *et al.* propose to modify the Tunstall algorithm, so as to avoid situations where multiple leaves are simultaneous candidates for extension. When several leaves have the same probability, the algorithm lexicographically orders them and chooses the first one. Otherwise, the criterion for choosing the leaf of extension remains unchanged. As a consequence, each p belongs only to one Tunstall region, and so the regions form a partition of $(0, 1)$ into distinct subintervals. Figure 4.4.6 shows an example from [17] demonstrating the segmentation of the interval $(0.5, 1)$ into three Tunstall regions in the case when the tree size is 6. The transitions between regions occur at the probabilities p_1 and p_2 that solve the equations $p^2 = 1 - p$ and $p^4 = 1 - p$, respectively. One can readily observe that the segmentation of the interval $(0, 0.5)$ corresponds to the symmetric trees and is therefore omitted.

The paper provides a simple method to compute the region boundaries and at the same

time, to construct all trees by executing the algorithm once for the first region and then adaptively constructing all other trees with no further executions. For a given tree size K, the number of Tunstall regions in the half-unit interval $N(R_{Tun}(K))$ is evaluated with the following upper and lower bounds:

$$K - \lceil \log(K) \rceil \leq N(R_{Tun}(K)) \leq \left(6/\pi^2\right) \cdot (\lceil \log(K) \rceil - 1) \cdot 2^{(\lceil \log(K) \rceil - 1)}. \quad (4.4.13)$$

Note that the lower bound is reported to be tight, at least up to $K = 16$, as opposed to the loose upper bound. Table 4.4.4 lists the exact number of Tunstall regions for tree sizes ranging from 4 to 10. An interesting pattern which arises from the above characterization involves the structure of the two Tunstall trees at the extremes of the half-unit interval, that is, the leftmost and the rightmost trees. These are always the balanced tree, which is optimal at least for $p = 0.5$, and the skew tree (the bit-stuffing tree), which is optimal in the neighborhood of $p = 1$.

Table 4.4.4 Number of Tunstall regions for small size trees.

Tunstall Tree Size	**Number of Regions**
4	2
5	2
6	3
7	4
8	5
9	5
10	6

The above-mentioned properties of Tunstall codes are especially appealing in the context of the complex problem with which we dealt in the former subsection. Recall that the problem entails the joint optimization of the tree and the bias. For every $(d, d + m)$ pair where $d \geq d_m$, the following two-stage approach greatly simplifies the optimization. At a first stage, one can carry out the algorithm described in [17] to compute the Tunstall regions and corresponding trees. Subsequently, one can numerically evaluate the

rate associated with the proper tree at each bias and proceed to optimize the *overall* rate. This way, optimization is restricted to a limited number of Tunstall trees, which can be easily constructed. As implied by the upper bound in (4.4.13), the number of candidate trees will not exceed the order of $(k-d)\log(k-d)$ – a significantly smaller number than the number of all parsing trees. In fact, the upper bound is quite loose and so the actual number seems to be much smaller as well as close to the lower bound. Lemma 4.4.5, in conjunction with the results of [17], also provides some insight into the asymptotic convergence pattern we observed in Tables 4.4.1 - 4.4.3. The lemma suggests that from a certain d onwards, only a few fixed Tunstall trees, among which is the bit-stuffing tree, are competing for the maximum. Moreover, one can verify that the asymptotically optimal trees in Tables 4.4.1 - 4.4.3 (as well as in the $k-d=6$ case) are always the balanced Tunstall trees. Although we can not infer that this will always be the case, we can narrow down the "asymptotic candidates" to the relatively small set of Tunstall trees.

As a final point, an attractive property of the (d,k)-codes studied in this subsection is that they provide a simple method for joint source-(d,k) (channel) coding. An alternative method for combined source and (d,k)-coding of p-biased sequences was proposed by Kerpez [4]. It is based on arithmetic coding techniques, and was shown to converge to the combined source-channel capacity [8] as the length of the input string approaches infinity. An example of a practical implementation of such a scheme, entailing a concatenation of two arithmetic encoder-decoder pairs, appears in [18]. Although the codes we study do not always seem to converge to the combined source-channel capacity, they have the advantage of limited complexity since the tree size is $k-d+1$, independently of the input size. It is also interesting to compare such a joint coding scheme to a scheme that separates the source from the constrained coding. For example, one can use a DT to remove the redundancy and subsequently apply an optimal parsing-tree (d,k)-code. The optimal code is found by the Lempel, Even, Cohn algorithm [12]. The separation of encoders clearly results in additional implementation complexity due to the addition of a DT. On the other hand, it has the advantage of being amenable to optimization for any (d,k) pair. Let us now consider the performances of the two approaches. For a given bias p, denote the rate that corresponds to the optimal tree by $R^{\star}(p)$. Then, the maximum achievable rate of the separating scheme equals $\frac{1}{h(p)}\cdot R^{\star}(0.5)$. The joint scheme will

outperform the separating scheme if $R^\star(p) > \frac{1}{h(p)} \cdot R^\star(0.5)$, or equivalently, if

$$h(p) \cdot R^\star(p) > h(\frac{1}{2}) \cdot R^\star(\frac{1}{2}). \tag{4.4.14}$$

One can see that both sides of (4.4.14) represent the optimized overall rates that are achieved for an arbitrary p and for $p = 0.5$ by the optimal constructions we considered in Section 4.4.2. Hence, different p's may lead to different relations.

4.5 Concluding Remarks and Open Problems

We conclude with some interesting open problems and with a qualitative discussion of the various constructions presented here.

We studied several (d, k)-codes of a special kind. All codes use a binary DT to bias the data before the actual constrained encoding takes place. The various constructions gradually build on each other, with the fundamental one being the bit stuffing algorithm. We have seen that the addition of controlled bit flipping resulted in improved rates over bit stuffing. A recent generalization to the symbol sliding algorithm demonstrated further improved performance. In this chapter, we extended symbol sliding into a general framework for constructing (d, k)-codes from variable-length source codes. We showed that the general framework gives rise to new code constructions which achieve improved performance over symbol sliding. In essence, we can say that more generalized algorithms tend to perform better. However, when searching for optimal codes, each level of extension requires the optimization of an additional parameter. This in turn makes analysis more complex and sometimes intractable. Moreover, at the highest level of extension, even numerical optimization is impractical.

We proceeded to investigate optimal variable-length (d, k)-codes under the general framework. Although limited to small $k - d$ values, our numerical optimization results indicate some possible trends. First, we found that symbol sliding is suboptimal in many cases. Rather surprisingly, the optimum in those cases corresponds to induced distributions which, unlike the bit stuffing distribution, do not resemble the structure of the maxentropic one. Second, as d increases (and $k - d$ is fixed), the optimal code tree converges to a certain specific tree, depending on the value of $k - d$. One question for future

research is whether these trees are indeed asymptotically optimal. If they are, then it is interesting to find out if convergence occurs for any $k - d$ and from what d value (as a function of $k - d$). Obviously, efficiently finding or constructing these trees is a challenging problem by itself. In this regard, it may be helpful to be able to characterize their exact form. Our results suggest that their skewness may be a possible characteristic.

We also considered the problem of finding optimal variable-length codes when the bias p is fixed. Our main result is that for a fixed $m = k - d$, there exists d_m such that for all $d \geq d_m$ and for all p, the optimal (d, k)-code corresponds to the Tunstall tree for p. However, we did not specify what d_m is or how to compute it. A natural direction for future research is to address this question. Furthermore, devising a general algorithm that generates the optimal tree given an arbitrary bias and an arbitrary (d, k) pair remains an open problem. This problem is also of interest in a wider context of the more general joint source-channel coding problem we discussed in Section 4.3. It forms a special case, where the transmission costs are determined by d and k, but the bias is arbitrary. In this regard, it is worth mentioning a duality, noted by Abrahams, between the general problem of Section 4.3 and another long-standing problem known as the Karp problem (see, for example, [5, Sec. III.1-2] and a reference therein). A recent advance on the dual problem is the work of Golin and Rote [11], which effectively solves it for a broad class of cases. Their work may provide insight for the solution of our problem.

Finally, we wish to emphasize a core difference between both bit stuffing and bit flipping and their two levels of extensions. We have seen that symbol sliding as well as the general framework rely substantially on the memoryless channel representation of (d, k)-sequences and, furthermore, on the independence of the maxentropic constrained phrases. However, these attributes are unique to (d, k) constraints and in general do not extend to other constraints of interest. Specifically, they do not apply to two-dimensional (2-D) constraints, which are of primary interest in current research on constrained coding. Consequently, an extension of such constructions to other constraints is not straightforward. The bit stuffing and bit flipping techniques, on the other hand, are based on entirely different principles. They operate in a streaming manner (“on the fly”) by locally satisfying the constraint. As such, they do not operate on a “phrase level” and are not limited to (d, k) constraints. In fact, bit stuffing has already been applied to various 2-D constraints

and was shown to achieve high rates [19]. In cases where it was amenable to analysis, it was used to derive lower bounds on the unknown capacities of several 2-D constraints. Bit-stuffing schemes for 2-D constraints are the topic of the next chapter.

Acknowledgment. This chapter is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," to appear in *Proc. 2006 IEEE Int. Symp. Inform. Theory*, Jul. 2006 and S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," submitted to *IEEE Trans. Inform. Theory*, Jan. 2006.

Bibliography

[1] G. N. N. Martin, G. G. Langdon, and S. J. P. Todd, "Arithmetic codes for constrained channels," *IBM J. Res. Develop.*, vol. 27, no. 2, pp. 94–106, Mar. 1983.

[2] S. J. P. Todd, G. G. Langdon, and G. N. N. Martin, "A general fixed rate arithmetic coding method for constrained channels," *IBM J. Res. Develop.*, vol. 27, no. 2, pp. 107–115, Mar. 1983.

[3] E. Arikan, "An implementation of Elias coding for input-restricted channels," *IEEE Trans. Inform. Theory*, vol. 36, no. 1, pp. 162–165, Jan. 1990.

[4] K. J. Kerpez, "Runlength codes from source codes," *IEEE Trans. Inform. Theory*, vol. 37, no. 3, pp. 682–687, May 1991.

[5] J. Abrahams, "Code and parse trees for lossless source encoding," in *Proc. Compression and Complexity of SEQUENCES 1997*, Positano, Italy, Jun. 1997, pp. 145–171.

[6] K. Jelinek and K. S. Schneider, "On variable-length-to-block coding," *IEEE Trans. Inform. Theory*, vol. 18, no. 6, pp. 765–774, Nov. 1972.

[7] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[8] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pt. I, pp. 379–423, Jul. 1948.

[9] E. Zehavi and J. K. Wolf, "On runlength codes," *IEEE Trans. Inform. Theory*, vol. 34, no. 1, pp. 45–54, Jan. 1988.

[10] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, "Codes for digital recorders," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[11] M. J. Golin and G. Rote, "A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs," *IEEE Trans. Inform. Theory*, vol. 44, no. 5, pp. 1770–1781, Sept. 1998.

[12] A. Lempel, S. Even, and M. Cohn, "An algorithm for optimal prefix parsing of a noiseless and memoryless channel," *IEEE Trans. Inform. Theory*, vol. 19, no. 2, pp. 208–214, Mar. 1973.

[13] Y. Sanakarasubramaniam and S. W. McLaughlin, "Capacity achieving code constructions for two classes of (d, k) constraints," *submitted to IEEE Trans. Inform. Theory*, Jun. 2004.

[14] Y. Sankarasubramaniam and S. W. McLaughlin, "Symbol sliding: improved codes for the (d, k) constraint," in *Proc. Int. Symp. Inform. Theory and Applic.*, Parma, Italy, Oct. 2004.

[15] B. P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 1967.

[16] S. A. Savari and W. Szpankowski, "On the analysis of variable-to-variable length codes," in *Proc. 2002 IEEE Int. Symp. Inform. Theory*, Lausanne, Switzerland, Jun./Jul. 2002, p. 176.

[17] F. Fabris, A. Sgarro, and R. Pauletti, "Tunstall adaptive coding and miscoding," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 2167–2180, Nov. 1996.

[18] H. Morita, M. Hoshi, and K. Kobayashi, " A reversible distribution converter with finite precision arithmetic," in *Proc. 2002 IEEE Int. Symp. Inform. Theory*, Lausanne, Switzerland, Jun./Jul. 2002, p. 404.

[19] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inform. Theory*, vol. 47, no. 3, pp. 1166–1176, Mar. 2001.

5

Two-Dimensional Bit-Stuffing Schemes with Multiple Distribution Transformers

5.1 Introduction

Recent advances in high-capacity optical storage technologies have motivated the study of two-dimensional constraints. These technologies use a two-dimensional (2-D) model of the recorded data, as opposed to the traditional one-dimensional (1-D) track model [1]. This approach gives rise to new types of error patterns, constraints and encoding algorithms. Two-dimensional constraints can be defined over different 2-D lattices, depending on the layout of the data on the recording medium. In this work, we consider the class of 2-D run-length-limited (RLL) (d, ∞) constraints as well as the 'no isolated bits' (n.i.b.) constraint, both defined on the square lattice. A 2-D (d, ∞) constraint consists of all binary arrays in which there are at least d zeros between any two successive ones in any row and in any column. The 2-D n.i.b. constraint requires that every bit equals to at least one of its four adjacent bits (i.e. the bit above, the bit below and the two bits to

its sides). In other words, it prohibits the occurrence of the patterns

$$\begin{array}{|c|c|c|}\hline & 0 & \\ \hline 0 & 1 & 0 \\ \hline & 0 & \\ \hline\end{array} \qquad \text{and} \qquad \begin{array}{|c|c|c|}\hline & 1 & \\ \hline 1 & 0 & 1 \\ \hline & 1 & \\ \hline\end{array} \quad .$$

Let $N(m, n)$ be the number of distinct binary arrays of size $m \times n$ that satisfy a given 2-D constraint. The *capacity* of the 2-D constraint is defined as

$$C = \lim_{m,n\to\infty} \frac{1}{mn} \log_2 N(m, n).$$

Unlike the 1-D case, there are no known methods for computing the capacity of many 2-D constraints of interest. Instead, several techniques for deriving upper and lower bounds on the capacity were suggested. One particular lower bounding technique is based on an analysis of a bit-stuffing encoding algorithm [2]. The algorithm converts the input sequence into another sequence having different statistical properties. It then encodes the latter sequence into a constrained array by inserting excess bits in a manner that guarantees that the constraint is satisfied.

Siegel and Wolf [2] initially introduced a bit-stuffing encoder for 2-D (d, ∞) constraints, for all $d \geq 1$. They computed a lower bound on the average rate of such a scheme and, a fortiori, on the capacity. Roth, Siegel, and Wolf [3] then proposed and analyzed a more general bit-stuffing scheme for the special case where $d = 1$. They showed that this scheme achieves improved performance over the original scheme. More recently, Halevy *et al.* [4] presented a bit-stuffing encoder for the n.i.b. constraint. Analysis of the encoder resulted in lower bounds on its average rate. Halevy *et al.* further obtained improved lower bounds on the rates of the (d, ∞)-encoders presented in [2], for $d \geq 2$. Additionally, a modified bit-stuffing scheme was proposed and analyzed by Forchhammer [5]. Application of this approach to the $(2, \infty)$ constraint yielded a further improved lower bound on its capacity.

In this chapter, we introduce two new bit-stuffing constructions. In the first construction, we extend the idea that underlies the improved $(1, \infty)$-construction in [3] to (d, ∞) constraints, where $d \geq 2$. The second construction is a bit-stuffing scheme for the n.i.b. constraint that is based on a capacity-achieving bit-stuffing scheme for a certain 1-D RLL constraint. Section 5.2 focuses on (d, ∞) constraints and Section 5.3 deals with the n.i.b.

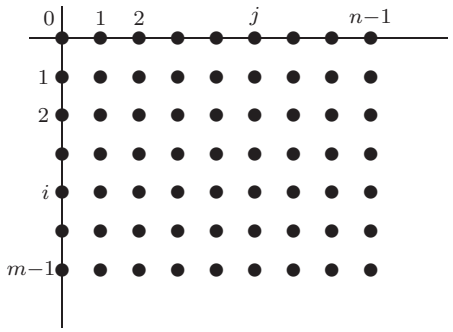


Figure 5.2.1 Rectangular array $B_{m,n}$.

constraint. In both sections, we begin by reviewing previous bit-stuffing schemes and proceed to describe our proposed scheme. We conclude each section with simulation results demonstrating the performance of these schemes. In Section 5.4, we summarize our findings and suggest directions for future research.

5.2 Bit-Stuffing Schemes for (d, ∞) Constraints

In this section we describe bit-stuffing schemes that encode arbitrary data sequences into 2-D (d, ∞)-constrained arrays. We start by introducing some notations and conventions, which will be used throughout the chapter.

We encode the input sequences into rectangular arrays of the form

$$B_{m,n} = \{(i, j) \in \mathbb{Z}^2 \ : \ 0 \leq i < m, \ \ 0 \leq j < n\},$$

where $B_{m,n}$ is shown in Figure 5.2.1. The random constrained array that is generated by the bit-stuffing encoder is denoted by X, where $X_{i,j}$ stands for the random bit at location $(i, j) \in B_{m,n}$. To properly define the encoding process, we assume zero entries outside of the quadrant, i.e., for all (i, j) such that $i < 0$ or $j < 0$.

The bit-stuffing construction that was originally proposed by Siegel and Wolf [2] works as follows. The encoder consists of a *binary distribution transformer* followed by a *bit stuffer*. The binary DT is the same element that was used by the one-dimensional coding schemes in the previous chapters. It bijectively converts an unbiased input sequence into a p*-biased* sequence that is subsequently fed into the bit stuffer. However, in this chapter, we change the convention used in previous chapters and let the bias p repre-

sent the probability that a bit is a 1, instead of the probability that it is a 0. This is done for convenience of presentation. The bit stuffer scans $B_{m,n}$ from its upper left corner to its lower right corner, by going down successive diagonals. It applies the following routine on each entry:

- If the current entry already contains a 0, then skip it and go to the next entry.
- If the current entry is empty, then assign the next p-biased bit into it. If the assigned bit is a 1, then check which of the d locations to the right of it and which of the d locations below it is empty. For each such empty location, insert (or *stuff*) a 0.

The 0's that we may encounter in some of the entries are always stuffed 0's that were inserted to the right of or below a previous p-biased 1. Hence, it is unnecessary to repeat stuffing at these entries. As a result, the number of 0's that are stuffed following a biased 1 is sometimes strictly less than $2d$.

At the decoder, we recover the p-biased sequence by applying a similar logic. We successively read the p-biased bits down diagonals, while discarding the stuffed 0's to the right of each 1 and below it. The inverse DT then recovers the unbiased input from the p-biased sequence. Now, note that biased sequences containing fewer 1's will generally result in fewer stuffed bits, yielding a higher average rate in the bit stuffing phase. On the other hand, as we decrease the probability of a 1, the rate of the transformer $h(p)$ decreases (when $p < \frac{1}{2}$). Similarly to the original 1-D bit-stuffing scheme, the overall rate is the product of these two rates, hence we need to optimize p to achieve the best rate.

The above technique was later extended by Roth, Siegel, and Wolf [3], for the special case where $d = 1$. They proposed to use two DT's in order to generate two distinct biased streams at the input to the bit stuffer. When assigning a biased bit into location (i, j), the value of $X_{i-1,j+1}$ determines the biased stream from which to take the bit. Specifically, a p_k-biased bit is assigned when $X_{i-1,j+1} = k$, for $k \in \{0, 1\}$, as illustrated in Figure 5.2.2. At the decoder, the same reasoning is applied to recover the two biased sequences. An analysis of the scheme showed that it achieves improved performance over the single-transformer scheme [3]. In addition, the optimal biases p_0^* and p_1^* were found to satisfy $p_0^* < p_1^* < 0.5$ (recall that $p_i = \mathsf{Pr}(1)$). To interpret this result, suppose that the current biased bit equals 1. If $X_{i-1,j+1} = 0$ then the assigned bit incurs the stuffing

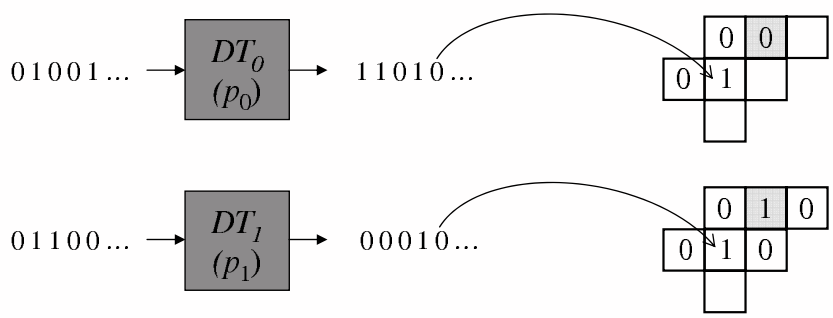


Figure 5.2.2 A bit-stuffing scheme with two biased bit streams for a 2-D $(1, \infty)$ constraint.

of two 0's. On the other hand, when $X_{i-1,j+1} = 1$, position $(i, j + 1)$ is already occupied by a stuffed 0. Thus, only a single 0 is stuffed in this case. Now, recall that biasing the data serves the purpose of reducing the average penalty from stuffing. Hence, it is reasonable to use a smaller bias for a pattern which incurs a higher penalty, as was found in this case. Motivated by the improved performance of this scheme and by the suggested interpretation, we now generalize it for $d \geq 2$.

5.2.1 Multiple-Transformer Schemes for $d \geq 2$

Consider the case where $d = 2$, and assume that the bit stuffer has just assigned a biased 1 into location (i, j). Figure 5.2.3 depicts examples of possible patterns that give rise to stuffing of $1, 2, 3$ and 4 0's, where the stuffed bits appear in bold. It can be seen that the number of stuffed bits depends on the occurrence of 1's in certain previously-filled neighboring locations. In this example, there are four such locations, all highlighted in Figure 5.2.3. Different combinations of 0's and 1's in these locations result in 1 to 4 stuffed bits. In the general case, it can be shown that the number of stuffed bits is determined by the patterns arising in a certain subset of d^2 previously-filled locations. These locations are characterized by the set

$$\begin{aligned}\Gamma(i,j) \quad = \quad & \Big\{(s,t) \in \mathbb{Z}^2 \;\; i{-}d \leq s < i \;\text{ and }\; j < t \leq j{+}d \;\text{ and }\; s{+}t \leq i{+}j\Big\} \\ & \bigcup \Big\{(s,t) \in \mathbb{Z}^2 \;:\; i < s \leq i{+}d{-}1 \\ & \text{and }\; j{-}d \leq t < j{-}1 \;\text{ and }\; s{+}t \leq i{+}j{-}1\Big\}\,.\end{aligned}$$

Figure 5.2.4 depicts this set for $d = 3$, where its entries are marked by thick dots. By accounting for the different patterns, we can show that they yield any number between 1 to $2d$ stuffed bits. The only location that is guaranteed to be empty, and therefore always stuffed with a 0, is location $(i + d, j)$. Following the same reasoning as in the $(1, \infty)$ case, we would like to minimize the expected number of stuffed bits by using smaller biases when encountering patterns that lead to more stuffed bits. Hence, we would generate $2d$ distinct biased streams with biases $p_1, p_2, \ldots p_{2d}$, each one to be used when the corresponding patterns arise. We then need to optimize for the $2d$ biases.

The performance of the scheme was studied for $d = 2$ and $d = 3$ by simulations. The biased streams were encoded into a rectangular array of size 400×400 and the rate was averaged over a number of iterations. To optimize the rate, we performed a brute force search over all possible combinations of the $2d$ biases (without restricting them to satisfy $0.5 > p_1 > p_2 > \ldots > p_{2d}$). Due to computational limitations, a coarse search was initially conducted. A more refined search on a narrower range of probabilities followed it. The finer search resolution consisted of increments of size 0.01 for each bias. The number of iterations was 500 for $d = 2$ and 25 for $d = 3$. For comparison, we simulated the single-transformer scheme. In this case, optimization used a brute force search with a resolution of 0.005. Table 5.2.1 shows the empirical rate estimates of the single-transformer and $2d$-transformers schemes for $d = 2, 3$. Also shown are analytical bounds on the single-transformer scheme that were derived in [4]. We would like to point out that for both $d = 2, 3$, the optimal biases indeed satisfied the expected relations, i.e., $0.5 > p_1^* > p_2^* > \ldots > p_{2d}^*$. In addition, we note that the improved analytical bound on the capacity of the $(2, \infty)$ constraint reported by Forchhammer [5] equals 0.44149. It can be seen from the table that the extension to multiple transformers results in minor improvements for $d = 2, 3$. Unfortunately, computational limitations prevented us from optimizing the scheme for larger values of d, where it may in fact prove to be more useful. Analysis of this approach may also produce improved bounds on capacity, as the current single-transformer analytical bounds are not tight.

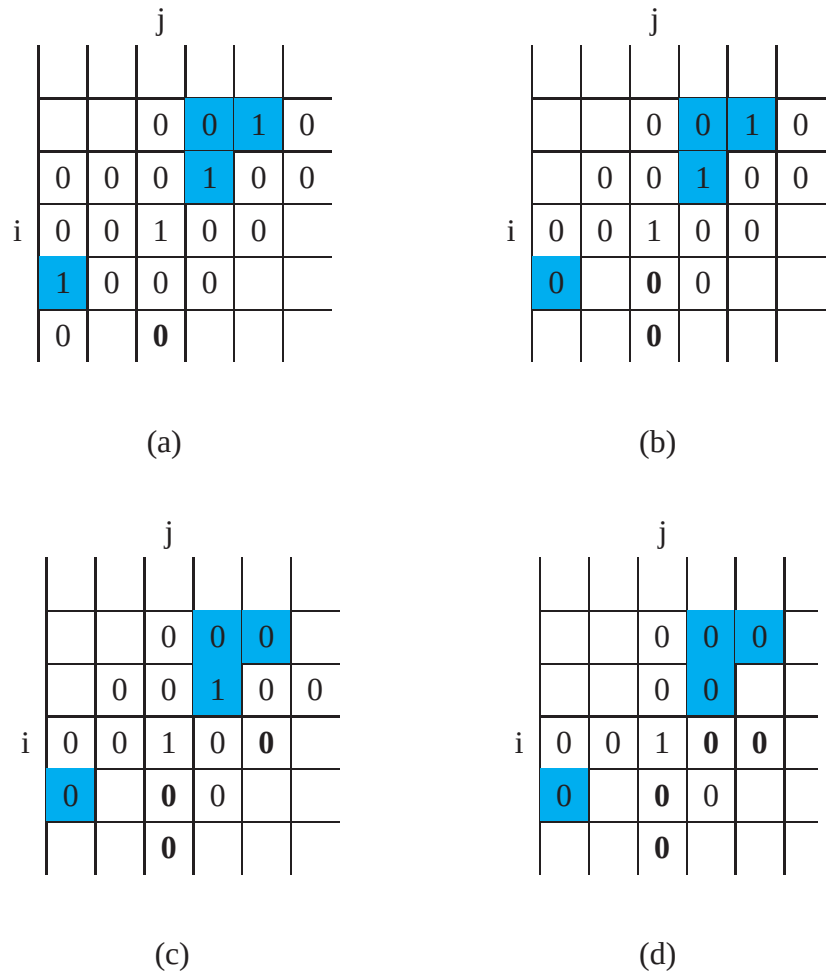


Figure 5.2.3 Bit stuffing for the $(2, \infty)$ constraint. Examples of four patterns that give rise to different numbers of stuffed bit when assigning a biased 1.

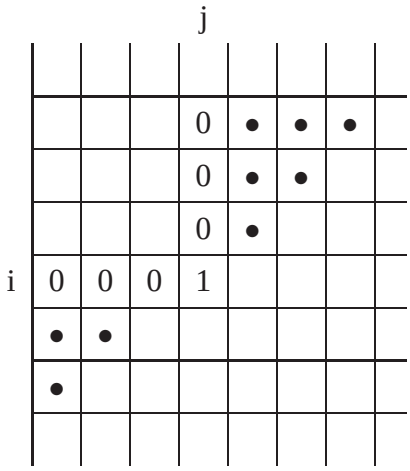


Figure 5.2.4 The set $\Gamma(i,j)$ when $d=3$. The set consists of all entries which affect the number of stuffed bits when assigning a biased 1.

5.3 Bit-Stuffing Schemes for the 'No Isolated Bits' Constraint

A bit-stuffing encoder for the 'no isolated bits' (n.i.b.) constraint was proposed by Halevy *et al.* [4]. The bit stuffer utilizes two distinct inputs - an unbiased stream, denoted by $\{U_n\}_{n=0}^{\infty}$, and a $\frac{2}{3}$-biased stream, denoted by $\{B_n\}_{n=0}^{\infty}$. Progressing down successive diagonals, the bit stuffer applies the following rules to determine the value of each entry $X_{i,j}$:

- If $(X_{i-1,j-1} = X_{i-2,j} = X_{i-1,j+1} \neq X_{i-1,j})$, then set $X_{i,j}$ to equal $X_{i-1,j}$.
- If $(X_{i,j-2} = X_{i-1,j-1} \neq X_{i,j-1})$ and either $(X_{i-1,j-1} \neq X_{i-2,j})$ or $(X_{i-1,j-1} = X_{i-1,j})$, then read the next biased bit B_n. If $(B_n = 1)$, then set $X_{i,j}$ to equal $X_{i,j-1}$. Else, set $X_{i,j}$ to equal the complement of $X_{i,j-1}$.
- Otherwise, set $X_{i,j}$ to equal the next unbiased bit U_n.

The above procedure checks if the bit at location $(i-1,j)$ is currently isolated by the bits to its sides and by the bit above it. If so, then a stuffing of an identical value at location (i,j) prevents a possible violation of the constraint. If this is not the case, then a specific

Table 5.2.1 Empirical estimates and analytical bounds on the rate of bit-stuffing encoders for (d, ∞) constraints.

d	Multiple-Transformers Avg. Rate	Single-Transformer Avg. Rate	Analytical Bounds From [4]
2	0.4447	0.4420	0.4267
3	0.3674	0.3647	0.3402

pattern is searched for. In this pattern, the bit to the left (i.e. $X_{i,j-1}$) is isolated by its neighbors to the left and above, while the bit above (i.e. $X_{i-1,j}$) is not isolated by its neighbors to the left and above. When this pattern occurs, we bias $X_{i,j}$ towards the value of $X_{i,j-1}$. For all other patterns, $X_{i,j}$ will assume equally likely values. One can verify that this process is invertible, hence we can recover the two streams at the decoder.

5.3.1 Schemes Based on One-Dimensional Maxentropic Probabilities

In this subsection, we construct a bit-stuffing scheme for the n.i.b. constraint by drawing a connection to a capacity-achieving scheme for the one-dimensional $(0, 3)$ constraint. The latter scheme is based on a multiple-transformer bit-stuffing method, which was recently proposed by Wolf [6]. A detailed description of this method appears in Section 3.2.2 of Chapter 3. Here, we briefly review it before we present our construction.

The idea behind the 1-D scheme is to emulate a walk on the graph with maxentropic probabilities, by using different biases at different states. Denote the maxentropic probability when moving from state i to state $i + 1$ by μ_i. We first generate a μ_i-biased stream for each state, i, which has a pair of emanating edges. Since the number of such states is $k - d$, we need to generate that many biased streams. Having multiple biased streams, the bit stuffer takes a μ_i-biased bit when in state i. Keeping track of the constraint graph, the random value that each biased bit assumes determines the next state. The single edges leaving all other $d + 1$ states correspond to stuffed bits. Clearly, this method produces maxentropic (d, k)-sequences.

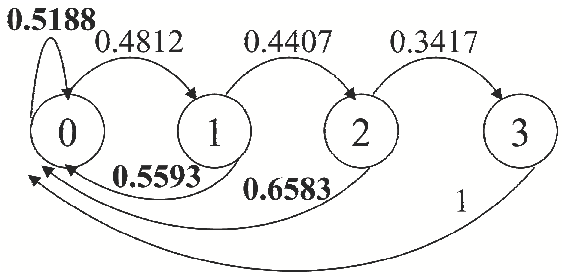


Figure 5.3.1 Maxentropic edge probabilities for the 1-D $(0, 3)$ constraint.

In the context of our work, we are interested in such a capacity-achieving construction for the $(0, 3)$ constraint. Figure 5.3.1 shows the constraint graph with the maxentropic edge probabilities. Looking at the scheme, we observe that the three biases (i.e., the probabilities of a 1) increase with increasing state labels. This can be interpreted by observing that stuffing occurs only when reaching state 3. Therefore, at all other states, we would rather have a 1 and move to state 0 than have a 0 and move to the right. Moreover, as we progress farther right, we are more likely to incur a stuffed-bit penalty. Thus, the closer we get to state 3, the more we wish to avoid it, which is reflected in an increasing bias towards a 1. It is important to note that bit stuffing with a single biased stream, i.e., with the same bias on all edges, does not achieve capacity [7]. Hence, the adjustment of the bias to the "foresight" or likelihood of future stuffed bits resulted in improved performance. Unfortunately, this technique is limited to constraints which have a finite-state graph description. It is not directly applicable to the 2-D n.i.b. constraint, as we are unaware of such a description in this case. However, we can adapt this approach to design a high-rate encoder.

We now describe the bit stuffer for the n.i.b. constraint and draw an analogy to the $(0, 3)$ construction. First, we maintain the stuffing strategy of the encoder that is described in Section 5.3. This means that stuffing occurs at location (i, j) only if the bit at $(i - 1, j)$ is already isolated by its 3 other nearest neighbors. If this is not the case, then a biased bit is assigned to location (i, j). A key observation is that the assignment of the current biased bit can help avoid possible isolation of several bits and hence result in fewer stuffed bits. To illustrate this idea, recall that the n.i.b. constraint prohibits the occurrence of two patterns, as shown in Section 5.1. These patterns involve 5 bits, arranged in the following

configuration:

	a	
b	c	d
	e	

.

For each such subset of bits, the central bit, c, should not be isolated. Now, observe that each bit we write takes any of the positions 'a' to 'e' with respect to prohibited patterns on different subsets of bits. Thus, it might affect the occurrence of these patterns at the corresponding subsets.

Recall that the bit stuffer first checks if we are about to violate the constraint. It then views location (i, j) as assuming position 'e' in the configuration. At this point, the bits at positions 'a', 'b', 'c' and 'd' determine whether or not stuffing occurs. We denote the event that leads to stuffing by $S_{i,j}$, i.e.,

$$S_{i,j} = \{X_{i-1,j} \neq X_{i-2,j}\} \cap \{X_{i-1,j} \neq X_{i-1,j-1}\} \cap \{X_{i-1,j} \neq X_{i-1,j+1}\}.$$

Figure 5.3.2(a) depicts one of the patterns that lead to stuffing, where the configuration entries are highlighted. In case stuffing is not required, we view (i, j) as occupying position 'd' in reference to locations $(i, j-1)$, $(i, j-2)$, $(i-1, j-1)$, and $(i+1, j-1)$. These entries are highlighted in Figure 5.3.2(b). In this case, we know the values at positions 'a', 'b' and 'c'. Thus, we may encounter the following relations: $c \neq b$ and $c \neq a$. Let

$$F_{i,j} = \{X_{i,j-1} \neq X_{i,j-2}\} \cap \{X_{i,j-1} \neq X_{i-1,j-1}\}$$

describe these relations. Then consider the case described by the event $A_{i,j} = \overline{S_{i,j}} \cap F_{i,j}$. Figure 5.3.2(b) shows a pattern that belongs to event $A_{i,j}$. Note that this is just one possible pattern, whereas some other patterns will fall into this category as well. In this case, the value assumed by $X_{i,j}$ might lead directly to stuffing at $(i+1, j-1)$ (i.e., position 'e'). We regard event $A_{i,j}$ as being "very close" to a future stuffing event. Still, we can bias $X_{i,j}$ towards the value at position 'c', to try to avoid this future stuffing event. Thus, we set $d = c$ or $d \neq c$ according to the value of a p_1-biased bit, for some p_1.

Next, we consider the case where $c = b$ or $c = a$ (i.e., event $\overline{F_{i,j}} \cap \overline{S_{i,j}}$), which guarantees that stuffing will not occur at $(i + 1, j - 1)$. We now view (i, j) as if it occupies position 'c', with respect to the highlighted entries shown in Figure 5.3.2(c).

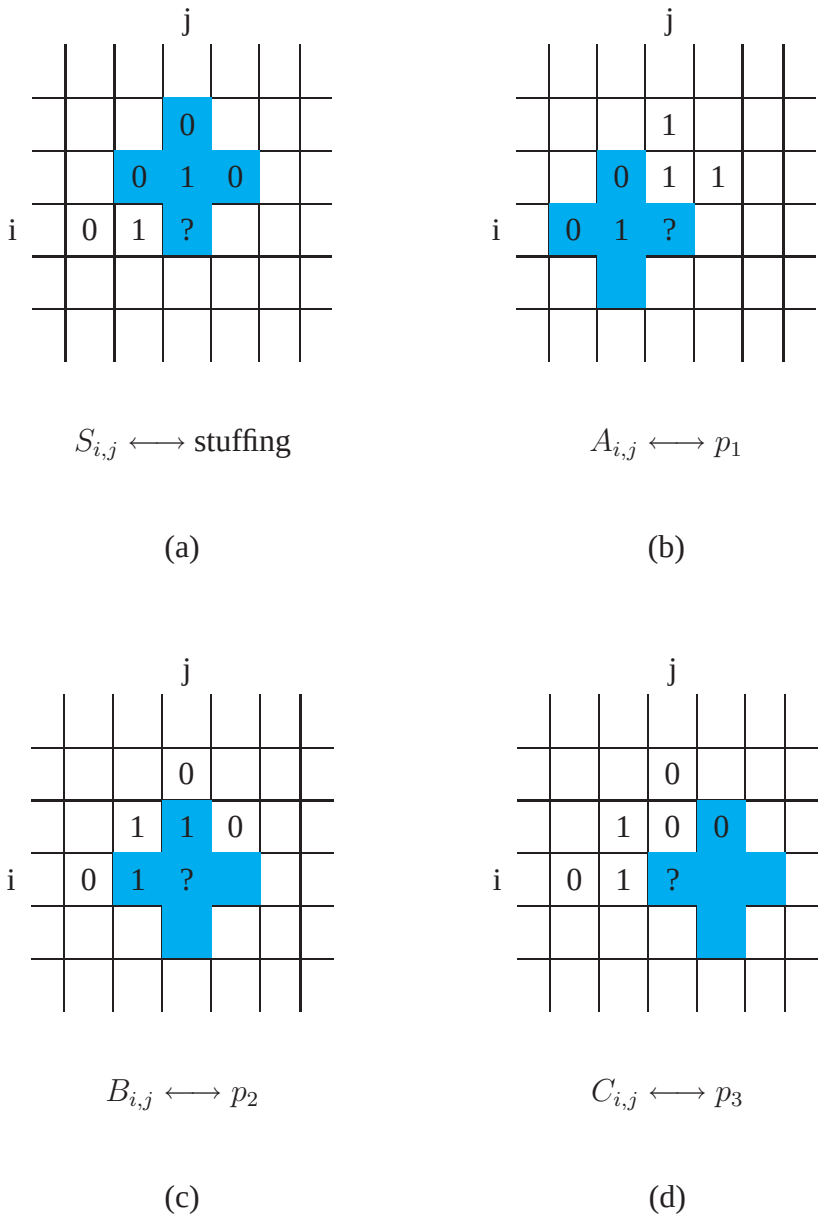


Figure 5.3.2 Bit stuffing for the n.i.b. constraint. Examples of four patterns which correspond to stuffing and to events $A_{i,j}$, $B_{i,j}$ and $C_{i,j}$.

Here we distinguish between two possible patterns: $a = b$ and $a \neq b$, where 'a' $= X_{i-1,j}$ and 'b' $= X_{i,j-1}$. The first pattern, denoted by

$$G_{i,j} = \{X_{i,j-1} = X_{i-1,j}\}$$

allows for possible stuffing at 'e' $= X_{i+1,j}$, depending on the future values at 'c' and 'd', whereas the latter eliminates the possibility of such event. Thus, if we encounter the first pattern, we would like to bias $X_{i,j}$ towards the value at positions 'a' and 'b' (see Figure 5.3.2(c) for an example). Let $B_{i,j}$ denote the event that corresponds to this case, i.e.,

$$B_{i,j} = \overline{S_{i,j}} \cap \overline{F_{i,j}} \cap G_{i,j}.$$

We now compare events $A_{i,j}$ and $B_{i,j}$ according to a criterion of "severity" or "proximity to a future stuffing event". We note that when $A_{i,j}$ occurs, then the value of the current bit determines the occurrence of a stuffing event. However, when $B_{i,j}$ occurs, the current bit can only increase the likelihood of such an event. In this case, stuffing depends on an additional future bit. Hence, we say that $B_{i,j}$ is "farther away" than $A_{i,j}$. Following the reasoning behind the 1-D maxentropic probabilities, the closer we get to stuffing a bit, the more we try to avoid it. Consequently, when $B_{i,j}$ occurs, we would use a bias p_2, such that $p_2 < p_1$. Finally, we consider the second pattern ($a \neq b$), in which case we view (i, j) as 'b' (see Figure 5.3.2(d)). Clearly, biasing the current bit towards the complement of $X_{i-1,j+1}$ ('a') would help prevent stuffing at $(i + 1, j + 1)$ ('e'). This case corresponds to the event

$$C_{i,j} = \overline{S_{i,j}} \cap \overline{F_{i,j}} \cap \overline{G_{i,j}}.$$

We rank it as the "farthest" among the three cases, as here 3 bits need to be assigned before stuffing is determined. We therefore use an even smaller bias p_3, such that $p_3 < p_2 < p_1$.

Having classified the possible patterns into three categories, we now search for the three biases. One approach would be to optimize the rate by a brute force search over all allowable triplets. However, we suggest to choose the biases based on a similarity to the 1-D $(0, 3)$ construction. According to this perspective, a pattern which requires stuffing is analogous to reaching state 3. Event $A_{i,j}$ corresponds to state 2, as this is the closest to stuffing. Hence, we set p_1 to equal the maxentropic bias at this state, i.e., $p_1 = 1-\mu_2 = 0.6583$. Similarly, $B_{i,j}$ corresponds to state 1 and so $p_2 = 1-\mu_1 = 0.5593$. Event $C_{i,j}$ corresponds to state 0, leading to $p_3 = 1 - \mu_0 = 0.5188$.

We simulated the proposed scheme using a 600×600 array and averaged the rate over 250 iterations. The average rate was approximately 0.92218. For comparison, Halevy *et al.* reported an empirical estimate of approximately 0.917 and an analytical bound of

0.91276 on the rate of their scheme [4]. To estimate the capacity, they applied the method proposed by Weeks and Blahut [8]. This resulted in an estimate of the first ten decimal places, namely 0.9238294367. In addition, we performed a brute force optimization of our scheme over all possible biases. A search with increments of 0.001 yielded an average rate of approximately 0.9223, where the optimal biases are $p_1 = 0.654$, $p_2 = 0.552$ and $p_3 = 0.52$. These optimal results are fairly close to the maxentropic $(0, 3)$ probabilities.

Finally, we note that this idea can be extended to an encoding scheme for the n.i.b. constraint defined on the hexagonal lattice. This constraint has been considered for use in future optical disks [1]. In this case, we use maxentropic probabilities from a 1-D $(0, 5)$ constraint. Simulations suggest that the achieved rate is very close to the optimized rate and that the optimal probabilities are close to the maxentropic ones as well. The average rate is approximately 0.9768. However, when applying the method of [8], we could not generate long enough sequences of bounds to get an estimate of the capacity. Still, we could bound it between 0.9583 and 0.9893.

The high rates achieved by our scheme suggest that the connection between the 1-D and the 2-D constraints may not be coincidental. Analysis could possibly provide further insight into this connection, as well as improved bounds on the capacity of the constraints.

5.4 Conclusion and Future Directions

We proposed two new bit-stuffing schemes, one for the class of 2-D run-length-limited (d, ∞) constraints and one for the 2-D 'no isolated bits' constraint. Both schemes are based on interleaving biased bits with multiple different biases into a 2-D array, while stuffing extra bits when necessary. We examined the performance of the suggested schemes through simulations. Results suggest that the scheme for (d, ∞) constraints did not yield significant gains over previous bit-stuffing schemes for small values of d. Since an optimization of this scheme for larger d's is currently impractical, we cannot draw any conclusions in these cases. Therefore, it may still be worthwhile to test the scheme's performance for d's larger than 3 once it is computationally feasible. In addition, analysis of this scheme is an interesting and challenging direction for future work. As we mentioned earlier, such analysis has the potential of producing improved bounds on the unknown

2-D capacity, as the current analytical bounds are not tight.

As opposed to the first scheme, the scheme for the n.i.b. constraint achieved improved empirical rates over a previously suggested scheme. The optimized rate and the rate obtained when assigning the maxentropic probabilities were shown to be very close to each other as well as close to the estimated capacity. Hence, analyzing the suggested scheme may yield improved lower bounds on the capacity. As analysis in this case seems to be difficult, it may be easier to consider a slightly simpler variant of this scheme, where only one bias is used. In other words, one should use the same decision rules as before when assigning the biased bits, but set $p_1 = p_2 = p_3$. Simulations of this scheme achieved an average rate of approximately 0.916, which is very close to the empirical rate of the scheme in [4] (which is approximately 0.917). Since the analytical lower bounds on the latter scheme are not tight, analysis of the former scheme may still improve on these bounds.

Acknowledgment. This chapter is in part a reprint of the material in the paper: S. Aviran, P. H. Siegel, and J. K. Wolf, "Two-dimensional bit-stuffing schemes with multiple transformers," in *Proc. 2005 IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 1478-1482.

Bibliography

[1] A. H. J. Immink, W. M. J. Coene, A. M. van der Lee, C. Busch, A. P. Hekstra, J. W. M. Bergmans, J. Riani, S. J. L. V. Benden, and T. Conway, "Signal processing and coding for two-dimensional optical storage," in *Proc. IEEE Globecom 2003*, San Fanscisco, CA, Dec. 2003, pp. 3904–3908.

[2] P. H. Siegel and J. K. Wolf, "Bit-stuffing bounds on the capacity of 2-dimensional constrained arrays," in *Proc. 1998 IEEE Int. Symp. Inform. Theory*, Cambridge, MA, Aug. 1998, p. 323.

[3] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inform. Theory*, vol. 47 , no. 3, pp. 1166–1176, Mar. 2001.

[4] S. Halevy, J. Chen, R. M. Roth, P. H. Siegel, and J. K. Wolf, "Improved bit-stuffing bounds on two-dimensional constraints," *IEEE Trans. Inform. Theory,* vol. 50, no. 5, pp. 824–838, May 2004.

[5] S. Forchhammer, "Analysis of bit-stuffing codes and lower bounds on capacity for 2-D constrained arrays using quasi-stationary methods," in *Proc. 2004 IEEE Int. Symp. Inform. Theory*, Chicago, IL, Jun./Jul. 2004, p. 161.

[6] J. K. Wolf, "An information theoretic approach to bit stuffing for network protocols," in *Proc. 3rd Asia-Europe Workshop on Information theory*, Kamogawa, Japan, Jun. 2003, pp. 18–21. Also presented at DIMACS Workshop on Network Information Theory, New Jersey, USA, 2003.

[7] P. E. Bender and J. K. Wolf, "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences," in *Proc. 1993 IEEE Int. Symp. Inform. Theory*, San Antonio, TX, Jan. 1993, p. 6.

[8] W. Weeks IV, R. E. Blahut, "The capacity and coding gain of certain checkerboard codes," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 1193–1203, May 1998.

6

Noise-Predictive Turbo Equalization for Partial-Response Channels

6.1 Introduction

Today's digital magnetic recording devices employ a noise-predictive maximum-likelihood (NPML) detection scheme in the readback process [1], [2]. This scheme was introduced to alleviate the effects of noise enhancement and coloration, present in conventional partial-response maximum-likelihood (PRML) systems. The idea is to first equalize the recording channel to a conventional low-degree partial-response (PR) transfer polynomial, $f(D) = 1 + f_1 D^1 + \ + f_M D^M$, where D is a delay operator and the coefficients $\{f_i\}_{i=1}^{M}$ are integers. As this colors the noise, the equalizer is concatenated with a noise whitening filter, whose transfer polynomial is $p(D) = 1 + p_1 D + ... + p_J D^J$. Consequently, the channel is shaped to a generalized partial-response (GPR) polynomial of the form $g(D) = f(D)p(D)$. The filtered samples are then decoded by a sequence detector that takes both $f(D)$ and $p(D)$ into account. Current high-density disk drives use degree-4 polynomials which closely match the recording channel, such as $g(D) = (1 - D)(1 + p_1 D + p_2 D^2 + p_3 D^3)$ and $g(D) = (1 - D^2)(1 + p_1 D + p_2 D^2)$ [3]. The associated NPML detector is simply a 16-state Viterbi detector, matched to the GPR channel model. This combination significantly mitigates the effects of noise enhancement at the cost of increased-complexity detection.

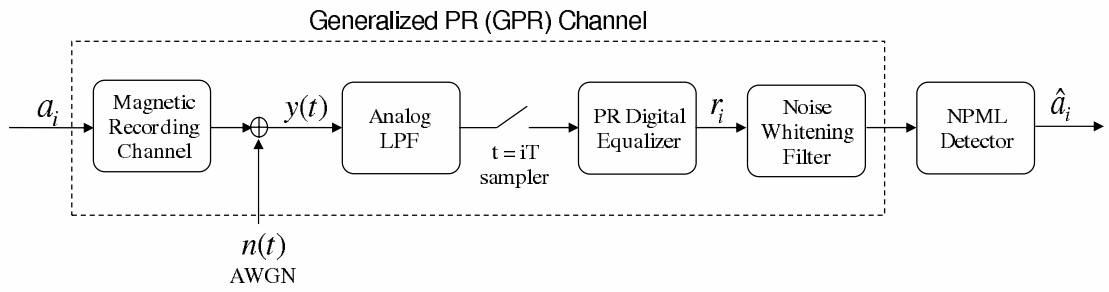


Figure 6.1.1 Block diagram of a partial-response system with an NPML detection scheme.

Iterative decoding and detection schemes have been found to dramatically improve the performance of many communication systems. As such, they are currently regarded as potential candidates for integration into future-generation recording systems. In particular, the framework of *turbo equalization* is the state-of-the-art method of iterative decoding and detection for intersymbol interference (ISI) channels. Within this framework, low-density parity-check (LDPC) error-correction codes are of special interest [4], [5]. They exhibit excellent performance in diverse applications and appear to have the potential to approach the maximum possible rate of transmission over the magnetic recording channel. Numerous authors considered the application of various turbo equalization schemes to partial-response channels [6], [7], [8]. However, these works treated the noise in the system as white noise - an assumption which does not accurately model a realistic recording system.

In [9], Mittelholzer, Dholakia and Eleftheriou modified a standard turbo equalization system for *conventional* PR channels so as to account for the spectral shaping of the noise. Their scheme fits naturally within the NPML framework, incorporating the GPR target $g(D) = (1 - D^2)(1 + p_1 D + p_2 D^2)$ into a standard turbo equalization architecture, as shown in Figure 6.1.2. The idea follows directly from the NPML approach, that is, one first whitens the noise at the output of the PR equalizer, creating a new GPR channel with enhanced ISI but approximately-white noise. Standard iterative detection and decoding methods, which were designed for channels with white noise, are then applied to the new ISI channel. The performance of the modified system was found to be substantially better than that of the baseline turbo equalization system. As with NPML systems, the penalty is in the form of increased detection complexity.

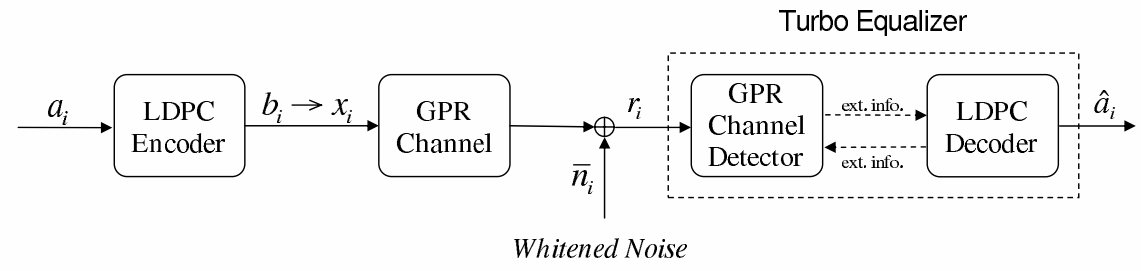


Figure 6.1.2 Block diagram of a generalized partial-response (GPR) system with a standard turbo equalization detection and decoding scheme. The channel detector in this scheme is matched to the GPR channel.

In this chapter, we propose a new approach to noise prediction in a turbo equalization system. Whereas the NPML-based method embeds noise prediction into the channel equalization and detection, the proposed system adjoins a separate noise prediction component to a turbo equalization system for *conventional* PR channels. This configuration offers reduced detection complexity and gives rise to new forms of noise prediction.

6.2 System Model

The block diagram in Figure 6.2.1 illustrates a standard turbo equalization scheme for magnetic recording systems. A stream of message bits $\{a_i\}$ is divided into L-bit blocks, where each block is encoded by a low-density parity-check (LDPC) encoder into a binary codeword of length N $(b_1, b_2, \cdots b_N)$. The bit stream is then mapped into a bipolar symbol stream $\{x_i\}$, $x_i \in \{+1, -1\}$, which is written on the disk. We model the recording channel by a Lorentzian channel model with additive white Gaussian noise (AWGN) due to system electronics. During the read process, the readback signal is passed through an analog low-pass filter, followed by a sampler. A zero-forcing equalizer shapes the overall transfer function to a target PR polynomial $f(D) = 1 + f_1 D^1 + \ + f_M D^M$. This leads to noise enhancement and spectral coloration. The output of the equalizer can be expressed as

$$r_i = x_i + \sum_{m=1}^{M} f_m x_{i-m} + n_i, \tag{6.2.1}$$

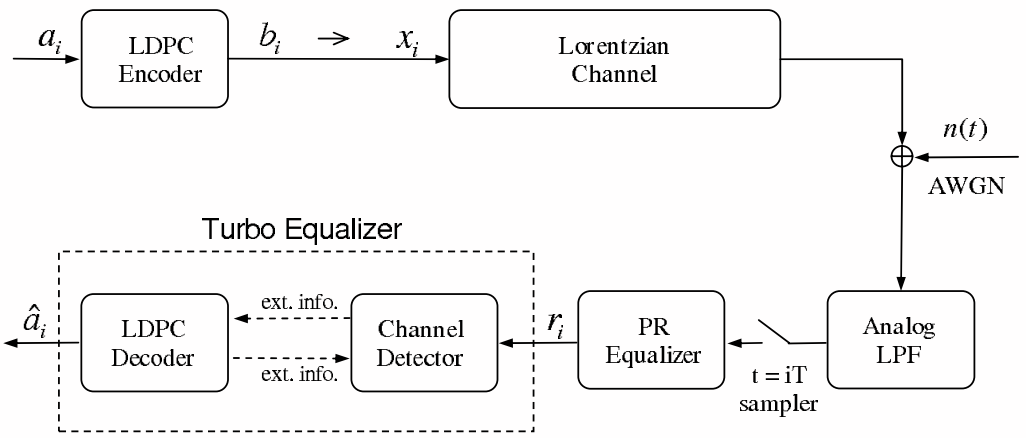


Figure 6.2.1 Block diagram of a partial-response system with a turbo-equalization detection and decoding scheme.

where n_i is sampled colored noise. The equalizer outputs are then decoded by a turbo equalization scheme, which includes a PR channel detector and an LDPC decoder.

6.3 Turbo Equalization for ISI channels

Turbo equalization (TE) is a technique for suppressing ISI by iteratively exchanging probabilistic or *soft* information between a channel detector and an error-correction code (ECC) decoder. The idea was introduced by Douillard *et al.* shortly after the advent of turbo codes and their iterative decoding technique [18]. It extended the turbo-decoding principles from the decoding of coded transmission over a memoryless channel to the joint equalization (detection) and decoding of coded transmission over channels with memory. Since then, this concept has evolved into a generic method, which has been studied and applied to diverse applications, including magnetic recording systems [7], [8], [9], [10], [11].

The main attribute of the constituent channel detector and ECC decoder is their ability to operate on bitwise soft information as their input, as well as producing bitwise soft information at their output. Such components are called soft-input soft-output (SISO)

modules, employing SISO algorithms. The SISO characteristic is what facilitates the iterative process of exchanging the soft information between the modules. When adequately performed, the process results in substantial performance gains over traditional non-iterative detection and decoding schemes. The soft information that flows within the turbo equalization system (henceforth the *turbo equalizer*) pertains to various probability estimates. The turbo equalizer's final outputs correspond to estimates of the probabilities that each of the transmitted bits takes on a value of 0 or 1. Compared to the hard bit-decisions (i.e., 0 or 1) delivered by traditional decoding/detection algorithms, these estimates convey additional information of how reliable their associated hard-decisions are. Moreover, by accounting for reliability information, SISO algorithms can make use of the more reliable estimates in order to provide improved-reliability estimates of bits whose initial estimates were not as reliable. The soft information pertaining to a bit is usually expressed in terms of the ratio of two probability estimates, each corresponding to one of the two possible hard-decisions. It is also common to work with the logarithm of these ratios, as it simplifies the required computations.

We distinguish between three types of soft information in the system: prior, posterior, and extrinsic. The input to a SISO algorithm serves as *prior* information with respect to that algorithm, and corresponds to bitwise probability estimates that are available before the algorithm's processing takes place. The algorithm then processes the prior information in conjunction with any additional knowledge it has on the coded bits, such as their respective noisy observations and/or certain relations between them. The processing results in two probabilistic quantities, the first being the exact (or approximate) conditional probability of a bit's value, given the information that is accessible to the algorithm. This is called the *a posteriori* probability (APP), as it forms an updated or *posterior* information on each bit. The second quantity is the *extrinsic* information, which can be straightforwardly computed for each bit from its prior and posterior information, as will be explained later. The latter form of output is used when a SISO module is incorporated within an iterative scheme that involves other SISO modules. In particular, it is the extrinsic output which is passed by a SISO module to the other modules. One can think of it as capturing new information, which the module has obtained by utilizing knowledge that is available *only* to it.

A generic turbo equalization scheme works as follows [12], [13]. The detector begins by producing extrinsic information based on noisy output samples from the channel, the channel model, and any prior information on the bits. Typically, the transmitted bits are assumed to be equally likely, in which case no prior information is available for the detector. Next, the detector's bitwise extrinsic information is provided to the ECC decoder which, in turn, treats it as prior information when generating its own extrinsic output. The ECC decoder's extrinsic output, reflecting the code constraints, is then fed back to the detector for use as prior information. This completes one detection and decoding iteration, and the process repeats for several iterations until a stopping criterion is met. The final output of the iterative process is the posterior information produced by the ECC decoder at the last iteration. It consists of estimates of the *a posteriori* probability ratios (APPR) of the coded bits, given the channel output samples, the channel model, and the code constraints. Final hard-decisions are then made by choosing the most likely value for each bit, or alternatively, by zero-slicing the logarithm of the APPR's. At this point, we note that some instances of turbo equalization require the interleaving of the encoded data before it is sent through the channel. At the turbo equalizer, matching interleaving and de-interleaving take place when information is passed between the two constituent modules. Interleaving serves the purpose of weakening the statistical dependencies among nearby bits. In this way, it contributes to more accurate estimation by the employed algorithms. In the case of LDPC codes, interleaving is inherently built into the code structure and need not be performed explicitly. Since our work focuses on LDPC codes, we chose to omit interleaving from the above description.

Various SISO algorithms and error-correcting schemes can fit within the framework we have described. A typical choice for a channel detector is the maximum *a posteriori* probability (MAP) detector, which is based on the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [14]. Other common choices include approximations to the BCJR algorithm [15], and the soft-output Viterbi algorithm (SOVA) [16]. Another type of detector, which was shown to perform well for several partial-response channels, is based on the application of the sum-product algorithm [17] to a particular graphical description of the ISI channel [8], [10]. Commonly used error-correction schemes other than LDPC codes include parallel concatenated convolutional codes (more familiar as "turbo codes") and a single

convolutional code [18]. Similar to channels with memory, convolutional codes can be decoded by the BCJR algorithm, by its approximations, or by the SOVA algorithm. In the case of turbo codes, two such decoding elements operate concurrently, while exchanging soft information according to the "turbo principle" described above. LDPC codes are usually decoded by the message-passing algorithm, also known as the sum-product or belief propagation algorithm [17], but may also be decoded by approximations to it [9].

In this work we focus on schemes that employ a MAP detector for the partial-response channel and a message-passing decoder for an LDPC code. In the next sections we review MAP detection, LDPC codes, and the message-passing algorithm for decoding LDPC codes. A detailed description and rigorous derivation of the BCJR and message-passing algorithms can be found, for example, in [12].

6.3.1 Maximum A Posteriori Probability (MAP) Detection for ISI Channels

As evident from its name, a maximum *a posteriori* probability detector finds the hard bit-decisions which maximize the *a posteriori* probability (APP) of each transmitted bit, given the sequence of channel outputs and given prior bit probabilities. It is well known that such a detector achieves the minimum bit-error rate and is thus optimal in that sense. The MAP detector is based on the BCJR algorithm, operating on the trellis description of the ISI channel. Assuming that the noise is white Gaussian with known variance, the BCJR algorithm computes the APP ratio (APPR) of each transmitted bit

$$Q(b_i|r_1, r_2, \cdots, r_N) \doteq \frac{\mathsf{Pr}(b_i = 1|r_1, r_2, \cdots, r_N)}{\mathsf{Pr}(b_i = 0|r_1, r_2, \cdots, r_N)} = \frac{\mathsf{Pr}(x_i = +1|r_1, r_2, \cdots, r_N)}{\mathsf{Pr}(x_i = -1|r_1, r_2, \cdots, r_N)}, \tag{6.3.1}$$

where $(r_1, r_2, \cdots, r_N)$ is the received sequence of channel outputs. The detector then sets the bit estimate $\hat{b_i}$ to the value with the maximum APP, i.e.,

$$\hat{b_i} = \arg \max_{b \in \{0,1\}} \mathsf{Pr}(b_i = b|r_1, r_2, \cdots, r_N).$$

Before we continue to describe the BCJR algorithm, we note that it was originally introduced in the broader context of estimating the transitions of a hidden Markov model. Hence, it is applicable to several other decoding and detection problems, such as the

decoding of convolutional codes [14] and the detection of constrained sequences [12]. In what follows, we review only the special case of detection for ISI channels, highlighting the key concepts of the algorithm and the points of relevance to our work.

Consider the following model, describing the samples $\{r_i\}$ obtained at the output of a noisy ISI channel

$$r_i = \sum_{m=0}^{M} h_m x_{i-m} + n_i. \tag{6.3.2}$$

Here, $\{x_i\}$ is the bipolar input to the channel, n_i is a white Gaussian noise sample with variance σ^2, and the h_m's represent the real-valued channel impulse response. The algorithm is best described using the trellis diagram description of the corresponding noiseless channel model

$$\rho_i = \sum_{m=0}^{M} h_m x_{i-m}. \tag{6.3.3}$$

The trellis has 2^M states of the form $(x_{i-M}, \cdots, x_{i-2}, x_{i-1})$, which maintain the memory of the channel. Let S be the set of channel states, and denote by s_i the state of the channel at time index i. From each state $s_{i-1} = (x_{i-M}, \cdots, x_{i-2}, x_{i-1})$, it is possible to reach only two states at the next time index i, depending on the value of x_i. Each such valid state transition corresponds to an edge or a branch in the trellis, connecting s_{i-1} to s_i. We label each branch with a pair $(x(s_{i-1}, s_i), \rho(s_{i-1}, s_i))$, where $x_i = x(s_{i-1}, s_i)$ and $\rho_i = \rho(s_{i-1}, s_i)$ are the input and the noiseless channel output that correspond to this transition, respectively. Finally, we use the notation $\mathbf{r}_j^k$ to represent the subsequence of consecutive channel output samples $(r_j, r_{j+1}, \cdots, r_{k-1}, r_k)$.

An efficient computation of the APP's in (6.3.1) relies on a particular decomposition of the joint probability

$$\mathsf{Pr}(s_{i-1} = s', s_i = s, r_1, r_2, \cdots, r_N), \tag{6.3.4}$$

where s' and s can be any two channel states. This probability relates to a particular transition (branch) at time index i. It can be shown that the channel's finite memory together with the white noise assumption lead to the following factorization:

$$\begin{aligned}
\mathsf{Pr}(s_{i-1}, s_i, r_1, r_2, \cdots, r_N) &= \mathsf{Pr}(s_{i-1}, s_i, \mathbf{r}_1^{i-1}, r_i, \mathbf{r}_{i+1}^N) \\
&= \mathsf{Pr}(s_{i-1}, \mathbf{r}_1^{i-1}) \cdot \mathsf{Pr}(s_i, r_i | s_{i-1}) \cdot \mathsf{Pr}(\mathbf{r}_{i+1}^N | s_i) \\
&\doteq \alpha_{i-1}(s_{i-1}) \cdot \gamma_i(s_{i-1}, s_i) \cdot \beta_i(s_i),
\end{aligned}$$

where

$$\alpha_{i-1}(s_{i-1}) \doteq \mathsf{Pr}(s_{i-1}, \mathbf{r}_1^{i-1})$$
$$\beta_i(s_i) \doteq \mathsf{Pr}(\mathbf{r}_{i+1}^N | s_i)$$
$$\gamma_i(s_{i-1}, s_i) \doteq \mathsf{Pr}(s_i, r_i | s_{i-1}).$$

The terms $\alpha_{i-1}(s_{i-1})$ and $\beta_i(s_i)$ are called *forward state metrics* and *backward state metrics*, respectively. The core of the BCJR algorithm is a recursive computation of these metrics for each state $s \in S$ and for each time index i, according to the following recursions:

$$\alpha_i(s) = \sum_{\forall s' \in S} \alpha_{i-1}(s')\gamma_i(s', s) \tag{6.3.5}$$

and

$$\beta_i(s) = \sum_{\forall s' \in S} \beta_{i+1}(s')\gamma_{i+1}(s, s'). \tag{6.3.6}$$

The forward recursion in (6.3.5) is typically initialized with $\alpha_0(\mathbf{0}) = 1$, and $\alpha_0(s) = 0$ for all $s \neq \mathbf{0}$, where $\mathbf{0}$ stands for the all-zero channel state. Similarly, the backward recursion in (6.3.6) is initialized with $\beta_N(\mathbf{0}) = 1$, and $\beta_N(s) = 0$ for all $s \neq \mathbf{0}$. These initial values require the last M bits of each block to be set to zero, which incurs a slight rate loss. Nonetheless, other initial values are possible as well, provided that they indeed reflect the distribution of the states at times $i = 0$ and $i = N$. Both recursions make use of *branch metrics* $\gamma_i(s_{i-1}, s_i)$, which correspond to the likelihood of the transitions between the states. These are computed before the algorithm carries out the forward and backward recursions.

The branch metric $\gamma_i(s_{i-1}, s_i)$ can be written as follows

$$\gamma_i(s_{i-1}, s_i) = \mathsf{Pr}(s_i, r_i | s_{i-1}) = \mathsf{Pr}(s_i | s_{i-1}) \cdot \mathsf{Pr}(r_i | s_{i-1}, s_i).$$

It equals zero for each pair of states s', s which are not connected by an edge. For each pair of connected states, i.e., for each state transition $s' \to s$, the branch metric takes the form

$$\gamma_i(s', s) = P^{\text{prior}}(x_i = x(s', s)) \cdot \mathsf{Pr}(r_i | s_{i-1} = s', s_i = s), \tag{6.3.7}$$

where $P^{\text{prior}}(x_i)$ is the prior bit probability that is fed to the detector. Under the assumption of additive Gaussian noise, the second probability in (6.3.7) is given by

$$\tilde{\gamma}_i(s', s) \doteq \mathsf{Pr}(r_i | s', s) = \mathsf{Pr}(r_i | \rho_i = \rho(s', s)) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(r_i - \rho(s', s))^2 / 2\sigma^2}, \tag{6.3.8}$$

where $\rho(s', s)$ is the corresponding noiseless channel output, given in (6.3.3). We call $\tilde{\gamma}_i(s', s)$ the *branch likelihood*, as it utilizes the channel outputs to estimate how likely a branch was traversed. It follows from (6.3.7) that the branch metric explicitly accounts for both the channel outputs and the prior bit information to indicate the probability of a transition. On the other hand, the recursive computation of the forward and backward state metrics incorporates the branch metrics, thus implicitly accounting for prior information and for the channel outputs. In the context of turbo equalization, it is important to note that the update of prior probabilities between iterations requires the rerunning of these recursions. However, the branch likelihoods do not change and need be computed only once, during the first iteration.

Finally, we combine the different metrics to produce the APPR values as follows. We first compute the joint probability $\mathsf{Pr}(x_i = x, \mathbf{r}_1^N)$ by summing the transition probabilities $\mathsf{Pr}(s_{i-1}, s_i, \mathbf{r}_1^N)$ over all the branches that assume an input of $x_i = x$, i.e.,

$$\mathsf{Pr}(x_i = x, \mathbf{r}_1^N) = \sum_{\forall s' \to s \,:\, x(s',s)=x} \mathsf{Pr}(s_{i-1} = s', s_i = s, \mathbf{r}_1^N). \tag{6.3.9}$$

Dividing (6.3.9) by $\mathsf{Pr}(\mathbf{r}_1^N)$, we obtain the bit's APP $\mathsf{Pr}(x_i = x|\mathbf{r}_1^N)$, and the APP ratio

$$Q(b_i|\mathbf{r}_1^N) = \frac{\mathsf{Pr}(x_i = +1|\mathbf{r}_1^N)}{\mathsf{Pr}(x_i = -1|\mathbf{r}_1^N)} = \frac{\sum_{\forall s' \to s \,:\, x(s',s)=+1} \alpha_{i-1}(s')\gamma_i(s', s)\beta_i(s)}{\sum_{\forall s' \to s \,:\, x(s',s)=-1} \alpha_{i-1}(s')\gamma_i(s', s)\beta_i(s)}. \tag{6.3.10}$$

At a last step, a bit-decision is made by comparing $Q(b_i|\mathbf{r}_1^N)$ to 1, thus choosing the value whose corresponding APP is larger.

As mentioned earlier, when incorporating a SISO detector in a turbo equalizer, one is interested in the extrinsic portion of the detector's output. We obtain the extrinsic information by factoring the term $P^{\text{prior}}(x_i)$ out of both sums in (6.3.10), thus writing the APPR as

$$Q(b_i|\mathbf{r}_1^N) = \frac{P^{\text{prior}}(x_i = +1) \cdot \sum_{\forall s' \to s \,:\, x(s',s)=+1} \alpha_{i-1}(s')\tilde{\gamma}_i(s', s)\beta_i(s)}{P^{\text{prior}}(x_i = -1) \cdot \sum_{\forall s' \to s \,:\, x(s',s)=-1} \alpha_{i-1}(s')\tilde{\gamma}_i(s', s)\beta_i(s)}.$$

The APPR is now split into two quantities, the prior information and the extrinsic information, as follows

$$Q(b_i|\mathbf{r}_1^N) = \frac{P^{\text{prior}}(x_i = +1)}{P^{\text{prior}}(x_i = -1)} \cdot Q^{\text{extrinsic}}(b_i|\mathbf{r}_1^N),$$

where

$$Q^{\text{extrinsic}}(b_i|\mathbf{r}_1^N) \doteq \frac{\sum_{\forall s' \to s \,:\, x(s',s)=+1} \alpha_{i-1}(s')\tilde{\gamma}_i(s',s)\beta_i(s)}{\sum_{\forall s' \to s \,:\, x(s',s)=-1} \alpha_{i-1}(s')\tilde{\gamma}_i(s',s)\beta_i(s)}. \tag{6.3.11}$$

6.3.2 Low-Density Parity Check (LDPC) Codes

Binary low-density parity-check (LDPC) codes are linear block error-correcting codes defined by binary parity-check matrices in which the proportion of 1's is very low. Such parity-check matrices are said to be *sparse* or to have a *low density* of 1's. LDPC codes were introduced more than 40 years ago by Gallager in his Ph.D. thesis, together with an associated iterative SISO decoding algorithm [4]. However, these codes were essentially forgotten until the mid-90's, when the advent of iterative decoding techniques has led researchers to rediscover them [5]. It has been shown that the performance of LDPC codes can get very close to theoretical limits on a memoryless channel with AWGN, although this requires the block length to be quite large. Nowadays, due to their acceptable decoding complexity and their remarkable performance, they are of great practical and theoretical interest to researchers.

A binary linear block code can be defined by a binary parity-check matrix H, where the code consists of all binary codewords $\mathbf{b} = (b_1, b_2, \cdots, b_N)$ that satisfy a set of linear equations (in modulo 2 arithmetic) given by $H\mathbf{b}^T = \mathbf{0}^T$. An example of a parity-check matrix and its associated linear equation system appears in Figure 6.3.1. We denote by M the number of rows in H, where each row corresponds to a *parity-check equation* that all codewords must satisfy. Hereafter, we assume that H is full rank, in which case the number of input message-bits is $L = N - M$ and the rate of the code is $\frac{L}{N}$. In case H is not full rank, i.e., it contains some linearly dependent rows, one can remove the extra dependent rows to obtain a matrix H', containing the remaining independent rows. In this case, M equals the number of rows in H', and the code rate is higher than in the full-rank case. Gallager considered parity-check matrices where there is a fixed number of 1's per column and a fixed number of 1's per row. Codes defined by such matrices are now called *regular* LDPC codes. In recent years, it has been found that matrices with varying numbers of 1 along rows and columns can yield codes which outperform regular codes. These are known as *irregular* LDPC codes.

The definition of LDPC codes through their parity-check matrices is closely related to their associated decoding algorithm, known as *message-passing decoding*. Such definition does not specify, however, any particular encoding method. An obvious approach to mapping L-bit input blocks into N-bit codewords is to obtain a systematic generator matrix for the code [19]. We obtain it by first reducing H to the form of $H^\star = [P|I_L]$ via Gaussian elimination, where I_L is an identity matrix of dimensions $L \times L$ and P is of dimensions $L \times M$. The systematic generator matrix then takes the form $G = \left[I_L|P^T\right]$. Encoding is done by multiplying the input block $\mathbf{a}$ by G, i.e., $\mathbf{b} = \mathbf{a}G$, where the first L bits of $\mathbf{b}$ are the information bits in $\mathbf{a}$. However, as opposed to the sparse parity-check matrix H, the matrix P is generally not sparse and hence the encoding complexity is high (proportional to N^2). This poses a practical problem, especially in view of the fact that in general, LDPC codes are very powerful only when the block length is quite large. In practice, however, one can use various methods that provide greatly reduced encoding complexity. Some of these methods take advantage of the sparseness of H, whereas others propose modifications of LDPC codes or structured LDPC codes that allow for simple encoding.

Before we continue to describe the iterative decoding of LDPC codes, it is helpful to introduce a graphical description of these codes, known as a *Tanner graph*. We will later use this graph to explain the principles of message-passing decoding. A Tanner graph of an LDPC code is a bipartite graph, containing two distinctive sets of nodes, with edges only connecting nodes from different sets. One set contains N *variable nodes*, each corresponding to a bit in the codeword, while the other set contains M *parity-check nodes*, each corresponding to one parity-check equation imposed on the bits. An *edge* connects variable node i with parity-check node j if the bit associated with variable node i participates in the parity-check equation that node j represents. Figure 6.3.1 illustrates the Tanner graph representation of the code that is defined by the matrix H shown in the figure. The squares denote the parity-check nodes (or simply *check nodes*) and the circles denote the variable nodes (or *bit nodes*). Switching between the two code representations is straightforward - each variable node is associated with a column in H and each parity-check node is associated with a row in H. The edges correspond to the 1's in H.

The Tanner graph provides a complete characterization of the code's structure. Its key

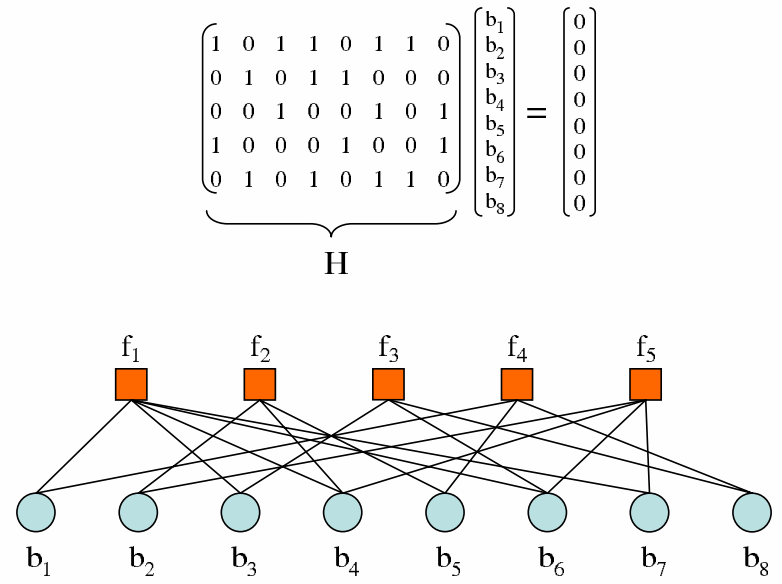


Figure 6.3.1 A parity-check matrix representation of an LDPC code and its associated Tanner graph.

property is the description of the complex overall dependencies between the bits through a system of simple relations between small subsets of the bits. These simple relations correspond to parity-check constraints that are imposed on subsets of the bits, and are referred to as *local* constraints. Their simplicity allows for simple local decoding, which amounts to evaluating the probabilities of each of the participating bits, based on these relations. The interrelations between the various local dependencies form the overall code structure. It is through these interrelations that the local structures communicate their evaluations and obtain new information from each other. The exchange of information takes place along the edges of the graph in the form of sent messages. The simple local decoding then repeats while accounting for the new information, yielding updated results which reflect the relationships between "distant" bits.

The described approach to the decoding of a complex structure illustrates the principles of the message-passing algorithm. In fact, Tanner graphs and message-passing decoding of LDPC codes are instances of the much wider concepts of *factor graphs* and the *sum-product* algorithm [17]. Factor graphs provide a means of expressing how a global function factors into local functions, and are suitable for modeling the dependencies among different elements in various communication systems. The sum-product algo-

rithm is a computationally efficient algorithm for evaluating the global function through local computations and an exchange of information along the edges of the factor graph. The global function is typically the posterior probability distribution of certain variables, given the system's structure as modeled by the graph. It is worthwhile noting that the BCJR algorithm is another special case of the sum-product algorithm, where messages are serially passed along a graphical description of an ISI channel.

6.3.3 Message-Passing Decoding of LDPC Codes

In the message-passing algorithm, messages are passed along the edges of the Tanner graph from bit nodes to check nodes and vice versa. The computation of messages is carried out at the bit nodes and check nodes, and a message corresponds to two probability estimates that are associated with bit values 0 and 1. It is also convenient to work with the ratio of the two estimates, as in the BCJR algorithm. Before presenting the algorithm, we introduce the following notations. Let b_i denote the i'th bit node and f_j denote the j'th check node, and suppose an edge e_{ij} connects b_i and f_j. A message from bit node b_i to check node f_j is composed of two values: $\mu_{b_i \to f_j}(0)$ and $\mu_{b_i \to f_j}(1)$. In the opposite direction, check node f_j sends bit node b_i a message with two values $\mu_{f_j \to b_i}(0)$ and $\mu_{f_j \to b_i}(1)$. Finally, we denote by F_i the set of check nodes that are connected to b_i by an edge, i.e., the set of all parity-check equations in which b_i is involved. Similarly, B_j stands for the set of bit nodes connected to f_j by an edge, or for all bits that participate in the j'th parity-check equation.

The input to the algorithm consists of initial estimates of each bit's probability. These estimates comprise the prior bit probabilities $P^{\text{prior}}(b_i)$, $i = 1, 2, \cdots, N$, with respect to the algorithm. They are available either from some prior processing of the channel outputs and/or from other SISO modules operating in conjunction with the LDPC decoder (as in turbo equalization). A single iteration entails the calculation and sending of messages from the bit nodes to the check nodes and from the check nodes to the bit nodes. The fundamental local decoding principle is as follows: the *outgoing* message from a node along an edge is a function of the *incoming* messages along all *other* edges connected to that node. Specifically, a message from a bit node b_i to a check node f_j is the following

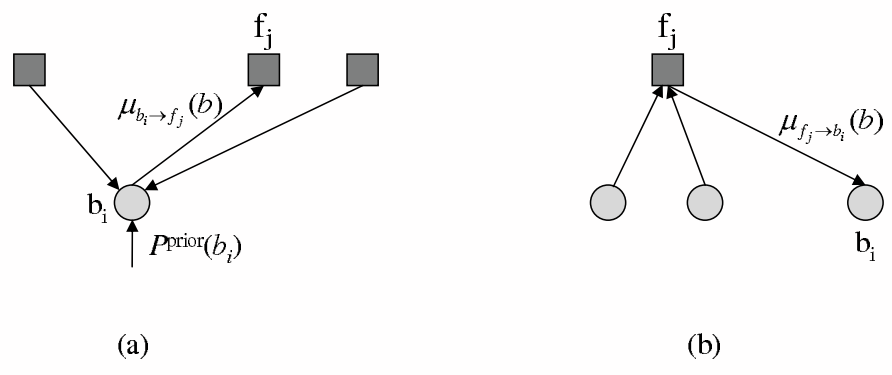


Figure 6.3.2 Message-passing local decoding rule: the outgoing message from a node along an edge is a function of the incoming messages along all *other* edges connected to that node.

product

$$\mu_{b_i \to f_j}(b) = k_{ij} \cdot P^{\text{prior}}(b_i = b) \cdot \prod_{f_{j'} \in F_i \setminus \{f_j\}} \mu_{f_{j'} \to b_i}(b), \qquad b \in \{0, 1\}, \qquad (6.3.12)$$

where k_{ij} is a normalization constant ensuring that $\mu_{b_i \to f_j}(0) + \mu_{b_i \to f_j}(1) = 1$. The $\mu_{f_{j'} \to b_i}(b)$'s are messages that were previously passed to b_i along all edges connected to it except for e_{ij}, as illustrated in Figure 6.3.2(a). Note that in the figure, we model the prior information as an incoming message from an outer source flowing along an extra edge. The message $\mu_{b_i \to f_j}(b)$ represents the probability that $b_i = b$ given information from adjacent check nodes except for f_j itself, and given prior information on the bits. An implicit assumption in its evaluation is that all incoming messages convey statistically independent probabilities, hence the product in (6.3.12).

The local knowledge that a check node possesses is that the values of its connected bit nodes sum to 0. The message from f_j to b_i $\mu_{f_j \to b_i}(b)$ reflects this information by capturing the likelihood that the j'th equation holds given that $b_i = b$, and given probability distributions of the other bits involved. Figure 6.3.2(b) demonstrates the participating incoming and outgoing messages. At first, it seems that the computation of the likelihood requires an exhaustive consideration of all bit-configurations that are consistent with a zero-sum. Fortunately, this seemingly complex calculation has a simplified solution, proved by Gal-

lager [4]. It turns out that the probability that I *independent* bits sum to 0 equals

$$\frac{1}{2} + \frac{1}{2}\prod_{i=1}^{I}(1-2p_i)\,,$$

where $\{p_1, p_2, \cdots, p_I\}$ are their respective probabilities of assuming a 1. Accordingly, the probability that the bits sum to 1 equals

$$\frac{1}{2} - \frac{1}{2}\prod_{i=1}^{I}(1-2p_i)\,.$$

Since the incoming messages from adjacent bit nodes are in the form of their probabilities, the likelihood of the j'th equation being satisfied, given that $b_i = b$, is

$$\begin{aligned}
\mu_{f_j\to b_i}(0) &= \frac{1}{2}\left(1 + \prod_{b_{i'}\in B_j\setminus\{b_i\}}\left(1-2\mu_{b_{i'}\to f_j}(1)\right)\right)\\
\mu_{f_j\to b_i}(1) &= \frac{1}{2}\left(1 - \prod_{b_{i'}\in B_j\setminus\{b_i\}}\left(1-2\mu_{b_{i'}\to f_j}(1)\right)\right).
\end{aligned} \tag{6.3.13}$$

Note that again, the decoding rule relies on the assumption of independence of the incoming probabilities.

The message-passing algorithm entails repeated calculations of messages by Equations (6.3.12) and (6.3.13) and their exchange according to a certain given schedule. This process stops once a certain condition is met, such as when a maximum number of iterations is reached. The algorithm outputs two types of probabilistic quantities - an estimate of each bit's *extrinsic* probability with respect to the code, and an estimate of each bit's *a posteriori* probability with respect to the code. The extrinsic probability describes the new information that was obtained from the knowledge of the code structure, i.e., from the parity-check relations imposed on the bits. It is the product of *all* incoming messages

$$P^{\text{extrinsic}}(b_i = b \mid H\mathbf{b}^T = \mathbf{0}^T) = k_i \cdot \prod_{f_j\in F_i}\mu_{f_j\to b_i}(b), \qquad b\in\{0,1\}, \tag{6.3.14}$$

where k_i normalizes the product such that $P^{\text{extrinsic}}(0) + P^{\text{extrinsic}}(1) = 0$. The APP relates to the bit probability conditioned on the knowledge of the code structure. It accounts for both the prior information and the extrinsic information as follows

$$P^{\text{posterior}}(b_i = b \mid H\mathbf{b}^T = \mathbf{0}^T) = k_i' \cdot P^{\text{prior}}(b_i = b) \cdot \prod_{f_j\in F_i}\mu_{f_j\to b_i}(b), \qquad b\in\{0,1\}, \tag{6.3.15}$$

where k_i' is another normalizing constant. As noted earlier, one can also take the quotient of the two probabilities of each type to form the following extrinsic output

$$Q^{\text{extrinsic}}(b_i \mid H\mathbf{b}^T = \mathbf{0}^T) = \prod_{f_j \in F_i} \frac{\mu_{f_j \to b_i}(1)}{\mu_{f_j \to b_i}(0)},$$

and APPR output

$$Q(b_i \mid H\mathbf{b}^T = \mathbf{0}^T) = \frac{P^{\text{prior}}(b_i = 1)}{P^{\text{prior}}(b_i = 0)} \cdot Q^{\text{extrinsic}}(b_i \mid H\mathbf{b}^T = \mathbf{0}^T).$$

The algorithm prescribes the rules for message calculation but leaves flexibility for different schedules of message passing. Basically, there is a choice among a range of schedules, from a fully parallel schedule to a serial schedule. In a fully parallel schedule, all nodes (both checks and bits) compute their messages concurrently, and exchange them afterwards. A serial schedule means that the nodes compute and pass their outputs one after the other, according to a certain order. A commonly used schedule is to divide each iteration into two halves. In the first half, all bit nodes compute and send their messages to the check nodes. In the second half, all check nodes make their estimates and send them to the bit nodes. We adopt the latter schedule in the description of the algorithm that we give below. Finally, we note that the choice of schedule may affect the decoding performance as well as implementation complexity.

Another factor which may affect the performance of a code is the presence of cycles in its Tanner graph, where cycles are paths which start and end at the same node. It is well known that when the graph has no cycles, message-passing can yield the exact bitwise APP's given the code structure. However, the existence of cycles can be seen to violate the independence assumption on which Equations (6.3.12) - (6.3.15) rely. In such cases, the algorithm outputs only an estimate or an approximation of the APP, which might be poor, thereby degrading performance. Still, there is ample empirical evidence that message-passing decoding yields good estimates for many codes whose corresponding graphs contain cycles. It is widely presumed, however, that very short cycles, especially those of length 4, can severly degrade message-passing performance, and should thus be avoided. A comprehensive analytical treatment of the effects of cycles on the decoding performance has not been developed to date.

We end this section with a summary of the message-passing algorithm for decoding of LDPC codes. We note that the description to follow is just one of a number of possi-

ble applications of message-passing principles to this problem. As we mentioned before, there is room for various schedules and stopping rules. We introduce the following shorthand notation. Let p_i denote the prior probability that the i'th bit equals 1, and let $q_i(0)$ and $q_i(1)$ denote the estimates of the bit's *a posteriori* probabilities at the output of the algorithm. The corresponding extrinsic probabilities are $q_i^{\text{ext}}(0)$ and $q_i^{\text{ext}}(1)$. We denote the bit-to-check and check-to-bit messages by $q_{ij}(b)$ and $r_{ji}(b)$, respectively. For each bit node b_i, the set $C_i = \{j : h_{ji} = 1\}$ consists of the indices of its adjacent check nodes. Similarly, for each check node f_j, the set $R_j = \{i : h_{ji} = 1\}$ represents the indices of its adjacent bit nodes.

Message-Passing Algorithm for Decoding LDPC Codes

- **Step 0.** Initialize. For all edges e_{ij} on the graph set

$$r_{ji}(0) = r_{ji}(1) = \frac{1}{2}.$$

  For all $i = 1, \cdots, N$, set p_i to the prior bit probabilities that are input to the algorithm.

- **Step 1.** Compute messages from bit nodes to check nodes. For all edges e_{ij} on the graph calculate

$$\begin{aligned} q_{ij}(0) &= k_{ij} \cdot (1 - p_i) \cdot \prod_{j' \in C_i \setminus \{j\}} r_{j'i}(0) \\ q_{ij}(1) &= k_{ij} \cdot p_i \cdot \prod_{j' \in C_i \setminus \{j\}} r_{j'i}(1), \end{aligned} \tag{6.3.16}$$

  where k_{ij} is chosen such that $q_{ij}(0) + q_{ij}(1) = 1$.

- **Step 2.** Compute messages from check nodes to bit nodes. For all edges e_{ij} on the graph calculate

$$\begin{aligned} r_{ji}(0) &= \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_j \setminus \{i\}} (1 - 2q_{i'j}(1)) \\ r_{ji}(1) &= 1 - r_{ji}(0). \end{aligned} \tag{6.3.17}$$

- **Step 3.** Compute tentative *posterior* outputs. For all i calculate

$$\begin{aligned} q_i(0) &= k_i' \cdot (1-p_i) \cdot \prod_{j \in C_i} r_{ji}(0) \\ q_i(1) &= k_i' \cdot p_i \cdot \prod_{j \in C_i} r_{ji}(1), \end{aligned} \tag{6.3.18}$$

  where k_i' ensures that $q_i(0) + q_i(1) = 1$.

  The corresponding *extrinsic* outputs are computed as

$$\begin{aligned} q_i^{\text{ext}}(0) &= k_i \cdot \prod_{j \in C_i} r_{ji}(0) \\ q_i^{\text{ext}}(1) &= k_i \cdot \prod_{j \in C_i} r_{ji}(1), \end{aligned} \tag{6.3.19}$$

  where k_i ensures that $q_i^{\text{ext}}(0) + q_i^{\text{ext}}(1) = 1$.

- **Step 4.** Check stopping criterion. For all i make a tentative bit decision from *posterior* output by

$$\hat{b_i} = \begin{cases} 1, & q_i(1) > q_i(0) \\ 0, & \text{otherwise.} \end{cases} \tag{6.3.20}$$

  If $H\hat{\mathbf{b}}^T = \mathbf{0}^T$ *or* if we have reached a maximum number of iterations, then stop. Else, go to step 1.

6.4 Noise Predictive Turbo Equalization

6.4.1 Background: Noise Prediction

In noise prediction, one uses a set of known noise values $\{n_{i+k} : k \in K\}$ to predict an unknown noise term n_i. Given a set of indices K we are interested in a linear predictor

$$p^K(D) = \sum_{k \in K} c_k D^k$$

that minimizes the mean-squared prediction error, defined as

$$E[e_K^2] \doteq E[(n_i - \hat{n_i}] = E[(n_i - \sum_{k \in K} c_k n_{i+k})^2].$$

Here, $\hat{n_i}$ stands for the predicted value of n_i. The optimal predictor coefficients $\{c_k^K\}$, together with the minimal mean-squared error (MMSE), can be derived from the noise autocorrelation [20]. In magnetic recording systems, where the noise is colored by a tapped delay line equalizer, the noise autocorrelation is merely a function of the equalizer's coefficients. Given the computed autocorrelation, the derivation of the optimal predictor coefficients amounts to the solution of a system of linear equations, known as the *Yule-Walker equations*. In order to simplify the presentation, we shall use the following terminology throughout the section. Given a set K, the predictor $p^K(D)$ always represents the *optimal* MMSE predictor for that set. The error power of $p^K(D)$ refers to the *minimal* mean-squared error $E[(e_K^\star)^2]$.

The following property is of considerable importance in the context of this work. If the noise n_i is correlated, then the MMSE is smaller than the noise power, i.e.,

$$E[(e_K^\star)^2] = E[(n_i - \sum_{k \in K} c_k^K n_{i+k})^2] < E[n_i^2].$$

As we will explain later, we attempt to predict the actual colored noise sample n_i and then replace it with the resulting prediction error $e_K^\star$. Thus, aside from a partial whitening effect on colored noise, it can be seen that the application of noise prediction has the benefit of reducing noise power. Both effects can contribute to improved detection and decoding.

6.4.2 A Noise Predictive Turbo Equalization Scheme

Figure 6.4.1 depicts a noise-predictive turbo equalization (NPTE) system. Each iteration starts with a standard turbo equalizer iteration, consisting of a single BCJR pass followed by T LDPC decoding iterations. The APPR's produced by the TE, $\{Q_i\}$, are used to determine a subset of the colored noise terms that can be reliably estimated. Given a predefined threshold, $R \geq 1$, the system performs the following actions for each bit.

1. If $Q_i > R$ or $Q_i < \frac{1}{R}$, then declare x_i *reliable* and set $\tilde{x_i} = 1$ or $\tilde{x_i} = -1$, respectively.

2. If the M+1 consecutive bits $\{x_i, x_{i-1}, , x_{i-M}\}$ are declared reliable, then compute a

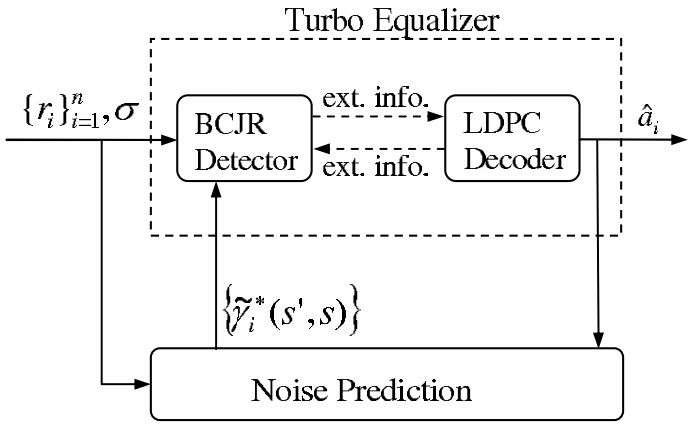


Figure 6.4.1 Noise-predictive turbo equalization system block diagram.

reliable colored noise estimate, $\tilde{n}_i$, by

$$\tilde{n}_i = r_i - \tilde{x}_i - \sum_{m=1}^{M} f_m \tilde{x}_{i-m}. \tag{6.4.1}$$

Otherwise, mark n_i as *non-estimated*.

Next, for each noise term, a variable length predictor of the form

$$p^K(D) = \sum_{k=-l}^{-1} c_k^K D^k + \sum_{k=1}^{m} c_k^K D^k, \quad l, m > 0,$$

is found, and the resulting predicted value and prediction error power are then determined, as follows.

3. For each bit, search for neighboring reliable colored noise estimates on both past and future samples. Stop at the first non-estimated noise term in each direction. Let $i - v_i$ and $i + z_i$ denote the indices of the outermost past and future estimated noise terms, respectively.

4. For a predefined integer $J > 0$, representing the maximum number of prediction taps, consider all possible sets of the form

$$K_{l,m} = \{k \in \mathbb{Z} \setminus \{0\} : -l \leq k \leq m\},$$

where l, m are nonnegative integers such that $l + m \leq J$.

- If $v_i + z_i \leq J$ then set $(l_i, m_i) = (v_i, z_i)$.
- If $v_i + z_i > J$ then find a pair (l_i, m_i) that minimizes the error power of $p^{K_{l,m}}(D)$, over all sets $K_{l,m}$ with $l \leq v_i$, $m \leq z_i$ and $l + m = J$.

Set $K^i = K_{l_i,m_i}$.

If $(l_i, m_i) \neq (0, 0)$ then set

$$\hat{n}_i = \sum_{k=-l_i}^{-1} c_k^{K_i} \tilde{n}_{i+k} + \sum_{k=1}^{m_i} c_k^{K_i} \tilde{n}_{i+k} \quad \text{and} \quad \sigma_i^{*2} = E[(e_{K_i}^{\star})^2]. \tag{6.4.2}$$

Otherwise, set $\hat{n}_i = 0$ and $\sigma_i^{*2} = \sigma^2$.

Note that predictors of different noise terms may use different sets of neighboring reliable estimates combined with the appropriate predictor coefficients. It is also possible that both l_i and m_i equal 0. In this case, no prediction will occur in the current iteration. Moreover, the predictor of any given noise term may change and need to be recalculated between TE iterations. It is also worth pointing out that the coefficients and error power that correspond to each of the sets $K_{l,m}$ are computed in advance using the known noise autocorrelation.

Finally, new sample values are produced for each bit. Updated BCJR branch likelihoods that correspond to these values are derived as follows, to be used in the next iteration.

5. Let $r_i^* = r_i - \hat{n}_i$. For each trellis branch calculate

$$\tilde{\gamma}_i^*(s', s) = \frac{1}{\sqrt{2\pi}\sigma_i^*} e^{-(r_i^* - \rho(s',s))^2/2\sigma_i^{*2}}. \tag{6.4.3}$$

Note that new likelihoods are calculated only when a prediction occurs. In case it does not, one merely needs to set the likelihoods to their initial values (i.e., where r_i and σ_i were used in (6.4.3) and not r_i^* and σ_i^*). In the next iteration, the BCJR will utilize the updated likelihoods along with updated prior probabilities obtained from the LDPC decoder.

It is through Equation (6.4.3) that the modified samples and their associated variances are accounted for in the next round of detection. To justify (6.4.3), we first express the

new sample as

$$r_i^* = \sum_{m=0}^{M} f_m x_{i-m} + \left(n_i - \left(\sum_{k=-l_i}^{-1} c_k^{K_i} \tilde{n}_{i+k} + \sum_{k=1}^{m_i} c_k^{K_i} \tilde{n}_{i+k} \right) \right). \tag{6.4.4}$$

We make the assumption that correct noise estimates are fed back, in view of the threshold-based selection of reliable bit estimates. Furthermore, as TE iterations progress, we expect that soft outputs of increasing reliability will be generated. Under this assumption, (6.4.4) reduces to

$$r_i^* = \sum_{m=0}^{M} f_m x_{i-m} + e_{K_i}^{\star}. \tag{6.4.5}$$

Observing that $e_{K_i}^{\star}$ is a zero-mean Gaussian random variable of variance $\sigma_i^{*2} = E[(e_{K_i}^{\star})^2]$, we obtain Equation (6.4.3), with the PR channel branch label $\rho(s', s) = \sum_{m=0}^{M} f_m x_{i-m}$.

We can now summarize the entire NPTE procedure with the following outline of the algorithm.

- **Step 0.** Initialize. Compute an initial set of branch likelihoods, based on channel outputs and variance.
- **Step 1.** Iterate once on TE. Check stopping criterion.
- **Step 2.** Determine reliable bits. Make tentative colored noise estimates from available reliable bit decisions.
- **Step 3.** Predict noise by using up to J tentative estimates.
- **Step 4.** Obtain *updated samples* based on the prediction of Step 3. Replace BCJR likelihoods with updated likelihoods that correspond to these samples. Go to Step 1.

6.5 Simulation Results

Simulations were performed for a Lorentzian channel with AWGN at recording densities of 2.60 and 2.85. The channel was equalized to two different PR targets using a finite-length approximation to an ideal lowpass filter and a 21-tap zero-forcing equalizer. The

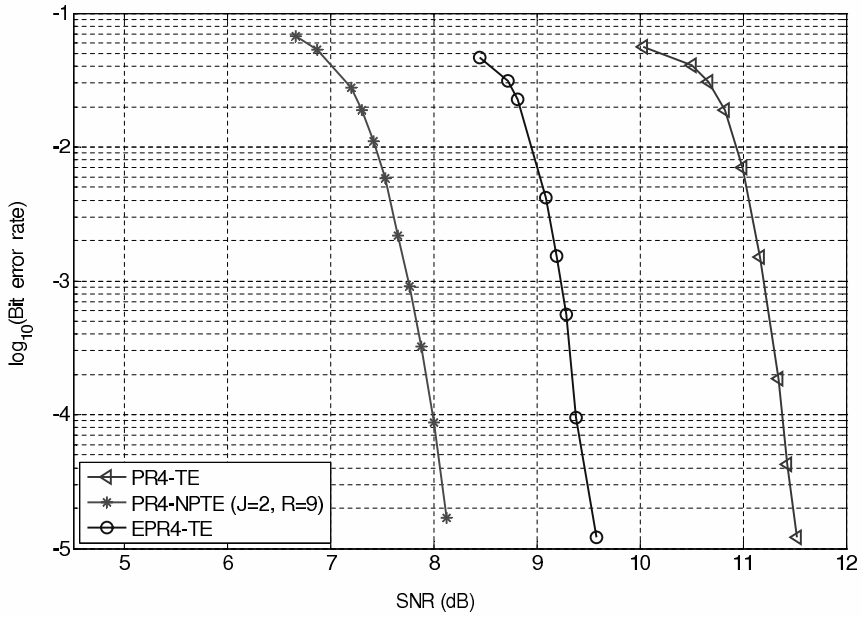


Figure 6.5.1 Performance of turbo equalization schemes at recording density $PW50/T_b = 2.60$.

equalization targets were the PR4 ($f(D) = 1-D^2$) and EPR4 ($f(D) = (1-D)(1+D)^2$) channels, giving rise to 4-state and 8-state trellises, respectively. The LDPC code has rate 8/9 and a block length of 4896. Each turbo equalizer iteration consists of running the BCJR algorithm once and subsequently performing a single message-passing decoding iteration, i.e., T=1. A single turbo equalizer iteration is followed by the above-mentioned noise prediction procedure, both together comprising a single detection/decoding iteration. The maximum number of detection/decoding iterations was 24. Simulations of the NPTE scheme with various threshold values between R=1 and R=50 indicated that a threshold of R=9 obtains good overall performance, hence R=9 was used for all simulations. The maximum number of prediction taps was set to J=2. This means that one can predict a noise sample by using either up to 2 past (to the left) nearest neighbors, or up to 2 future (to the right) nearest neighbors, or up to 1 neighbor on each side. The signal-to-noise-ratio (SNR) is defined as the ratio of the mean-square signal value and the variance of the noise, both measured at the input to the equalizer.

Figure 6.5.1 shows performance curves for a recording density of 2.60. We simulated a standard turbo equalizer for a PR4 channel (PR4-TE) as well as the suggested NPTE

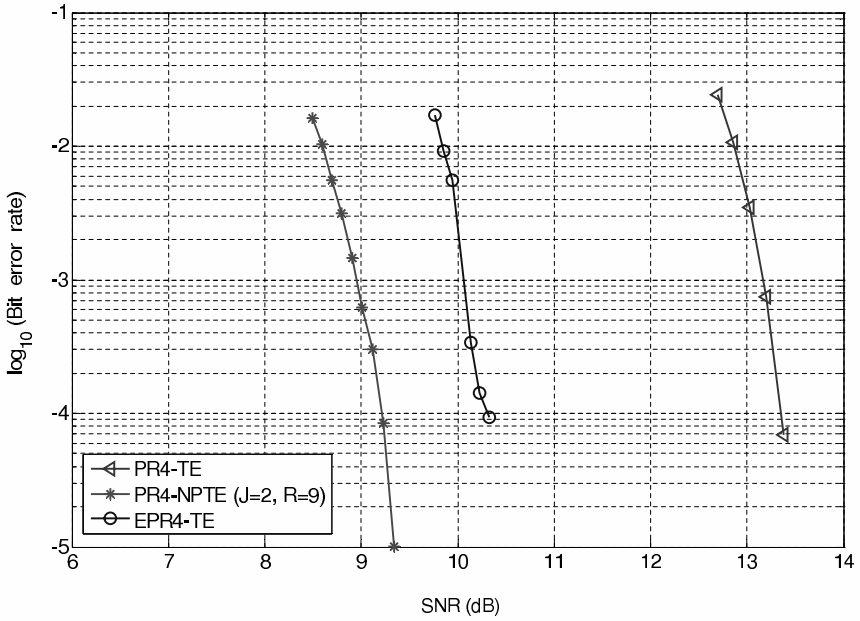


Figure 6.5.2 Performance of turbo equalization schemes at recording density $PW50/T_b = 2.85$.

scheme for the same channel (PR4-NPTE). We also considered a standard turbo equalizer matched to an EPR4 channel (EPR4-TE) as an alternative scheme. The reason is that at high densities, an EPR4 channel provides a better fit to the real channel characteristics than a PR4 channel, but in turn, is more complex (i.e., has a larger memory). The statistics were gathered using simulation runs of 3,000 codewords or approximately 10^7 transmitted bits. It can be seen that NPTE with R=9 and J=2 on a PR4 channel outperforms standard TE on an EPR4 channel by about 1.5 dB. Similar gains were reported in [9] for a standard TE on the 16-state trellis target $g(D) = (1 - D^2)(1 + p_1 D + p_2 D^2)$. A similar effect, but of a slightly smaller magnitude, holds for a density of 2.85, as one can observe from Figure 6.5.2. Note that the same TE and NPTE parameters were used at this density as well.

As noted earlier, we also tested the sensitivity of the NPTE system performance to the reliability threshold R. The performance curves shown in Figure 6.5.3 pertain to selected threshold values ranging from R=1 to R=50. All other system parameters remain fixed at the values reported above. The curves demonstrate that NPTE performance improves with increasing value of R up to approximately $9 \leq R \leq 12$. Then, performance starts

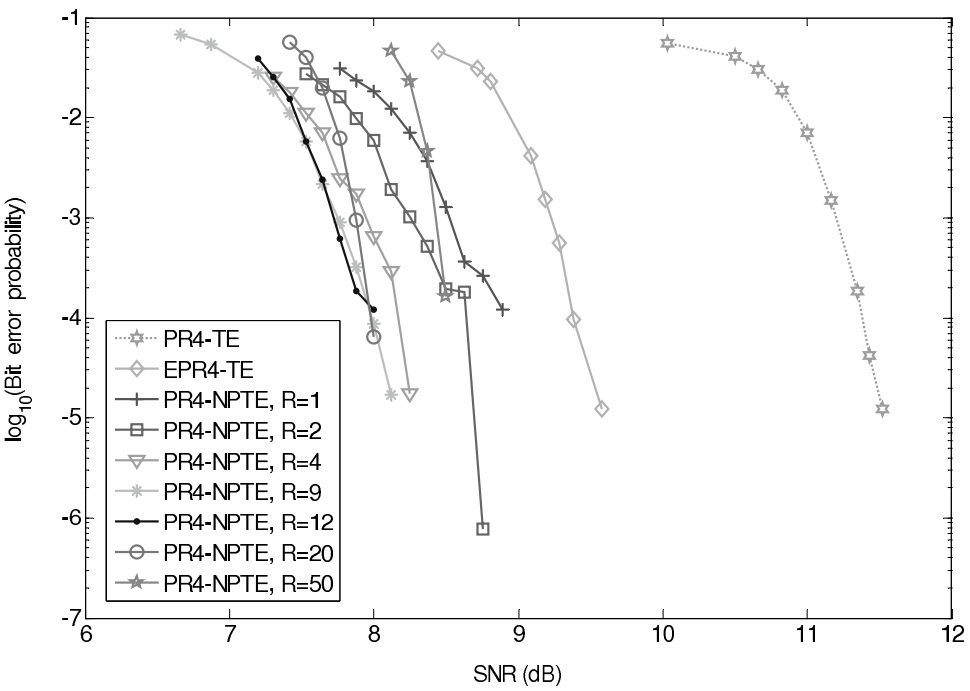


Figure 6.5.3 Sensitivity of the NPTE system to the reliability threshold R at recording density $PW50/T_b = 2.60$.

degrading when further increasing R. A possible reason is that a too small R may result in feedback of many incorrect estimates leading to mispredictions, whereas a too large R may yield too few reliable estimates and thus limited and infrequent predictions.

6.6 Conclusion and Discussion

The emergence of iterative decoding principles has opened the door to new methods for handling colored noise. In this chapter, we proposed a new method for incorporating noise prediction into a standard turbo equalizer. It is based on iteratively whitening the noise in a selective manner, while utilizing the soft information produced by a standard turbo equalizer. When applied to a PR4 channel, the proposed scheme substantially improves upon conventional turbo equalizers for both PR4 and EPR4 channels. It achieves performance comparable to previously suggested methods that are based upon turbo equalization of generalized PR channels, while reducing the overall system complexity.

In order to comprehend the differences between present technologies and the proposed new method, recall that the former integrate noise prediction into the channel structure and detection. The present conventional approach is rooted in the characteristics of existing detection and decoding techniques. Specifically, noise prediction is irrelevant to the traditional ECC decoder (e.g., for decoding of Reed-Solomon codes [19]), which operates on hard-decisions. In contrast, an iterative SISO ECC decoder can both benefit from and contribute to noise processing. Consider, for example, the transmission of LDPC encoded data over a memoryless channel with additive colored noise. One could design a noise-predictive LDPC decoder, based on the ideas presented here. That is to say, it is possible to slightly adapt the noise prediction procedure described in Section 6.4.2 to operate on the soft inputs and outputs of the LDPC decoder.

In the case of ISI channels, prediction is a stand-alone module that exchanges soft information with the turbo equalizer. One benefit of our approach is the use of a simpler channel model with fewer states than corresponding generalized PR channel models. In addition, iterative systems provide soft or reliability information for the entire block on a bit-basis. This facilitates an adequately reliable estimation in Step 2 of the NPTE procedure, as well as a judicious and noncausal prediction in Step 3. Reliable estimation combined with a careful choice of tentative prediction taps helps reduce misprediction. Consequently, it results in increasingly refined noisy samples and thereby in improved turbo equalization performance. The causal prediction filters used in NPML systems and in [9] arise from the serial nature of maximum-likelihood Viterbi detectors [1]. Specifically, at each time instance (i.e., for each bit), only information on prior bits has been processed and can be fed back in the form of tentative noise estimates. Nevertheless, *noncausal* prediction filters may enhance noise reduction compared to *causal* prediction filters with the *same* number of taps. Indeed, for the noise models considered here and when J=2, prediction through one adjacent past sample and one adjacent future sample incurs a smaller error power than that associated with the 2 previous nearest or the 2 next nearest neighbors. This effect might not hold for arbitrary noise models, but can be shown to hold in our case (i.e., when equalizing from densities 2.60 and 2.85 to a PR4 channel).

The proposed prediction module is self-contained, and as such, its integration is not limited to the systems described earlier. In general, it may be added to a variety of SISO

systems, as long as the noise model is known or has been estimated. For example, it can be used for joint equalization and decoding of two-dimensional (2-D) ISI channels, arising in novel page-oriented storage technologies (see, e.g., [21]). Noise prediction for a two-dimensional correlation model was proposed in [22], where a 2-D ISI channel was equalized to a memoryless channel, using a zero-forcing equalizer. The power of the arising colored noise was reduced through 2-D noise prediction. The noise whitening process was incorporated into a 2-D multilevel coding with multistage decoding scheme, where multiple LDPC codewords are interleaved into an array. In a similar manner, one can use a zero-forcing equalizer in conjunction with a single LDPC code that encodes the whole array. Then, the NPTE algorithm can be adapted to whiten the 2-D colored noise between message-passing iterations. Obviously, this approach is not limited to zero-forcing equalization. For example, it can work jointly with 2-D detection schemes such as the iterative multi-strip (IMS) detector, proposed in [23].

We conclude this chapter by mentioning a recently published related work by Kaynak, Duman, and Kurtas [24] on noise predictive belief propagation (NPBP) detection. The authors suggest an alternate way of harnessing the advantages of state-of-the-art detection techniques to combat the effects of colored noise. They incorporate causal noise prediction filters into the SISO partial-response channel detector that was proposed in [8] (see also [10] for further results). This detector applies the message-passing algorithm to a certain graphical description of the PR channel. Its prime advantage is in its parallel message-passing schedule, facilitating a fully parallel implementation of the algorithm, in contrast to the serial schedule used by the BCJR algorithm. The proposed NPBP detector obtains tentative bit decisions and noise estimates in a similar manner to the NPTE algorithm. However, its whitening process is not selective and utilizes fixed and causal predictors. All noise samples are predicted systematically, regardless of the reliability of the relevant estimates. Although a certain threshold-based technique is applied in order to alleviate feedback of erroneous bit decisions, it does not benefit from the reliability information provided by the SISO detector. An important quality of NPBP is that its integration of noise prediction retains the detector's parallel architecture. This allows for fast detection as well as for fast parallel implementation of NPBP turbo equalization with LDPC codes. It would be interesting to compare the performances of NPTE and NPBP,

and also to consider the integration of NPTE principles into the parallel detector in [8].

Acknowledgment. The author is grateful to Joseph Soriaga and Panu Chaichanavong for helpful discussions and for their assistance in the implementation of the NPTE scheme. This chapter is in part a reprint of the material in the papers: S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," in *Digests IEEE Int. Magnetics Conf. INTERMAG 2005*, Nagoya, Japan, Apr. 2005, pp. 981-982 and S. Aviran, P. H. Siegel, and J. K. Wolf, "Noise-predictive turbo equalization for partial-response channels," *IEEE Trans. Magnetics*, vol. 41, no. 10, pp. 2959-2961, Oct. 2005.

Bibliography

[1] E. Eleftheriou and W. Hirt, "Improving performance of PRML/EPRML through noise prediction," *IEEE Trans. Magnetics*, vol. 32, no. 5, pp. 3968–3970, Sept. 1996.

[2] J. D. Coker, E. Eleftheriou, R. L. Galbraith, and W. Hirt, "Noise-predictive maximum likelihood (NPML) detection," *IEEE Trans. Magnetics*, vol. 34, no. 1, pp. 110–117, Jan. 1998.

[3] R. D. Cideciyan, J. D. Coker, E. Eleftheriou, and R. L. Galbraith, "Noise predictive maximum likelihood detection combined with parity-based post-processing," *IEEE Trans. Magnetics*, vol. 37, no. 2, pp. 714–720, Mar. 2001.

[4] R. G. Gallager, *Low-Density Parity-Check Codes*. The M.I.T. Press, Cambridge, MA, USA, 1963.

[5] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[6] J. L. Fan, A. Friedmann, E. M. Kurtas, and S. W. McLaughlin, "Low density parity check codes for magnetic recording," in *Proc. 37th Allerton Conf. Commun. Control, and Computing*, Monticello, IL, Sept. 1999, pp. 1314–1323.

[7] T. V. Souvignier, M. Öberg, P. H. Siegel, R. E. Swanson, and J. K. Wolf, "Turbo decoding for partial response channels," *IEEE Trans. Commun.*, vol. 48, no. 8, pp. 1297–1308, Aug. 2000.

[8] B. M. Kurkoski, P. H. Siegel, and J. K. Wolf, "Joint message-passing decoding of LDPC codes and partial-response channels," *IEEE Trans. Inform. Theory*, vol. 48, no. 6, pp. 1410–1422, June 2002.

[9] T. Mittelholzer, A. Dholakia, and E. Eleftheriou, "Reduced-complexity decoding of low density parity check codes for generalized partial response channels," *IEEE Trans. Magnetics*, vol. 37, pp. 721–728, Mar. 2001.

[10] G. Colavolpe and G. Germi, "On the application of factor graphs and the sum-product algorithm to ISI channels," *IEEE Trans. Commun.*, vol. 53, no. 5, pp. 818–825, May 2005.

[11] W. E. Ryan, L. L. McPheters, and S. W. McLaughlin, "Combined turbo coding and turbo equalization for PR4-equalized Lorentzian channels," in *Proc. Conf. Information Sciences and Systems (CISS'98)*, Princeton, NJ, Mar. 1998, pp. 489–493.

[12] J. L. Fan, *Constrained Coding and Soft Iterative Decoding*. Boston: Kluwer Academic Publishers, 2001.

[13] R. Koetter, A. C. Singer, and M. Tüchler, "Turbo equalization," *IEEE Signal Proc. Magazine*, vol. 21, no. 1, pp. 67–80, Jan. 2004.

[14] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[15] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

[16] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Globecom 1989*, Dallas, TX, Nov. 1989, pp. 1680–1686.

[17] F. R. Kschischang, B. J. Frey, and H. -A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[18] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE ICC'93*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[19] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1977.

[20] D. G. Manolakis, V. K. Ingle, and S. M. Kogon, *Statistical and Adaptive Signal Processing*. New York: McGraw-Hill, 2000.

[21] A. H. J. Immink, W. M. J. Coene, A. M. van der Lee, C. Busch, A. P. Hekstra, J. W. M. Bergmans, J. Riani, S. J. L. V. Benden, and T. Conway, "Signal processing and coding for two-dimensional optical storage," in *Proc. IEEE Globecom 2003*, San Fanscisco, CA, Dec. 2003, pp. 3904–3908.

[22] J. B. Soriaga, P. H. Siegel, J. K. Wolf, and M. Marrow, "On achievable rates of multistage decoding on two-dimensional ISI channels," in *Proc. IEEE Int. Symp. Inform. Theory 2005*, Adelaide, Australia, Sept. 2005, pp. 1348–1352.

[23] M. Marrow and J. K. Wolf, "Detection of 2-dimensional signals in the presence of ISI and noise," in *Proc. Int. Symp. Inform. Theory and Applications*, Parma, Italy, Oct. 2004, pp. 891–894.

[24] M. N. Kaynak, T. M. Duman, and E. M. Kurtas, "Noise predictive belief propagation," *IEEE Trans. Magnetics*, vol. 41, no. 12, pp. 4427–4434, Dec. 2005.