INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1353092

Error-correcting two-dimensional modulation codes

Erxleben, Wayne Henry, M.S.

The University of Arizona, 1993

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

ERROR-CORRECTING TWO-DIMENSIONAL MODULATION CODES

by

Wayne Henry Erxleben

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1 9 9 3

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Wayne Erleben

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Michael W. Marcellin
Michael W. Marcellin
Assistant Professor of
Electrical and Computer Engineering

April 24, 1993
Date

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Michael W. Marcellin for his technical guidance, advice, patience, and financial support, all of which were essential to the completion of my thesis. I also wish to thank my thesis committee members, Dr. Larry C. Schooley and Dr. Ming-Kang Liu, for their helpful suggestions.

Additionally, I am grateful to International Business Machines Corporation for providing the funding that made this research possible. Finally, I want to acknowledge the moral support of my family, friends, and colleagues.

*Dedicated to my mother, whose love and support has helped sustain me
through graduate school, and to the memory of my father, who encouraged
my interest in mathematics at an early age*

TABLE OF CONTENTS

LIST OF FIGURES	7
LIST OF TABLES	8
ABSTRACT	9
1. Introduction	10
1.1. The magnetic recording channel	10
1.2. History of modulation coding	17
1.3. Two-dimensional modulation codes	20
1.4. Error correction	23
1.5. Scope, baseline, and goals	26
1.5.1. Why (1,3)?	26
1.5.2. Baseline performance of existing codes	28
1.5.3. Goals of this research	29
2. Block coding techniques	31
2.1. Fredrickson and Wolf—Multiple block (d,k) codes	32
2.2. French—Distance-preserving mappings	34
2.3. Blaum—Mapping Reed-Solomon encoded symbols to runlength- limited blocks	35
2.4. Ferreira and Lin—Integer compositions	36
2.5. Abdel-Ghaffar and Weber	38
2.6. Ytrehus—The mixed-error channel	38
2.7. Patapoutian and Kumar—Subcodes of linear block codes	40
2.8. Hilden, Howe, and Weldon	41
3. Trellis coding techniques	48
3.1. Computer search	50
3.2. State splitting	52
3.3. Ungerboeck-type trellis coding	62
3.3.1. Background	62

3.3.2.	Application to RLL codes; restriction of the set of terminal states	65
3.3.3.	Ferreira	68
3.3.4.	Multi-track codes found by Ungerboeck's methods	71
3.3.4.1.	$n \times l = 2 \times 3$, rate = $2/6$ codes	72
3.3.4.2.	$n \times l = 2 \times 4$, rate = $3/8$ codes	80
3.3.4.3.	Higher rate 2×1 codes with distance 5	85
3.3.4.4.	$n \times 2$ codes with distance 5	95
3.3.4.5.	Other possibilities	104
4.	Summary, conclusions, and future work	106
REFERENCES		112

LIST OF FIGURES

1.1. State transition diagram for a $(d, k) = (1, 3)$ constrained channel. . .	17
1.2. State transition diagram for a $(d, k) = (1, 3)$ 2-track constrained channel. . .	21
1.3. Magnetic recording viewed as a communication system.	25
1.4. Magnetic recording system with RLL and error correction combined into a single coding step.	27
3.1. Lee and Wolf's $(d, k) = (1, 3)$ trellis code with distance 3.	51
3.2. Lin and Wolf's $(d, k) = (1, 3)$ trellis code with distance 4.	51
3.3. Third power of 2-track $(1, 3)$ FSTD, after elimination of states S_4 and S_5	60
3.4. State diagram for 2-track $(1, 3)$ encoder with distance 3 and rate $2/6$. . .	61
3.5. General finite-state encoder.	64
3.6. Rate $2/6$ trellis that achieves $d_{free} = 3$	75
3.7. Encoder for the trellis of Figure 3.6.	75
3.8. Rate $2/6$ trellis that achieves $d_{free} = 4$	77
3.9. Encoder for the trellis of Figure 3.8.	78
3.10. Rate $2/6$ trellis that achieves $d_{free} = 5$	79
3.11. Encoder for the trellis of Figure 3.10.	80
3.12. Rate $3/8$ encoder that achieves $d_{free} = 3$ or 4 using the trellises of Figure 3.6 or 3.8 respectively.	82
3.13. Encoder for 64-state trellis.	84
3.14. Encoder for 256-state trellis.	93
3.15. Encoder for 1024-state trellis.	98

LIST OF TABLES

1.1. Existing codes	29
3.1. 2×3 words satisfying the (1,3) constraint.	73
3.2. 2×4 words satisfying the (1,3) constraint and ending with zeroes on both tracks.	83
3.3. Fibonacci numbers and powers of two.	86
3.4. 2×5 codewords grouped into pairs with intrapair distance 6.	89
3.5. Eight disjoint foursomes of even-weight 2×6 words with intrafoursome distance ≥ 6	91
3.6. A set of 32 disjoint pairs of 2×6 codewords with intrapair distance ≥ 6	94
3.7. A set of 32 disjoint foursomes of even-weight 2×7 codewords with intrafoursome distance ≥ 6	96
3.8. A set of 32 disjoint foursomes of odd-weight 2×7 codewords with intrafoursome distance ≥ 6	97
3.9. The 3×2 codewords.	100
3.10. The 4×2 codewords.	101
3.11. Parameters of $n \times 2$ codes for $3 \leq n \leq 9$	104
4.1. Summary of (1,3) codes derived in this thesis.	109

ABSTRACT

Modulation coding, to limit the number of consecutive zeroes in a data stream, is essential in digital magnetic recording/playback systems. Additionally, such systems require error correction coding to ensure that the decoded output matches the recorder input, even if noise is present. Typically these two coding steps have been performed independently, although various methods of combining them into one step have recently appeared. Another recent development is two-dimensional modulation codes, which meet runlength constraints using several parallel recording tracks, significantly increasing channel capacity. This thesis combines these two ideas. Previous techniques (both block and trellis structures) for combining error correction and modulation coding are surveyed, with discussion of their applicability in the two-dimensional case. One approach, based on trellis-coded modulation, is explored in detail, and a class of codes developed which exploit the increased capacity to achieve good error-correcting ability at the same rate as common *non*-error-correcting one-dimensional codes.

CHAPTER 1

Introduction

1.1 The magnetic recording channel

Magnetic recording/playback systems, such as tapes and disks together with their respective drives, can be thought of as communication channels. Whereas most communication channels, such as telephone lines, radio broadcasts, and microwave links, transmit information from one *location* to another, virtually instantaneously, recording channels transmit information from one *time* to another (and possibly from one location to another as well, if the medium is transported). The information is stored for an indefinite period of time as a spatially varying level of magnetization of some medium made of a magnetic material. Audio and video cassettes are examples of analog magnetic recording channels, in which the magnetization pattern is both written and read as a continuous waveform. In this thesis, however, we will only be concerned with digital recording channels, such as computer memories, in which the input to and output from the channel are defined in discrete time and take only a

finite number of values. As we shall see, however, the actual magnetization pattern on the medium is still a continuous waveform.

The advantages of digital recording over analog should be obvious and well-known to most readers of this thesis. In systems where the information source is itself digital, there is of course no alternative. But even for analog information such as audio or video, the increased complexity and bandwidth required to convert the analog signal to digital is outweighed by the advantages of doing so, except in low-cost, low-fidelity systems. Namely, since the signal takes on only a finite number of values, it can be perfectly regenerated (recorded and played back) an infinite number of times with no degradation, given proper coding techniques. Furthermore, powerful signal processing can be performed.

In most practical digital recording systems, the “finite number of values” referred to above is two, so the channel is binary. (Nonbinary signalling has been explored recently; see for instance, [1, 2] and especially the partial response techniques discussed in [3, 4].) The binary data are written to the medium by a technique known as saturation recording, in which the current through the write head is switched between two levels which are equal in magnitude but opposite in polarity, and sufficiently large to fully magnetize the medium in one direction or the other. In NRZ (Non-Return to Zero) signalling, the two magnetization polarities directly represent the data bits “0” and “1”. However, most modern recording systems use NRZI (Non-Return to Zero

Inverse) precoding, in which a “1” is represented as a transition in magnetization from one polarity to the other, and a “0” is represented as no such change.

The data are read back from the medium by a read head, which like the write head is an inductive coil. Thus, the voltage induced across the head is proportional to the derivative of the magnetic field. If the system were linear and had infinite bandwidth and no noise or interference, the magnetization pattern on the medium, and thus the magnetic field seen by the head, would be a square wave, and the readback voltage would be a sequence of impulses of alternating signs, at times that were integer multiples of the bit duration T_b . In fact, the magnetic material responds nonlinearly to the field applied by the write head, but most analyses assume the minimum spacing between transitions is large enough that saturation is reached, in which case linear superposition holds reasonably well. This is a valid assumption at the densities achievable with present-day technology.

Due to the finite bandwidth of the channel, the voltage induced across the read head corresponding to an isolated current step through the write head, say from $-A$ to $+A$, is not an impulse but a pulse of the form $2Ag(t)$ where $g(t)$ is the unit step response of the channel, usually modelled as a Lorentzian pulse:

$$g(t) = \frac{K}{1 + (2t/\tau)^2}$$

where τ is the half-amplitude pulse width (in units of time) and K is a proportionality constant [3, 5, 6].

(Note that above, as throughout the thesis, we interchangeably refer to the spatially varying magnetization of the medium and the time-varying field at the heads. The two views are equivalent since the medium is assumed to be moving past the heads at a constant linear velocity v .)

This signal is input to a peak detector, which detects every time the signal's magnitude exceeds some threshold and interprets this as corresponding to a transition on the medium (a "1" at the input to the channel) [7]. This information is also input to a phase locked loop, which adjusts the system clock so the average position of the transitions is the middle of the bit cells (*i.e.*, at the clock ticks).

The position of transitions in the bit cell does vary, due to noise and to interference from adjacent transitions and adjacent tracks of the medium. A technique known as write precompensation is sometimes used to adjust the positions of the transitions *before* they are written to the medium, to compensate for the shifts expected to occur because of interference [7]. As Katz and Campbell [8] showed, reducing the average bitshift improves the performance (*i.e.*, the error rate) of the channel. Another technique known as pulse slimming equalization is sometimes applied between the read head and the peak detector [7] (also sometimes before the write head [9, 10]). As the name implies, it shapes the signal in such a way as to make the pulses narrower, thus allowing them to be closer together without interaction.

Both of the above methods can improve the performance of the peak detection channel at a given storage density, or increase the density while getting the same performance. They impose on the system a certain minimum spacing between transitions, which we will call T_{min} , which must be met to avoid intersymbol interference and achieve the desired performance. A third technique known as modulation coding can further increase the storage density while potentially offering other benefits as well. Modulation coding is the focus of this thesis¹.

If no coding were performed on the information sequence input to the channel, then T_b (the bit duration) could be no smaller than T_{min} since consecutive “1”s in the data sequence would create transitions separated by only T_b . If, however, we constrain the data to always have at least one “0” between consecutive “1”s, then T_b can be half of T_{min} . In general, if the data has at least d “0”s between consecutive “1”s, then we can have $T_{min} = (d + 1)T_b$. That is, the achievable bit rate has been increased by a factor of $d + 1$ over the unconstrained case. Although some rate loss occurs in mapping the unconstrained sequences to constrained sequences, there is a net increase in information density if the rate R of this mapping is greater than $1/(d + 1)$. (By rate we mean the ratio of the number of bits in the unconstrained sequence to the

¹The partial response techniques mentioned on page 11 do not utilize peak detection. In partial response systems, intersymbol interference is not a problem; in fact, it is intentionally introduced in a controlled way. Thus, the modulation coding concepts we are about to discuss are not applicable to such systems. Rather, partial response systems present an entirely different class of coding problems, which are outside of the scope of this thesis.

number of bits in the constrained sequence to which it maps. Naturally, R is always less than 1.)

In addition to the d constraint, there is usually a constraint that the maximum number of consecutive zeroes between ones is some integer k . The purpose of this constraint is to ensure that the phase locked loop gets information often enough to keep the bit clock properly adjusted. The more stable the oscillator in the phase locked loop, the larger k can be.

These two constraints together are, not surprisingly, called (d, k) constraints or runlength constraints, since they restrict the lengths of “runs” of zeroes in the data. (Note that an unconstrained sequence corresponds to $d = 0$ and $k = \infty$.) The mapping which takes unconstrained data sequences to sequences satisfying the (d, k) constraint is called a runlength-limited (RLL) code or more generally a modulation code.

At this point some clarification of terminology and notation is in order, because different authors have used different names and symbols for the same things. Runlength code, runlength-limited code, RLL code, and (d, k) code are all synonymous. Modulation codes are more general, as they may include other types of constraints which we are not considering: charge constraints, separate runlength constraints for the entire sequence and for the subsequences interleaved therein, and constraints that the runlengths take on only certain values between d and k . However, the terms

are often used interchangeably. Additionally, the literature sometimes refers to the codes as recording codes (particularly in magnetics papers, such as [11]), data translation codes (particularly in information theory texts, such as [12]), or constrained codes. In this thesis we shall most often refer to them as RLL codes, even though the multi-track codes on which we will concentrate were introduced as “two-dimensional modulation codes” [13]. Our choice of terminology is meant to reduce confusion between “modulation coding” and “coded modulation” which is described in Section 3.3 and is an entirely different concept.

The use of d and k to denote the constraints is almost universal but not quite. Ferreira [14, 15, 16, 17] refers to them as b and l respectively, because traditionally in the theory of error correction codes (with which we will overlap) d has denoted (Hamming) distance and k has been the number of input bits which map to n coded bits. Also, Blahut [18] uses r and s , and Kolesnik and Krachkovsky [19] use l and m . We retain the use of d and k . In this thesis, distance will always be a d with a subscript (d_{min} or d_{free}). Error correction codes will not be said to map k bits to n bits, but rather they will map p bits to q bits. Meanwhile, n and l will have different meanings in the context of two-dimensional modulation codes, which we introduce in Section 1.3.

Figure 1.1: State transition diagram for a $(d, k) = (1, 3)$ constrained channel.

1.2 History of modulation coding

Before explaining two-dimensional modulation codes, brief mention should be made of the techniques which have been used to develop one-dimensional modulation codes over the years, to provide historical perspective. For a more thorough discussion of most of these techniques the reader is referred to the surveys by Siegel [11], Blahut [18], or Immink [20].

Shannon [21] introduced the concept of a constrained channel, described it by a state transition diagram, showed how to compute its capacity, and proved that codes exist at any rate less than capacity. Figure 1.1 is the state transition diagram for a channel with a $(1, 3)$ constraint. Each state indicates the number of “0”s which have occurred in the coded data since the last “1”. The number of states is therefore equal to $k + 1$.

Shannon did not construct codes but just proved their existence. Freiman and Wyner [22] gave the first real construction method for modulation codes, although

they essentially dealt with only the k constraint ($d = 0$). The method is to form each codeword from a prefix which takes the channel out of one of a set of “terminal states” and a suffix which puts the channel back into a terminal state.

Other 1960’s researchers in RLL codes were Franaszek, and Tang and Bahl. Franaszek [23, 24, 25] extended the idea of a set of terminal states, allowed codewords to vary in length, and devised an algorithm to eliminate potential codewords until the desired code is found. Only a few codes have been constructed by his method, but one of them (for the $(d, k) = (2, 7)$ constraint) has seen very widespread use. Tang and Bahl [26] found an algorithm to enumerate the potential codewords of given length satisfying arbitrary (d, k) constraints, as well as specifying merging bits for concatenability, and thus defined an encoder and decoder mapping. (Their method was later improved by Beenker and Immink [27] by further restricting the codewords so fewer merging bits were needed.) Meanwhile, two simple codes known as the Manchester code and the Miller code were designed using no theory, and implemented in many systems [11].

In the 1970’s and early 1980’s, Jacoby, Lempel, Cohn, Patel, and others [28, 29, 30, 31] invented the class of look-ahead codes, which do a basic block encoding, then substitute for certain bit patterns if the upcoming codewords would create constraint violations. This work has yielded a number of practical codes, but perhaps more importantly, introduced in the code construction the ideas of an approximate eigen-vector inequality and state-splitting. These ideas became crucial to the landmark

1982 paper by Adler, Coppersmith, and Hassner [32], which for the first time gave a mathematically rigorous formulation of the problem and a general algorithm to design modulation codes at any rate below capacity, with finite error propagation. This algorithm will be described in detail in Section 3.2. Related techniques and applications can be found in [6, 33, 34, 35].

A few new approaches to RLL coding have been introduced since 1982. French, Dixon, and Wolf [1] generalized to channels with more than two possible symbols, in which case d and k become matrices, expressing different runlengths allowed between different symbols. Fredrickson and Wolf [36] suggested cycling among several block codes, with different codeword lengths in the different blocks, to increase the overall rate. We will have more to say about this in Section 2.1. Immink and Hollmann [37] and Helberg and Ferreira [38] have introduced codes that use some of the “excess” capacity of the (d, k) constraints (above the code rate) to allow synchronization patterns or an ancillary low-rate channel, respectively. Perhaps one of the most interesting new developments has been the introduction of two-dimensional modulation codes by Marcellin and Weber [13]. It is on these codes that the rest of this thesis will focus.

1.3 Two-dimensional modulation codes

Most magnetic tape used for digital data storage has several tracks which are written in parallel by multiple write heads, and read in parallel by several read heads. Existing systems treat these tracks independently. The data sequence written to each track is encoded to meet the (d, k) constraint. Recall though that the purpose of the k constraint is to ensure frequent enough transitions that the phase locked loop can keep the clock properly adjusted. Since the tracks are all read at the same time, suppose a common clock is used for several, say n , tracks. Then we only need a transition on *one* of those n tracks every k bits, not necessarily on every track. In other words, any individual track is allowed to have runs of more than k consecutive zeroes, as long as the n tracks, viewed together, do not have a run of more than k consecutive all-zero n -tuples. The d constraint must still be satisfied on each track. Clearly the system's old (d, k) constraint has been transformed to a somewhat looser, n -track (d, k) constraint without sacrificing performance.

The channel is now described by a different state transition diagram than in the one-track case. Figure 1.2 is the state diagram for a channel with a 2-track $(1, 3)$ constraint. Note that the channel now has 6 states, as opposed to the 3 that it had in Figure 1.1. In general the number of states is $(d + 1)^n + k - d$, as shown in [13], when $k > d$. For $n = 1$ this of course reduces to $k + 1$ as one expects for a single-track code.

Figure 1.2: State transition diagram for a $(d, k) = (1, 3)$ 2-track constrained channel.

In [13], Marcellin and Weber not only introduced the concept of multi-track constraints but also calculated and tabulated the associated channel capacities. Channel capacity is still computed as it has been since Shannon, namely the logarithm of the largest eigenvalue of the state transition matrix [21]. However, the number of channel states becomes so large so quickly that finding the largest eigenvalue is a formidable task. Marcellin and Weber thus determined the structure of the state transition matrix and exploited the recursive nature of this structure to develop fast algorithms to compute the capacity. An important result of these computations was that substantial increases in capacity are possible by using multi-track constraints instead of single-track, particularly when k is relatively small. In fact, some constraints can

be met by using two-dimensional codes which were impossible on one track, or had zero capacity, namely those with $k \leq d$. They also found that most of the increase in capacity occurs from just increasing n from 1 to 2. This fact should be borne in mind in future chapters.

In the context of two-dimensional codes, we will use the symbol l to denote the codeword *length* in n -track symbols. The codeword *size* will be given as $n \times l$. The rate of a two-dimensional code will always be given in bits/bit, as p/q where $q = nl$. Generally we will not reduce the fraction p/q , to emphasize the actual number of bits and not just the ratio.

Marcellin and Weber did not develop any technique for generating codes in [13], and it appears that a number of existing techniques can be applied to these new constraints. This in fact has been done in the dissertation by Orcutt [35] and the papers taken from it [39, 40]. We mention here the code published in [40] as an example illustrating the rate increase possible by encoding two tracks together instead of independently.

The code in [40] satisfies the (1,3) constraint, for which most existing systems use the Miller code. The capacity associated with this constraint on one track is $C = 0.552$, and the Miller code operates at rate $R = 1/2$. By using $n = 2$ tracks, the capacity goes up to $C = 0.680$, and the code in [40] achieves $R = 4/6$, a 33% rate increase.

1.4 Error correction

Despite the use of write precompensation, pulse slimming equalization, and RLL coding, errors still occur as a result of noise, timing jitter, and media defects. There are several different classes of errors that occur in a recording channel. *Random additive errors* occur due to thermal noise and electronics noise. As the name suggests, randomly distributed bits are changed from “0” to “1” or vice versa, *i.e.*, a “1” is added to the correct bit, modulo 2. These, of course, are the kind of errors that error correction codes have traditionally dealt with, and that are usually assumed in theoretical analyses of communication systems. *Burst errors* are also additive, but the incorrect bits occur in bursts rather than in a uniform distribution. Burst errors usually occur due to media defects and are generally corrected with a separate layer of coding working outside of the random error correction.

Peak shift errors are more characteristic of magnetic recording channels than other channels. Although Seymour [41] states that dropouts—additive errors that change a “1” to a “0”—are the “primary cause of data errors in high-density magnetic recording,” other analyses [42] suggest that peak shifts are the most prevalent class of errors for recording channels. A peak shift happens when a “1” in the data, corresponding to a transition in the magnetization pattern and a peak at the read head, is detected in a neighboring bit cell instead of the proper one. Note that a peak shift is equivalent to two additive errors.

Bit slip errors or synchronization slippages occur when the clock recovery circuit has added or deleted a bit cell, so the bit stream is offset from its proper position by one bit. An uncorrected bit slip could be equivalent to an infinite number of additive errors, since its effects would propagate until there was another slip in the opposite direction.

To combat these errors, error correction coding is typically done on the information sequence before it enters the RLL encoder. That is, the information sequence is mapped to a data sequence with redundancy built into it, so if (a few) errors occur in the channel, the resulting sequence out of the RLL decoder is still “closer” to the correct sequence (in some distance metric, typically Hamming distance) than to any other valid sequence. Thus it can be mapped back to the correct sequence, correcting the channel errors. The entire system is shown in Figure 1.3.

Traditionally the processes of error correction and RLL coding have been done separately, with the RLL encoder, constrained channel, and RLL decoder together considered as an unconstrained (but noisy) channel, as indicated in Figure 1.3. Shannon [21] proved that this separation of functions does not prevent one from attaining arbitrarily low probability of error, while transmitting information at any rate below channel capacity. His proof, however, said nothing about the complexity of the codes required to achieve such performance. Indeed, in many communication systems, combining the error correction and modulation coding into a single block, as

Figure 1.3: Magnetic recording viewed as a communication system.

shown in Figure 1.4, allows comparable performance with simpler implementation than is possible when the functions are separated.

While error correction codes have been the subject of countless papers over the last 40 years, and modulation codes have also been studied in a significant (albeit *much* smaller) body of research, the combination of these mappings into a single encoding step has only in recent years been explored. In this thesis we will not deal with burst errors; however, for the other types of errors we will combine error correction with RLL coding in ways that take advantage of the increased capacity available by working with multi-track rather than single-track constraints.

1.5 Scope, baseline, and goals

1.5.1 Why (1,3)?

For the remainder of this thesis we concentrate on the $(d, k) = (1, 3)$ constraint, for a number of reasons. First, it is a constraint that has been utilized in many commercial systems, usually implemented with the Miller code [11]. Second, the number of states and thus the complexity of the codes grows roughly as the n^{th} power of d (where n is the number of tracks), and mildly with k [13], so our procedures can only be shown in explicit detail for relatively small d and k . It is hoped that these methods become the basis for automated procedures that could be used to devise codes for other sets of constraints. Finally, with the (1,3) constraint, a larger

Figure 1.4: Magnetic recording system with RLL and error correction combined into a single coding step.

increase in capacity is available by changing from single-track to multi-track codes than with some other common constraints. The capacity of a 1-track $(1,3)$ code is 0.552; this increases to 0.680 by using 2 tracks, and to 0.694 by using 4 or more. By contrast, the capacity of $(2,7)$ codes increases only from 0.517 for single-track codes to 0.551 when 3 or more tracks are encoded jointly [13].

1.5.2 Baseline performance of existing codes

Table 1.1 lists the specifications of existing codes, against which any new codes should be judged. The Miller code has no error-correcting ability, but is the standard code used for a $(1,3)$ constraint. It would be nice to get the same rate $(1/2)$ in an error-correcting $(1,3)$ code. The Ferreira, Hope, and Nel code is not a $(1,3)$ code, but it has rate and distance properties that we would like in a $(1,3)$ code; in other words we wish to reduce k *vis a vis* this code, whereas *vis a vis* the others we wish to increase rate or distance. We will discuss the derivation of Ferreira's code in Section 3.3.3. The Lin and Wolf code, which will be described in Section 3.1, has the largest distance, large enough to not only correct single errors but to also detect double errors. To our knowledge no $(1,3)$ codes with distance 5 and rate at least $1/3$ have yet been published.

CODE	(d, k)	RATE	DISTANCE
Miller	(1,3)	1/2	1
Ferreira Hope & Nel	(1,7)	4/8	3
Lin & Wolf	(1,3)	1/3	4

Table 1.1: Existing codes

1.5.3 Goals of this research

We wish to improve upon the codes listed in Table 1.1 by taking advantage of the capacity increase associated with meeting multi-track rather than single-track constraints. There are three directions we can go to make specific improvements. One is to try to get distance 5 in a rate $1/3$ (or $p/3p$) code, *i.e.*, better distance than Lin and Wolf at the same rate. Upon achieving this, which we do in Section 3.3.4.1, a natural thing to do is to try to increase the rate while keeping distance 5. This is done in Sections 3.3.4.2 and 3.3.4.3. A second direction is to see whether we can do error correction (*i.e.*, get distance ≥ 3) at the same rate as the Miller code, and if so, how much distance can we get? Clearly both of these directions close in on the same goal: a distance 5, rate $1/2$ code. We shall see that such a code is attainable (Section 3.3.4.3), with the tradeoff being complexity.

The third direction in which to make improvement is in codes designed to correct peak shift errors. Note that a distance 5 code can correct two additive errors. Consider the set of all error patterns consisting of two additive errors. A subset of this is the set of all error patterns consisting of a single peak shift. Thus a distance 5 code can correct a single peak shift. But a code optimized to correct peak shifts could do

so more powerfully and more efficiently than one built on Hamming distance. Since there is evidence that peak shifts are the dominant class of errors on some channels [42], we make some investigation of such codes, in Sections 2.8 and 3.3.4.4.

Additionally, we survey methods by which previous researchers have combined error correction and runlength coding in the single-track case, and comment on the applicability of these methods to multi-track constraints. As most of these methods are block coding methods, and our most successful original work has been in trellis coding, the reader will find that survey material comprises the bulk of Chapter 2, while Chapter 3 primarily reports on our own work.

CHAPTER 2

Block coding techniques

Codes that separate the data stream into non-interacting blocks, and encode and decode each block independently of all others, are called *block codes*. The alternative is to have the encoding and decoding depend on past and/or future data; in other words the channel has *memory* and the coding is dependent on the *state* of the channel. This is called *trellis coding* and will be described in the next chapter.

If a block code maps p bits to q bits, then an obvious way to decode is to look up, for each possible q -tuple, the corresponding “most likely” p -tuple from a table in memory. Of course, if such a decoding scheme is used, the size of the lookup table increases exponentially with q , and becomes prohibitively large if q is big enough to make a very powerful code. For this reason most block codes for error correction are linear and have a rich enough algebraic structure that other decoding methods can be used.

Runlength codes, however, are inherently nonlinear, *i.e.*, the sum of two codewords is not necessarily another codeword. Thus table lookup has been the primary decoding method. To get codewords that not only satisfy the constraints but also

build distance requires relatively large codeword sizes, and thus very large lookup tables.

Tang and Bahl’s enumeration algorithm [26], later refined by Beenker and Immink [27], allowed block RLL codes to be encoded and decoded with only a linear relation between codeword size and complexity. This scheme relied on a one-to-one mapping between codewords and contiguous integers. Recently a similar enumeration scheme has been derived for (non-error-correcting) two-dimensional RLL codewords [39]. But since we are trying to use codewords with a certain amount of distance between them, there would now be “holes” in the numbering, and so such an enumeration scheme would not work, or at least would not give us the benefit that it does in the non-error-correcting case.

In this chapter we present a survey of techniques which have been used to combine error correction and runlength coding in the one-dimensional case, using a block structure. Several of these techniques appear to be applicable to the two-dimensional case. In the final section of the chapter, we work through a detailed example applying the technique of Hilden, Howe, and Weldon [43] to the 2-track (1,3) constraint.

2.1 Fredrickson and Wolf—Multiple block (d,k) codes

Fredrickson and Wolf recently introduced *multiple block (d, k) codes* [36], in which an ensemble of block codes is used, with different codeword lengths in the different

codes. The encoder cycles through the various codes, so the overall result is a time-varying block code. The reason for doing this is that in some cases a higher average rate can be attained, since some of the codes can have shorter codeword lengths than is possible with a single block, since they do not have to be self-concatenable.

Shortly after introducing multiple block (d, k) codes, Fredrickson and Wolf published another paper [44] which uses the technique to get (d, k) codes that can also detect errors in the same coding step. Note that errors are only detected, not corrected, since the minimum distance between words is 2, not 3. Moreover, the errors with which the paper deals are not additive errors, but peak shifts and bit slips.

In [36], Fredrickson and Wolf gave recursive algorithms for enumerating the set of codewords that have a certain choice of the number of beginning and ending zeroes, and the *cost* in channel bits due to that choice. In [44] they refine these algorithms to enumerate the subsets of those sets that have intrasubset distance at least 2, in either of the two distance metrics they define. Geometrically this means subsets that do not contain any edges connecting them in whatever space they occupy. These subsets become the various block codes through which the channel cycles. The order of this cycling is then determined by applying Karp's algorithm [45] for the minimum cycle mean of a digraph. The actual words within a block are then encoded by application of Beenker and Immink's enumerative coding scheme [27].

This sort of concept may be applicable to two-dimensional constraints; however, the signal space would have a very different structure, and an entirely different set of algorithms would be necessary. Since the resulting codes would only be effective against peak shifts and bit slips, not against additive errors, and even then they could only *detect* errors, not *correct* them, it was decided to not pursue this approach.

2.2 French—Distance-preserving mappings

The technique suggested by French in [46] is as much applicable to trellis codes as to block codes. In trellis codes, however, the same concept is discussed more fully in Ferreira [15, 16, 17], and so we save that discussion until Section 3.3.3. Here we mention only the block code examples of French’s technique.

The idea is simply to map the codewords of a standard error-correcting code to (d, k) constrained codewords in such a way that the distance between any pair of words in the original code is at least preserved, and possibly increased, between the images of those words under the mapping. A simple, but not very efficient, way of doing this is to leave the bits of the original codewords intact and insert appropriate “0”s and “1”s between them in such a way as to meet the constraints. Larger and more efficient examples are also given in [46] which were obtained through Franaszek’s sequence state method [23, 25] and a computer search.

2.3 Blaum—Mapping Reed-Solomon encoded symbols to runlength-limited blocks

In [47], Blaum notes that in traditional concatenated coding schemes for magnetic recording channels, the outer (error-correcting) code is most often a Reed-Solomon code. A Reed-Solomon code is not a binary code but rather treats a ν -tuple of data bits as a single symbol, an element of the finite field $\text{GF}(2^\nu)$, and performs all encoding and decoding operations over that field. By interleaving these symbols, Reed-Solomon codes can correct burst errors. Although magnetic recording channels do tend to have burst errors due to physical media defects, another important reason that outer error correction codes have to deal with bursts is the error propagation of the inner modulation codes. This has also been noted by several other authors [42, 43, 48, 49] as one of the reasons why combining the two layers of coding into one is more efficient than concatenating them.

From this background, Blaum proposes a scheme which uses a 2^ν -ary Reed-Solomon code on the data, then maps the resulting ν -tuples to (d, k) -constrained codewords. The set of possible codewords is found and enumerated by one of Beenker and Immink's constructions [27]. A subset of this set is selected with cardinality as large a power of 2 as possible, and the words in this subset are put into one-to-one correspondence with the elements of $\text{GF}(2^\nu)$, *i.e.* with the encoded ν -tuples. This

correspondence defines the encoder and decoder mappings. (As in [27], merging bits are required between words to ensure concatenability.)

Blaum's scheme is really not much different than concatenation. The biggest difference is that the inner encoder does not act ignorantly on the bit stream entering it; rather, it knows the data are actually symbols from $\text{GF}(2^\nu)$, as encoded by the outer encoder, and acts directly on them. Likewise, if a word entering the decoder does not satisfy the (d, k) constraints, the inner decoder can just declare an erasure of the whole word, which the outer Reed-Solomon decoder can easily correct. Furthermore, the Reed-Solomon code can be interleaved just as in a traditional concatenated scheme, in order to correct burst errors.

It appears that Blaum's method would be fully applicable for two-dimensional codes. The Reed-Solomon code's output symbols can just as easily be put into one-to-one correspondence with multi-track codewords as with single-track codewords. This has not been the main focus of our research, however.

2.4 Ferreira and Lin—Integer compositions

Ferreira and Lin, in [50], developed an entirely different approach to the problem of putting error-correcting ability into a set of (d, k) codewords. Their idea was to think of each constrained n -tuple not just as a string of bits but as a string of integers representing the runlengths of zeroes contained in the word. For instance, a

“1” followed by three “0”s is represented by the integer 4. If only n -tuples beginning with a “1” are considered, it is clear that each such n -tuple can be represented by an ordered set of integers, known as a *composition block*, which uniquely determines the locations of all the “1”s.

Ferreira and Lin then show how to construct sets of constrained codewords with certain distance properties by following certain rules concerning the associated composition blocks. The specific rules vary depending on whether it is desired that the code correct single random errors, peak shifts, double adjacent errors, or multiple adjacent erasures. The simplest case is single random errors, and two different families of codewords are given, based on two different composition rules. The concept that both have in common is that the set X of integers used in forming composition blocks is such that if $a, b \in X$, then $(a + b) \notin X$. This guarantees that if a “1” is changed to a “0” or vice versa, the resulting composition block will contain an integer not allowed to be there.

Encoding and decoding procedures are given in [50] as well as the constructions themselves. A problem with these codes is that the blocks have to be quite long in order to get good rates. As the block length gets larger and larger, of course, the ability to correct only, say, a single error in that block becomes less and less satisfactory.

It does not appear that these codes can be applied to the two-dimensional case. The composition block concept does not include any way to describe the multi-track k constraint.

2.5 Abdel-Ghaffar and Weber

Abdel-Ghaffar and Weber [48] present constructions for codes to detect single random additive errors and single peak shifts. However, they can only detect them, not correct them. Their methods, like Ferreira and Lin's [50], represents codewords not by bits but by integer compositions, or as they say, "one-locators." The constructions involve finding sets of codewords whose one-locators satisfy certain modulo arithmetic equations. Their peak shift-detecting codes are said to be a generalization of Fredrickson and Wolf's codes [44]. Their additive error-detecting codes are said to be a generalization of Lee and Wolf's construction [51]. We will not be too interested in Abdel-Ghaffar and Weber's constructions, as we will be able to do much better in both rate and error-correcting power.

2.6 Ytrehus—The mixed-error channel

To this point, the codes we have discussed are optimized to work against a certain class of errors—additive, peak shift, or whatever. We know, though, from experimental data [41, 42] that real magnetic channels contain multiple classes of errors.

Ytrehus [52] recognized this and devised a coding scheme for use on “mixed-error channels” (MC’s), capable of simultaneously correcting additive and peak shift errors. Of course, the resulting codes can also handle single-error-class channels as special cases.

In addition to classifying binary words on the usual basis of Hamming weight and Hamming parity (Hamming weight modulo 2), Ytrehus also classifies them by their “shift weight” and “shift parity” (shift weight modulo 2). The shift weight is found by simply counting the “1”s weighted by their positions in the word (1, 2, 3, ..., *wordlength*). In other words, it is nothing more than the sum of the one-locators of the word, as discussed in Section 2.5.

Ytrehus’ codes are a form of “generalized concatenated codes” [53], using two outer codes $\mathcal{O}_1$ and $\mathcal{O}_2$, and four inner codes $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$, and $\mathcal{I}_4$. Each of the inner codes contains a set of (d, k) constrained words with a different combination of even/odd Hamming parity and shift parity.

The input word is split into two subwords, the first one encoded by $\mathcal{O}_1$ and the second one by $\mathcal{O}_2$. The first subword must have even length, as $\mathcal{O}_1$ is a quaternary code—in Ytrehus’ example, a rate 3/5 systematic quaternary Hamming code. The second outer code $\mathcal{O}_2$ can be a very simple code with $d_{min} = t + 1$ where t is the number of MC errors which the overall code is meant to correct. In Ytrehus’ example,

$\mathcal{O}_2$ is a rate $4/5$ systematic 2^5 -ary parity-check code. Whatever codes are chosen, it is important that both have the same output wordlength (in symbols, not bits), N .

Thus there are now N pairs, each consisting of an $\mathcal{O}_1$ quaternary symbol and an $\mathcal{O}_2$ word. For each pair, the $\mathcal{O}_1$ symbol chooses one of the four inner codes, and the $\mathcal{O}_2$ word is input to the selected inner code. If either an additive error or a shift error occurs in the channel, the inner decoder detects it, by virtue of one of the parities being wrong, and this is manifested as an erased symbol, which the outer decoder corrects.

Ytrehus' approach could be applied to two-dimensional codewords meeting n -track constraints. However, the number of inner codes required would no longer be four, but rather 4^n , since the Hamming and shift parities of each track are independent. This then becomes an extremely complicated set-partitioning exercise with rather long block lengths. We have not pursued this because a similar, but simpler, set-partitioning [51] is done in our trellis-coding techniques (see Section 3.3.4), with resulting codes that are both shorter and more powerful than Ytrehus'.

2.7 Patapoutian and Kumar—Subcodes of linear block codes

Patapoutian and Kumar [54] recently introduced a very straightforward construction of error-correcting (d, k) block codes. They start with *any* error-correcting linear block code, then find the coset of that code that contains the most words meeting

the (d, k) constraints. Some subset of those constrained words can then be used for the error-correcting runlength code.

An algorithm for systematic encoding is given in [54]. The beauty of this approach, however, is that a standard decoder for the base linear code can be used as the decoder for the (d, k) code (after translating codewords by the coset leader, of course). Also, if instead of starting with, say, a Hamming code, one starts with the even-weight subcode of the Hamming code, then one can correct peak shifts as well as additive errors.

Patapoutian and Kumar's approach may be useful for two-dimensional codes, although we have not pursued it. One could start with a one-dimensional error-correcting linear block code, and find the coset of it that contained the most words meeting the d constraint. Words of this coset code could then be stacked on top of each other to construct two-dimensional words. Finally, those multi-track words that violated the k constraint could be purged, leaving an error-correcting two-dimensional (d, k) code. The encoder would have to insert merging bits, as in Tang and Bahl [26], to ensure that the constraints were not violated when words were concatenated.

2.8 Hilden, Howe, and Weldon

Another very interesting recent approach to combining error correction and RLL coding is due to Hilden, Howe, and Weldon [43]. Their approach is not so much a new

class of codes as it is a new method of combining existing codes. Unlike many of the other techniques discussed, it does not combine the functions into a single encoding step. There are still two layers of coding but, as in [52], they are put together in a more clever way than mere concatenation. Virtually any RLL code (either block or trellis) can be utilized as the first step. The output of the RLL encoder is broken into blocks, and the second layer of coding utilizes a systematic ternary error correction block code such as a ternary BCH code.

Hilden, Howe, and Weldon's paper deals exclusively with peak shift errors. In [55], Hilden and Weldon modify these codes to correct isolated additive errors as well as peak shifts. The modified codes, with which we will not deal herein, require a much more complicated error-correcting code in the second layer, and only work if the additive errors are separated by at least $k + 1$ bit positions.

The encoding technique is as follows:

1. Use any RLL code to map the data to constrained sequences, and partition the constrained data into blocks.
2. Treat the *runlengths*, instead of the bits, as information symbols, from the set of integers between $d + 1$ and $k + 1$.
3. Map the block of runlengths to a new block whose symbols are the sums of the runlengths accumulated from the beginning of the block up to that point.

4. Reduce these integers modulo 3. In general this would be modulo $2t + 1$ if the code is to correct errors in which a peak is shifted by t bit cells, but we will only consider the case $t = 1$, as the probability of a peak shifting more than one bit cell is negligible.
5. Use a systematic ternary error-correcting block code on these ternary symbols, generating a string of parity-check symbols to be appended to the information block.
6. Map the ternary digits back to strings of “0”s separated by “1”s.

This technique appears to be fully applicable for multi-track constraints, as a two-dimensional code can be used for the RLL coding. Beyond that step though, the tracks must be treated independently, so the advantage of increased capacity from multi-track constraints is not fully realized. In some cases however, there may still be a significant advantage to working on multiple tracks.

We now give an example to illustrate the application of this technique to two-dimensional codes. Unlike the codes presented in some sections of this thesis, this example is not the result of a major effort to find the best code within its class, as this technique has not been a major focus of our research. Rather, we have selected an example which easily lends itself to comparison with other codes. The constraints with which we deal, here as throughout the thesis, are $(d, k) = (1, 3)$. The RLL code is the rate $4/6$, 2-track code described in [40] because it takes advantage of

the increased capacity to a greater extent than any other two-dimensional code yet found. This is in fact a trellis code, but the resulting sequences are then broken into blocks of size $(n \times l) = (2 \times 36)$ before the second step of encoding is performed. This step utilizes the same ternary BCH code as in [43]. We will go through all the steps of the procedure and at the same time apply them to a sample data stream.

Suppose our information sequence contains the following subsequence:

1011 1110 0100 1111 0100 0000 1011 1011 1100 1100 0100 1010

The 2-track (1,3) coder would then produce the following sequence of codewords (assuming that the initial encoder state is S_0 (see [40])):

001 000 010 101 001 000 001 010 100 000 100 001
000 101 010 100 100 010 000 100 001 001 001 000

We need to add some dummy bits at the beginning to ensure that both tracks start with a “1”. We also add some dummy bits at the end to ensure that the parity runs which will be appended can follow without violating constraints. It appears we need a 2×3 block at the beginning and a 2×2 block at the end, if the number of bits is to be data-independent. The bits are chosen as follows:

If both tracks begin with (00), append to beginning: $\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$

Otherwise, append to beginning: $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$

If both tracks end with (00), append to end: $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

Otherwise, append to end: $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

Our sequence is now:

$$\begin{array}{c|cccccccccccccccc} 101 & 001 & 000 & 010 & 101 & 001 & 000 & 001 & 010 & 100 & 000 & 100 & 001 & & 00 \\ 100 & 000 & 101 & 010 & 100 & 100 & 010 & 000 & 100 & 001 & 001 & 001 & 000 & & 00 \end{array}$$

In runlength space, this is:

$$R_d = \begin{pmatrix} 2 & 3 & 5 & 2 & 2 & 3 & 6 & 2 & 2 & 6 & 5 & 3 \\ 6 & 2 & 2 & 2 & 3 & 4 & 5 & 5 & 3 & 3 & 6 & \end{pmatrix}$$

Note that the two tracks need not have the same number of runs.

In sum space,

$$S_d = \begin{pmatrix} 2 & 5 & 10 & 12 & 14 & 17 & 23 & 25 & 27 & 33 & 38 & 41 \\ 6 & 8 & 10 & 12 & 15 & 19 & 24 & 29 & 32 & 35 & 41 & \end{pmatrix}$$

It can be shown that any set of 41-tuples which may result on a track when the dummy bits are appended can contain no more than 19 runs, even though $41 > 19 \times 2$.

Reduced modulo 3, we get:

$$T_d = \begin{pmatrix} -1 & -1 & 1 & 0 & -1 & -1 & -1 & 1 & 0 & 0 & -1 & -1 \\ 0 & -1 & 1 & 0 & 0 & 1 & 0 & -1 & -1 & -1 & -1 & \end{pmatrix}$$

Representing the two sequences as polynomials, we get:

$$T_{d,1}(x) = -x^{11} - x^{10} + x^9 - x^7 - x^6 - x^5 + x^4 - x - 1$$

$$T_{d,2}(x) = -x^9 + x^8 + x^5 - x^3 - x^2 - x - 1$$

The generator polynomial for the BCH code is:

$$g(x) = x^7 - x^5 - x^4 + x^3 - x^2 + 1$$

Multiplying both information polynomials by x^7 and dividing by $g(x)$ gives:

$$r_1(x) = x^4 + x^3 - x^2 - x + 1$$

$$r_2(x) = -x^5 - x^4 - x^3 - x^2 + x$$

and the codeword polynomials are:

$$\begin{aligned} T_1(x) &= x^7 T_{d,1}(x) - r_1(x) \\ &= -x^{18} - x^{17} + x^{16} - x^{14} - x^{13} - x^{12} + x^{11} - x^8 - x^7 - x^4 - x^3 + x^2 + x - 1 \end{aligned}$$

$$\begin{aligned} T_2(x) &= x^7 T_{d,2}(x) - r_2(x) \\ &= -x^{16} + x^{15} + x^{12} - x^{10} - x^9 - x^8 - x^7 + x^5 + x^4 + x^3 + x^2 - x \end{aligned}$$

The reader may verify that $g(x)$ divides both of these.

The parity parts of these polynomials, namely $-r_1(x)$ and $-r_2(x)$, are written in vector form as:

$$T_{p,1} = (0 \ 0 \ -1 \ -1 \ 1 \ 1 \ -1)$$

$$T_{p,2} = (0 \ 1 \ 1 \ 1 \ 1 \ -1 \ 0)$$

These map back to sum space as:

$$S_{p,1} = (3 \ 6 \ 8 \ 11 \ 13 \ 16 \ 20)$$

$$S_{p,2} = (3 \ 7 \ 10 \ 13 \ 16 \ 20 \ 24)$$

and to runlengths as:

$$R_{p,1} = (3 \ 3 \ 2 \ 3 \ 2 \ 3 \ 4)$$

$$R_{p,2} = (3 \ 4 \ 3 \ 3 \ 3 \ 4 \ 4)$$

The parity segments thus determined could be as long as 28 bits or as short as 14 bits, depending on the data. Pad them with predetermined dummy sequences to make both tracks 30 bits long:

```
10010010100101001000|1001001000
10010001001001001000|100100
```

So what is finally written to the tape is:

```
10100100001010100100000101010000010000100|100100101001010010001001001000
10000010101010010001000010000100100100000|100100010010010010001000100100
```

The overall rate is $48/142 = 0.338$, and two peak shifts on each track can be corrected. By comparison, if all the same encoding steps had been taken using a rate $1/2$ single-track $(1,3)$ code on both tracks independently, instead of the rate $2/3$ two-track $(1,3)$ code, the overall rate would have been only $36/142 = 0.254$. Thus going to two tracks offers a 33% increase in rate¹.

¹This was computed by assuming the same ternary BCH code was used, thus mandating that the output sequences of the $(1,3)$ coders were still broken into 41-tuples. When the dummy bits are stripped off, that leaves 36 channel bits, which correspond to 18 information bits, on each of the two tracks.

CHAPTER 3

Trellis coding techniques

We now turn to *trellis codes*, which, as opposed to block codes, do not break the data stream into non-interacting blocks. Any data bit entering the encoder may affect not only the codeword being generated at that time, but several future codewords as well.¹ At the decoder, the output word depends on past and/or future codewords as well as the present one. In all useful codes, the number of codewords affected by any one bit is finite, to avoid *catastrophic error propagation*, in which a single channel error can cause infinitely many errors at the decoder output.

When a trellis code is linear, as is usually the case when the code's only function is error correction, it is referred to as a *convolutional code*. As pointed out, though, in the previous chapter, runlength codes are inherently nonlinear. In either case, trellis coding has some key advantages over block coding. One is that a higher rate can often be obtained, as well as a simpler encoder and decoder implementation for

¹It is actually a misnomer to refer to "codewords" in the context of trellis codes, as the sequences are not broken into non-interacting "words" as they are in block codes. Our $(n \times l)$ blocks should rightly be called "symbols" (in $GF(2)^q$), and we do sometimes call them that. But we also continue to call them "codewords" to avoid ambiguity as to whether "symbols" means bits, n -tuples, or $n \times l$ -blocks.

comparable performance. Another advantage is additional coding gain through the use of soft-decision decoding.

The usual way to describe the dependence of the encoder mapping on past data is to say the encoder has *memory* and to speak of the contents of that memory as the encoder *state*. The operation of the encoder is then depicted on a state transition diagram, which becomes a *trellis* if the time dependence is explicitly shown. Recall that the channel itself is also described by a state transition diagram, which shows how symbols are allowed to follow each other, or *concatenate*.² The trick then is to get from the state description of the channel to that of the encoder. The encoder state transitions must correspond to allowable channel state transitions.

This concatenability is a central issue in the design of constrained codes. That is, not only must codewords satisfy the runlength constraints internally, but any concatenation of codewords that the encoder allows must also result in sequences that meet the constraints. For example, the words 0100 and 0010 both satisfy the single-track (1,3) constraint, but when they are concatenated, the resulting sequence (01000010) violates $k = 3$. Thus the set of codewords and/or the encoder state transition diagram must be carefully designed to avoid any constraint violations when words are concatenated.

²Note that we earlier used the the word *concatenate* to describe the cascaded connection of inner and outer encoders (or decoders). In the present context, however, we just mean the juxtaposition of two codewords following each other in the data stream.

Since our goal is to correct errors as well as meeting channel constraints, we also require that the code build a certain amount of distance. This distance now is measured not between individual codewords, but rather between all semi-infinite sequences of codewords that begin at a given state. It is usually denoted by d_{free} .

We will consider three ways to derive an encoder: The first is through state splitting, which in a mathematically rigorous way makes a smooth transition from channel states to encoder states. The second approach, Ungerboeck-type trellis coding, restricts the set of states in which the channel can be left at the end of a codeword, thus allowing the encoder states to be assigned without regard to channel state, and the codewords to concatenate freely. The third approach, which we will actually discuss first, in Section 3.1, is to assume a particular form for the encoder and do a computer search to find a satisfactory assignment of codewords.

3.1 Computer search

Lee and Wolf in [56] and Lin and Wolf in [49] published trellis codes which achieve a better tradeoff between rate and distance than any previously published. The codes have no obvious algebraic structure. Some of them are rather asymmetric, and one even has the number of states not a power of 2. Lee and Wolf's $(d, k) = (1, 3)$ code is shown in Figure 3.1, and Lin and Wolf's in Figure 3.2. Lin and Wolf's code, by merit of its achieving distance 4 at rate $1/3$, is one of the codes we listed in Table

Figure 3.1: Lee and Wolf's $(d, k) = (1, 3)$ trellis code with distance 3. The labels on the edges indicate "input bit"/"codeword."

Figure 3.2: Lin and Wolf's $(d, k) = (1, 3)$ trellis code with distance 4. The labels on the edges indicate "input bit"/"codeword."

1.1 as one of the "baseline performance" codes to which any new codes should be compared.

No details of the computer search are presented in [56] or [49] so one supposes that some general class of trellises was assumed (more general than those associated with linear convolutional codes), and then an exhaustive search was performed to find, by trial and error, a code of that form with the desired rate and distance properties.

We attempted to find a rate $2/4$, distance-3, 2×2 , two-dimensional $(1,3)$ code by searching through ways of placing 2×2 words on one of Ungerboeck's 16-state trellises [57]. For reasons that later became clear (see Section 3.3.4), this search was

unfruitful and was stopped after 15,000 CPU-minutes. This, of course, assumed one specific trellis structure. The computational load that a truly exhaustive search would entail is mind-boggling. If the criteria which Wolf and his colleagues used to limit their searches were known, perhaps a similar search could yield good two-dimensional codes. In the meantime, though, we have been able to utilize computer searches for certain aspects of our trellis code designs, which will be described in Section 3.3.

Such computer searching, of course, constitutes a very *ad hoc* approach, whereas the techniques which we will describe in Sections 3.2 and 3.3 are more systematic and thus applicable in more general situations, not to mention more mathematically satisfying. The computer search method is nevertheless an important part of this thesis' survey of existing codes, since it has yielded several good ones.

3.2 State splitting

In Section 1.2 we referred to the 1982 paper by Adler, Coppersmith, and Hassner [32] as the first mathematically rigorous formulation of the general problem of coding to meet the constraints of a channel with a finite number of states. The main theorem of that paper was that encoders and decoders, with finitely many states, could be found to map between unconstrained data sequences and sequences meeting the channel constraints, at any rational rate below channel capacity, and that the decoders would exhibit finite error propagation. The proof of that theorem was constructive;

that is, in showing that such codes (known as “sliding block codes”) exist, it developed an algorithm for constructing them. In this section we give a brief description of that algorithm, known as the “state splitting” algorithm, then explain how it can be applied to multi-track constraints, and propose a variation on the algorithm which would generate codes for combined modulation and error correction.

As is well known, a constrained channel can be described by a state transition diagram G , as in Figure 1.1, or equivalently by a state transition matrix A . The state transition matrix is a square matrix with the same number of rows (and columns) as there are states. The $(i, j)^{\text{th}}$ element (the element in the i^{th} row and j^{th} column) is an integer equal to the number of edges in G which begin at state i and end at state j . It is also well known (see, for instance, [12]) that the state transition matrix describing the change of state after l transitions (the l^{th} power of G) is equal to A^l . If at least 2^p edges leave every state of G^l , or equivalently if all the rowsums of A^l are at least 2^p , then a rate p/q binary code that meets the channel constraints can easily be found, where $q = nl$ and n is the number of tracks being coded in parallel. Indeed, if by deleting certain states from G , and thus deleting the corresponding rows and columns from A , one can arrive at a subsystem that meets the rowsum criterion, then the code construction is still straightforward.

In situations where p/q is less than the channel capacity, but not all of the rowsums of A^l are greater than 2^p , Adler *et al.* have shown that a rate p/q code still exists, and that it can be found through the process of state splitting. The Perron-Frobenius

theory of non-negative matrices [6] ensures that a vector v with integer entries, called an *approximate eigenvector*, can be found that satisfies the inequality:

$$A^l v \geq 2^p v$$

The i^{th} component of v can be thought of as the *weight* of state i . The edges of G can also be assigned weights, corresponding to the weights of the states at which they end. The approximate eigenvector inequality can then be thought of as saying that the sum of the weights of the edges leaving state i is at least 2^p times the weight of state i [6].

In the state splitting algorithm introduced in [32] and clarified in [6], the approximate eigenvector is used as a guide to splitting channel states to derive new state transition diagrams and matrices. A channel state is split by replacing it with two or more *offspring* states, each of which has the same incoming edges as the parent state had, but only some of the outgoing edges. Each edge which left the parent state is assigned to leave exactly one of the offspring states. The number of states into which a parent state may be split is determined by its weight, and to some extent the assignment of edges to offspring states is determined by the edges' weights, although the designer is left enough freedom in this regard to assign edges cleverly to achieve other goals, such as minimizing error propagation and simplifying implementation. Each time a state is split, the number of edges leaving the remaining states increases, since any edge that went to the split state must now go to all its offspring. This

process continues until all states have at least 2^p edges leaving them, at which point one assigns input words to the output words which label the edges, completing the encoder definition.

A number of techniques are given in [6] that simplify the state splitting process or allow steps to be combined. The most important one is *state merging*, in which states which were offspring of different parent states can be combined if they have identical outgoing edges. Splitting and merging can even be combined into a single step using a partial ordering of states. These ideas are applied in [6] to derive some existing (d, k) codes, including the Miller code.

In [35] Orcutt applied the state splitting algorithm to multi-track recording channels with two-dimensional (d, k) constraints. The process is exactly the same as in the one-dimensional case, except that the initial state transition diagram and matrix are those which describe the multi-track constraint. For example, for the 2-track $(1,3)$ constraint we start with the diagram of Figure 1.2 instead of Figure 1.1. The state transition matrix is:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

([13] explains how to compute these matrices.) As mentioned previously, the code in [40] was derived using state splitting on the “higher 3-block graph” of the 2-track $(1,3)$ -constrained channel (as defined in [32] and represented by the third power of A),

and achieved rate $4/6$, which is 98% of channel capacity, by splitting the 6 channel states into 13 encoder states.

In all of the above examples, both one- and two-dimensional, the implicit goal is to maximize the code rate, with no concern for any error-correcting ability. A rational rate was chosen which was as close to the channel capacity as could be attained with p and q small enough to be practically implemented. This led to the computation of an approximate eigenvector—there are in fact infinitely many choices of an approximate eigenvector, but the one was always chosen with the smallest maximum component or the smallest norm. From this point on, the code construction was guided by the approximate eigenvector.

If, however, one wishes to adapt this algorithm to construct *error-correcting* modulation codes, such a guide is not applicable. The rates at which we are trying to get error-correcting modulation codes are well below capacity, with the consequence that the rowsums of A^l are often all greater than 2^p to start with. This is equivalent to saying that the all-one vector is an approximate eigenvector. Thus, state splitting is not necessary to meet the constraints at these rates. Instead, we would like to split and merge states in such a way as to increase the *distance* between paths through the diagram. The all-one approximate eigenvector gives no clue as to how to do this.

However, some observations can be made about possible distance-building mechanisms which could be used to guide state splitting. Suppose a state is split into two

offspring, with the even-weight codewords leaving one offspring state, and the odd-weight words leaving the other. Both of those states will build distance 2 between the words leaving them. All the other states, on the other hand, will have added another outgoing edge whose label is at distance 0 from another outgoing edge. If the even/odd splitting is done to every state, many excess edges result. Eventually, the excess edges must be purged in such a way that the words entering any state are all distinct.

The above implies that every state in G^l must have at least 2^p even words *and* 2^p odd words leaving it. State splitting only increases the number of paths leaving a state—not the number of distinct words. Therefore if G^l does not meet this criterion, state splitting will do no good.

We now illustrate the state splitting technique with an example. The code we will derive will be a 2-track (1,3) code with rate and distance the same as Lee and Wolf's single-track (1,3) code [56] which we described in Section 3.1. Specifically we will take $l = 3$ to get a rate $p/q = 2/6$ code with distance 3. We begin with the 3rd power of the state transition matrix:

$$A^3 = \begin{bmatrix} 1 & 2 & 2 & 3 & 0 & 1 \\ 2 & 3 & 4 & 4 & 1 & 1 \\ 2 & 4 & 3 & 4 & 1 & 1 \\ 4 & 6 & 6 & 5 & 3 & 0 \\ 3 & 5 & 5 & 5 & 3 & 0 \\ 3 & 4 & 4 & 2 & 3 & 0 \end{bmatrix}$$

All the rowsums are greater than 2^2 , so the all-one vector is an approximate eigenvector. If the codewords are written out, it is seen that each state has at least 4

even-weight words and at least 4 odd-weight words leaving it. (In practice, we have found that the number of even-weight and the number of odd-weight words are always the same, plus or minus one word.)

Our mechanism for building distance 3 is to split the states into odd and even to get distance 2 between words *leaving* a state, and then ensure that the words *entering* a state are all distinct so we get distance 1 there. (This is the same scheme described in [51] in a slightly different context.) Although it is not a requirement that all the columnsums be greater than 2^2 , it is a good idea, since there must be an average of 4 words entering each state. States S_4 and S_5 , *i.e.*, the states that correspond to the last two rows and columns of A , each have fewer than four distinct words entering them. The only way to increase the number of distinct words entering a state is to merge it with another state, and that can only be done if they have the same words leaving. This seemed unlikely to happen. Therefore, states S_4 and S_5 were eliminated, since the resulting rowsums were still large enough. The resulting state transition diagram is shown in Figure 3.3. The codewords which appear on the edges in Figure 3.3 are abbreviated such that each integer represents the bits on both tracks at a given position. Specifically,

$$\text{"0"} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \quad \text{"1"} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \text{"2"} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad \text{"3"} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

(This abbreviated notation for two-dimensional codewords will be used throughout the remainder of this thesis.) The labeling of the states reflects the way they are

identified in [13], and also indicates (in braces) the possible strings of symbols that could put the channel in that state.

The next step is to split each state S_i into two offspring: E_i to which all the even-weight words are assigned, and O_i to which all the odd-weight words are assigned. The resulting FSTD is too large to include here, but the number of edges leaving each state are as follows:

E_0 :	8
O_0 :	8
E_1 :	14
O_1 :	12
E_2 :	13
O_2 :	12
E_3 :	22
O_3 :	20

We want only four edges leaving each state, so obviously this is way too many. Edges were eliminated by trial and error to give the encoder state diagram shown in Figure 3.4. (Many other encoder diagrams are possible.) An arbitrary assignment of input words (pairs of bits) to edges (preferably one to minimize the complexity of the logic circuitry implementation) completes the encoder definition. Decoding is accomplished by the Viterbi algorithm [58] rather than the deterministic mappings described in [32], since we are now trying to find the most likely path through a trellis (albeit a highly nonlinear one).

This example code is not all that great. It does achieve the same rate and distance as Lee and Wolf's (1,3) code, but that code's performance has been surpassed anyway

Figure 3.3: Third power of 2-track (1,3) FSTD, after elimination of states S_4 and S_5 .

Figure 3.4: State diagram for 2-track (1,3) encoder with distance 3 and rate 2/6.

by Lin and Wolf's code [49], which achieved distance 4 at the same rate, and also by some of the codes we will derive later in this thesis. Also, Lee and Wolf's encoder only has 3 states, whereas ours has 8. Nevertheless, this example serves to demonstrate how more powerful codes could be obtained by the same method. Distance can be built between words leaving a state by splitting that state—the even/odd split, which builds distance 2, is simplest, but there are other possibilities. Distance can then be built between words *entering* a state by elimination of excess edges, and possibly by state-merging. The beauty of this method is that the designer need not be concerned with concatenability—it is automatically achieved. Thus one may concentrate on distance-building. The drawback of this method is that the state transition diagrams become too large to be drawn and manipulated manually. For that reason, the process will have to be automated before truly powerful codes can be derived by state splitting.

3.3 Ungerboeck-type trellis coding

3.3.1 Background

The next method of generating an encoder is based on the concept of *trellis coded modulation* invented by Ungerboeck [57], which has become the technique of choice for bandpass digital signalling. In a digital communication system utilizing a band-pass channel, for instance a telephone line, the data must be encoded to allow error

correction, and it must be mapped to sinusoidal signals within a specified frequency range to be able to pass through the channel. This is quite analogous to the magnetic recording channel in that the data is mapped to a form compatible with the particular channel. Just as we are combining the steps of error correction coding and modulation coding, Ungerboeck combined the steps of error correction coding and bandpass modulation.

The reader is no doubt familiar with phase-shift keying (PSK) and quadrature amplitude modulation (QAM) as bandpass modulation techniques for digital signals. The bandpass symbols used are often represented as a constellation of points in the complex plane. The greater the Euclidean distance between these points, the lower the probability of error. What Ungerboeck did was to increase the number of symbols in the constellation and partition them into subsets in such a way that the minimum distance (d_{min}) between points in the same subset was larger than the d_{min} of the entire set. The partitioning continues until subsets either consist of only a single point or have $d_{min} \geq d_{free}$, the minimum distance that is desired between coded sequences.

The input data stream is encoded by a finite-state encoder like the one shown in Figure 3.5. Of the p input bits, $\tilde{p}$ of them go into a finite-state machine (FSM) that maps them to $\tilde{p} + i$ bits, which are used to select one of $2^{\tilde{p}+i}$ subsets of bandpass symbols. Each subset contains $2^{p-\tilde{p}}$ symbols, so the remaining $p - \tilde{p}$ input bits select a symbol out of the subset. If the FSM has ν memory elements, then the encoder is

Figure 3.5: General finite-state encoder.

described by a state transition diagram, or a trellis, with 2^v states. The assignment of subsets to trellis branches is done in such a way as to directly build Euclidean distance between paths. If the constellation has been partitioned to the extent that the subsets consist of only one symbol, then $\tilde{p} = p$ and the code's d_{free} is the distance built by the trellis structure of the FSM. If subsets contain more than one symbol, then d_{free} is the lesser of the subsets' d_{min} and the distance built by the trellis structure.

(For simplicity, we examine only the special case of $i = 1$, which is sufficient for our purposes. The rate of the FSM is $\tilde{p}/(\tilde{p} + 1)$ and hence the size of the constellation is double that of the uncoded case. From here on we will assume $i = 1$ and not write out the i anymore.)

As with convolutional codes, the output of the channel is decoded by a Viterbi decoder [58]. But instead of demodulating and then running the Viterbi algorithm

on the demodulated bits, with Hamming distance as a metric, Ungerboeck's scheme runs the algorithm directly on the QAM signal points, with Euclidean distance as a metric. TCM has allowed coding gains (reductions in channel signal-to-noise ratio required for the same probability of error) of more than 6 dB, a major breakthrough.

Pollara, McEliece, and Abdel-Ghaffar [59] have applied the same concept to ordinary error correction coding, thus generalizing both block codes and convolutional codes into the larger class of *finite-state codes*. The $\tilde{p}+1$ output bits of the finite-state machine (FSM) are used to select one of $2^{\tilde{p}+1}$ subsets of codewords, each of which can be thought of as a $(q, p - \tilde{p})$ block code. The remaining input bits select a word out of the subset. If $\nu = 0$ or $\tilde{p} = 0$, then we just have a block code. If $\tilde{p} = p$ and the FSM is linear then the result is a convolutional code.

3.3.2 Application to RLL codes; restriction of the set of terminal states

In addition to bandpass modulation, Ungerboeck and Wolf, among others, have applied these trellis-coding concepts to baseband signalling on partial response channels [60], and Marcellin and Fischer have applied them to source quantization [61]. A number of factors lead one to expect trellis coding to be useful for RLL codes as well. For one thing, Kobayashi long ago proposed the use of Viterbi decoding in magnetic recording channels [62]. Furthermore, as pointed out earlier, the situation

in a magnetic recording channel is very similar to a bandpass channel, as far as needing to increase distance between symbols from a non-linear set. However, there are important differences that make the magnetic channel a little more difficult.

The dimensionality of the signal space tends to be higher for the magnetic channel. Ungerboeck's original paper [57] only worked in two dimensions (QAM), but later he and others extended it to 4- and 8-dimensional spaces [63, 64], where the higher dimensions could be realized as specifying time windows within the symbol duration. But for us each dimension of a codeword can only take two values (0 and 1), so the number of dimensions is the codeword size $q = nl$. For instance, one of the codes we developed is a nonlinear subset of a 12-dimensional subspace of $\text{GF}(2)^{14}$. As a consequence, in some cases we cannot use the finite-state machines (convolutional coders) that Ungerboeck has tabulated [57, 64], but must search for new ones.

A second difference is that the signal constellations are not nice evenly spaced lattices, but irregularly spaced points scattered here and there through q -dimensional space. This makes set-partitioning much more of an *ad hoc* process. We cannot for instance use the construction of Calderbank and Sloane [65], who identified the subsets as cosets of a sublattice. If coding is thought of as a sphere-packing problem, we now have q -dimensional spheres of different sizes, a fact that was noted in [19].

The third difference is the fact that Euclidean distance is not the appropriate metric for the magnetic channel. Conceivably one of the metrics that Fredrickson

and Wolf used to detect peak shifts and bit slips [44] could be used, but we have not done this. We have used only Hamming distance.

The most fundamental difference, however, is that in Ungerboeck's papers the channel has no memory. There is no state transition diagram associated with the channel, only with the encoder. Thus any symbol can follow any other symbol; concatenability is not a concern. In RLL coding, on the other hand, the fact that the channel *has* memory is the very essence of the problem; concatenating symbols in such a way as to meet the constraints implied by the channel's state transition diagram is *exactly* what we are trying to do. In Section 3.2 we discussed how to work within the channel's state description and move cleanly from that to an encoder via state splitting. While this method is conceptually attractive, it became clear that it does not lend itself to distance-building very readily, because no guide analogous to the approximate eigenvector has yet been discovered. Ungerboeck's method, as an alternative, is designed to build distance, but in order to use it we must essentially "drown out" the underlying state structure of the channel, and *force* codewords to all be concatenable with each other. One way to do this is to have all codewords end in certain non-information-bearing bits which can be followed by any possible n -tuple that may begin the next word.

This "drowning out" of the channel states may seem like a "brute force" approach; however, it has always been used in block RLL codes. Freiman and Wyner [22], and later Franaszek [23, 24, 25], limited the channel to being in one of a reduced set

of “terminal states” at the ends of blocks. These states were those which could be followed by any of the codewords in the set. A somewhat similar concept was used by Tang and Bahl [26]. Their method allowed the channel to be put in any state at the end of a codeword, but then specified non-information-bearing “merging bits” to be placed *between* codewords to ensure concatenability. These merging bits were determined by the preceding and following codewords. Although restriction of the set of terminal states to obscure the channel’s state structure may seem like “brute force,” so far it has yielded the best codes of any method we have explored.

3.3.3 Ferreira

If the set-partitioning described above is continued all the way until subsets contain only one codeword, so that $\tilde{p} = p$, then what the encoder really does is map the output sequences of a linear convolutional code onto the nonlinear set of constrained sequences. The distance between constrained sequences totally depends on the distance built by the convolutional code’s structure. Thus the mapping must preserve distance, similarly to French’s mappings of codes [46] which we discussed in Section 2.2. In a series of papers [66, 14, 15, 16, 17], Ferreira developed a technique for combining error correction and RLL coding that draws on the large amount of existing knowledge about convolutional codes.

In addition to (d, k) constraints, many of Ferreira’s codes have been designed to also satisfy a charge constraint. If the bits on the tape, *after* the NRZI precoding, are

represented as $+1$ and -1 (corresponding to the two directions of magnetization), then the sum of all the bits from the start of recording up to any point on the tape must always lie between $-C$ and $+C$ for some small integer C . This makes the signal have no DC component, which is important in some applications. In fact, some of Ferreira's codes [17] are designed primarily for the d and C constraints, with k determined after the fact. However, his method remains applicable to (d, k) codes. When reading Ferreira it should be borne in mind that he uses b and l instead of the more standard d and k .

Ferreira identified nine properties of good convolutional codes that contribute to their distance-building [15]. A code is chosen that has these properties, with rate equal to $\tilde{p}/(\tilde{p} + 1)$ and minimum distance equal to that desired for the RLL code. The convolutional code sequences are then mapped to constrained sequences in such a way that the mapping preserves all nine properties and thus preserves, or possibly increases, Hamming distance. In [15] and [16] specific codes were found by this process. The one in [16] was presented in Table 1.1 as one of our baseline codes. In [17] a generalized procedure is given, although it still requires a computer search to find the appropriate mapping.

The codes derived in [14] and [15] had $d = 0$, so concatenability was easily achieved. The code in [16] had $d = 1$, so the set of constrained symbols was chosen to only include those with the last bit initially a "0". This is again the concept of leaving the channel in a terminal state to allow free concatenation of symbols. Use of

Ferreira's method for $d > 1$ would require more such non-information-bearing bits. Since these terminal bits are all the same, they do not contribute to distance building. Therefore, after the code sequences are put together, a terminal "0" can be changed to a "1" if doing so will not violate the d constraint. This often has the effect of reducing the maximum runlength; what started out as a (d, k) code may now satisfy a $(d, k - 1)$ constraint.

The concepts in Ferreira's papers are very similar to those in Ungerboeck's [57, 67], although interestingly enough, they never cite each other. The main difference (other than the symbol sets over which they are working) is the following: Ferreira starts with the state transition diagram and uses his distance-building properties as criteria to determine the mapping to channel symbols. Ungerboeck starts by building distance between the channel symbols, using his "set-partitioning" idea, then finds a convolutional code that will put these symbols on its trellis in such a way as to exploit their distance structure. Ungerboeck's codes are in a sense more general because they allow the base code to be a more general finite-state code than just a convolutional code. On the other hand, Ferreira's approach is a little more formalized, in that it identifies exactly what aspects of a convolutional code make it suitable.

In the next subsection we apply these concepts to multi-track RLL codes and achieve performance superior to Ferreira's. We mostly use Ungerboeck's approach, although certain aspects of Ferreira's work play important roles, particularly the idea

of changing non-information-bearing terminal bits from “0” to “1” whenever possible, to reduce runlengths.

3.3.4 Multi-track codes found by Ungerboeck’s methods

As mentioned in section 1.3, most of the capacity increase associated with multi-track constraints occurs from merely increasing n from 1 to 2. For this reason, and to keep the number of channel states small, most of our work has been with $n = 2$. Consider first the $(n \times l) = (2 \times 2)$ codewords satisfying a $(d, k) = (1, 3)$ constraint on the two tracks. There are nine of them:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

(This is the set of words that we attempted to place on a 16-state trellis by computer search, as described in Section 3.1.) There are too few words in this set to let us choose some subset of them that restricts the set of terminal states. If for instance we only used those words with zeroes in the last position of both tracks, there would only be four of them, and the minimum distance between them only 1.

Since the FSM operates at rate $\tilde{p}/(\tilde{p} + 1)$, and its output chooses one of $2^{\tilde{p}+1}$ subsets, the code would only have an overall rate of $1/4$, and no error-correcting ability.³

3.3.4.1 $n \times l = 2 \times 3$, rate = $2/6$ codes

Table 3.1 shows all the 2×3 words that satisfy a (1,3) constraint internally. Of the 25 words, 13 have even weight and 12 have odd weight. Since 12 and 13 are both greater than 2^3 , one might think we could encode at rate $3/6$. The problem, of course, is that the words cannot be freely concatenated. Any word that ends with a “1” in the top track, for instance, cannot be followed by any of the 10 words that begin with a “1” in the top track. Every state in the trellis would have 8 words entering it, each restricting the set of words that can leave that state. It is clear that sets of 8 words cannot be found that can then leave any of the states, so a rate $3/6$ code is impossible.

Instead let us consider only those 2×3 (1,3) words that end with “0”s on both tracks. These are simply the 2×2 words considered earlier, with “0”s appended to both tracks; hence there are nine of them. Throwing out the all-zero word so we don’t have to worry about concatenating with it, we are left with four even-weight words and four odd-weight words, any of which can follow any other. These can easily be placed on a standard trellis, as follows.

³It may at first appear that even this modest code is no good, since concatenation of two all-zero words would violate the k constraint. But in that situation, one of the “0”s in the second column of the first word can be changed to a “1” just as in Ferreira’s codes [16].

$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$

Table 3.1: 2×3 words satisfying the (1,3) constraint.

Since we want rate $2/6$, we need $2^2 = 4$ words leaving every state. The obvious first step of set-partitioning is into the four even-weight and four odd-weight words. In fact, it is easy to see that there is no other way to partition the set into two sets of four and get intraset distance ≥ 2 . (Such a partitioning is not actually necessary, but without it the number of states required to get the same d_{free} is much larger.) The next step of partitioning is all the way down to individual words.

To get $d_{free} = 3$, as Lee and Wolf achieved at rate $1/3$ (see Section 3.1), we follow a construction similar to that in [51], which makes the number of states equal to the number of branches leaving a state and allows paths which diverge from a state to remerge in just two transitions. In our construction, the paths build distance 2 in the first transition and distance 1 in the second transition, whereas in [51] they built distance 1 in the first transition and distance 2 in the second. The reason for

this variation is to allow a simple encoder implementation via a systematic FSM with the least significant output bit coming directly from the last shift register stage, consistent with the larger Ungerboeck-type trellises used later in the thesis. Since there are four branches leaving each state, we only need four states. Two states are assigned to have only even-weight words leaving, and two states are assigned to have only odd-weight words leaving. The four words leaving any state are distinct. The words are ordered in such a way that the four words *entering* any state are distinct too. Figure 3.6 shows the resulting trellis with codewords assigned. The encoder for this trellis appears in Figure 3.7. Note that in the encoder, an even-numbered FSM output, or *label*, maps to an even-weight codeword, and an odd-numbered label maps to an odd-weight word. This convention will be retained throughout the sequel.

To get $d_{free} = 4$, as Lin and Wolf achieved at rate $1/3$ (see Section 3.1), a slightly larger trellis is required. Again four branches leave every state, and paths which diverge from a state may remerge in only two transitions. Now however, distinct paths must build distance 2 at the first transition *and* distance 2 at the second transition. Therefore, states must not only have all even-weight or all odd-weight words *leaving* them, they must also have all even-weight or all odd-weight words *entering* them. This cannot be achieved with four states; Ungerboeck's 8-state trellis from [57] works though⁴. There are four kinds of states in this trellis:

⁴This is the trellis which appears in Figure 7 of Ungerboeck's original paper [57]. However, the one specified in all of his tables (in [57] and [64]) has the two inputs permuted. This of course yields an equivalent code but with different assignment of labels to branches.

Figure 3.6: Rate 2/6 trellis that achieves $d_{free} = 3$.

Figure 3.7: Encoder for the trellis of Figure 3.6.

1. Even words enter, even words leave—states 0 and 2
2. Even words enter, odd words leave—states 1 and 3
3. Odd words enter, even words leave—states 4 and 6
4. Odd words enter, odd words leave—states 5 and 7

This trellis is shown with the codewords placed on its branches in Figure 3.8, and its encoder circuit in Figure 3.9.

Clearly, to get $d_{free} = 5$ at this rate, we need a trellis with the following properties:

1. The four words leaving every state have distance 2 between them.
2. The four words entering every state have distance 2 between them.
3. Paths which diverge from a common state must require at least three transitions to remerge. This implies that the 16 paths of length 2 that begin at any given state must end at 16 distinct states; thus the trellis must have at least 16 states.
4. For every set of paths which diverge and remerge in three transitions (there are 4 paths in each such set), the words associated with the *middle* transition must be distinct.

Thus distinct paths build distance 2 in the first transition, 1 in the second, and 2 in the third, for a total of at least 5. The 16-state trellis tabulated in [57] is shown in Figure 3.10. To reduce the clutter, the codeword assignments are shown to the left

$$C_0 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad C_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$C_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad C_3 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$C_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad C_5 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$C_6 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad C_7 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Figure 3.8: Rate 2/6 trellis that achieves $d_{free} = 4$.

Figure 3.9: Encoder for the trellis of Figure 3.8.

of the states they leave, with the first word assigned to the top branch leaving that state, and so forth. Property 1 is clearly satisfied by these assignments. To verify property 2, the codewords *entering* each state are written to the right of the state, again with the first word corresponding to the top branch, and so forth. As in the 8-state trellis, there are four kinds of states:

1. Even words enter, even words leave—states 0, 2, 4, 6
2. Even words enter, odd words leave—states 1, 3, 5, 7
3. Odd words enter, even words leave—states 8, 10, 12, 14
4. Odd words enter, odd words leave—states 9, 11, 13, 15

Property 3 is verified by tracing out the 16 2-transition paths that begin at each state and seeing that they do indeed end at 16 distinct states. The reader may do this if he wishes. As for property 4, the number of steps of verification was reduced by a factor of 16, because it was recognized that the 3-transition paths beginning in

Figure 3.10: Rate 2/6 trellis that achieves $d_{free} = 5$.

Figure 3.11: Encoder for the trellis of Figure 3.10.

the same set of four and ending in the same set of four have the same set of middle transitions. This allowed the verification to in fact be done by hand.

Since all these properties are satisfied, this trellis achieves distance 5 at rate $2/6$, which was one of our goals as stated in Section 1.5.3. Its encoder is shown in Figure 3.11. It seems reasonable that one could continue to increase distance at this same rate, using this same set of codewords, by increasing the number of states. The larger trellises would be chosen to increase the number of transitions for divergent paths to remerge. Perhaps distance 6 could be gotten with 64 states and distance 7 with 256 states. We have not pursued this avenue though, but instead tried to increase the rate.

3.3.4.2 $n \times 1 = 2 \times 4$, rate = $3/8$ codes

A rather easy way to increase the rate of the distance 3 and 4 codes found in the last subsection (Figures 3.6 and 3.8) is to replace the 2×3 words by *pairs* of 2×4 words. To see how this works, consider the set of all 2×4 words that end in double

zeroes, shown in Table 3.2. These 24 words are the 2×3 words we saw in Table 3.1, with “0”s appended on both tracks, minus the all-zero word, which violates the k constraint. As before, the first partition is into 12 even and 12 odd words. But now, the second level of partitioning is not to individual words but to pairs with distance 4 between them. Additionally, we restrict the choice of words to those that begin with no more than one double zero and end with no more than two double zeroes. This guarantees that the words can all be freely concatenated without violating the k constraint. The resulting subsets are shown in Figure 3.12. If these are placed on the 4-state trellis of Figure 3.6 in the same places as the correspondingly numbered 2×3 words, we get a rate $3/8$, distance 3 code. If they are placed on the 8-state trellis of Figure 3.8, again in the respectively numbered locations, the result is a rate $3/8$, distance 4 code. The encoder for these codes is also shown in Figure 3.12, with the “convolutional coder” block representing the “FSM” part of either Figure 3.7 or Figure 3.9.

This neat trick, however, only works up to distance 4. Pairs with distance 5 or 6 cannot be found among the even or odd subset (more precisely, not *enough* such pairs can be found). So if we try to put pairs of 2×4 words in place of the 2×3 words on the 16-state trellis of Figure 3.10, d_{free} is limited not by the trellis’ structure but by the intrapair distance, so we only get $d_{free} = 4$. Obtaining distance 5 at rate $3/8$ requires a different approach, namely that we go back to subsets consisting of individual words and increase the number of states again. We again wish to build

$$\begin{aligned}
 D_0 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\} & D_1 &= \left\{ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\} \\
 D_2 &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\} & D_3 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \right\} \\
 D_4 &= \left\{ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \right\} & D_5 &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\} \\
 D_6 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \right\} & D_7 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \right\}
 \end{aligned}$$

Figure 3.12: Rate 3/8 encoder that achieves $d_{free} = 3$ or 4 using the trellises of Figure 3.6 or 3.8 respectively.

$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$	
$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$
$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$

Table 3.2: 2×4 words satisfying the (1,3) constraint and ending with zeroes on both tracks.

distance 2 in the first transition, distance 1 in the second transition, and distance 2 in the third transition. Since there now must be $2^3 = 8$ branches leaving each state, we now require $8^2 = 64$ states. Since the codewords lie in a 6-dimensional subspace of $\text{GF}(2)^8$, a logical place to look for a trellis is in Ungerboeck's table of 8-dimensional trellis codes (Table V of [64]). There is a 64-state code with $\tilde{p} = 3$ listed therein, and this code is indeed the one we want.

Because of its size, this trellis has not been drawn out, but its encoder is shown in Figure 3.13. Verifying that this trellis has all the properties we require to achieve $d_{\text{free}} = 5$ is a tedious task, but it has been done.

Figure 3.13: Encoder for 64-state trellis.

3.3.4.3 Higher rate $2 \times l$ codes with distance 5

Since increasing the length l of the codewords from 3 to 4 allowed a higher code rate with the same distance, a natural question is whether this can be continued. How high a rate can be reached by making the codewords longer? Can we get rate 0.5? Recall that the capacity of a channel with a 2-track (1,3) constraint is 0.68, so clearly rate 0.5 can be reached. However, if the codewords are too long, distance 5 will no longer be a very powerful code.

Let us examine how the number of eligible 2-track codewords grows with l . Each track must individually satisfy the $d = 1$ constraint. Tang and Bahl showed in [26] that the number of $(1, \infty)$ one-track words increased with l as the Fibonacci sequence. The words in an $n \times l$ (1,3) code are made by choosing n one-track $(1, \infty)$ words independently and stacking them on top of each other, then deleting those that contain more than three consecutive all-zero n -tuples. If the Fibonacci sequence is defined as:

$$\mathbf{x}_1 = 1$$

$$\mathbf{x}_2 = 2$$

$$\mathbf{x}_m = \mathbf{x}_{m-2} + \mathbf{x}_{m-1} \quad \text{for } m \geq 3$$

so that $(\mathbf{x}_m) = (1, 2, 3, 5, 8, 13, 21, 34, \dots)$, then the number of $n \times (l - 1)$ words is somewhat less than $\mathbf{x}_l^n$. We form our potential $n \times l$ words by appending a zero onto each track of these.

l	x_l	x_l^2	2^{l+1}
1	1	1	4
2	2	4	8
3	3	9	16
4	5	25	32
5	8	64	64
6	13	169	128
7	21	441	256
8	34	1156	512

Table 3.3: Fibonacci numbers and powers of two.

A necessary condition for the existence of any rate p/q trellis code found by Ungerböck's methods is that the number of eligible codewords is at least 2^{p+1} . Since we want $q = 2p$ and are working on two tracks so that $q = 2l$, this condition becomes $x_l^2 \geq 2^{l+1}$. Table 3.3 shows the two sides of this inequality for several values of l .

We have already seen the rate $2/6$ codes with $l = 3$ and the rate $3/8$ codes with $l = 4$, and Table 3.3 shows that we cannot get rate 0.5 with those sets of codewords. With $l = 5$ there would be just enough words if we could use them all; however, some must be eliminated to avoid violating the $k = 3$ constraint when words are concatenated. We now show that there are enough for a rate $4/10$ code.

There are at least four ways to choose which of the 64 words to eliminate:

1. Keep only those with no more than two double-zero pairs at the beginning and no more than one double-zero pair at the end. This option eliminates 28 words, leaving 36 (18 even-weight and 18 odd-weight).

2. Keep only those with no more than one double-zero pair at the beginning and no more than two double-zero pairs at the end. This option eliminates 17 words, leaving 47 (23 even-weight and 24 odd-weight).
3. Keep only those with no double-zero pairs at the beginning and no more than three double-zero pairs at the end. This option eliminates 28 words, leaving 36 (18 even-weight and 18 odd-weight).
4. Allow the k constraint to be initially violated when words are concatenated, then fix it by possibly changing one or both of the (non-information-bearing) “0”s at the end of the word to a “1” as Ferreira did in his codes. Then we only need to eliminate words that violate k internally, not counting the last position (only the all-zero word falls into this category), and words that would *prevent* the last-position “0”s from becoming “1”s (words that have double-ones in either the first position or the second-to-the-last position). This option eliminates 18 words, leaving 46 (22 even-weight and 24 odd-weight).

Since all four options leave more than 16 even-weight and more than 16 odd-weight words, we can use any of them to get a rate $4/10$ code. But if we want distance 5, then by Property 3 listed on page 76, the number of encoder states must be at least the square of the number of branches leaving each state. So if the second step of set-partitioning goes all the way down to individual codewords (the first step being the breaking into even and odd weights), then we need $16^2 = 256$ states. If on the

other hand, the second step is into pairs with intrapair distance ≥ 5 , then there are only 8 branches (subsets) leaving each state, and we can use the same 64-state trellis as in the rate $3/8$, distance 5 code of Section 3.3.4.2, but with the FSM outputs now selecting 1 of 16 *pairs*, and the 4th input bit selecting a codeword from the pair.

The distance between two even-weight words can only be even, and the distance between two odd-weight words can also only be even. So really we need intrapair distance 6 to get a 64-state trellis to work. If option 1, 3, or 4 is used in eliminating words, the remaining even-weight words cannot be paired up in such a way. This was determined by simply writing down all the words and trying it, noting how many weight-4 words there were. (To get distance-6 pairs among the evens, weight-2 words must be paired with weight-4 words, so there must be enough weight-4 words to go around.) But option 2 works quite nicely. The resulting pairs, shown in Table 3.4, were put together such that if a track in a word has weight 1, the corresponding track in the other word of the pair has weight 2, and vice versa. No words were used with zero-weight tracks. Thus the pairs not only have Hamming distance 6, they have *infinite* distance between them if one of Fredrickson and Wolf's metrics is used. So the code should exhibit good performance against peak shifts and bit slips as well.

Table 3.3 indicates that for $l = 6$ it may be possible to get a rate 0.5 code, as there are 169 potential codewords, and only 128 are needed. However, none of the four elimination options leaves that many:

$$\begin{aligned}
D_0 &= \left\{ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \right\} & D_1 &= \left\{ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \right\} \\
D_2 &= \left\{ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \right\} & D_3 &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \right\} \\
D_4 &= \left\{ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \right\} & D_5 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \right\} \\
D_6 &= \left\{ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \right\} & D_7 &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \right\} \\
D_8 &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \right\} & D_9 &= \left\{ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \right\} \\
D_{10} &= \left\{ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \right\} & D_{11} &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \right\} \\
D_{12} &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \right\} & D_{13} &= \left\{ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \right\} \\
D_{14} &= \left\{ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \right\} & D_{15} &= \left\{ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \right\}
\end{aligned}$$

Table 3.4: 2×5 codewords grouped into pairs with intrapair distance 6.

- Option 1 leaves 100 words (50 even, 50 odd)
- Option 2 leaves 123 words (61 even, 62 odd)
- Option 3 leaves 100 words (50 even, 50 odd)
- Option 4 leaves 118 words (60 even, 58 odd)

There are certainly enough for a rate $5/12$ code using any of the four options. In all four cases, the words can be grouped into 32 pairs with intrapair distance 6, but in none of the four cases can they be grouped into 16 foursomes with intrafoursome distance 6. Option 2 comes very close; a computer search found that 162 even foursomes can be put together, and it found within those a set of eight even foursomes which are disjoint, *i.e.* they contain 32 distinct words. These eight foursomes are shown in Table 3.5. (To save space in this table and in subsequent tables, we represent 2-track words the same way as in Figures 3.3 and 3.4. The pair of bits on the two tracks at a given position are represented as an integer between 0 and 3.) Among the odd words, however, although 48 odd foursomes could be put together, there is no set of eight odd foursomes which are disjoint. The upshot of all this is that each state must have 16 pairs of words leaving it to get rate $5/12$; hence to get distance 5 we need a trellis with $16^2 = 256$ states instead of just 64.

A 256-state trellis with $\tilde{p}/(\tilde{p} + 1) = 4/5$ could not be found in either the trellis-coding or convolutional-coding literature. So a computer search was performed to find one. To look through all systematic convolutional coders with feedback that

{212010, 030300, 300120, 121010}
 {210210, 030120, 121200, 303030}
 {030030, 212100, 300210, 103020}
 {212020, 030210, 120120, 301010}
 {203010, 012120, 120030, 300300}
 {201210, 012030, 302100, 121020}
 {201030, 210300, 120210, 102120}
 {021030, 210120, 302010, 120300}

Table 3.5: Eight disjoint foursomes of even-weight 2×6 words with intrafoursome distance ≥ 6 .

have $\nu = 8$ and rate $4/5$ was not possible, as the number of taps which may or may not be connected is $(\tilde{p} + 1)(\nu - 1) = (5)(7) = 35$, which gives 2^{35} possible encoders. Analysis of the relation between encoder configuration and code properties limited the search in several ways. For one thing, if every state is to be reachable in two steps, there cannot be two consecutive adders with no input bits coming into them. This also implies that since there is no adder on the feedback path, there must be inputs to the first and last adders. Without loss of generality, the most significant input bit can always be input to the first adder, since permutations of the input bits yield equivalent codes. This last fact also allows us to avoid counting equivalent codes multiple times. Furthermore, we can eliminate encoders in which the coefficients of the taps off of some input line, viewed as a vector, are a linear combination of the coefficients of taps off of other lines. This is because such a situation would mean that two different input words could lead to identical state transitions.

These restrictions still leave a vast multitude of encoders. Unlike the searches described in Ungerboeck and other TCM literature, we are not necessarily looking

for the one code with the very best Euclidean distance properties, but just for a code that meets the requirements listed on page 76. There may be many such codes. Therefore, a particular class of encoders was assumed for the initial search, namely those with two adder inputs from each of the three most significant input bits, one from the least significant input bit and none from the feedback path. Moreover, the tap off of the least significant input bit was only allowed to go into the last adder. Some of the encoders that were tested actually had a few additional taps because the program did not always disconnect old taps before trying new ones. The ninth encoder that was tested met all the requirements. It is shown in Figure 3.14.

A possible set of 32 pairs of words (from the “Option 4” list of 118 words) is shown in Table 3.6. Again the even-numbered pairs contain even-weight codewords, and the odd-numbered pairs contain odd-weight codewords. Recall that in Option 4, the k constraint may be initially violated when words are concatenated, but is fixed by changing the terminal “0” on one or both tracks to a “1.” This particular pairing has some nice symmetry properties just as our pairing of the 2×5 words had. Namely, all of the odd words have weight 3, with two “1”s on one track and one “1” on the other. Thirty of the 32 even words have only weight-1 tracks and weight-2 tracks. The words 212020 (in D_2) and 101210 (in D_4) both have a weight-3 track, but they are opposite tracks. Finally, there are no all-zero tracks.

Finally when we get to $l = 7$ there are enough words to get rate 0.5, if Option 2 or 4 is used for elimination. Using Option 4, a computer search found 143,648 foursomes

Figure 3.14: Encoder for 256-state trellis.

D_0	=	{203010, 030000}	D_1	=	{200210, 002120}
D_2	=	{002010, 210120}	D_3	=	{202010, 020120}
D_4	=	{020010, 212020}	D_5	=	{020300, 003020}
D_6	=	{200010, 030300}	D_7	=	{200120, 021200}
D_8	=	{000120, 101210}	D_9	=	{200300, 021020}
D_{10}	=	{000300, 212010}	D_{11}	=	{202100, 001210}
D_{12}	=	{002100, 210210}	D_{13}	=	{201020, 030200}
D_{14}	=	{020100, 103020}	D_{15}	=	{201200, 012020}
D_{16}	=	{200100, 120210}	D_{17}	=	{203000, 030020}
D_{18}	=	{001020, 210300}	D_{19}	=	{003010, 210020}
D_{20}	=	{001200, 212100}	D_{21}	=	{021010, 210200}
D_{22}	=	{003000, 120120}	D_{23}	=	{201010, 010120}
D_{24}	=	{201000, 120300}	D_{25}	=	{212000, 120020}
D_{26}	=	{021000, 102120}	D_{27}	=	{010210, 102020}
D_{28}	=	{010020, 121200}	D_{29}	=	{012010, 120200}
D_{30}	=	{010200, 121020}	D_{31}	=	{030010, 100120}

Table 3.6: A set of 32 disjoint pairs of 2×6 codewords with intrapair distance ≥ 6 .

of even codewords, and 119,825 foursomes of odd codewords, all with intrafoursome distance 6. It is still unknown whether 32 disjoint foursomes can be found among either of these lists. A computer program was written to search for such a set, but it ran approximately 14,000 cpu-minutes without finding one in the even set, and about 6000 cpu-minutes without finding one in the odd set, and was subsequently stopped. Using Option 2, however, a set of 32 disjoint even foursomes and a set of 32 disjoint odd foursomes were found. They are shown in Tables 3.7 and 3.8. Thus the $2^7 = 128$ words which must leave a state are grouped into 32 branches, and only $32^2 = 1024$ states are needed instead of the 4096 that would be required if we could only make pairs. This is very significant for implementation because 1024 states is,

for many practical purposes, the largest trellis that can be decoded in real time by the Viterbi algorithm at today's levels of technology.

(Incidentally, the computer found sets of 16 disjoint foursomes very quickly, even using Option 4, so a rate $6/14$ distance 5 code could be made with the same 256-state trellis as we used for the rate $5/12$ code. In applications where a 1024-state trellis is too complex, this code would represent a slight improvement in rate over the $5/12$ code, with the same distance and number of states, the only tradeoff being the codeword length. Simulation would be required to determine whether the slight reduction in performance due to increased codeword length outweighs the slight increase in rate.)

To find a 1024-state trellis on which to put all these foursomes, the same kind of computer search was performed as the one that produced the 256-state trellis. Again the search discovered a good trellis relatively quickly (a little over 4 cpu-hours), although this time it had to test 143 encoders before a satisfactory one was found, instead of only 9. The result is shown in Figure 3.15.

3.3.4.4 $n \times 2$ codes with distance 5

We have not done any research on codes using size $2 \times l$ words with $l > 7$ for a number of reasons. For one thing, the goal of a rate $p/2p$ (1,3) code with distance 5 was reached. Maybe even higher rates could be found with longer words, but the number of codewords, the difficulty of partitioning them into subsets, the number of

D_0	=	{1203030, 3021210, 1210200, 3030120}
D_2	=	{1201200, 3021030, 1212120, 3030300}
D_4	=	{3002100, 1021020, 1212030, 3030210}
D_6	=	{2121210, 1201010, 3030030, 1012100}
D_8	=	{2121030, 1003010, 1030200, 3012120}
D_{10}	=	{2103030, 3020100, 1001210, 1210020}
D_{12}	=	{2101010, 1201020, 3012030, 1030100}
D_{14}	=	{0303030, 3000300, 1021010, 1012020}
D_{16}	=	{0301010, 1021200, 3003030, 3010100}
D_{18}	=	{0121010, 1202100, 3001200, 1010030}
D_{20}	=	{0103010, 1200300, 3001020, 1030010}
D_{22}	=	{0101210, 1020300, 1003020, 3010010}
D_{24}	=	{0101030, 1202010, 1020120, 3010200}
D_{26}	=	{2101200, 3002020, 1020210, 1210100}
D_{28}	=	{2101020, 3020200, 1200120, 1012010}
D_{30}	=	{0301200, 1202020, 3001010, 1010300}
D_{32}	=	{2120100, 0301020, 1002120, 1210010}
D_{34}	=	{2102100, 0121200, 3020010, 1010120}
D_{36}	=	{2100300, 0121020, 1200210, 3010020}
D_{38}	=	{0300300, 0103020, 3000120, 1010210}
D_{40}	=	{0302010, 2100120, 3000210, 1030020}
D_{42}	=	{2030100, 2102010, 0300120, 1001030}
D_{44}	=	{2012100, 2100210, 0120120, 1200030}
D_{46}	=	{2010300, 2120010, 0302100, 1002030}
D_{48}	=	{2010120, 0300030, 0120300, 3002010}
D_{50}	=	{0212100, 2102020, 0300210, 1020030}
D_{52}	=	{0210300, 0120210, 0102120, 3020020}
D_{54}	=	{2030010, 0210120, 0102030, 1000200}
D_{56}	=	{2012010, 0302020, 2120200, 1000100}
D_{58}	=	{2021010, 2012020, 0210210, 0100100}
D_{60}	=	{2003010, 0212020, 2030200, 0120030}
D_{62}	=	{0202120, 2020300, 0201210, 2100030}

Table 3.7: A set of 32 disjoint foursomes of even-weight 2×7 codewords with intrafoursome distance ≥ 6

D_1	=	{3021200, 1203010, 3010030, 1210300}
D_3	=	{3021020, 1201210, 3012010, 1210120}
D_5	=	{3003020, 1021210, 1212010, 3030100}
D_7	=	{1203020, 3021010, 1210210, 3012100}
D_9	=	{3020300, 3003010, 1030030, 1212100}
D_{11}	=	{3002120, 3001210, 1210030, 1030300}
D_{13}	=	{2121010, 1201030, 3010210, 1030120}
D_{15}	=	{2103010, 1021030, 3030200, 1012120}
D_{17}	=	{2101210, 1003030, 1030210, 3010120}
D_{19}	=	{2101030, 1001200, 1212020, 3030010}
D_{21}	=	{0303010, 1202120, 3001030, 3010300}
D_{23}	=	{0301210, 3020210, 1200100, 1012030}
D_{25}	=	{0121210, 1202030, 3020120, 1010200}
D_{27}	=	{2121200, 0301030, 1020100, 3012020}
D_{29}	=	{2120300, 0121030, 1002100, 3030020}
D_{31}	=	{2120120, 0101010, 3002030, 1010100}
D_{33}	=	{2100100, 0103030, 1200200, 3020030}
D_{35}	=	{2120210, 0300100, 2103020, 1010010}
D_{37}	=	{2120030, 0302120, 0101200, 1010020}
D_{39}	=	{0302030, 0100300, 2121020, 1001010}
D_{41}	=	{2030300, 2100010, 0303020, 1000120}
D_{43}	=	{2030120, 0300200, 2102030, 1001020}
D_{45}	=	{0210100, 0120010, 2102120, 3000200}
D_{47}	=	{2030210, 2012120, 0101020, 1000300}
D_{49}	=	{2030030, 0212120, 0120100, 1200010}
D_{51}	=	{0212030, 2010100, 0100210, 1020020}
D_{53}	=	{0210200, 2012030, 0100120, 1020010}
D_{55}	=	{2021210, 0210010, 0102020, 3000100}
D_{57}	=	{2021030, 2010200, 0300010, 1002020}
D_{59}	=	{2001010, 0210020, 0102100, 1020200}
D_{61}	=	{2020100, 2003030, 0300020, 1000210}
D_{63}	=	{0200300, 0203030, 2010010, 0120020}

Table 3.8: A set of 32 disjoint foursomes of odd-weight 2×7 codewords with intrafoursome distance ≥ 6

Figure 3.15: Encoder for 1024-state trellis.

encoder states, the difficulty of the search for an encoder, and the complexity of the decoding algorithm are all increasing exponentially with codeword size. Additionally, as the words, and hence the divergent/remergent trellis-path pairs (error events) get longer, a distance 5 code gets less powerful; a code which corrects 2 errors in a string of, say, 27 bits (per track) is not as good as one which corrects 2 errors in a string of 18 bits (per track).

Hence, we now leave $2 \times l$ codewords and explore the other dimension of codeword size, *viz.* the number of tracks. Specifically, l will be fixed at 2, and $n \times 2$ codewords will be used with increasing n . Again the last (*i.e.*, second) position will be initially set to all “0”s to leave the channel in a terminal state and allow free concatenation of words. This may seem to be a huge waste of bits, as half the bits in a word carry no information; however, this is partially compensated by the fact that the other position is then unconstrained and can contain any n -tuple of bits.

The first codeword size in this series is 2×2 , but we have already discussed this set of words at the beginning of Section 3.3.4 and saw that there aren’t enough words to make a useful code. The 3×2 words are shown in Table 3.9. Obviously these eight words all satisfy the $d = 1$ constraint, but if two consecutive all-zero words occurred, the $k = 3$ constraint would be violated. To avoid this situation, we use the same technique as Ferreira did in [16] and we did in the “Option 4” method of Section 3.3.4.3. Namely, whenever two consecutive all-zero words occur, one of the “0”s in the second position of the first word is changed to a “1”. There is no danger

$$\begin{array}{l}
\text{EVEN: } \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \\
\text{ODD: } \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}
\end{array}$$

Table 3.9: The 3×2 codewords.

of violating d by doing this, since there are “0”s on both sides of this position. The choice of which track or tracks to put a “1” on is completely arbitrary; in fact, this choice could conceivably be used to convey some other piece of information, as we will discuss in Section 3.3.4.5.

The eight codewords again partition into four even-weight and four odd-weight words, with intraset distance 2 in both these sets, just as they did in the 2×3 case. Thus the trellises of Figures 3.6, 3.8, and 3.10 can be used to get codes with the same rate ($2/6$) and the same distances as in Section 3.3.4.1. Specifically we achieve distance 3 with 4 states, distance 4 with 8 states, and distance 5 with 16 states. Depending on the error distribution, though, the 3-track codes may give better performance than the 2-track, because the error events in which a given distance is built between paths is shorter (fewer bit positions).

Likewise, the 4×2 codewords of Table 3.10 can be used in rate $3/8$ codes with the same distances and the same trellises as the 2×4 words used in Section 3.3.4.2. This is because the codewords partition into 8 even-weight and 8 odd-weight words, and both these sets then partition quite easily into pairs with intrapair distance 4,

EVEN:	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}$
ODD:	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$

Table 3.10: The 4×2 codewords.

by complementing the first bit on each track. This grouping is shown in Table 3.10 by placing the words of a pair in the same column. Thus, just as in the 2×4 case, the 4-state and 8-state trellises of Figures 3.6 and 3.8 can be used with each edge representing a pair, yielding distances 3 and 4 respectively. Also as in the 2×4 case, distance 6 pairs cannot be put together, so to get distance 5 we again need the 64-state trellis generated by the circuit of Figure 3.13.

The above pattern does not however hold for all codeword sizes. That is, for some values of q , the set of $q/2 \times 2$ words cannot be partitioned into subsets as large as the

set of $2 \times q/2$ words can, given the same required intrasubset distance. For instance, we were able in Section 3.3.4.3 to divide the set of 2×5 codewords into pairs with intrapair distance 6 (8 such pairs with even weight and 8 with odd weight), leading to a rate 4/10 distance 5 code with 64 encoder states. The 5×2 words cannot be grouped into distance 6 pairs, since there are only 5 information-carrying bits in which to build distance! Distance 5 pairs could be constructed, of course, by complementing the first bit on each track, but then the even/odd partition would be lost, so only one unit of distance could be built at each transition. So to get a distance 5 code at rate 4/10 with 5×2 words, a 256-state trellis is needed, such as the one generated by the circuit of Figure 3.14, with each branch representing only a single codeword.

There seem to be both advantages and disadvantages to using $n \times 2$ codes as opposed to $2 \times l$ codes. The $n \times 2$ codes—or at least the sets of codewords—are easier to construct (and easier to count) for large q . This is because the $n \times 2$ codes are in fact linear. The code is a vector space rather than a set of irregularly spaced points taken from a vector space. This suggests that more of the rich algebraic structure of modern coding theory could be used to generate more powerful codes. On the other hand, the way we constructed these codewords is intimately tied to the fact that $d = 1$, so they may not be a very general class of codes after all. Performance-wise, there may be a great advantage in the fact that the shortest error events, those corresponding to the smallest intrapath distances, are only 6 bits long on each track instead of $3l$.

An even greater advantage would seem to be the fact that only every other bit on a track contributes to distance-building. The n -tuple that occurs in the second half of each codeword is totally ignored by the decoder. Any errors that occur there don't have to be corrected or detected. One implication of this is that the error-correcting ability of the code is spread over fewer bits. Another very important implication is that peak shifts, which as mentioned earlier are equivalent to two consecutive additive errors, are now reduced to only one additive error as far as the decoder is concerned, since it never sees the second one.

A disadvantage of $n \times 2$ codes is that no matter how many tracks are encoded together, a rate $p/2p$ code can never be achieved. The number of codewords is always 2^n , of which half, or 2^{n-1} , are available from any state. Hence the rate is $(\log_2 2^{n-1})/(2n) = (n-1)/2n$. If n could be made arbitrarily large (an impossibility for a practical system), the rate could be arbitrarily close to 0.5, but it can never reach it, whereas we reached it with 2×7 codewords before. Another disadvantage is that the trellis complexity increases with codeword size faster than it did for $2 \times l$ codewords, because the subsets that can be put together are not always as large, as we saw comparing 5×2 and 2×5 codes. However, because the error-correcting ability of these codes is more efficient than that of the $2 \times l$ codes, codes with $d_{free} = 3$ or 4 should perhaps be considered as well as those with $d_{free} = 5$. Table 3.11 lists the parameters of all the $n \times 2$ (1,3) codes with rate $(n-1)/2n$ and distance 4 or 5

Codeword dimensions	Rate	Distance 4			Distance 5		
		Subset size	No. of subsets	No. of states	Subset size	No. of subsets	No. of states
3×2	$2/6$	1	8	8	1	8	16
4×2	$3/8$	2	8	8	1	16	64
5×2	$4/10$	2	16	16	1	32	256
6×2	$5/12$	4	16	16	2	32	256
7×2	$6/14$	4	32	32	2	64	1024
8×2	$7/16$	8	32	32	2	128	4096
9×2	$8/18$	8	64	64	4	128	4096

Table 3.11: Parameters of $n \times 2$ codes for $3 \leq n \leq 9$.

for n up to 9. The last code listed is probably one of the most interesting from an application standpoint, since 9-track tape is a popular recording medium.

3.3.4.5 Other possibilities

Since there are advantages to both of the codeword shapes we have considered ($2 \times l$ and $n \times 2$), it is possible that when q is greater than 8, and not equal to 2 times a prime, a compromise can be reached by using other shapes. For example, $q = 12$ can be realized as 2×6 , 3×4 , 4×3 , or 6×2 codewords.

In some of the codes described above there were situations where terminal “0”s had to change to “1”s to meet the k constraint. There was usually a choice of what track or tracks these “1”s should go on. Additionally, in *all* the codes there are situations where terminal “0”s *can* turn into “1”s although they don’t *have* to. In both these types of situations it is conceivable that these bits could be put to good use by stuffing some information into them—perhaps some control information,

or a secondary low-rate channel carrying non-critical information that doesn't need protection from errors. Just as intriguingly, the decision of whether or not to invert these bits when there is a choice could be based on DC content of the signals on the individual tracks. The terminal zero on a track could be inverted or not inverted, whichever would reduce the accumulated charge (running digital sum). It seems unlikely that such a scheme could transform our (d, k) codes into (d, k, C) DC-free codes, but at least the DC content could be reduced. These are possible areas of future investigation.

CHAPTER 4

Summary, conclusions, and future work

The background material in Chapter 1 of this thesis has, it is hoped, provided the reader with sufficient justification for what we have attempted to do in the remainder of the thesis. Specifically, it should be clear that the magnetic recording channel, in order to perform reliably at high storage densities, requires both modulation coding and error correction coding. It should also be clear what is gained by combining these two functions, and why there are benefits associated with using multiple tracks to do so.

We have explored many coding techniques in Chapters 2 and 3, and found a number of ways to incorporate error correction into two-dimensional modulation codes, or equivalently, to extend error-correcting modulation codes into two dimensions. A direct comparison of these approaches is not always straightforward, since there are many code parameters which vary from one to the next, and different combinations may be more advantageous in different applications. However, an attempt will be made here to summarize the techniques and to identify the relative strengths and weaknesses of the resulting codes.

The single-track error-correcting RLL coding techniques we surveyed in Chapter 2 have varying levels of applicability to multi-track codes. The approaches of Ferreira and Lin, and of Abdel-Ghaffar and Weber, use integer composition representations of the codewords, rather than bit representations, and appear to not be at all applicable in two dimensions, since the k constraint cannot be dealt with. Hilden, Howe, and Weldon's method also uses integer compositions in the form of runlengths, but the example given in Section 2.8 illustrated that good two-dimensional codes could indeed be generated by this method. The increased channel capacity associated with multi-track constraints is not fully exploited, since the tracks are only treated jointly up to a certain step. Nevertheless, the code rate is certainly higher than what could be attained with a single-track Hilden, Howe, and Weldon construction.

Patapoutian and Kumar's technique of finding the coset of a linear error correction code containing the most (d, k) words could theoretically be applied in two dimensions, but it seems to be a less direct way of building a set of constrained words with distance between them than some of the other methods. It is also not clear whether the resulting code rates would be competitive with other methods unless the blocks were made quite long. The concepts used by Ytrehus and by Fredrickson and Wolf could also be used for two-dimensional codes, but their application would be much more complicated than in the one-dimensional case.

Of all the block coding techniques, it seems that those of Blaum and of French are the most well-suited for two-dimensional codes. The concept common to them, to

Ferreira’s trellis codes, to the Ungerboeck-type trellis codes that we derived, and, to some extent, to the block codes of Patapoutian and Kumar and Hilden, Howe, and Weldon, is that of mapping from some error-correcting code to a set of (d, k) constrained sequences, in such a way that the distance structure of the error-correcting code is preserved.

The other trellis coding techniques—state-splitting and computer search—also have potential for finding new error-correcting two-dimensional modulation codes. For state-splitting, however, much of the design procedure needs to be automated before it can create high-performance codes for realistic constraints, since the state-transition diagrams quickly become too large and cumbersome to work with manually. As for computer searches, more work is needed to limit the class of codes being searched.

Table 4.1 presents a comparison of the important parameters of all the error-correcting two-dimensional $(d, k) = (1, 3)$ codes we have explicitly derived in this thesis. For the Hilden, Howe, and Weldon code, d_{free} is not given because it is a block code, not a trellis code like the others, and because it is designed to build peak shift distance, not Hamming distance.

The results shown in Table 4.1 compare favorably with the baseline codes in Table 1.1 and the goals we stated in Section 1.5.3. We achieved distance 5 in a rate $p/3p$ code quite easily, with 2×3 or 3×2 codewords and 16 states. In fact we achieved

Type of code	Codeword dimensions $n \times l$	Code rate p/q	Distance d_{free}	Number of states
Hilden, Howe, and Weldon	2×71	48/142	(Able to correct 2 peak shifts per track per codeword)	1
Adler, Copper-smith, Hassner	2×3	2/6	3	8
Ungerboeck	2×3	2/6	3	4
Ungerboeck	2×3	2/6	4	8
Ungerboeck	2×3	2/6	5	16
Ungerboeck	2×4	3/8	3	4
Ungerboeck	2×4	3/8	4	8
Ungerboeck	2×4	3/8	5	64
Ungerboeck	2×5	4/10	5	64
Ungerboeck	2×6	5/12	5	256
Ungerboeck	2×7	6/14	5	256
Ungerboeck	2×7	7/14	5	1024
Ungerboeck	3×2	2/6	4	8
Ungerboeck	3×2	2/6	5	16
Ungerboeck	4×2	3/8	4	8
Ungerboeck	4×2	3/8	5	64
Ungerboeck	5×2	4/10	4	16
Ungerboeck	5×2	4/10	5	256
Ungerboeck	6×2	5/12	4	16
Ungerboeck	6×2	5/12	5	256
Ungerboeck	7×2	6/14	4	32
Ungerboeck	7×2	6/14	5	1024
Ungerboeck	8×2	7/16	4	32
Ungerboeck	8×2	7/16	5	4096
Ungerboeck	9×2	8/18	4	64
Ungerboeck	9×2	8/18	5	4096

Table 4.1: Summary of (1,3) codes derived in this thesis.

distance 5 at higher rates, up to $p/2p$, with reasonable complexity (2×7 codewords and 1024 states).

Since 2×7 was the smallest codeword size for which a rate $p/2p$ code was possible, we did not bother to explicitly generate a distance 3 code at this rate, but went straight to distance 5. A distance 3 code would not be all that powerful when the error events in the trellis are so long. However, if such a code were desired, it could be generated in the same fashion as the others quite readily.

As for peak shifts, not only is the Hilden, Howe, and Weldon code specifically designed to attack them, but any of the distance 5 codes can correct a single peak shift as a special case of a double additive error pattern. Furthermore, the $n \times 2$ codes, although they can never reach rate $p/2p$, are very effective on peak shifts because a peak shift shows up as only a single erroneous information-bearing bit instead of two.

There is no single “best” code listed. The choice of an error-correcting two-dimensional modulation code will depend on the specific application: the probabilities of different classes of errors, the number of tracks, the correlation of errors along a track, the correlation of errors across tracks, the rate requirements, and the allowable decoder complexity.

There are many directions in which this work can be extended. As mentioned in Section 3.3.4.5, there may be advantages to using codeword dimensions with $n \neq 2$ and $l \neq 2$, or perhaps charge constraints or an ancillary channel could be incorporated

into the non-information-bearing bits. Of course, constraints other than (1,3) should be considered. With multi-track constraints it is even possible to have $k \leq d$, if analysis shows that some channel would benefit from such a constraint. As the state-splitting and/or set-partitioning procedures become more automated, it should become feasible to generate codes such as these for arbitrary sets of constraints.

REFERENCES

- [1] C. A. French, G. S. Dixon, and J. K. Wolf, "Results involving (D, K) constrained M -ary codes," *IEEE Trans. Mag.*, vol. MAG-23, pp. 3678–3680, Sept. 1987.
- [2] C. S. Chi, "Triplex code for quarternary high density recording," *IEEE Trans. Mag.*, vol. MAG-20, pp. 888–890, Sept. 1984.
- [3] K. A. S. Immink, "Coding techniques for the noisy magnetic recording channel: A state-of-the-art report," *IEEE Trans. Comm.*, vol. 37, pp. 413–419, May 1989.
- [4] A. R. Calderbank, C. Heegard, and T. A. Lee, "Binary convolutional codes with application to magnetic recording," *IEEE Trans. Inform. Th.*, vol. IT-32, pp. 797–815, Nov. 1986.
- [5] L. J. Fredrickson, "Coding for maximum likelihood detection on a magnetic recording channel," *IEEE Trans. Mag.*, vol. 26, pp. 2315–2317, Sept. 1990.
- [6] B. H. Marcus, P. H. Siegel, and J. K. Wolf, "Finite-state modulation codes for data storage," *IEEE Journal on Selected Areas in Communications*, vol. 10, pp. 5–37, Jan. 1992.
- [7] P. H. Siegel, "Applications of a peak detection channel model," *IEEE Trans. Mag.*, vol. MAG-18, pp. 1250–1252, Nov. 1982.
- [8] E. R. Katz and T. G. Campbell, "Effect of bitshift distribution on error rate in magnetic recording," *IEEE Trans. Mag.*, vol. MAG-15, pp. 1050–1053, May 1979.
- [9] H. D. L. Hollmann, "The general solution of write equalization for RLL (d, k) codes," *IEEE Trans. Inform. Th.*, vol. 37, pp. 856–862, May 1991.
- [10] R. C. Schneider, "Write equalization in high-density magnetic recording," *IBM J. Res. Dev.*, vol. 29, pp. 563–568, Nov. 1985.

- [11] P. H. Siegel, "Recording codes for digital magnetic storage," *IEEE Trans. Mag.*, vol. MAG-21, pp. 1344–1349, Sept. 1985.
- [12] R. E. Blahut, *Principles and Practice of Information Theory*. Reading, MA: Addison-Wesley, 1987.
- [13] M. W. Marcellin and H. J. Weber, "Two-dimensional modulation codes," *IEEE Journal on Selected Areas in Communications*, vol. 10, pp. 254–266, Jan. 1992.
- [14] H. C. Ferreira, "Lower bounds on the minimum Hamming distance achievable with runlength constrained or DC free block codes and the synthesis of a $(16,8)$ $d_{min} = 4$ DC free block code," *IEEE Trans. Mag.*, vol. MAG-20, pp. 881–883, Sept. 1984.
- [15] H. C. Ferreira, "The synthesis of magnetic recording trellis codes with good Hamming distance properties," *IEEE Trans. Mag.*, vol. MAG-21, pp. 1356–1358, Sept. 1985.
- [16] H. C. Ferreira, J. F. Hope, and A. L. Nel, "Binary rate four eighths, runlength constrained, error correcting magnetic recording modulation code," *IEEE Trans. Mag.*, vol. MAG-22, pp. 1197–1199, Sept. 1986.
- [17] H. C. Ferreira, D. A. Wright, and A. L. Nel, "Hamming distance preserving mappings and trellis codes with constrained binary symbols," *IEEE Trans. Inform. Th.*, vol. 35, pp. 1098–1103, Sept. 1989.
- [18] R. E. Blahut, *Digital Transmission of Information*. Reading, MA: Addison-Wesley, 1990.
- [19] V. D. Kolesnik and V. Y. Krachkovsky, "Generating functions and lower bounds on rates for limited error-correcting codes," *IEEE Trans. Inform. Th.*, vol. 37, pp. 778–788, May 1991.
- [20] K. A. S. Immink, "Runlength-limited sequences," *Proc. IEEE*, vol. 78, pp. 1745–1759, Nov. 1990.
- [21] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, 1948.

- [22] C. V. Freiman and A. D. Wyner, "Optimum block codes for noiseless input restricted channels," *Inform. Contr.*, vol. 7, pp. 398-415, 1964.
- [23] P. A. Franaszek, "Sequence-state coding for digital transmission," *Bell Syst. Tech. J.*, vol. 47, pp. 143-157, Jan. 1968.
- [24] P. A. Franaszek, "On synchronous variable length coding for discrete noiseless channels," *Inform. Contr.*, vol. 15, pp. 155-164, 1969.
- [25] P. A. Franaszek, "Sequence-state methods for run-length-limited coding," *IBM J. Res. Dev.*, pp. 376-383, July 1970.
- [26] D. T. Tang and L. R. Bahl, "Block codes for a class of constrained noiseless channels," *Inform. Contr.*, vol. 17, pp. 436-461, 1970.
- [27] G. F. M. Beenker and K. A. S. Immink, "A generalized method for encoding and decoding run-length-limited binary sequences," *IEEE Trans. Inform. Th.*, vol. IT-29, pp. 751-754, Sept. 1983.
- [28] G. V. Jacoby, "A new look-ahead code for increased data density," *IEEE Trans. Mag.*, vol. MAG-13, pp. 1202-1204, Sept. 1977.
- [29] G. V. Jacoby and R. Kost, "Binary two-thirds rate code with full word look-ahead," *IEEE Trans. Mag.*, vol. MAG-20, pp. 709-714, Sept. 1984.
- [30] A. Lempel and M. Cohn, "Look-ahead coding for input-restricted channels," *IEEE Trans. Inform. Th.*, vol. IT-28, pp. 933-937, Nov. 1982.
- [31] A. M. Patel, "Zero-modulation encoding in magnetic recording," *IBM J. Res. Dev.*, vol. 19, pp. 366-378, July 1975.
- [32] R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes—An application of symbolic dynamics to information theory," *IEEE Trans. Inform. Th.*, vol. IT-29, pp. 5-22, Jan. 1983.
- [33] C. D. Heegard, B. H. Marcus, and P. H. Siegel, "Variable-length state splitting with applications to average runlength-constrained (ARC) codes," *IEEE Trans. Inform. Th.*, vol. 37, pp. 759-777, May 1991.

- [34] A. D. Weathers and J. K. Wolf, "A new rate $2/3$ sliding block code for the $(1, 7)$ runlength constraint with the minimal number of encoder states," *IEEE Trans. Inform. Th.*, vol. 37, pp. 908–913, May 1991.
- [35] E. K. Orcutt, "Encoding of multi-track (d, k) modulation codes," Ph.D. dissertation, Univ. of Arizona, Tucson, 1992.
- [36] L. J. Fredrickson and J. K. Wolf, "Coding using multiple block (d, k) codes," in *Proceedings IEEE Conf. on Communications*, pp. 1623–1627, June 1989.
- [37] K. A. S. Immink and H. D. L. Hollmann, "Prefix-synchronized run-length-limited sequences," *IEEE Journal on Selected Areas in Communications*, vol. 10, pp. 214–222, Jan. 1992.
- [38] A. S. J. Helberg and H. C. Ferreira, "Runlength constrained recording codes with ancillary channels," *IEEE Trans. Mag.*, vol. 26, pp. 2321–2323, Sept. 1990.
- [39] E. K. Orcutt and M. W. Marcellin, "Enumerable multi-track (d, k) block codes," *IEEE Trans. Inform. Th.*, to appear.
- [40] E. K. Orcutt and M. W. Marcellin, "2-track $(1, 3)$ code with rate $2/3$," *Electronic Letters*, vol. 28, pp. 1112–1114, June 1992.
- [41] D. B. Seymour, "Characterization of dropouts with data read-back errors for digital magnetic-tape recorders," *IEEE Trans. Mag.*, vol. MAG-22, pp. 44–46, Jan. 1986.
- [42] T. D. Howell, "Analysis of correctable errors in the IBM 3380 disk file," *IBM J. Res. Dev.*, vol. 28, pp. 206–211, Mar. 1984.
- [43] H. M. Hilden, D. G. Howe, and E. J. Weldon, Jr., "Shift error correcting modulation codes," *IEEE Trans. Mag.*, vol. 27, pp. 4600–4605, Nov. 1991.
- [44] L. J. Fredrickson and J. K. Wolf, "Error detecting multiple block (d, k) codes," *IEEE Trans. Mag.*, vol. 25, pp. 4096–4098, Sept. 1989.
- [45] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.

- [46] C. A. French, "Distance preserving run-length limited codes," *IEEE Trans. Mag.*, vol. 25, pp. 4093–4095, Sept. 1989.
- [47] M. Blaum, "Combining ECC with modulation: Performance comparisons," *IEEE Trans. Inform. Th.*, vol. 37, pp. 945–949, May 1991.
- [48] K. A. S. Abdel-Ghaffar and J. H. Weber, "Bounds and constructions for runlength-limited error-control block codes," *IEEE Trans. Inform. Th.*, vol. 37, pp. 789–800, May 1991.
- [49] Y. Lin and J. K. Wolf, "Combined ECC/RLL codes," *IEEE Trans. Mag.*, vol. 24, pp. 2527–2529, Nov. 1988.
- [50] H. C. Ferreira and S. Lin, "Error and erasure control (d, k) block codes," *IEEE Trans. Inform. Th.*, vol. 37, pp. 1399–1408, Sept. 1991.
- [51] P. Lee and J. K. Wolf, "A general error-correcting code construction for run-length limited binary channels," *IEEE Trans. Inform. Th.*, vol. 35, pp. 1330–1335, Nov. 1989.
- [52] O. Ytrehus, "Runlength-limited codes for mixed-error channels," *IEEE Trans. Inform. Th.*, vol. 37, pp. 1577–1585, Nov. 1991.
- [53] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.
- [54] A. Patapoutian and P. V. Kumar, "The (d, k) subcode of a linear block code," *IEEE Trans. Inform. Th.*, vol. 38, pp. 1375–1382, July 1992.
- [55] H. M. Hilden and E. J. Weldon, Jr., "Codes for correcting additive errors, shift errors and synch slippages," Tech. Rep. KRL-3, PICHTR, 1988.
- [56] P. Lee and J. K. Wolf, "Combined error correction/modulation codes," *IEEE Trans. Mag.*, vol. MAG-23, pp. 3681–3683, Sept. 1987.
- [57] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Th.*, vol. IT-28, pp. 55–67, Jan. 1982.
- [58] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE* (Invited Paper), vol. 61, pp. 268–278, Mar. 1973.

- [59] F. Pollara, R. J. McEliece, and K. Abdel-Ghaffar, "Finite-state codes," *IEEE Trans. Inform. Th.*, vol. 34, pp. 1083–1089, Sept. 1988.
- [60] J. K. Wolf and G. Ungerboeck, "Trellis coding for partial-response channels," *IEEE Trans. Comm.*, vol. COM-34, pp. 765–773, Aug. 1986.
- [61] M. W. Marcellin and T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Trans. Comm.*, vol. 38, pp. 82–93, Jan. 1990.
- [62] H. Kobayashi, "Application of probabilistic decoding to digital magnetic recording systems," *IBM J. Res. Dev.*, pp. 64–74, Jan. 1971.
- [63] L. F. Wei, "Trellis-coded modulation with multidimensional constellations," *IEEE Trans. Inform. Th.*, vol. IT-33, pp. 483–501, July 1987.
- [64] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets—Part II: State of the art," *IEEE Commun. Mag.*, vol. 25, pp. 12–21, Feb. 1987.
- [65] A. R. Calderbank and N. J. A. Sloane, "New trellis codes based on lattices and cosets," *IEEE Trans. Inform. Th.*, vol. IT-33, pp. 177–195, Mar. 1987.
- [66] H. C. Ferreira, "On DC free magnetic recording codes generated by finite state machines," *IEEE Trans. Mag.*, vol. MAG-19, pp. 2691–2693, Nov. 1983.
- [67] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets—Part I: Introduction," *IEEE Commun. Mag.*, vol. 25, pp. 5–11, Feb. 1987.