1992

Loop transversal codes

Frank Agrell Hummer

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Mathematics Commons

Recommended Citation
https://lib.dr.iastate.edu/rtd/10004

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
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.
Loop transversal codes

Hummer, Frank Agrell, Ph.D.
Iowa State University, 1992
Loop transversal codes

by

Frank Agrell Hummer

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Mathematics

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1992
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1. INTRODUCTION</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 2. ERROR-CORRECTING CODES</td>
<td>3</td>
</tr>
<tr>
<td>General concepts</td>
<td>3</td>
</tr>
<tr>
<td>Error patterns and decoding</td>
<td>6</td>
</tr>
<tr>
<td>Evaluation of codes</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER 3. LINEAR CODES</td>
<td>17</td>
</tr>
<tr>
<td>CHAPTER 4. EXHAUSTIVE CODE CONSTRUCTIONS</td>
<td>20</td>
</tr>
<tr>
<td>Overview</td>
<td>20</td>
</tr>
<tr>
<td>Code extensions and extended error patterns</td>
<td>21</td>
</tr>
<tr>
<td>Lexicodes and greedy codes</td>
<td>23</td>
</tr>
<tr>
<td>Linearity of binary lexicodes and greedy codes</td>
<td>27</td>
</tr>
<tr>
<td>Loop transversal codes</td>
<td>39</td>
</tr>
<tr>
<td>Construction</td>
<td>40</td>
</tr>
<tr>
<td>Maximality</td>
<td>44</td>
</tr>
<tr>
<td>Binary loop transversal codes and lexicodes</td>
<td>46</td>
</tr>
<tr>
<td>CHAPTER 5. PERFORMANCE OF LOOP TRANSVERSAL CODES</td>
<td>49</td>
</tr>
<tr>
<td>General results</td>
<td>49</td>
</tr>
<tr>
<td>Binary loop transversal codes</td>
<td>49</td>
</tr>
<tr>
<td>Non-binary loop transversal codes</td>
<td>52</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>56</td>
</tr>
<tr>
<td>APPENDIX. FORTRAN SOURCE CODE FOR GREEDY LOOP TRANSVERSAL ALGORITHM</td>
<td>57</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

This paper discusses a class of error-correcting codes called loop transversal codes (henceforth, LT codes) introduced in [5]. These are linear codes constructed with attention to the syndrome, which is constructed in a more or less greedy fashion, the goal being to find optimal or near optimal (linear) codes of arbitrary word length for a given set of error parameters. LT codes are similar in spirit to lexicodes in that they search greedily through sets in an effort to construct "good" codes — codes of a sufficient size. LT codes however can be constructed more efficiently because they search through the (usually) smaller syndrome space rather than search through the code space, as do lexicode constructions.

This comparison of construction efficiency is particularly relevant in those binary cases for which the lexicode construction produces linear codes, for in these cases, as we will show, the corresponding LT code which has a syndrome constructed by a lexicographically greedy algorithm (called the greedy algorithm) is always identical to the lexicode. It has been shown by Conway and Sloane [2] and also by Brualdi and Pless [1] that binary lexicodes are linear when the codes are constructed so as to correct white-noise error patterns. Such white-noise lexicodes are (naturally) constructed by ensuring that an adequate Hamming distance exists between a new codeword "candidate" and each of the words previously admitted to the code. The literature concerning lexicodes is not explicit with regard to methods for constructing lexicodes in non-white-noise cases. In this paper we present a very natural generalization of lexicographic construction which includes both the white-noise and the non-white-noise cases (for both binary and non-binary codes) — a generalization which reduces in the white-noise case to the construction using Hamming distances. This generalization was motivated by a need to construct lexicodes for "unusual" error patterns which could serve as data for comparison to the corresponding LT codes. It is surprising that in the binary case, these non-white-noise lexicodes also turn out to be linear, just as in
white-noise case, and we will show that, given a set of errors having certain weak conditions, we can construct an ad hoc metric which, although it is not likely to provide direct insights into the constructions of codes, can nevertheless serve the same function in every detail as the (white-noise) Hamming metric in the proof by Brualdi and Pless which shows linearity of binary white-noise lexicodes. Thus we show the linearity of all binary lexicodes constructed so as to correct a set of errors containing at least the all-zero "error", and the equality between binary lexicodes and those LT codes constructed by the greedy algorithm is extended from the white-noise case to include the non-white-noise case.

In non-binary cases, lexicodes are not generally linear, and so the LT codes have an advantage in their linearity (as well as in the efficiency of their construction). Data for ternary LT codes suggests that they compare well — in both white-noise and non-white-noise cases — to the best linear codes known and also to lexicodes. The ternary LT code construction by the greedy algorithm for random single and double errors produces the perfect ternary [11, 6, 5] Golay code, a [43, 34, 5] code (1 dimension better than previously known for n=43 and d=5), a [44, 35, 5] code, and a [45, 36, 5] code, each 2 dimensions better than any previously known.
CHAPTER 2. ERROR-CORRECTING CODES

General concepts

Let $Q$ be an alphabet having $q$ distinct symbols. For convenience we will take $Q = \{0, 1, \ldots, q-1\}$ (with $q \geq 2$). A set $C$ is called a $q$-ary block code of length $n$, or just a $q$-ary code of length $n$ if $C \subseteq Q^n$. The elements of $Q^n$ are called words (or $q$-ary words of length $n$) and the elements of $C$ are called codewords (or $q$-ary codewords of length $n$). We will represent words in $Q^n$ as strings of characters, or digits, from $Q$, with the digits numbered from 1 to $n$, going right to left. (The convention of numbering bits from right to left will help us in chapter 4 where we will often associate a string which represents an element in $Q^n$ with the integer having that string as its base-$q$ representation.) For $x \in Q^n$ we will designate the $i$-th character in $x$ as $x^i$. Thus in the word $0110201$, $x^3 = 2$.

The value $M = |C|$ is called the size of the code $C$. If $q$ is a prime or a power of a prime, we can view $Q$ as the field $GF(q)$ (if $q$ is prime, this will be identical to the ring $Z_q$) and we can view $Q^n$ as the vector space $GF(q)^n$ over $GF(q)$. If $q = p^e$ for some prime $p$ and some $e \geq 2$, then we can alternatively view $Q$ as the ring $Z_{q^e}$, and view $Q^n$ as the $Z_{q^e}$-module $\prod_{i=1}^{n} Z_{q^e}$. In any of these cases, scalar multiplication of words (whether the words are elements of a vector space or a module) will be defined on $Q^n$. We may not always require a multiplicative structure, and in any event we can view $Q$ as the abelian group $Z_q$, treating $Q^n$ as the product group $\prod_{i=1}^{n} Z_q$. We will represent addition in $Q^n$ by $\oplus_q$ and subtraction by $\ominus_q$. We will represent scalar multiplication by juxtaposition. Usually we study binary codes, with $q = 2$ and we treat $Q^n$ as the vector space $GF(q)^n$.

In order to discuss error-correcting codes, consider the following diagram:
The code $C$ described earlier, in the context of error-correcting codes, serves within the communication system pictured in figure 1 as a set of messages which can be sent from the sender to the receiver, via the medium of communication called the channel. The channel takes as input not only the intended message but also noise which can cause the message received by the receiver to differ from the message sent. It is hoped that we know something about the types of noise which are likely to occur in the channel, and that given this knowledge, we are able to select our code $C$ in such a way that we can (usually) infer, by inspecting the received word, which codeword was actually sent. To make this inference correctly is to correctly decode the received message.

An **error-correcting block code** then is a code $C = \{c_i\}_{i=1}^{M} \subseteq \mathbb{Q}^n$ together with a collection of subsets of $\mathbb{Q}^n$, $\{W_i\}_{i=1}^{M}$, such that $W_i \cap W_j = \emptyset$ for $i \neq j$, and a **decoding function** $\text{dec} : \bigcup_{i=1}^{M} W_i \rightarrow C$ defined by

$$\text{dec}(w) = c_i \quad \text{if} \quad w \in W_i.$$

Intuitively, if a received word $w$ is to be decoded as the codeword $c_i$, then we want $w$ to be a likely result of distorting $c_i$ by the noise in the channel. We think of such a distortion as a vector addition of $c_i$ and an error vector, $e \in \mathbb{Q}^n$. ($c_i \oplus e = w$) For a given $i \in \{1, 2, ..., M\}$ then, $W_i$ will consist of the set of all sums $c_i \oplus e$ such that $e$ is an error we wish to correct in the event that the message
$c_i$ is sent. Typically in our discussion, if we wish to correct a given error $e$, then we will wish to correct it regardless of which message $c_i$ is sent, so that for all $i,j \in \{1,2,...,M\}$, $c_i \oplus_q e \in W_i \Leftrightarrow c_j \oplus_q e \in W_j$. In practice, this last condition on our specification of the sets $W_i$ will cause $\bigcup_{i=1}^{M} W_i$ to be a proper subset of $Q^n$: that is, there can occur received words $w$ such that $\text{dec}(w)$ is undefined. In this situation we have incomplete decoding. Likewise, if we insist on complete decoding so that $\bigcup_{i=1}^{M} W_i = Q^n$, then it may be the case that we can correct a given error $e$ when certain codewords are sent, but not when other codewords are sent. In any event, given a code $C$ and collection of sets $W_i$ we will call the set $E = \{ e \in Q^n | c_i \oplus_q e \in W_i, \forall i \in \{1,...,M\} \}$ the set of errors corrected by the code $C$. To say then that a code $C$ corrects the errors in the set $E$ is to say that there exists a collection of disjoint sets $\{W_i\}_{i=1}^{M}$ such that $c_i \oplus_q e \subseteq W_i$ for all $i \in \{1,2,...,M\}$. In this case we will say that $C$ is an $E$-correcting code, and the sets $W_i$ may be unspecified, or we may take $W_i$ to be equal to $c_i \oplus_q E$ for each $i$. We will often refer to a set $E$ as an error pattern due to the structure that is usually present among the errors we wish to correct. $C$ is an $E$-correcting code then if $c_1 \oplus_q e_1 \neq c_2 \oplus_q e_2$ for all $c_1, c_2 \in C$, $c_1 \neq c_2$, for all $e_1, e_2 \in E$.

While we have defined errors to be elements of $Q^n$ for a given $n$, it is common, and sometimes convenient, to refer to the differences between individual digits in the message sent and the message received as "errors." Thus if the message sent and the message received differ in three digits, we will often say that three errors have occurred, rather than say that a single error has occurred which contains three non-zero digits. Alternatively, it may be said for the above situation that three digit-errors have occurred, and this is the form we will attempt to use in this paper. The meaning should in any case be clear from the context.

Below is an example of a binary error-correcting code together with the set of errors it corrects:
Note that if we take $W_i = c_i \oplus E$ then $\bigcup_{i=1}^{M} W_i$ is not all of $Q^n$: $| \bigcup_{i=1}^{M} W_i | = M \times 7 = 56$, but $| Q^n | = 64$. For example, if $w = 001100$ then $w \notin \bigcup_{i=1}^{M} W_i$ and it is not clear how we are to decode because none of the codewords differs from this $w$ by any of the error vectors. If we wish to decode this $w$ when it is received, then we can either allow the decoding to be arbitrary (in which case of course it is still true that $C$ corrects the errors in $E$), or if we wish to decode it more “reasonably” we can invoke the principle that error vectors with fewer 1’s are more likely to occur, and we can decode $w$ as either $000000$, $011101$, or as $101110$, assuming that one of the vectors $001100$, $010001$, or $100010$ occurred as an error.

### Error patterns and decoding

As mentioned in the previous section, the set of errors, $E$, that we wish to correct with a code, $C$, will be chosen so as to increase the likelihood of decoding correctly. However, as the sets $W_i$ get larger, the restriction that they must be disjoint allows fewer codewords, and less information can be sent. It is efficient then, in designing an error-correcting code, to chose to correct those elements of $Q^n$ which are most likely to occur as errors. This will require a knowledge of the statistical properties of the channel noise. Channel statistics can present many complications. For example the assumption suggested in the previous section that
we will want to target for correction certain errors regardless of the codeword sent is not realistic in asymmetric channels, in which (take $q = 2$) the probability that a 1 is incorrectly received as a 0 is different from the probability that a 0 is incorrectly received as a 1. We will make the assumption in this paper that errors occur in a way independent of messages — that is, that our channels are symmetric.

Another simplifying assumption that is often made for purposes of code design is that digit-errors occur independently from each other. Such a channel is called a memoryless channel, a white-noise channel, or a random error channel. We will discuss this case in some detail here because certain general concepts in coding theory are most easily understood in the context of white-noise channels.

The assumption that our channel is white-noise has a natural effect upon the selection of a set $E$ of errors to be corrected. Errors are admitted into $E$ on the basis of the number of non-zero digits they contain. Consider the binary case. If an error occurs in any given digit with probability $p (0 \leq p < \frac{1}{2})$, then the probability that an error with a single 1-digit occurs is $(1 - p)^{n-1} p$, and the probability that a particular error with two non-zero digits occurs is $(1 - p)^{n-2} p^2$, which is less than the first probability. If we admit two-digit errors into $E$, then we will want to admit single-digit errors into $E$ because they are more likely. Given any $q$, we define the Hamming weight of a vector $x \in \mathbb{Q}^n$:

$$\text{wt}_H(x) = \left| \left\{ i \mid x^i \neq 0 \right\} \right| .$$

Using Hamming weight, we devise a metric upon $\mathbb{Q}^n$ called the Hamming metric:

$$d_H(x, y) = \text{wt}_H(x \oplus_q y) .$$

It is clear that an alternative definition of Hamming distance is given by $d_H(x, y) = \left| \left\{ i \mid x^i \neq y^i \right\} \right|$. This metric is used extensively in research into coding theory. Indeed, most of the research in coding theory has dealt with the white-noise case, for which this metric is an appropriate tool. If we wish to
correct all errors \( e \) such that \( \text{wt}_H(e) \leq h \) for some \( 0 \leq h \leq n \) (that is, we want \( E = \{ e \mid \text{wt}_H(e) \leq h \} \)), we say that \( E \) is a random-error pattern or a white-noise error pattern. To correct the errors in such an error pattern, it is sufficient to design a code \( C \) such that \( d_H(c_i, c_j) \geq 2h + 1 \) for all \( c_i, c_j \in C \). So the minimum Hamming distance between codewords is an important property of a code for it tells us how many digit-errors it can correct. We define the minimum distance of a code \( C \) as

\[
d_{\text{MIN}}(C) = \min \{ d_H(c, c') \mid c, c' \in C \}.
\]

From the preceding discussion, it is clear that if \( d_{\text{MIN}}(C) = n \), then \( C \) is able to correct all errors of Hamming weight less than or equal to \( \left\lceil \frac{m-1}{2} \right\rceil \).

If \( w \) is a received word, our goal is to decode \( w \) as that element of the code, \( C \), which maximizes the conditional probability

\[
P(c \text{ is the message sent} \mid w \text{ is the message received}).
\]

This is the principle of maximum likelihood decoding, and it reflects the practice, described above, that ideally we will choose our set \( E \) to consist of the set of most likely error vectors. In the white-noise case, maximum likelihood decoding consists of finding a vector \( e \in E \) of smallest possible Hamming weight such that \( w \Theta e \in C \), and then we decode \( w \) as \( w \Theta e \). Because \( d_H(w, w \Theta e) = \text{wt}_H(e) \), this is equivalent to decoding \( w \) as an element of the code \( C \) which is as near as possible (using Hamming distance) to the word \( w \). To decode using distance in this way is to perform minimum distance decoding.

The previous discussion shows that in the white-noise case, maximum likelihood decoding is the same as minimum distance decoding using Hamming distance.

It is not surprising to see a case in which the sets \( W_i \) (\( \supseteq c_i \Theta E \)) can be described essentially in terms of a metric: intuitively, the words which we would like to decode as a given codeword \( c \) should be those which are "closest" to \( c \) in some sense.
The white-noise channel model is often unrealistic. Often the occurrence of an error in digit $w^i$ will affect the probability of an error in digit $w^j$, particularly if these digits are close to each other in $w$. Such a channel we will call a non-white-noise channel. We can imagine something like electrical interference from a storm which, if it caused an error in digit $w^i$, is likely to have persisted long enough to have caused an error in $w^{i+1}$ or $w^{i-1}$. Error vectors in which the non-zero digits are located "near" to each other in the word are called burst errors. More precisely, an $l$-burst error is an error such that the non-zero digits are confined to a region of $l$ consecutive digits, with the first and last digits within the region both being non-zero. $l$ is called the length of the burst. The error vector \[00000401001200000000000\] is a 7-burst error. A code which corrects burst errors of up to length $l$ (and which, by convention, does not correct bursts of greater length) is called an $l$-burst-error-correcting code.

If dependencies among digit-errors exist as described above, it may be reasonable, in our effort to correct errors of the highest probability, to design a code which corrects burst errors of length up to $l$, but which doesn't correct all random errors of Hamming weight up to $l$. In practice codes are designed to correct individual burst errors, or up to some specified number of burst errors, or combinations of random and burst errors. The methods for designing such codes become very sophisticated. We should say here that the notion of Hamming distance does not seem to provide useful insights into the construction of codes for non-white-noise channels. Later however, we will present method by which, given an arbitrary set of errors, $E$, having certain weak conditions, we can construct an ad hoc metric $d_E$ which, although it is not likely to serve as a tool for efficient code constructions, will nevertheless play a role in a proof involving lexicodes. While postponing the presentation of this metric until later, it is appropriate to comment here that such a metric allows us to extend to the non-white-noise cases the intuitive relationship between maximum likelihood decoding and minimum (Hamming) distance decoding which we had in the white-noise case: if $E$ contains
the set of error vectors which are most likely to occur in the channel, then maximum likelihood decoding is equivalent to minimum distance decoding under the metric $d_E$.

A typical assumption that is often made regarding error patterns is that they always contain the all-zero vector. This convention reflects the simple channel property that, regardless of any dependencies among digit-errors, the (unconditional) probability of a 0 in given error vector digit position is always greater than the probability of any non-zero value in that position. Another restriction on error patterns that will sometimes be made is that they are *self-subordinate*: $E$ is self-subordinate if $e \in E$ implies $e' \in E$, where $e'$ is derived from $e$ by replacing any number of non-zero digits of $e$ with zeros. The following error pattern in $\text{GF}(2)^7$ is not self-subordinate:

```
0000000
0000011
0000110
0001100
0011000
0110000
1100000
```

We use the special symbol $B$ to denote the set of all single-digit errors having a 1 in the only non-zero digit position. In later discussions of loop transversal codes we will make the restriction that $B \subseteq E$. In the binary case, if $E$ is self-subordinate then $B \subseteq E$ unless there is a digit position $i$ such that $e^i = 0$ for all $e \in E$. But this later situation would be unusual in most applications because it suggests that there is a digit position for which we have no intention of correcting any errors.
Given that \( Q \) and \( Q^n \) have at least an (additive) abelian group structure, we say that an error pattern \( E \) is closed under negation if \( e \in E \) implies \(-e \in E\) for all \( e \in E \). Given that \( Q \) has a multiplicative operation due to our treatment of it as \( GF(q) \) or \( Z_q \), and that \( Q^n \) has the corresponding vector space or module structure, we say that an error pattern \( E \) is closed under scalar multiplication if \( e \in E \) implies \( \alpha e \in E \) for all \( \alpha \in Q \), for all \( e \in E \).

Besides considering error-correcting codes, we will also consider error-detecting codes. Let \( E \) and \( F \) be subsets (error patterns) of \( Q^n \). We say that a code \( C \subseteq Q^n \) is \( E \)-correcting and \( F \)-detecting if \( C \) is \( E \)-correcting and \((c_1 \oplus_q E) \cap (c_2 \oplus_q F) = \emptyset \) for all \( c_1, c_2 \in C, c_1 \neq c_2 \); that is, if \( C \) is \( E \)-correcting and \( c_1 \oplus_q e \neq c_2 \oplus_q f \) for all \( c_1, c_2 \in C, c_1 \neq c_2 \), for all \( e \in E \), for all \( f \in F \). From the previous discussion of the white-noise case, we can see that a code \( C \) corrects single-digit errors and detects double-digit errors if \( d_{MIN}(C) = 4 \), while \( C \) corrects single-digit errors (that is, \( C \) is \( E \)-correcting where \( E = B \)) if \( d_{MIN}(C) = 3 \).

If \( C \) is an \( F \)-detecting code, then when an error in \( F \) occurs, we can diagnose the fact that an error has occurred without necessarily being able to decode uniquely.

**Evaluation of codes**

It has already been suggested that a major goal in designing an \( E \)-correcting code, \( C \), is to have \( C \) contain a large number of words. Of course if there is a limit on the number of digits we can send in a given period of time, then we will be able to send words more quickly if \( n \) is small. Generally when we consider the problem of finding good codes to correct certain types of errors, we don’t presume that the value \( n \), the length of the codewords, is predetermined in any particular way, and we tend to consider error patterns as being “extended” across the spaces \( Q^n \) for various values of \( n \). For example, in the white-noise case, we will hope to find a good code which will correct enough random errors — more or less regardless of the \( n \) value required to do so — which will allow us to send information with a certain degree of reliability (low probability of error). It turns
out that even though having a larger value of $n$ actually allows (in most cases) a greater opportunity for errors to occur in a word (because there are more places for errors to happen), the combinatorial opportunities for finding more codewords outweighs (in some precise sense) the increased risk of errors. The net effect of all this is that the interests of (1) small $n$ value, (2) many codewords, and (3) protection from errors all compete against each other in the design of codes. It is the precise relation between these interests which we will largely be dealing with in this section. There is a fourth concern, which we will treat more or less separately near the end of the section: (4) efficient decoding. It is apparent that if we demand the elegant combinatorial structures in our codes which will facilitate decoding, we run the risk of limiting our selection of codewords, and so our fourth interest competes with (at least) our second interest.

We should make precise the way in which large $n$ values give us an advantage in communication applications. Shannon's theorem does this in the white noise case. This was proven by C.E. Shannon in 1948, and it launched the fields of Information Theory and Coding Theory. In this paper we are steering away from issues of probability and information theory proper. But we will present just enough here to allow us to understand the statement of this landmark theorem. These basic concepts will also be important to the understanding of codes we will examine later.

The rate of a code of length $n$ which contains $M$ words is $R = \frac{\log_q M}{n}$. The rate of a code essentially tells us what proportion of the digits in a codeword are actually containing "information". In the code presented on page 5, $M=8$, so that with the code we can send 8 different messages, which is the number of messages that can be sent with 3 ($= \log_2 8$) binary digits (with no errors). So it is as if 3 information digits out of the total of 6 actually contain information, for a rate of $\frac{1}{2}$. The other half of the digits are essentially redundant and provide the error correction capability of the code. The redundancy of a code $C$ is defined as $\text{red}(C) = n - \log_q M$. Note that if $M$ is not an integer power of $q$, then $R$ and
red(C) are not integers, but this presents no problems.

The preceding paragraph gives definitions applicable to all error-correcting codes. In what follows the terminology is most relevant to the white-noise case.

For a given $Q$, an $(n, M, d)$ code is a code of length $n$ having $M$ codewords such that $d_{MIN}(C)$ (using Hamming distance) is $d$. $A_q(n, d)$ is defined to be the largest value of $M$ such that a q-ary $(n, M, d)$ code exists. (Usually the subscript $q$ does not appear and $A(n, d)$ is understood to apply to the binary case.) So the largest possible q-ary $(n,*,*d)$ code is an $(n, A_q(n, d), d)$ code; such a code is called optimal.

For a q-ary white-noise channel, we define the entropy function:

$$H_q(p) = p \log_q(q - 1) - p \log_q p - (1 - p) \log_q(1 - p) \quad \text{for } p \in \left[0, \frac{q - 1}{q}\right]$$

$H_q(p)$ is essentially a measure of the proportion of “disorder” that exists in a white-noise q-ary channel in which the probability of an given digit-error is $p$. The capacity of such a channel is given by $1 - H_q(p)$, and this is the portion of the digits in the channel which can carry information. The notion of capacity is made precise by using the probabilistic formulations of information theory, but for our purposes we can think of the capacity of a channel as a ceiling for the rate which can be achieved with a code sent through that channel such that we have any degree of reliability that information is actually arriving at the receiver’s end.

**Theorem 2.1 (Shannon’s theorem):** If $R < 1 - H_q(p)$, then for any $\varepsilon > 0$, and for some $n$ sufficiently large, there exists an $(n, q^k, *)$ code with rate $\frac{k}{n} \geq R$ such that the probability that a decoding error occurs is less than $\varepsilon$.

We shall not present here a proof of this theorem, but its promise that codes are available which will satisfy our need for reliable communication has served as the impetus for the study of error-correcting codes, and within the coding theory community it is held to be one of the greatest ironies that the proof of this important theorem is an “existence” proof — it is non-constructive, and suggests no method for finding these codes.
This theorem makes it clear (in the white-noise case) that the interests of having short \( n \) values and of error correction do compete with each other. Given an achievable rate value, \( R \), we can find a code \( C \) which will provide any degree of reliability in the received message which we desire, as long as we pick \( n \) sufficiently large.

A lower bound exists on the value of \( A_q(n,d) \), to which we will refer when we discuss lexicodes and loop transversal codes.

**Theorem 2.2 (Gilbert bound):** Let \( V_q(n,r) \) equal the number of elements of \( Q^n \) contained in a ball of Hamming radius \( r \) about a word \( w \). Then

\[
A_q(n,d) \geq \frac{q^n}{V_q(n,d-1)} .
\]

**Proof:** Let \( C \) be an optimal \((n, M, d)\) code. Then we must have \( M \cdot V_q(n,d-1) \geq q^n \). Otherwise, the (disjoint) balls of radius \( d-1 \) about the codewords in \( C \) don’t cover \( Q^n \), and so there would be some word \( w \not\in C \) having a distance of at least \( d \) to all of the elements of \( C \).

A code that satisfies the Gilbert bound is called a *good* code. The proof of the Gilbert bound suggests that given any \((n, *, d)\) code, we can always extend it to a maximal code which satisfies the Gilbert bound, merely by successively adding words to the code which have distance at least \( d \) to each word already in the code. Such a code may not be optimal, but it will be maximal with respect to subset inclusion in the power set of \( Q^n \).

Upper bounds for \( A_q(n,d) \) have been extensively studied, but we will consider here only a very simple one:

**Theorem 2.3 (sphere packing bound):**

\[
A_q(n,d) \leq \frac{q^n}{V_q(n, \left\lfloor \frac{d-1}{2} \right\rfloor)} .
\]

**Proof:** If \( C \) is a \( q \)-ary \((n, M, d)\) code, then balls of radius \( \left\lfloor \frac{d-1}{2} \right\rfloor \) are disjoint, so \( M \cdot V_q(n, \left\lfloor \frac{d-1}{2} \right\rfloor) \leq q^n \).
The preceding theorem is easily generalized to the non-white noise case. If $C$ is a $q$-ary $E$-correcting code of length $n$, then $M \leq \frac{q^n}{|E|}$, due to the requirement that the sets $c_i \oplus_q E$ be disjoint.

A perfect code is one which meets the limit set by the sphere packing bound, although this term is generally only used in the white-noise case. In this dissertation, when we refer to codes as being perfect, it will be understood that such codes are perfect with respect to some white-noise error pattern, unless otherwise specified. The following example is a classic perfect binary code which corrects single-digit errors:

```
0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This is an example of a Hamming code. It has been shown that if $n = (q^k - 1)/(q - 1)$, where $q$ is a prime or a power of a prime, then there exists a perfect $(n, q^{n-k}, 3)$ code. These codes are called Hamming codes. A trivial code consisting of a single codeword is always a perfect $(n, 1, n+1)$ code, the code $C = Q^n$ is always a perfect $(n, q^n, 1)$ code, and the binary repetition codes (each containing the two codewords of the form 0000...00 and 1111...11) are all perfect $(n, q, n)$ codes for odd values of $n$. It has been proven that the only perfect codes (in the white-noise case) having parameters $(n, M, d)$ different from these codes are the binary Golay code with parameters $(23, 2^{12}, 7)$ and the ternary Golay code with parameters $(11, 3^6, 5)$. (Other perfect codes exist that are "equivalent" to those described above under, for example, permutations of the alphabet $Q$ or permutations among digit positions, which change the code but do not change the parameters of the code.)

We mentioned earlier that in evaluating codes, we have the additional concern of decoding complexity. Let's consider the white-noise case. In its simplest form,
decoding will involve essentially the following work: given a received word \( w \), we find \( d_H(w, c) \) for all \( c \in C \), and we compare these distances, decoding \( w \) as some element of \( C \) which minimizes the distance. If we consider codes which have rates kept away from 0, then \( M = |C| \) increases exponentially with \( n \), and so does the work of decoding. To see this, note that the work in computing the Hamming distance between \( w \) and \( c \) is \( O(n) \) and that we must do \( |C| = q^nR \) such comparisons, where \( R \) is the rate of the code. Again, that is in the white-noise case. In the general case we can't assume that it is so easy to find the "distance" between \( w \) and a given \( c \in C \); to determine whether or not a given \( c \) was (most probably) the codeword sent when \( w \) was the word received, we essentially must subtract every element of \( E \) from \( w \) and check to see if the difference is a codeword. This work is \( O(|C| \cdot |E|) \), which, in most cases (if the code approaches optimality in size among \( E \)-correcting codes) is roughly \( O(q^n) \).

To reduce decoding complexity, it seems that we must require some kind of structure for our codes which will facilitate decoding. It appears though that to do this will probably diminish either our rate of communication or our error correcting capability. For example there is a class of codes called BCH codes which can be designed to correct \( h \) random errors, and there exist \( O(n) \) algorithms for decoding them. Unfortunately these decoding algorithms do not correct all of the errors correctable by the codes (sub-optimal decoding), and as \( n \) increases, their rates go to 0. The next chapter is devoted largely to linear codes (defined to be vector subspaces of \( Q^n \), where \( Q \) has a field structure), which, although they do not have enough structure to avoid the orders of decoding complexity discussed above, are nevertheless much more efficient to decode within the ranges of \( n \) and \( R \) which typically prevail in a practical setting. And while the class of linear codes doesn't solve our asymptotic decoding problems, it has been shown that the restriction that a code be linear is not so strong a restriction that we can't find linear codes fulfilling the promise of Shannon's theorem and the Gilbert bound.
CHAPTER 3. LINEAR CODES

A linear code is an error-correcting code that is a vector subspace of \( Q^n \), where \( Q \) has a field structure. Note that this will require that \( q \) is prime or a power of a prime. In the examples of linear codes we will see throughout this paper, \( q \) will be prime. Due to the properties of finite vector spaces over finite fields, a linear code \( C \) must have \( M = q^k \) for some integer \( k \), \( 0 \leq k \leq n \), where \( k \) is the dimension of the code. To indicate the parameters of a linear code, we sometimes use an alternative notation: an \([n, k, d]\) code is a linear \((n, q^k, d)\) code.

The rate of a linear code then is equal to its dimension. For a linear code we can quite literally think of a certain collection of \( k \) digits as being the digits which contain the “information” while the remaining \( n - k \) digits provide the redundancy. In fact, without making any compromises in the quality of codes which we can form, we can assume in the white-noise case that, say, the leftmost \( k \) digits are the information digits and that the rightmost \( n-k \) digits are the redundancy digits; it has been shown that permutations of the digit positions for a linear code can always be performed which result in this grouping, and the random error correcting capability of the code is left unchanged. This is not generally true in non-white-noise cases.

A generator matrix for a linear code \( C \) is a matrix whose rows form a basis for \( C \). If \( G \) is a generator matrix for \( C \), and \(|C| = q^k\), then \( G \) is a \( k \times n \) matrix. In the white-noise case, in keeping with the notion of positioning the information digits to the left, \( G \) can be given the form

\[
G = (I_k \mid P).
\]

There exists a corresponding parity check matrix, with dimensions \((n - k) \times n\), having the property that \( GH^T = 0 \). In the white noise case where \( G = (I_k \mid P) \), \( H \) is given by

\[
H = \left( -P^T \mid I_{n-k} \right).
\]
Any codeword in $C$ can be viewed as a linear combination of the rows of $G$. That is, for all $c \in C$, $c = xG$ for some row vector $x \in \mathbb{Q}^k$. Because $GH^T = 0$ we have that for any $c \in C$, $cH^T = 0$, and that if $w = c \oplus_q e$ is a received word, then

$$wH^T = (c \oplus_q e)H^T = cH^T \oplus_q eH^T = eH^T \in \mathbb{Q}^{n-k}.$$ 

The vector $s = wH^T$ is called the *syndrome* of $w$. We will also express this equality by writing $\text{syn}(w) = s$.

The equation above shows then that the syndrome of a received word is the same as the syndrome of the error vector which occurred as noise in the channel. Of course many vectors $e \in \mathbb{Q}^n$ can have the same syndrome, but, given that $s = wH^T$ we decode $w$ as $w \oplus_q e$ where $e$ is an error most likely to occur from among errors having syndrome $s$. Still this choice of $e$ may not be unique if we insist on complete decoding, but if we restrict our decoding function to the domain $C \oplus_q E \equiv \bigcup_{i=1}^M (c_i \oplus_q E)$ and ensure that the syndrome is injective on the set of errors corrected by the code, $C$, then whenever $s = wH^T$ is in the set $\text{syn}(E)$, we will be able to uniquely decode, and the decoding will be correct whenever the error that occurs is actually an element of $E$.

If $\text{syn}(e) = s$, and if $e$ is the particular vector we have chosen to interpret as the channel error whenever the syndrome of the received word is $s$, then $e$ is called a *coset leader*. It is a leader in the probabilistic sense in that it is a vector in $\mathbb{Q}^n$ that is at least as likely as any other vector, having that syndrome, to occur as an error given the channel statistics. The word “coset” refers to the fact that, for any $s$ in the range of the syndrome, $\{w \in \mathbb{Q}^n \mid \text{syn}(w) = s\}$ is a group coset in $\mathbb{Q}^n$ of the subgroup $C$. If $C$ is an $E$-correcting code, then $E$ will be contained in the set of coset leaders.

Having a linear code gives us an advantage in decoding in the general case (where we don’t presume white noise). In order to decode, we need only store the $q^{n-k}$ possible syndrome values and their associated coset leaders. This is far
better than $O(q^n)$, although if we restrict ourselves to codes with rate bounded away from 0 or 1, this is still exponential in $n$.

In the white-noise case, if $k > \frac{n}{2}$ then $q^{n-k}$ is less than $q^k$ ($= q^n$) and so linearity of the code gives us an advantage in decoding (Compare this with the analysis at the end of chapter 2.) However, if $k < \frac{n}{2}$ then it may be cheaper (we are still considering the white-noise case) to store the $q^k$ codewords and compare distances between the received word and the codewords as discussed near the end of chapter 2.

An important subclass of the linear codes is the class of cyclic codes. A code $C$ is cyclic if it is linear and if every wrap-around shift of a codeword in $C$ is also a codeword in $C$. That is, $c^u c^{u-1} c^{u-2} \ldots c^2 c^1 \in C \Rightarrow c^1 c^u c^{u-1} \ldots c^3 c^2 \in C$. The additional structure of cyclicity gives cyclic codes some advantage with respect to decoding complexity. Also cyclic codes are important because good burst-error correcting codes are often most easily found among the cyclic codes. Later in the paper we will be comparing some burst-error correcting loop transversal codes to some cyclic codes.
CHAPTER 4. EXHAUSTIVE CODE CONSTRUCTIONS

Overview

With advancements made in computer technology in recent years, coding theorists have anticipated that in the future there will be less importance placed on efficient decoding algorithms and sophisticated code structures, and more emphasis placed on finding optimal or near optimal codes. This is evident in research that has been done concerning lexicodes, greedy codes, and loop transversal codes. These codes have in common an approach toward code construction which doesn’t involve searching for elegant code structures, but which instead seeks to admit new codewords into a code through a more or less exhaustive search using greedy algorithms. Lexicodes and greedy codes result from looking through the space $\mathbb{Q}^n$ in search of codewords, while loop transversal codes attempt to greedily build the syndrome function with domain $E$, the set of errors. The goal of the loop transversal syndrome construction is to keep the redundancy of the code small, thus increasing the size of the code.

In spite of the fact that their constructions don’t presuppose linearity, lexicodes and greedy codes turn out to be linear in the binary case. This linearity has generally been recognized only in the binary white-noise case, but in this chapter we will extend this result to show that all binary lexicodes and greedy codes constructed so as to be $E$-correcting codes, for any error pattern $E$ containing the $0$ vector, are linear. This will require a natural generalization of lexicode and greedy code constructions to non-white-noise cases. Loop transversal codes on the other hand are constructed so as to be linear, yet this restriction apparently is not so strong as to prevent them from competing well, in terms of codesize, with lexicodes and greedy codes. In the binary case, it will be shown that a binary loop transversal code having a syndrome constructed by a particular greedy algorithm, for a given error pattern $E$ which is required only to be self-subordinate and to
contain the set $B$ of single-digit errors, is identical to the lexicode constructed so as to correct the same error pattern $E$. The loop transversal codes have the advantage that their construction is much more efficient than the lexicode construction, and the syndrome decoding algorithm is given automatically by the loop transversal construction.

In the non-binary case, lexicodes (and greedy codes) are not generally linear, and so non-binary loop transversal codes have a clear advantage in their linearity. Also, they seem to compare well to lexicodes with regard to size. The ternary random double error loop transversal codes, constructed by the greedy syndrome construction algorithm, include the perfect ternary Golay code, and several record-breaking codes.

**Code extensions and extended error patterns**

In our discussions of codes in the rest of this paper, we will largely be concerned with the possibility of *extending* a given code $C \in Q^n$ by considering it also as a code in $Q^{n+1}$ where (in keeping with our convention of numbering digits from right to left) if $c^n c^{n-1} \ldots c^1 = c \in C \subseteq Q^n$, then $c$ is identified with $0 c^n c^{n-1} \ldots c^1 \in Q^{n+1}$, and by considering codes in $Q^{n+1}$ which contain $C$. If $C$ is an $E$-correcting code in $Q^n$ (where $E \subseteq Q^n$) then it is obvious that $C$ is also an $E$-correcting code in $Q^{n+1}$, if we make it clear that we are identifying the set $E \subseteq Q^n$ with its counterpart in $Q^{n+1}$ in the natural way as we did with $C$. However when we consider $C$ in $Q^{n+1}$ we will often be concerned with extending $C$ to correct a set of errors $E'$ where $E \subseteq E' \subseteq Q^{n+1}$. It is not clear that we can always do this, or that even $C$ by itself is capable of correcting the errors in $E'$. In what follows we will give a necessary and sufficient condition on $E'$ that will allow an $E'$-correcting extension of the code $C$. First we will establish some definitions.

We will say that $E$ is an *extended error pattern* if $E = \bigcup_{n=1}^{\infty} E_n$ where for each $n$, $E_n$ is an error pattern in $Q^n$ and $E_n \subseteq E_{n'}$ for $n \leq n'$. If $E$ is an extended
error pattern and $C \subseteq Q^n$ we will say that $C$ is an $E$-correcting code if $C$ is an $E_u$-correcting code, where $E_u = E \cap Q^n$. We will say that $C'$ is an $E$-correcting extension of the code $C$ (or just that $C'$ is an extension of $C$ if $E$ is understood from the context) if $C \subseteq Q^n$, $C' \subseteq Q^p$ for some $p \geq u$, $C \subseteq C'$, and $C'$ is an $E$-correcting code.

The motivation behind these definitions is that we would like to consider an extended error pattern $E$, such as the set of all random double errors for any word length, and we hope to build a nested series of codes $C_0 \subseteq C_1 \subseteq C_2 \subseteq C_3 \ldots$ such that $C_u \subseteq Q^n$ (and such that, typically, $C_u \not\subseteq Q^{u-1}$) and each $C_u$ is an $E$-correcting code. Now it is not generally true that if $C_u$ is $E$-correcting (that is, if $C_u$ corrects $E_u$) then $C_u$, considered as a code in $Q^p$ for some $p \geq u$, is $E_p$-correcting. This inability to "nest" codes occurs in the following example which uses the extended error pattern $E$ given below.

\[
\begin{align*}
0 \\
1 \\
10 \\
11 \\
100 \\
101 \\
110 \\
1000 \\
10001 \\
10010 \\
10011 \\
11100 \\
11101 \\
11110
\end{align*}
\]

The code $C_4 = \{0000, 1111\}$ corrects $E_4 = \{0, 1, 10, 11, 100, 101, 110, 1000\}$ ($= \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 1000\}$), but does not correct $E_3$ as we can see from the equation $00000 \oplus_2 11100 = 01111 \oplus_2 10011$. 
A condition on an extended error pattern $E$ that is sufficient for the sort of code nesting we would like is given in the following theorem. Remember that $E$ is self-subordinate if $e \in E \Rightarrow e' \in E$, where $e'$ is derived from $e$ by replacing some number of its non-zero digits with zeros.

**Proposition 4.1:** Let $E$ be an extended error pattern. If $E$ is self-subordinate, and if $C_u$ is an $E_u$-correcting code, then it is also an $E_p$-correcting code for $p \geq n$.

**Proof:** Let $C_u$ correct $E_u$. We will suppose that $C_u$ does not correct $E_p$ for some $p \geq n$, and conclude that $E$ is not self-subordinate.

Under our supposition, $\exists c_1, c_2 \in C_u$ such that $c_1 \neq c_2$ and such that $c_1 \oplus_q e_1 = c_2 \oplus_q e_2$ for some $e_1, e_2 \in E_p$. (Actually, under the assumption that $C_u$ is $E$-correcting, it must be that either $c_1$ or $c_2$ is in $E_p - E_u$. ) This implies that the $n$ right-most digits of $c_1 \oplus_q e_1$ and $c_2 \oplus_q e_2$ agree. Because $C_u$ is $E_n$-correcting, it must be that $e'_1$ and $e'_2$, which are derived from $e_1$ and $e_2$ by having their non-zero digits to the left of the $n$-th digit changed to zero, are not in $E_u$. But that is to say that $E$ is not self-subordinate.

The self-subordinate property is a very useful one: the requirement that an error pattern be self-subordinate usually makes it unnecessary to specify, in talking about an $E$-correcting code $C$, whether the error pattern $E$ is extended or not. The reasons for this are apparent from the previous theorem.

**Lexicodes and greedy codes**

Lexicodes gained attention when they were discussed in a paper by Conway and Sloane in 1986 [2]. I have found mention of them in an exercise given by van Lint in [6] (page 41) in 1982. Because they have so far been presented in the literature only in the white-noise case (not necessarily binary), that is the context in which we will first discuss them here.

Let $E$ be a white-noise error pattern in $Q^h$ consisting of words of Hamming weight $h$. An $E$-correcting code of length $n$ must have minimum Hamming
distance of at least $2h+1$. The lexicode of minimum Hamming distance $2h+1$ is constructed by considering the elements of $\mathbb{Q}^n$ in lexicographic order, beginning with $00000...00$, then $00000...01$, etc., and we allow a new word $x$ into the code if and only if $d_H(x, c) \geq 2h + 1$ for all words $c$ that have already been admitted into the code. The all-zero word then is the first word admitted into the code. This construction ensures that, if $L$ is the resulting lexicode, then $d_{MIN}(L) = 2h + 1$.

Below is an example with $q=3$, $h=2$, $n=7$:

\[
\begin{align*}
0 & 0 0 0 0 0 0 \\
0 & 0 1 1 1 1 1 \\
0 & 0 2 2 2 2 2 \\
1 & 1 0 0 1 1 2 \\
1 & 1 1 1 0 2 0 \\
1 & 2 0 1 2 0 1 \\
2 & 1 2 2 0 0 1 \\
2 & 2 0 2 1 2 0
\end{align*}
\]

We can generalize this slightly so as to consider lexicodes of even minimum distance. That is, we can construct a lexicode $L$ with $d_{MIN}(L) = 2h$ so that $L$ is an $(h-1)$-error-correcting $h$-error-detecting code. In such a case it may difficult to simply say that we have actually constructed the lexicode so as to correct an error set $E$ for any particular $E$; the construction is performed with respect to Hamming distance only. If, as in the ternary example above, a lexicode $L$ was constructed strictly with regard to a given error pattern $E$, then we will say that $L$ is the lexicode constructed about $E$.

In the white-noise case, the extended error pattern of errors having up to a certain fixed Hamming weight is always self-subordinate. In building a lexicode in this case it makes no difference whether we perform our construction using a predetermined wordlength or if we instead approach the task as though we are building a code of arbitrarily long wordlength which corrects the extended error pattern. If we consider the words lexicographically, then, according to our
conventions with regard to digit-numbering, we will consider all of the words in \( \mathbb{Q}^n \) for admission into the code before we consider any words in \( \mathbb{Q}^{n+1} \), and we can stop when we like.

A lexicode is **unrestricted** if the code is built, as we have described, by considering all possible words in \( \mathbb{Q}^n \). A lexicode is **restricted** if instead we only consider, from the start, some proper subset of words in \( \mathbb{Q}^n \) to search through lexicographically. For example, in [2] Conway and Sloane discuss the construction of constant weight codes by only searching lexicographically only through the set of words having a certain Hamming weight. In what follows, wherever we don’t specify as to restricted or unrestricted lexicodes, we mean to imply that the lexicodes are unrestricted.

In the white-noise case, lexicodes always satisfy the Gilbert bound. This becomes clear upon reflection of the proof of the Gilbert bound. Any maximal code in the white-noise case satisfies that bound, and lexicodes are constructed so as to be maximal. In fact, as Conway and Sloane report, lexicodes seem to do well with regard to codesize. In particular, lexicode construction in the binary white-noise case, with \( d_{MIN} = 3 \), generates all of the perfect Hamming codes and the perfect \((23, 12, 7)\) Golay code, as well as other well known codes.

As we said earlier, lexicodes have only been discussed in the context of white-noise cases and the Hamming metric. Here we will consider a more general lexicode construction for non-white-noise error patterns. Because we do not yet have available a metric such that \( d_{MIN}(C) \) with regard to that metric corresponds to error-correcting capability for non-white-noise error patterns (although one will be presented later), we will use a more explicit criterion for admitting new codewords into the lexicode.

Let \( E \) be an (arbitrary) error pattern in \( \mathbb{Q}^n \). The lexicode constructed about \( E \) is formed by considering the elements of \( \mathbb{Q}^n \) in lexicographic order, and we allow a new word \( x \) into the code if and only if \( x \oplus_q e_1 \neq c \oplus_q e_2 \) for all words
c that have already been admitted into the code for all \( e_1, e_2 \in E \). The all-zero word then is the first word admitted into the code. For example, the following is a binary 2-burst-error-correcting lexicode:

| 0 0 0 0 0 0 0 0 | 1 0 0 0 1 0 0 1 |
| 0 0 0 1 0 1 0 1 | 1 0 0 1 1 1 0 0 |
| 0 0 1 0 1 0 1 0 | 1 0 0 1 0 0 1 1 |
| 0 0 1 1 1 1 1 1 | 1 0 0 1 0 1 1 0 |
| 0 1 0 0 0 0 1 1 | 1 1 0 0 0 1 1 0 |
| 0 1 0 0 1 0 0 1 | 1 1 0 0 1 1 0 1 |
| 0 1 0 1 1 1 0 0 | 1 1 0 1 0 0 1 0 |
| 0 1 0 1 1 1 0 0 | 1 1 0 1 1 0 0 1 |

We should note that the binary 2-burst-error pattern is understood, by the conventions in terminology discussed in chapter 2, to include the single-digit errors, and that this error pattern is self-subordinate. However there is no problem in constructing a lexicode about \( E \) when \( E \) is not self-subordinate, as long as we understand that we cannot do this by first constructing \( C_1 \) about \( E_1 \), then \( C_2 \) about \( E_2 \), etc., without risking the possibility that we will have to remove codewords from our code at some point, in which case it is not clear how to proceed. If \( E \) is not self-subordinate, and we want to construct a lexicode \( C_n \subseteq Q^n \) for some \( n \), then we take \( E_n = E \cap (Q^n) \) as our error pattern and construct the lexicode as described above, and the construction is not problematic.

Greedy codes are a generalization of lexcodes discussed (for the binary case) by Brualdi and Pless in [1]. Greedy codes are constructed by successively considering words in \( Q^n \) (in the paper [1], \( q=2 \)) for inclusion in a code, but words are not necessarily considered in their usual lexicographic order. Rather, an ordered basis \( B = \{y_1, y_2, ..., y_n\} \) for \( Q^n \) is determined, where \( y_i < y_j \) in the \( B \)-order if \( i < j \), and this ordered basis induces an order on \( Q^n \). Specifically, the order on \( Q^n \) is induced as follows. Let \( Y \) be the matrix whose \( i \)-th row is
$y_{(i+1)-i}$, and let $v$ and $w$ be words in $Q^n$. Then the word that is the matrix product $vY$ comes before the word $wY$ in the $B$-order if $v$ comes before $w$ lexicographically (that is if $v$ is less than $w$ when $v$ and $w$ are identified as base 2 representations of integers). It is with respect to this induced order that words are considered for inclusion into the code. The greedy code construction which uses the lexicographically ordered standard basis $\{000...0001, 000...0010, 000...0100, ..., 100...0000\}$ produces a lexicode. We will not deal extensively with the special properties of greedy codes as contrasted with lexicodes.

**Linearity of binary lexicodes and greedy codes**

Conway and Sloane establish in [2] a correspondence between lexicodes and heap games, objects of study in combinatorial game theory. We will not explain the correspondence in detail here, but heap games use turning sets, and it is shown that the words in a q-ary lexicode correspond to winning moves in a heap game. Conway and Sloane prove, using game theory, that in the binary case, lexicodes defined via this correspondence by any family of turning sets are are linear. Now, we have discussed lexicode construction with regard to white-noise error patterns and Hamming distance, and it is not clear how we can in general apply the result involving game theory to determine exactly when a binary $E$-correcting lexicode will be linear. However, Conway and Sloane show that white-noise lexicographic constructions correspond to a particular class of heap games, called Grundy games, so we can infer that binary white-noise lexicodes are linear, where this includes lexicodes constructed for both odd and even-valued minimum Hamming distances.

Brualdi and Pless [1] give an alternative proof of the linearity of binary greedy codes (which includes the binary lexicodes as a special case) in the white-noise case which makes no use of game theory. The proof is given for greedy codes constructed so as to maintain a minimum Hamming distance for the code. We will next show (constructively) that, if $Q$ has an (additive) group structure, then given any q-ary error pattern $E \subseteq Q^n$ such that $0 \in E$ and $E$ is closed under negation,
there exists a metric $d_E$ on $Q^n$ such that constructing a greedy code about $E$ is the same as constructing a greedy code so as to maintain a minimum distance for the code, where distance is according to the metric $d_E$. Also, given the further hypothesis that we have an error pattern $F \subseteq Q^n$ closed under negation, we will show that there exists a metric $d_{E,F}$ such that constructing a greedy code so as to be $E$-correcting and $F$-detecting is the same as constructing a greedy code so as to maintain a minimum distance for the code, where distance is according to the metric $d_{E,F}$. After that we will show that each of these metrics can replace the Hamming metric $d_H$ in the proof by Brualdi and Pless which show the linearity of binary greedy codes. We will conclude that any binary greedy code constructed about an error pattern $E$ such that $0 \in E$ is a linear code, and that, also, binary greedy codes constructed so as to be $E$-correcting and $F$-detecting (for some $E \subseteq Q^n$ and $F \subseteq Q^n$, with $0 \in E$) are linear. (All binary error patterns are automatically closed under negation.)

**Theorem 4.1:** Let $Q$ have an additive group structure. Let $E$ be a $q$-ary error pattern in $Q^n$ such that $0 \not\in E$ and $E$ is closed under negation. Let $A = E \ominus_q E = \{e_1 \ominus_q e_2 \mid e_1, e_2 \in E\}$. Then (1) the function $d_E : Q^n \times Q^n \rightarrow \{0, 1, 2, 3\}$ defined by

$$d_E(x, y) = \begin{cases} 
0 & \text{if } x \ominus_q y = 0 \\
1 & \text{if } x \ominus_q y \in E - \{0\} \\
2 & \text{if } x \ominus_q y \in A - E \\
3 & \text{if } x \ominus_q y \not\in A
\end{cases}$$

is a metric on $Q^n$, and (2) $C$ is an $E$-correcting code iff $d_{E,\text{int}}(C) = 3$.

**Proof:** (1) Note that $0 \in E$ implies that $E \subseteq A$ and that the sets $\{0\}, E - \{0\}, A - E,$ and $Q^n - A$ form a partition of $Q^n$ so that $d_E$ is well defined. It is clear that $d_E(x, y) \geq 0$ for all $x, y \in Q^n$, and that $d_E(x, y) = 0$ iff $x = y$.

To see that $d_E$ is symmetric, we note that $d_E(x, y)$ and $d_E(y, x)$ are determined, respectively, by the locations of $x \ominus_q y$ and $y \ominus_q x$ within our partition of $Q^n$, but these expressions are negatives of each other. The fact that all four sets in our partition are closed under negation gives us the result.
Not surprisingly the triangle inequality is the most tedious. We must show that \( d_E(x, y) + d_E(y, z) \geq d_E(x, z) \) for all \( x, y, z \in Q^n \).

Case 1. \( d_E(x, z) = 0 \). The proof is trivial.

Case 2. \( d_E(x, z) = 1 \). The triangle inequality can fail only if both terms on the left side are 0, which is impossible because then we have \( d_E(x, z) = 0 \), so the inequality can’t fail in case 2.

Case 3. \( d_E(x, z) = 2 \). The triangle inequality can fail only if at least one of the terms on the left side is 0. But in that case the left side and right side are equal, so the inequality can’t fail in case 3.

Case 4. \( d_E(x, z) = 3 \). The triangle inequality can fail only if \( \min\{d_E(x, y), d_E(y, z)\} \leq 1 \). If \( \min\{d_E(x, y), d_E(y, z)\} = 0 \), then either \( x = y \) or \( y = z \); in either case, the inequality holds. If \( \min\{d_E(x, y), d_E(y, z)\} = 1 \) then the triangle inequality is false only if \( d_E(x, y) = d_E(y, z) = 1 \). This equation says that \( x \otimes_q y, y \otimes_q z \in E - \{0\} \). But

\[
x \otimes_q z = (x \otimes_q y) \oplus_q (y \otimes_q z) \in A,
\]

so \( d_E(x, z) < 3 \) and the triangle inequality can’t fail in case 4.

(2) Suppose \( d_{E_{\min}}(C) < 3 \). Then for some \( c_1, c_2 \in C \) such that \( c_1 \neq c_2 \), \( d_E(c_1, c_2) < 3 \), so \( c_1 \otimes_q c_2 \in A \). This implies that \( c_1 \otimes_q c_2 = e_1 \oplus_q e_2 \) for some \( e_1, e_2 \in E \). Therefore we have \( c_1 \otimes_q e_1 = c_2 \otimes_q e_2 \), but because \( E \) is closed under negation, we have \( -e_1 \in E \), and the equation \( c_1 \otimes_q (-e_1) = c_2 \otimes_q e_2 \) tells us that \( C \) is not an \( E \)-correcting code.

Likewise if \( C \) is not an \( E \)-correcting code, then there exist \( c_1, c_2 \in C \) such that \( c_1 \neq c_2 \) and \( c_1 \otimes_q e_1 = c_2 \otimes_q e_2 \) for some \( e_1, e_2 \in E \). Therefore \( c_1 \otimes_q c_2 = e_2 \otimes_q e_1 \in A \) (using again the observation that \( E \) is closed under negation), so that \( d_E(c_1, c_2) < 3 \), and \( d_{E_{\min}}(C) < 3 \).

**Theorem 4.2:** Let \( Q \) have an additive group structure. Let \( E \) be a q-ary error pattern in \( Q^n \) such that \( 0 \in E \) and \( E \) is closed under negation. Let \( F \)
be a q-ary error pattern in $Q^n$ closed under negation. Then (1) the function $d_{E,F} : Q^n \times Q^n \rightarrow \{0, 1, 2, 3\}$ defined by

$$d_E(x,y) = \begin{cases} 
0 & \text{if } x \Theta_q y = 0 \\
1 & \text{if } x \Theta_q y \in E - \{0\} \\
2 & \text{if } x \Theta_q y \in [(E \cup F) \Theta_q E] - E \\
3 & \text{if } x \Theta_q y \notin [(E \cup F) \Theta_q E]
\end{cases}$$

is a metric on $Q^n$, and (2) $C$ is an $E$-correcting, $F$-detecting code iff $d_{E,F_{mix}}(C) = 3$.

**Proof:** (1) Note that $0 \in E$ implies that $E \subseteq [(E \cup F) \Theta_q E]$ and that the sets $\{0\}, E - \{0\}, [(E \cup F) \Theta_q E] - E,$ and $Q^n - [(E \cup F) \Theta_q E]$ form a partition of $Q^n$ so that $d_{E,F}$ is well defined. It is clear that $d_{E,F}(x,y) \geq 0$ for all $x,y \in Q^n$, and that $d_{E,F}(x,y) = 0$ iff $x = y$.

To see that $d_{E,F}$ is symmetric, we note that $d_{E,F}(x,y)$ and $d_{E,F}(y,x)$ are determined, respectively, by the locations of $x \Theta_q y$ and $y \Theta_q x$ within our partition of $Q^n$, but these expressions are negatives of each other. The fact that all four sets in our partition are closed under negation gives us the result.

It remains to show that $d_{E,F}(x,z) + d_{E,F}(y,z) \geq d_{E,F}(x,y)$ for all $x,y,z \in Q^n$.

Case 1. $d_{E,F}(x,z) = 0$. The proof is trivial.

Case 2. $d_{E,F}(x,z) = 1$. The triangle inequality can fail only if both terms on the left side are $0$, which is impossible because then we have $d_{E,F}(x,z) = 0$, so the inequality can't fail in case 2.

Case 3. $d_{E,F}(x,z) = 2$. The triangle inequality can fail only if at least one of the terms on the left side is $0$. But in that case the left side and right side are equal, so the inequality can't fail in case 3.

Case 4. $d_{E,F}(x,z) = 3$. The triangle inequality can fail only if $\min\{d_{E,F}(x,y), d_{E,F}(y,z)\} \leq 1$. If $\min\{d_{E,F}(x,y), d_{E,F}(y,z)\} = 0$, then either $x = y$ or $y = z$; in either case, the inequality holds. If $\min\{d_{E,F}(x,y), d_{E,F}(y,z)\} = 1$ then the triangle inequality is false only if
\[ d_E(x, y) = d_E(y, z) = 1. \] This equation says that \( x \ominus_q y, y \ominus_q z \in E - \{0\} \). But then
\[ x \ominus_q z = (x \ominus_q y) \ominus_q (y \ominus_q z) \in E \ominus_q E \subseteq (E \cup F) \ominus_q E, \]
so \( d_{E,F}(x, z) < 3 \) and the triangle inequality can't fail in case 4.

(2) Suppose \( d_{E,F,\text{mut}}(C) < 3 \). Then for some \( c_1, c_2 \in C \) such that \( c_1 \neq c_2 \), \( d_{E,F}(c_1, c_2) < 3 \), so \( c_1 \ominus_q c_2 \in (E \cup F) \ominus_q E \). This implies that either \( c_1 \ominus_q c_2 = e_1 \ominus_q e_2 \) for some \( e_1, e_2 \in E \) or that \( c_1 \ominus_q c_2 = e \ominus_q f \) for some \( e \in E, f \in F \). In the first case we have \( c_1 \ominus_q e_1 = c_2 \ominus_q e_2 \), but because \( E \) is closed under negation, we have \( -e_1 \in E \), and the equation \( c_1 \ominus_q (-e_1) = c_2 \ominus_q e_2 \) tells us that \( C \) is not an \( E \)-correcting code. In the second case we have \( c_1 \ominus_q f = c_2 \ominus_q e \), but because \( F \) is closed under negation, we have \( -f \in F \), and the equation \( c_1 \ominus_q (-f) = c_2 \ominus_q e \) tells us that \( C \) is not an \( E \)-correcting, \( F \)-detecting code.

Likewise if \( C \) is not an \( E \)-correcting, \( F \)-detecting code, then there exist \( c_1, c_2 \in C \) such that \( c_1 \neq c_2 \) and such that either \( c_1 \ominus_q e_1 = c_2 \ominus_q e_2 \) for some \( e_1, e_2 \in E \) or that \( c_1 \ominus_q f = c_2 \ominus_q e \) for some \( e \in E, f \in F \). Therefore, in the first case,
\[ c_1 \ominus_q c_2 = e_2 \ominus_q e_1 \in E \ominus_q E \subseteq (E \cup F) \ominus_q E \]
only or, in the second case,
\[ c_1 \ominus_q c_2 = e \ominus_q f \in E \ominus_q F \subseteq (E \cup F) \ominus_q E \]
(assuming again the observation that \( E \) and \( F \) are closed under negation), so that in either case \( d_{E,F}(c_1, c_2) < 3 \), and \( d_{E,F,\text{mut}}(C) < 3 \). ■

Remark: The metric \( d_{E,F} \) does not have the property that maintaining some minimum distance with regard to \( d_{E,F} \) is necessary and sufficient for obtaining an \( E \)-correcting code, unless we explicitly take \( F \) to be the empty set, in which case it is clear from the definitions of these metrics that \( d_{E,F} = d_E \).

The generalization of the sphere packing bound given on page 14 following the proof of the (white noise) sphere packing bound can be understood in terms...
of the metric $d_E$: the minimum distance of an $E$-correcting code is $d_{E_{\text{mix}}}(C) = 3$, and the number of elements in a ball of $d_E$ radius $\left\lfloor \frac{3-1}{2} \right\rfloor = \left\lfloor \frac{2}{2} \right\rfloor$ is $|E|$. Similarly we can apply this new metric to get a generalization of the Gilbert bound:

**Proposition 4.2 (generalized Gilbert bound):** Let $Q$ have an additive group structure. If $E$ is a $q$-ary error pattern such that $0 \in E$ and $E$ is closed under negation, then there exists an $E$-correcting code having at least $\frac{q^n}{|E \oplus_q E|}$ words.

**Proof:** Under the hypotheses of the proposition, the function $d_E$, as defined earlier, is a metric. The number of elements in any ball of radius 2 (under the metric $d_E$) is $|E \oplus_q E|$. Suppose an optimal code $C \subseteq Q^n$ has $M$ words with $M < \frac{q^n}{|E \oplus_q E|}$. Then the balls of radius 2 about the elements of $C$ do not cover all of $Q^n$. Therefore there exists $x \in Q^n$ such that $d_E(x, c) = 3$ for all $c \in C$, implying that $C \cup \{x\}$ is an $E$-correcting code. Therefore $C$ is not an optimal $E$-correcting code and we have a contradiction. □

We should note that in the binary case, we can replace the symbol $\oplus_q$ appearing in the definition of the metrics $d_E$ and $d_{E,F}$ by the symbol $\oplus_2$, and that we do not need to explicitly state the conditions that $E$ and $F$ are closed under negation.

From Theorems 4.1 and 4.2, we can see that to construct a greedy code about an error pattern $E$ (having the necessary conditions) is to construct a greedy code with $d_{E_{\text{mix}}} = 3$. Similarly, to construct a greedy code so as to be $E$-correcting and $F$-detecting is to construct a greedy code with $d_{E,F_{\text{mix}}} = 3$. To show that we can substitute $d_E$ or $d_{E,F}$ in place of the Hamming metric $d_H$ in the proof by Brualdi and Pless that binary lexicoes constructed according to the metric $d_H$ are linear, we will present a proof that is essentially identical to that in [1], with the only differences being due to the replacement of the metric. This replacement of the Hamming metric will be justified by Lemmas 4.1 and 4.2 which follow.

**Lemma 4.1:** Let $C_i$ be a binary linear code in $Q^i (Q = \{0, 1\})$. Index the cosets with superscripts, taken from $Q'^n$ for some $m$, as follows: $C_i^0 = C_i$ and
Let \( C_i^\beta \oplus_2 C_i^\gamma = C_i^{\alpha \oplus_2 \beta} \), where \( \oplus_2 \) between cosets represents coset addition in the (additive) quotient group \( Q^i / C_i \). Let \( y_{i+1} \in Q^{i+1} \setminus Q^i \). Let \( z \in C_i^\alpha \). Let \( E \) be an error pattern in \( Q^{i+1} \) containing 0, and let \( d_E \) be the metric defined in Theorem 4.1. Then

\[
d_E \left( y_{i+1} \oplus_2 z, C_i^\beta \right) = d_E \left( y_{i+1}, z \oplus_2 C_i^\beta \right) = d_E \left( y_{i+1}, C_i^{\alpha \oplus_2 \beta} \right) = d_E \left( y_{i+1} \oplus_2 C_i^\alpha, C_i^\beta \right).
\]

**Proof:**

\[
d_E \left( y_{i+1} \oplus_2 z, C_i^\beta \right) = \min \left\{ d_E (y_{i+1} \oplus_2 z, x) \mid x \in C_i^\beta \right\}.
\]

The value on the right is determined by the set \( S_1 = \left\{ (y_{i+1} \oplus_2 z) \oplus_2 x \mid x \in C_i^\beta \right\} \) and the location of its elements in the partition consisting of \( \{0\} \), \( E \setminus \{0\} \), \( A \setminus E \), and \( Q^{i+1} \setminus A \).

\[
d_E \left( y_{i+1}, z \oplus_2 C_i^\beta \right) = \min \left\{ d_E (y_{i+1}, z \oplus_2 x) \mid x \in C_i^\beta \right\}.
\]

The value on the right is determined by the set \( S_2 = \left\{ y_{i+1} \oplus_2 (z \oplus_2 x) \mid x \in C_i^\beta \right\} \) and the location of its elements in the partition given above. But \( S_1 = S_2 \), and so we have established the first of the four equalities expressed in the claim. The second and third equalities hold because \( z \oplus_2 C_i^\beta = C_i^\alpha \oplus_2 C_i^\beta = C_i^{\alpha \oplus_2 \beta} \), which follows simply from the properties of coset addition. The fourth equality expressed in the claim is proven in a manner similar to the way in which the first equality was proven; for each side of the equality, the distance between the sets is determined by a set of distances which is in turn determined by the location (within the partition established by the metric) of the elements of identical subsets of \( Q^{i+1} \).

**Lemma 4.2:** Let \( C_i \) be a binary linear code in \( Q^i \) (\( Q = \{0, 1\} \)). Index the cosets with superscripts, taken from \( Q^m \) for some \( m \), as follows: \( C_i^0 = C_i \) and \( C_i^\alpha \oplus_2 C_i^\beta = C_i^{\alpha \oplus_2 \beta} \), where \( \oplus_2 \) between cosets represents coset addition in the (additive) quotient group \( Q^i / C_i \). Let \( y_{i+1} \in Q^{i+1} \setminus Q^i \). Let \( z \in C_i^\alpha \). Let \( E \) be an error pattern in \( Q^{i+1} \) containing 0, let \( F \) be any error pattern in \( Q^{i+1} \), and let \( d_{E,F} \) be the metric defined in Theorem 4.2. Then

\[
d_{E,F} \left( y_{i+1} \oplus_2 z, C_i^\beta \right) = d_{E,F} \left( y_{i+1}, z \oplus_2 C_i^\beta \right) = d_{E,F} \left( y_{i+1}, C_i^{\alpha \oplus_2 \beta} \right) = d_{E,F} \left( y_{i+1} \oplus_2 C_i^\alpha, C_i^\beta \right).
\]
Proof: The proof is essentially the same as the proof of Lemma 4.1.

As another preliminary to our generalization of the proof in [1] concerning the linearity of binary greedy codes, we present the following Lemma, without proof, as it appears in [1]. Remember that in much of what follows we will rely implicitly on the natural identity between binary words and the integers which they represent when we view them as base 2 representations.

Lemma 4.3: Let \( \alpha \) and \( \beta \) be two integers such that \( \beta < \beta \oplus_2 \alpha \). The number of integers \( x \) such that \( \beta \leq x < \beta \oplus_2 \alpha \) is at most \( \alpha \) with equality iff (in their base 2 representations) \( \alpha \) and \( \beta \) have no powers of 2 in common. In particular, \( \beta < 2^k \) implies that there are exactly \( 2^k \) integers \( x \) with \( \beta \leq x < \beta \oplus_2 2^k \), and hence there do not exist integers \( \alpha \) and \( \beta \) such that either \( \alpha < \beta < \beta \oplus_2 2^k \leq \alpha \oplus_2 2^k \) or \( \alpha < \beta \oplus_2 2^k < \beta \leq \alpha \oplus_2 2^k \) holds.

Before presenting the generalization of the proof of the linearity of binary greedy codes, some definitions are required. We can represent the \( B \)-order on the space \( Q^n \) (where \( Q=\{0, 1\} \)) by indexing its elements as \( z_1, z_2, ..., z_{2^n} \), where \( i<j \) implies that \( z_i < z_j \) in the \( B \)-order. We define a function \( g : Q^n \rightarrow \mathbb{Z}_{\geq 0} \) by

\[
g(z_i) = \min\{t \in \mathbb{Z}_{\geq 0} : \text{dist}(z_i, z_j) = 3 \text{ for all } j < i \text{ such that } g(z_j) = t\}.
\]

It is implicit in this definition of \( g \) that if the set on the right is empty, then \( g(z_i) \) equals the smallest integer not in \( \{g(z_1), g(z_2), ..., g(z_{i-1})\} \). The designed distance of a code is the minimum distance sought for a code: in the present context, it is the minimum distance sought for the code during the greedy construction, whether it be Hamming distance, or distance determined by the metrics \( d_B \) or \( d_{E,F} \), in which case the designed distance is 3. We define the index of the \( B \)-order on \( Q^n \) relative to the designed distance \( d \) as the smallest value \( m \) such that \( g(Q^n) \subseteq Q^m \), or alternatively, using our identity between words and integers, as the smallest value \( m \) such that \( g(z) \leq 2^m - 1 \) for all \( z \in Q^n \).

Next we give our generalization of the proof by Brualdi and Pless of the linearity of binary greedy codes.
Theorem 4.3: Let $Q = \{0, 1\}$, and consider $Q^n$ to be the vector space $GF(2)^n$. Let $E$ be an error pattern in $Q^n$ containing 0, and let $F$ be an error pattern in $Q^n$. Let $\text{dist}$ be one of the metrics $d_E$ or $d_{E,F}$ given in Theorems 4.1 and 4.2. Let $m$ be the index of the $B$-order on $Q^n$ relative to the designed distance $\text{dist}_{MIN} = 3$. Then the function

$$g : Q^n \rightarrow Q^m$$

is a surjective homomorphism whose kernel equals the $B$-greedy code $C$ of length $n$ and designed distance 3. In particular, $C$ is a linear code of dimension $n - m$. A parity check matrix for $C$ is the $m$ by $n$ matrix

$$H = [g(e_1) \cdots g(e_2) g(e_1)]$$

(where the $e_i$s are the standard basis vectors), and for each $z \in Q^n$, $g(z)$ is the syndrome of $z$ relative to $H$.

Proof: Let the order basis $B$ be $\{y_1, y_2, \ldots, y_n\}$ and let $V_i = \text{span}(y_1, \ldots, y_i)$ $(0 \leq i \leq n)$. We prove by induction on $i$ that

$$g : V_i \rightarrow Q^m$$

is a homomorphism. Since $V_n = Q^n$, we get that $g : Q^n \rightarrow Q^m$ is a homomorphism, from which all of the conclusions of the theorem easily follow.

For each $\alpha$ which occurs as a $g$-value of a vector in $V_i$, let

$$C^\alpha_i = \{x \in V_i : g(x) = \alpha\} \quad (0 \leq i \leq n).$$

If $\alpha = (0\ldots0)$ we write $C_i$ in place of $C^\alpha_i$.

We have $V_0 = \{(0\ldots0)\}$ and $g(0\ldots0) = (0\ldots0)$. Hence, $g : V_0 \rightarrow Q^m$ is indeed a homomorphism. Let $i \geq 0$ and assume that $g : V_i \rightarrow Q^m$ is a homomorphism. Thus $C_i$ is a linear code, and its cosets are the $C^\alpha_i$. Moreover, for any two cosets $C^\alpha_i$ and $C^\alpha'_i$, we have

$$C^\alpha_i \oplus C^\alpha'_i = C^{\alpha + \alpha'}_i.$$  \hspace{1cm} (1)
We now consider the map

\[ g : V_{i+1} \rightarrow Q^{\alpha} \]  

(2)

where

\[ V_{i+1} = V_i \cup (y_{i+1} \oplus_2 V_i). \]

To conclude that (2) is a homomorphism it suffices to prove that

\[ g(y_{i+1} \oplus_2 z) = g(y_{i+1}) \oplus_2 g(z) \]

for all \( z \) in \( V_i \). We consider two cases.

Case 1: \( \text{dist}(y_{i+1}, C_i^\alpha) < 3 \) for all \( C_i^\alpha \).

Consider any vector \( y_{i+1} \oplus_2 z \) with \( z \) in \( V_i \). For each \( C_i^\alpha \), Lemmas 4.1 and 4.2 show that we have

\[ \text{dist}(y_{i+1} \oplus_2 z, C_i^\alpha) = \text{dist}(y_{i+1}, z \oplus_2 C_i^\alpha) = \text{dist}(y_{i+1}, C_i^\beta) \]

for some \( \beta \). Hence

\[ \text{dist}(y_{i+1} \oplus_2 z, C_i^\alpha) < 3, \quad \forall C_i^\alpha. \]

From the algorithm for computing \( g \), it follows that no vector in \( y_{i+1} \oplus_2 V_i \) receives the same \( g \)-value as a vector in \( V_i \). Let \( \gamma \) be the smallest integer which is not a \( g \)-value of any vector in \( V_i \). Since we are assuming inductively that \( g : V_i \rightarrow Q^{\alpha} \) is a homomorphism, it follows that \( \gamma \) is a power of 2; that is \( \gamma \) has a single non-zero digit as a binary word. Therefore we have \( g(y_{i+1}) = \gamma \). It follows from the definitions of \( d_E \) and \( d_{E,F} \) that

\[ \text{dist}(y_{i+1} \oplus_2 z, y_{i+1} \oplus_2 z') = \text{dist}(z, z') \]

for all \( z, z' \in V_i \).

Therefore it follows from the definition of the \( B \)-order and the definition of \( g \) that computing \( g \)-values of vectors in \( y_{i+1} \oplus_2 V_i \) is the same as computing the \( g \)-values of “corresponding” vectors in \( V_i \) using the initial value \( \gamma \). Hence

\[ g(y_{i+1} \oplus_2 z) = \gamma + g(z) = \gamma \oplus_2 g(z) = g(y_{i+1}) \oplus_2 g(z) \]

for all \( z \) in \( V_i \). (3)
Hence (2) is a homomorphism in this case.

Case 2: There is a $\beta$ such that $\text{dist}(y_{i+1}, C_i^{\beta}) = 3$.

We choose $\beta$ to be the smallest integer satisfying the assumption of this case, and hence

$$g(y_{i+1}) = \beta.$$ 

Suppose $z \in C_i^{\alpha}$. Then by (1), and by Lemmas 4.1 and 4.2, for all $\tau$,

$$\text{dist}(y_{i+1} \oplus_2 z, C_i^{\tau}) = \text{dist}(y_{i+1}, z \oplus_2 C_i^{\tau}) = \text{dist}(y_{i+1}, C_i^{\alpha_1 \oplus_2 \tau}).$$

Thus for each $\alpha$ and for each $\tau$, all of the vectors in $y_{i+1} \oplus_2 C_i^{\alpha}$ have the same distance to the coset $C_i^{\tau}$. Since the vectors in $y_{i+1} \oplus_2 V_i$ are considered in the same order as the vectors in $V_i$, each of the vectors in $y_{i+1} \oplus_2 C_i^{\alpha}$ has the same $g$-value, and for $\alpha \neq \alpha'$, vectors in $y_{i+1} \oplus_2 C_i^{\alpha}$ have different $g$-values from vectors in $y_{i+1} \oplus_2 C_i^{\alpha'}$. For $x \in C_i^{\alpha}$, we can now write $g(y_{i+1} \oplus_2 C_i^{\alpha})$ in place of $g(y_{i+1} \oplus_2 x)$.

Consider a $g$-value $\gamma$ of $V_i$. By (1),

$$C_i^{\beta} \oplus_2 C_i^{\gamma} = C_i^{\beta \oplus_2 \gamma}.$$ 

We have by Lemmas 4.1 and 4.2

$$\text{dist}(y_{i+1} \oplus_2 C_i^{\gamma}, C_i^{\beta \oplus_2 \gamma}) = \text{dist}(y_{i+1}, C_i^{\beta}) = 3,$$

which implies that $\beta \oplus_2 \gamma$ is a possible $g$-value for the vectors in $y_{i+1} \oplus_2 C_i^{\gamma}$. By taking $\gamma = \beta$ and using the fact that 0 is the smallest possible $g$-value, we now conclude that

$$C_{i+1} = C_i^0 = C_i^0 \cup (y_{i+1} \oplus_2 C_i^{\beta}),$$

and thus that $C_{i+1}$ is a linear code.

We now start another induction on increasing values of $\gamma$ and show that

$$g(y_{i+1} \oplus_2 C_i^{\gamma}) = \beta \oplus_2 \gamma,$$
that is, the cosets of the linear code $C_{i+1}$ are given by

$$C^{i\oplus_2 \gamma}_{i+1} = C^i_{i+1} \cup (y_{i+1} \oplus_2 C^\gamma_i).$$

In particular, this implies that each vector in $y_{i+1} \oplus_2 V_i$ gets the same $g$-value as a vector in $V_i$. For $\gamma = 0$, (5) holds by the definition of $\beta$. Now suppose that $\tau \neq 0$ and the (5) holds for all $\gamma < \tau$. This implies that

$$g(y_{i+1} \oplus_2 C^\gamma_i) \neq \beta \oplus_2 \tau, \text{ for all } \gamma < \tau. \quad (6)$$

Let

$$\rho = g(y_{i+1} \oplus_2 C^\tau_i).$$

Since $\beta \oplus_2 \tau$ is a possible $g$-value for $y_{i+1} \oplus_2 C^\tau_i$ and since by (6), $\beta \oplus_2 \tau$ has not been given away by the time we reach the first vector in $y_{i+1} \oplus_2 C^\tau_i$, we now conclude that $\rho \leq \beta \oplus_2 \tau$.

There is a smallest power $2^k$ such that $\tau \oplus_2 2^k < \tau$. Let

$$\mu = g(y_{i+1} \oplus_2 C_i^\tau \oplus_2 2^k) = \beta \oplus_2 (\tau \oplus_2 2^k).$$

Then

$$\rho \leq \beta \oplus_2 \tau = \mu \oplus_2 2^k.$$

We also have by Lemmas 4.1 and 4.2 that

$$\text{dist}(y_{i+1} \oplus_2 C_i^\tau \oplus_2 2^k, C_i^\mu \oplus_2 2^k) = \text{dist}(y_{i+1} \oplus_2 C_i^\tau, C_i^\mu) = 3. \quad (7)$$

We claim that $\mu \leq \rho \oplus_2 2^k$. Assume to the contrary that $\rho \oplus_2 2^k < \mu$. Then using (7), we see that there exists an $\alpha < \tau \oplus_2 2^k$ such that $g(y_{i+1} \oplus_2 C_i^\alpha) = \rho \oplus_2 2^k$ and hence $\beta \oplus_2 \alpha = \rho \oplus_2 2^k$. Now $\alpha < \tau \oplus_2 2^k < \tau$ and Lemma 4.3 imply that $\alpha \oplus_2 2^k < \tau$ and so

$$g(y_{i+1} \oplus_2 C_i^\alpha \oplus_2 2^k) = \beta \oplus_2 \alpha \oplus_2 2^k = \rho.$$
contradicting $g(y_{i+1} \oplus_2 C_i^r) = \rho$. Hence, 
\[ \mu \leq \rho \oplus_2 2^k. \]

We now make the following claim:

**Claim:** $\rho = \mu \oplus_2 2^k$.

**Proof of claim:** The proof proceeds by assuming to the contrary that $\rho \neq \mu \oplus_2 2^k$. Since $g(y_{i+1} \oplus_2 C_i^r \ominus_2 2^k) = \mu$, we also know that $\rho \neq \mu$. This allows us to use Lemma 4.3 to show that one of the following two cases holds:

\[ \mu < \rho < \mu \oplus_2 2^k < \rho \oplus_2 2^k, \quad (8) \]

or

\[ \rho < \mu < \rho \oplus_2 2^k < \mu \oplus_2 2^k. \quad (9) \]

Further applications of Lemma 4.3 to these two cases gives the result. ■

Since $\mu = \beta \oplus_2 (\tau \ominus_2 2^k)$ we now conclude that $\rho = \beta \oplus_2 \tau$. Therefore $g : V_{i+1} \rightarrow Q^m$ is a homomorphism. ■

Our conclusion then is that, if $E$ is a binary error pattern and $0 \in E$, then the binary greedy code constructed about $E$ is a linear code. If, further, $F$ is a binary error pattern then the greedy code constructed so as to be $E$-correcting and $F$-detecting is a linear code. As an example, note that the example given earlier on page 26 of the binary double-burst-error-correcting lexicode is a linear code.

**Loop transversal codes**

Loop transversal codes (henceforth called LT codes) were introduced by Smith in [5]. They are linear codes given by a function, $\text{syn}$, having a domain consisting of an error pattern $E$ which we wish to correct. The function $\text{syn}$ is constructed so that it can serve as the syndrome function for a code; the code can then be taken as the kernel of the syndrome map. LT codes get their name from their theoretical foundation, which is given in [5] and which involves the class of algebraic structures called loops and transversals to loops.
In practice, the function \( \text{syn} \) which is to serve as the syndrome is constructed so as to minimize the redundancy of the code, thus increasing the code size. Without presenting the algorithm in detail just yet, we should say that in this paper LT codes are in every case constructed by the \textit{greedy} syndrome construction algorithm, by which we consider the elements of \( Q^m \) in lexicographic order, allowing them into the range of the syndrome if they preserve (together with that portion of the syndrome function already constructed) the particular properties which we must require of the syndrome. In the LT codes discussed in this dissertation, this greedy algorithm is used exclusively, and we will refer to LT codes constructed in this way as \textit{greedy LT codes}.

\textbf{Construction}

In the paper [5], Smith expresses as one of the purposes of the LT code construction the goal of providing codes with arbitrarily long wordlength designed to provide error correction for a given set of channel statistics, where the code for any wordlength is nested in a code of longer wordlength. That is, the LT code algorithm provides us with a series of code extensions for an extended error-pattern, \( \mathbb{E} \). An appropriate restriction on our error patterns, as discussed in section 2 of this chapter, is that the error pattern \( \mathbb{E} \) is self-subordinate. This implies that \( 0 \in \mathbb{E} \). We further require that \( B \subseteq \mathbb{E} \), where \( B \) is the set of all words having only a single non-zero digit, which is a 1. As mentioned before, under the restriction that \( \mathbb{E} \) is self-subordinate, the restriction that \( B \subseteq \mathbb{E} \) is equivalent to the fairly weak requirement that for any digit position, there is some error with a 1 in that position (for either binary or non-binary cases).

Let \( \mathbb{E} \) be an extended self-subordinate error pattern. We want to build a syndrome \( \text{syn} \) defined on \( \mathbb{E} \) such that, for any \( n \), \( \text{syn}|_{\mathbb{E}_n} \) serves as the syndrome for a code \( \mathbb{C}_n \) which is an extension of \( \mathbb{C}_{n-1} \). If our code is to be built using the
syndrome function, then we must require that

\begin{enumerate}
\item \(\text{syn}|_{E_n} : E_n \to \mathbb{Q}^n\) is injective
\item \(\text{syn}(e_1) \oplus \text{syn}(e_2) = \text{syn}(e_1 \oplus e_2)\)
\end{enumerate}

whenever \(e_1, e_2, e_1 \oplus e_2 \in E\)

\begin{enumerate}
\item \(\text{syn}(\alpha e) = \alpha \text{syn}(e)\)
\end{enumerate}

whenever \(\alpha, e \in E\), where \(\alpha \in \mathbb{Q}\).

Further, if we want the extensional property for the sequence of codes of various lengths, we require that \(\text{syn}|_{E_n \cap \{b \in B \mid \text{syn}(b) \in B\}}\) is onto \(B \cap \mathbb{Q}^n\) for some \(m\). That these conditions will suffice is established in [5] using the properties of loop transversals. The greedy algorithm we will use, for constructing \(\text{syn}|_{E_n}, \text{syn}|_{E_{n+1}}, \text{syn}|_{E_{n+2}}, \ldots\) in turn, will accord to the requirements.

In [5], Smith presents the algorithm for syndrome construction in concrete terms only in the binary case. The algorithm extends naturally to the general \(q\)-ary case. Remember that we are not under the assumption that \(E\) is a white-noise error pattern. In order to make the induction method clear in what follows, we will let \(E_0\) equal \(\{0\}\).

Step (1) Let \(\text{syn}(0) = 0\).

Step (2) Suppose we have defined \(\text{syn}\) on \(E_n\). Let \(\text{syn}(b_{n+1}) = \min(S)\) where \(S\) is the set
\[
\{ s \mid (e_1^{n+1}s) \oplus \text{syn}(e_1 \Theta_q (e_1^{n+1}b_{n+1})) \neq (e_2^{n+1}s) \oplus \text{syn}(e_2 \Theta_q (e_2^{n+1}b_{n+1})) \text{ for all } e_1 \in (Q^{n+1} - Q^n) \cap E, \text{ for all } e_2 \in Q^{n+1} \cap E \}.
\]

Step (3) for \(e \in E_{n+1} - E_n\), let \(\text{syn}(e) = e^{n+1} \text{syn}(b_{n+1}) \Theta_q \text{syn}(e\Theta_q e^{n+1}b_{n+1})\).

In step (2), \(e_i^{n+1}\) refers to the \((n+1)\)-th digit of \(e_i\) so \(e_i^{n+1}\) is an element of \(Q\), and \(e_i^{n+1}s\) and \(e_i^{n+1}b_{n+1}\) represent scalar multiples of vectors.

\(\text{syn}(b_{n+1})\) then is given the smallest possible value such that, when we extend \(\text{syn}\) from \(E_n \cup \{b_{n+1}\}\) to \(E_{n+1}\) using requirements (2a) and (2b), we preserve the
injectivity of syn. Also, this algorithm gives us that \( \text{syn}|_{E \cap \{b \in B \mid \text{syn}(b) \in B \}} \) is onto \( B \cap Q^m \) for some \( m \) dependent on \( i \); if \( \text{syn}|_{E \cap \{b \in B \mid \text{syn}(b) \in B \}} \) is onto \( B \cap Q^m \) for some \( m \), then \( b_{n+1} \) is always an element of the set in step (2) over which we take the minimum.

If \( E \) is closed under scalar multiplication, then we can give an alternative expression to step (2):

**Step (2')** Let \( \text{syn}(b_{n+1}) = \min \{ s \mid s \not\in A_{n+1} \} \) where

\[
A_{n+1} = \{ \text{syn}(e_2) \ominus q \text{syn}(e_1) \mid e_1, e_2 \in E_n \text{ and } b_{n+1} \ominus q e_1 \in E \}.
\]

In the binary case \( E \) is always closed under scalar multiplication and \( \ominus q = \ominus 2 \), so we have another alternative for step (2) in the binary case:

**Step (2'')** Let \( \text{syn}(b_{n+1}) = \min \{ s \mid s \not\in A_{n+1} \} \) where

\[
A_{n+1} = \{ \text{syn}(e_1) \ominus 2 \text{syn}(e_2) \mid e_1, e_2 \in E_n \text{ and } b_{n+1} \ominus 2 e_1 \in E \}.
\]

In fact, step (2'') is the form given in [5], where the set \( A_{n+1} \) is called *anathema*.

Given that the function \( \text{syn}|_{E_n} \) has been constructed, the \( E \)-correcting LT code of length \( n \) is given by \( C_n = \{ c \mid c \in Q^n \text{ and } \text{syn}(c) = 0 \} \) which we will call the *LT code constructed about \( E \).* From the properties of linear codes we have \( \text{red}(C_n) = \max \{ 1 + \left\lfloor \log_q(\text{syn}(e)) \right\rfloor \mid e \in E_n \} \) and \( \dim(C_n) = n - \text{red}(C_n) \). From these facts it is clear that \( \text{red}(C_{n+1}) \geq \text{red}(C_n) \).

Also, \( \dim(C_{n+1}) > \dim(C_n) \) only if \( \text{red}(C_{n+1}) = \text{red}(C_n) \), in which case \( \dim(C_{n+1}) = \dim(C_n) + 1 \). (\( \text{red}(C_{n+1}) = \text{red}(C_n) \) occurs iff the left-most non-zero digit of \( \text{syn}(b_{n+1}) \) is no further to the left than is the left-most non-zero digit of \( \text{syn}(b_i) \) for all \( i \) such that \( 0 \leq i \leq n \).) It is impossible that \( \dim(C_{n+1}) < \dim(C_n) \) due to the fact that \( \text{syn}|_{E_{n+1}} \) is an extension of \( \text{syn}|_{E_n} \), which is guaranteed by our requirement that \( E \) is self-subordinate. Thus we have proven the following:

**Lemma 4.4:** If \( C_n \) and \( C_{n+1} \) are the LT codes constructed (using the greedy algorithm) about \( E_n \) and \( E_{n+1} \) for some extended error pattern \( E \), then \( \dim(C_n) \leq \dim(C_{n+1}) \leq \dim(C_n) + 1 \).
The following is an example of an LT code construction:

\[
\begin{array}{c|c}
  e & \text{syn}(e) \\
  0 & 0 \\
  1 & 1 \\
  10 & 10 \\
  11 & 11 \\
  100 & 100 \\
  110 & 110 \\
  1000 & 1000 \\
  1100 & 1100 \\
  10000 & 1001 \\
  11000 & 1101 \\
  100000 & 1010 \\
  110000 & 1111 \\
\end{array}
\]

We are assisted in the code construction by the fact that \( \text{red}(C_{n+1}) = \text{red}(C_n) \) (and therefore \( \text{dim}(C_{n+1}) > \text{dim}(C_n) \)) occurs iff the left-most non-zero digit of \( \text{syn}(b_{n+1}) \) is no further to the left than is the left-most non-zero digit of \( \text{syn}(b_i) \) for all \( i \) such that \( 0 \leq i \leq n \). Inspection of the syndrome shows us that the smallest value of \( n \) such that we get a non-zero codeword in \( C_n \) is \( n = 5 \). We construct the first non-zero codeword \( c_1 \), remembering that we must have \( \text{syn}(c_1) = 0 \) as follows:

\[
\begin{array}{c|c}
  \text{syndromes} & \text{errors} \\
  101 & 10000 \\
  100 & 100 \\
  \oplus_2 1 & \oplus_2 1 \\
\end{array}
\]

\[
\text{syn}(c_1) = 0 \quad c_1 = 10101
\]

In the previous example, the syndrome is not an increasing function, and even the syndrome's restriction to the set \( B \) is not an increasing function. The
latter only happens in non-white-noise cases. In white-noise cases, we have the property that error patterns are closed under scalar multiplication, and so we can use the algorithm which makes use of anathema. Inspection of the definitions of the anathema sets $A_n$ and $A_{n+1}$ shows that, in the white-noise case, $A_n \subseteq A_{n+1}$, so that $\text{syn}(b_n) < \text{syn}(b_{n+1})$.

**Maximality**

We have seen earlier that a lexicode of length $n$ constructed about an error pattern $E$ is always maximal in the collection of $E$-correcting codes, and that in the white-noise case (for which the Gilbert bound is applicable) lexicodes satisfy the Gilbert bound. Here we will show that LT codes are maximal among the collection of linear codes, and that under the restriction that the error pattern $E$ is closed under scalar multiplication, LT codes are maximal within the collection of all codes, linear or non-linear. Thus (1) LT codes are maximal and satisfy the Gilbert bound in the white-noise case, for any $q$, and (2) LT codes are maximal $E$-correcting codes for any $E$ in the binary case (as long as $E$ satisfies the loop transversal code requirements that $B \subseteq E$ and $E$ is self-subordinate). This is mildly surprising when we consider that LT codes are restricted in their construction so as to be linear.

As a first step, we will show that LT codes are maximal within the collection of linear codes.

**Theorem 4.4:** Let $C_n$ be the LT code of length $n$ constructed (by the greedy syndrome construction algorithm) about an error pattern $E$. If $C$ is an $E$-correcting linear code of length $n$ such that $C_n \subseteq C$, then $C_n = C$.

**Proof:** Suppose $C_n \neq C$. Let $j$ be the smallest integer such that $C_n \cap Q^j \neq C \cap Q^j$. Because $C_n \subseteq C$, it must be that there exists $c \in C \cap (Q^j - Q^{j-1})$, while $C_n \cap (Q^j - Q^{j-1}) = \emptyset$. Now the code $C$ does not come to us supplied with a syndrome function which we can specify. But the existence of $c$ shows that it should have been possible for the loop transversal greedy
algorithm to assign a syndrome value to $b_j$ in such a way that it would have produced a proper extension of $C_n \cap Q^{i-1}$ ($= C \cap Q^{i-1}$) which would include $c$ or some other word in $Q^j - Q^{i-1}$. Because the greedy algorithm did not find such a syndrome value, we have a contradiction. ■

**Theorem 4.5:** Let $E$ be an error pattern that is closed under scalar multiplication. If a linear code $C$ is maximal among linear $E$-correcting codes of length $n$, then it is maximal among all $E$-correcting codes of length $n$.

**Proof:** Let $C$ be a linear code that is maximal among all linear $E$-correcting codes of length $n$, where $E$ is closed under scalar multiplication. Suppose $D$ is a (non-linear) $E$-correcting code of length $n$ such that $C \subseteq D$. Let $x \in D - C$. Let $C' = \text{span}(C \cup \{x\})$. Then $C'$ is a linear $E$-correcting code of length $n$ which contradicts the maximality of $C$ among linear $E$-correcting codes of length $n$.

To show that $C'$ is $E$-correcting, note that $C \cup \{x\}$ is $E$-correcting because it is a subset of $D$, which is $E$-correcting. An arbitrary element of $C'$ has the form $c \oplus \alpha x$, where $c \in C$ and $\alpha \in \mathbb{Q}$. Suppose that $C'$ is not $E$-correcting. Then for some $c'_1, c'_2 \in C'$ we have

$$c'_1 \oplus \alpha e_1 = c'_2 \oplus \alpha e_2$$

for some $e_1, e_2 \in E$. Let $c'_1 = c_1 \oplus \alpha x$ and $c'_2 = c_2 \oplus \alpha x$. If $\alpha = \beta$ then

$$c'_1 \oplus \alpha e_1 = c'_2 \oplus \alpha e_2 \Rightarrow c_1 \oplus \alpha e_1 = c_2 \oplus \alpha e_2$$

giving us a contradiction of the hypothesis that $C$ is an $E$-correcting code. If $\alpha \neq \beta$ then

$$c'_1 \oplus \alpha e_1 = c'_2 \oplus \alpha e_2 \Rightarrow c_1 \oplus \alpha e_1 = c_2 \oplus \beta e_2$$

$$c_1 \oplus \alpha e_1 = c_2 \oplus (\beta - \alpha) e \Rightarrow c_1 \oplus \gamma e_1 = c_2 \oplus \gamma e_2$$

where $\gamma = (\beta - \alpha)^{-1}$, giving us a contradiction of the hypothesis that $C \cup \{x\}$ is $E$-correcting; here we have used the fact that $E$ is closed under scalar multiplication and that $C$ is a linear code. ■
Theorem 4.6: Let $E$ be a self-subordinate error pattern such that $B \subseteq E$ and $E$ is closed under negation. Then the LT code constructed about $E$ by the greedy algorithm is a maximal code.

Proof: The result follows from the previous two theorems. ■

Binary loop transversal codes and lexicodes

We have seen that in the binary case, both lexicodes and LT codes are maximal. In fact, for any binary error pattern $E$ satisfying the requirements for the loop transversal code construction (regardless of whether $E$ is a white-noise or non-white-noise pattern), the LT code constructed about $E$ is identical to the (unrestricted) lexicode constructed about $E$. The proof of this uses the following theorem within an induction proof, and uses a base case result that for wordlength $n=1$ the binary LT code and lexicode both consist of the code $\{0\}$, together with the result shown in section 3 of this chapter that all binary lexicodes are linear.

Theorem 4.7: For any positive integer $i$, let $L_i$ and $C_i$ be the lexicode and LT code, respectively, of length $i$ for any $i$, constructed about an extended binary error pattern $E$ (which has $B \subseteq E$ and which is self-subordinate), where $C_i$ is constructed by the greedy syndrome construction algorithm. If $L_u = C_u$ then $L_{u+1} = C_{u+1}$.

Proof: We have shown in section 3 of this chapter that $L_u$ and $L_{n+1}$ are linear. A lexicode does not automatically determine a unique syndrome, but under our hypothesis that $L_u = C_u$, we can take $\text{syn}_{L_{u+1}}$ to be that extension of $\text{syn}_{L_u}$ induced by choosing $\text{syn}_{L_{u+1}}(b_{u+1})$ to be the lexicographically smallest word such that $L_{u+1}$ is the kernel of $\text{syn}_{L_{u+1}}$.

If $\dim(L_{u+1}) = \dim(L_u)$, then, because lexicodes are maximal, we must have $\dim(C_{n+1}) = \dim(C_n)$, so that $L_{n+1} = C_{n+1}$.

If $\dim(L_{u+1}) > \dim(L_u)$, then $\dim(L_{u+1}) = \dim(L_u) + 1$. (If $\dim(L_{u+1}) > \dim(L_u) + 1$, then there exist $x, y \in Q^{u+1} - Q^u$ such that $L_{n+1} = \text{span}(L_u \cup \{x, y\})$ but $y \not\in \text{span}(L_u \cup \{x\})$. However $x \oplus y \in Q^u - L_u$, contra-
predicting the maximality of \( L_n \). In this case we will have \( \dim(C_{n+1}) > \dim(C_n) \) because the increase in dimension in the lexicode construction shows that some choice of syndrome will be possible in the loop transversal construction which will also increase the dimension. We showed earlier that in this case we have \( \dim(C_{n+1}) = \dim(C_n) + 1 \). Combining results we have that in any case under our hypotheses, \( \dim(L_{n+1}) = \dim(C_{n+1}) \).

Let \( l \) and \( c \) be the lexicographically smallest codewords in \( L_{n+1} - L_n \) and \( C_{n+1} - C_n \) respectively. Using basic linear algebra and the result that \( \dim(L_{n+1}) = \dim(C_{n+1}) \) it can be shown that

\[
l = c \Leftrightarrow L_{n+1} = C_{n+1}.
\]

By the lexicode construction we know that \( l \leq c \). Let's construct \( l \) and \( c \). Let

\[
\text{syn}_{C_{n+1}}(b_{n+1}) = v = v_1^m v_2^{m-1} \ldots v_{n}^1
\]

and let

\[
\text{syn}_{L_{n+1}}(b_{n+1}) = w = w_1^m w_2^{m-1} \ldots w_{n}^1.
\]

Then

\[
l = b_{n+1} \oplus_2 \left( \bigoplus_{\{j \mid v_j = 1\}} b_{\text{m}(j)} \right)
\]

where \( \text{m}(j) \) is the smallest integer \( i \) such that \( (\text{syn}(b_i))^j = 1 \). In fact, \( \text{syn}(b_{\text{m}(j)}) \) equals \( b_j \in B \) due to properties of the greedy algorithm: loosely speaking, whenever we have to venture into a larger range space to build our syndrome, the first word we come to (lexicographically) will always suffice, and that is always some \( b_j \), an element of \( B \). More formally, we always have

\[
(b_j \oplus_2 \text{syn}(E \cap \{0, 1\}^{(b_{\text{m}(j)} - 1)})) \cap \text{syn}(E \cap \{0, 1\}^{(b_{\text{m}(j)} - 1)}) = \emptyset.
\]

Also,

\[
c = b_{n+1} \oplus_2 \left( \bigoplus_{\{j \mid v_j = 1\}} b_{\text{m}(j)} \right).
\]
By the loop transversal construction using the greedy syndrome construction algorithm, \( v \leq w \). Now if \( v < w \), then if \( w^k \) is the left-most digit in which \( w \) differs from \( v \), then \( w^k = 1 \) and \( v^k = 0 \). This together with the expressions given for \( l \) and \( c \) above implies that \( c \leq l \). Combining results we get \( c = l \). ■

**Theorem 4.8:** Let \( L \) and \( C \) be the lexicode and LT code, respectively, of some length \( n \) constructed about a binary error pattern \( E \) (which has \( B \subseteq E \) and which is self-subordinate), where \( C \) is constructed by the greedy syndrome construction algorithm. Then \( L=C \).

**Proof:** The theorem follows from a simple proof by induction which uses the previous theorem in the inductive step, together with the base case that \( L_1 = C_1 = \{0\} \). This equation is easily verified when we note that \( E_1 = \{0, 1\} \). ■
CHAPTER 5. PERFORMANCE OF LOOP TRANSVERSAL CODES

General results

In this chapter we will examine the performance of LT codes with respect to codesize. Earlier we showed that in the cases for which the error pattern $E$ is closed under scalar multiplication, that LT codes are maximal. We have seen that in the binary case, LT codes are identical to lexicodes. Therefore, like lexicodes, the loop transversal algorithm produces all of the binary perfect codes: the binary hamming codes and the binary [23,12,7] Golay code. Inspection of the greedy syndrome algorithm shows that even in non-binary cases, the LT code constructed about the single-digit errors will always pack the range of the syndrome function into the smallest space possible: anathema for each $n$ always consists of only those values already in the range of the syndrome. In what follows we will make some comparisons of LT codes with the best known codes where possible. For unusual error patterns, it is difficult to determine from the literature what the best codes are, and in some of these cases we have generated the corresponding lexicodes to provide some basis for judging the performance of LT codes.

Binary loop transversal codes

Table 1 below gives the dimensions of the binary white-noise greedy LT codes for various values of $n$ and minimum Hamming distance. Note that this information agrees with the information given by Conway and Sloane in [2] for lexicodes. Larger $n$ values are achievable in some cases due to the relative efficiency of the loop transversal algorithm. In this table, numbers which appear in parentheses are the dimensions of the best linear codes known for the given values of $n$ and $d$. In the binary case, this information comes from [7], and in the ternary case, this information comes from [3].
Table 1. Dimensions of binary greedy LT codes

<table>
<thead>
<tr>
<th>n</th>
<th>d=5</th>
<th>d=7</th>
<th>d=9</th>
<th>n</th>
<th>d=5</th>
<th>n</th>
<th>d=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>38</td>
<td>27</td>
<td>64</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>39</td>
<td>27</td>
<td>65</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>40</td>
<td>28</td>
<td>66</td>
<td>53</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>41</td>
<td>29</td>
<td>67</td>
<td>54</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>42</td>
<td>30</td>
<td>68</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>43</td>
<td>31</td>
<td>69</td>
<td>56</td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>44</td>
<td>32</td>
<td>70</td>
<td>56</td>
</tr>
<tr>
<td>19</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>45</td>
<td>33</td>
<td>71</td>
<td>57</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>46</td>
<td>34</td>
<td>72</td>
<td>58</td>
</tr>
<tr>
<td>21</td>
<td>12</td>
<td>10</td>
<td>5</td>
<td>47</td>
<td>35</td>
<td>73</td>
<td>59</td>
</tr>
<tr>
<td>22</td>
<td>12</td>
<td>11</td>
<td>6</td>
<td>48</td>
<td>36</td>
<td>74</td>
<td>60</td>
</tr>
<tr>
<td>23</td>
<td>13</td>
<td>12</td>
<td>6</td>
<td>49</td>
<td>37</td>
<td>75</td>
<td>61</td>
</tr>
<tr>
<td>24</td>
<td>14</td>
<td>12</td>
<td>6</td>
<td>50</td>
<td>38</td>
<td>76</td>
<td>62</td>
</tr>
<tr>
<td>25</td>
<td>15</td>
<td>12</td>
<td>6</td>
<td>51</td>
<td>39</td>
<td>77</td>
<td>63</td>
</tr>
<tr>
<td>26</td>
<td>16</td>
<td>12</td>
<td>6</td>
<td>52</td>
<td>40</td>
<td>78</td>
<td>64</td>
</tr>
<tr>
<td>27</td>
<td>17</td>
<td>13</td>
<td>7</td>
<td>53</td>
<td>40</td>
<td>79</td>
<td>65</td>
</tr>
<tr>
<td>28</td>
<td>18</td>
<td>13</td>
<td>7</td>
<td>54</td>
<td>41</td>
<td>80</td>
<td>66</td>
</tr>
<tr>
<td>29</td>
<td>19</td>
<td>14</td>
<td>7</td>
<td>55</td>
<td>42</td>
<td>81</td>
<td>67</td>
</tr>
<tr>
<td>30</td>
<td>19</td>
<td>15</td>
<td>7</td>
<td>56</td>
<td>43</td>
<td>82</td>
<td>68</td>
</tr>
<tr>
<td>31</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>57</td>
<td>44</td>
<td>83</td>
<td>69</td>
</tr>
<tr>
<td>32</td>
<td>21</td>
<td>16</td>
<td>8</td>
<td>58</td>
<td>45</td>
<td>84</td>
<td>70</td>
</tr>
<tr>
<td>33</td>
<td>22</td>
<td>16</td>
<td>8</td>
<td>59</td>
<td>46</td>
<td>85</td>
<td>71</td>
</tr>
<tr>
<td>34</td>
<td>23</td>
<td>17</td>
<td>8</td>
<td>60</td>
<td>47</td>
<td>86</td>
<td>72</td>
</tr>
<tr>
<td>35</td>
<td>24</td>
<td>18</td>
<td>9</td>
<td>61</td>
<td>48</td>
<td>87</td>
<td>73</td>
</tr>
<tr>
<td>36</td>
<td>25</td>
<td>19</td>
<td>9</td>
<td>62</td>
<td>49</td>
<td>88</td>
<td>74</td>
</tr>
<tr>
<td>37</td>
<td>26</td>
<td>20</td>
<td>9</td>
<td>63</td>
<td>50</td>
<td>89</td>
<td>75</td>
</tr>
</tbody>
</table>
In the non-white-noise case we compared \( l \)-burst-correcting LT codes with the best cyclic \( l \)-burst correcting codes listed in [4] for \( l=2,3,4, \) and \( 5 \). In Table 2 below, the symbol \((n,k)\) means that a code of length \( n \) and dimension \( k \) was found. The LT codes reported here are not generally cyclic.

Table 2. The best binary \( l \)-burst correcting cyclic \((n, k)\) codes, and corresponding greedy LT codes for \( l = 2, 3, 4, \) and \( 5 \)

<table>
<thead>
<tr>
<th>2-burst cyclic</th>
<th>2-burst LT code</th>
<th>3-burst cyclic</th>
<th>3-burst LT code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7, 3)</td>
<td>(7, 2)</td>
<td>(15, 9)</td>
<td>(15, 8)</td>
</tr>
<tr>
<td>(15,10)</td>
<td>(15, 9)</td>
<td>(17, 9)</td>
<td>(17, 10)</td>
</tr>
<tr>
<td>(21, 15)</td>
<td>(21, 15)</td>
<td>(21, 14)</td>
<td>(21, 13)</td>
</tr>
<tr>
<td>(31, 25)</td>
<td>(31, 24)</td>
<td>(27, 20)</td>
<td>(27, 19)</td>
</tr>
<tr>
<td>(63, 56)</td>
<td>(63, 55)</td>
<td>(51, 42)</td>
<td>(51, 42)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(63, 55)</td>
<td>(63, 54)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(85, 76)</td>
<td>(85, 75)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4-burst cyclic</th>
<th>4-burst LT code</th>
<th>5-burst cyclic</th>
<th>5-burst LT code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(15, 7)</td>
<td>(15, 6)</td>
<td>(15, 5)</td>
<td>(15, 5)</td>
</tr>
<tr>
<td>(19, 11)</td>
<td>(19, 10)</td>
<td>(21, 10)</td>
<td>(21, 10)</td>
</tr>
<tr>
<td>(21, 12)</td>
<td>(21, 12)</td>
<td>(23, 12)</td>
<td>(23, 12)</td>
</tr>
<tr>
<td>(31, 21)</td>
<td>(31, 21)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(51, 41)</td>
<td>(51, 41)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We can see from Table 2 that the greedy loop transversal algorithm produces a binary 3-burst correcting (17,10) code, 1 dimension better than the best binary 3-burst correcting cyclic code. However this particular LT code has the relative disadvantage that it is not cyclic.

**Non-binary loop transversal codes**

Tables 3 and 4 below gives the dimensions of ternary white-noise greedy LT codes for \(d=5\) and \(d=7\). For the non-linear lexicodes the "dimension" of the code is \(nR\), the rate times the wordlength. The numbers in parentheses are the dimensions of the best linear codes known for the given values of \(n\) and \(d\).

**Table 3. Dimensions of ternary greedy LT codes and lexicodes for up to random double errors \((d=5)\)**

<table>
<thead>
<tr>
<th>(n)</th>
<th>dim of LT code</th>
<th>dim of lexicode</th>
<th>size of lexicode</th>
<th>(n)</th>
<th>dim of LT code</th>
<th>(n)</th>
<th>dim of LT code</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1 (1)</td>
<td>1.00</td>
<td>3</td>
<td>19</td>
<td>11 (12)</td>
<td>33</td>
<td>24 (25)</td>
</tr>
<tr>
<td>6</td>
<td>1 (1)</td>
<td>1.00</td>
<td>3</td>
<td>20</td>
<td>12 (13)</td>
<td>34</td>
<td>25 (26)</td>
</tr>
<tr>
<td>7</td>
<td>2 (2)</td>
<td>1.89</td>
<td>8</td>
<td>21</td>
<td>13 (14)</td>
<td>35</td>
<td>26 (27)</td>
</tr>
<tr>
<td>8</td>
<td>3 (3)</td>
<td>2.63</td>
<td>18</td>
<td>22</td>
<td>15 (15)</td>
<td>36</td>
<td>27 (28)</td>
</tr>
<tr>
<td>9</td>
<td>4 (4)</td>
<td>3.33</td>
<td>39</td>
<td>23</td>
<td>16 (16)</td>
<td>37</td>
<td>29 (29)</td>
</tr>
<tr>
<td>10</td>
<td>5 (5)</td>
<td>4.02</td>
<td>83</td>
<td>24</td>
<td>17 (17)</td>
<td>38</td>
<td>30 (30)</td>
</tr>
<tr>
<td>11</td>
<td>6 (6)</td>
<td>4.75</td>
<td>185</td>
<td>25</td>
<td>18 (18)</td>
<td>39</td>
<td>30 (31)</td>
</tr>
<tr>
<td>12</td>
<td>6 (6)</td>
<td></td>
<td></td>
<td>26</td>
<td>19 (19)</td>
<td>40</td>
<td>31 (32)</td>
</tr>
<tr>
<td>13</td>
<td>6 (7)</td>
<td></td>
<td></td>
<td>27</td>
<td>18 (19)</td>
<td>41</td>
<td>32 (33)</td>
</tr>
<tr>
<td>14</td>
<td>7 (8)</td>
<td></td>
<td></td>
<td>28</td>
<td>19 (20)</td>
<td>42</td>
<td>33 (33)</td>
</tr>
<tr>
<td>15</td>
<td>8 (8)</td>
<td></td>
<td></td>
<td>29</td>
<td>20 (21)</td>
<td>43</td>
<td>34 (33)</td>
</tr>
<tr>
<td>16</td>
<td>8 (9)</td>
<td></td>
<td></td>
<td>30</td>
<td>21 (22)</td>
<td>44</td>
<td>35 (33)</td>
</tr>
<tr>
<td>17</td>
<td>9 (10)</td>
<td></td>
<td></td>
<td>31</td>
<td>22 (23)</td>
<td>45</td>
<td>36 (34)</td>
</tr>
<tr>
<td>18</td>
<td>10 (11)</td>
<td></td>
<td></td>
<td>32</td>
<td>23 (24)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4. Dimensions of ternary greedy LT codes and lexicodes for up to random triple errors (d=7)

<table>
<thead>
<tr>
<th>n</th>
<th>dim of LT code</th>
<th>dim of lexicode</th>
<th>size of lexicode</th>
<th>n</th>
<th>dim of LT code</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1 (1)</td>
<td>1.00</td>
<td>3</td>
<td>13</td>
<td>4 (4)</td>
</tr>
<tr>
<td>8</td>
<td>1 (1)</td>
<td>1.00</td>
<td>3</td>
<td>14</td>
<td>5 (5)</td>
</tr>
<tr>
<td>9</td>
<td>1 (1)</td>
<td>1.00</td>
<td>3</td>
<td>15</td>
<td>5 (5)</td>
</tr>
<tr>
<td>10</td>
<td>2 (2)</td>
<td>2.00</td>
<td>9</td>
<td>16</td>
<td>6 (6)</td>
</tr>
<tr>
<td>11</td>
<td>3 (3)</td>
<td>2.77</td>
<td>21</td>
<td>17</td>
<td>7 (7)</td>
</tr>
<tr>
<td>12</td>
<td>3 (3)</td>
<td>3.42</td>
<td>42</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that for the ternary random double errors (minimum Hamming distance =5), the LT greedy algorithm produces the perfect Golay code with parameters [11, 6, 5] and 729 codewords, significantly better than the 185 codewords for the corresponding lexicode. In the white-noise case then, the LT greedy algorithm produces every perfect code: as mentioned earlier, the only non-ternary (and non-trivial) perfect codes are binary, and because binary greedy LT codes are identical to lexicodes, and also because (as reported in [2]) the binary lexicodes include all of the perfect binary codes (the binary Hamming codes, the [23, 12, 7] binary Golay code, and the trivial perfect binary codes), it follows that the LT greedy algorithm produces all of the perfect binary codes as well. For random triple errors, the LT greedy algorithm consistently keeps up with the best known linear codes within the range of values of n given in Table 4. Also, in the case of random double errors, Table 3 shows that for n = 43, 44, and 45, and d=5, the LT greedy algorithm produces three record breaking codes. Table 5 shows the syndrome for the LT codes indicated in Table 3.
Table 5. Syndrome for ternary random double error correcting greedy LT codes (d=5)

<table>
<thead>
<tr>
<th>i</th>
<th>syn(b_i)</th>
<th>i</th>
<th>syn(b_i)</th>
<th>i</th>
<th>syn(b_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>16</td>
<td>10000000</td>
<td>31</td>
<td>102001112</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>17</td>
<td>10100012</td>
<td>32</td>
<td>102101211</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>18</td>
<td>10200110</td>
<td>33</td>
<td>102202212</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>19</td>
<td>11000022</td>
<td>34</td>
<td>110002001</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>20</td>
<td>11100100</td>
<td>35</td>
<td>110110000</td>
</tr>
<tr>
<td>6</td>
<td>10000</td>
<td>21</td>
<td>11201000</td>
<td>36</td>
<td>110202120</td>
</tr>
<tr>
<td>7</td>
<td>10112</td>
<td>22</td>
<td>12001002</td>
<td>37</td>
<td>111010212</td>
</tr>
<tr>
<td>8</td>
<td>11021</td>
<td>23</td>
<td>12100210</td>
<td>38</td>
<td>111101120</td>
</tr>
<tr>
<td>9</td>
<td>11202</td>
<td>24</td>
<td>12201122</td>
<td>39</td>
<td>111211100</td>
</tr>
<tr>
<td>10</td>
<td>12120</td>
<td>25</td>
<td>100000000</td>
<td>40</td>
<td>112011010</td>
</tr>
<tr>
<td>11</td>
<td>12211</td>
<td>26</td>
<td>100100021</td>
<td>41</td>
<td>112110021</td>
</tr>
<tr>
<td>12</td>
<td>100000</td>
<td>27</td>
<td>100201001</td>
<td>42</td>
<td>112202011</td>
</tr>
<tr>
<td>13</td>
<td>1000000</td>
<td>28</td>
<td>101000202</td>
<td>43</td>
<td>120012101</td>
</tr>
<tr>
<td>14</td>
<td>1100011</td>
<td>29</td>
<td>101101000</td>
<td>44</td>
<td>120111201</td>
</tr>
<tr>
<td>15</td>
<td>1200101</td>
<td>30</td>
<td>101200210</td>
<td>45</td>
<td>120220201</td>
</tr>
</tbody>
</table>

We have said that non-binary lexicodes are not generally linear. Table 6 below however, indicates several ternary non-white-noise lexicodes having integral values for their “dimension”, nR, suggesting the possibility that these codes are actually linear. Inspection of these codes verified that the 2-burst correcting ternary lexicodes for n = 5, 6, 7, 8, and 9 are in fact linear.
Table 6. Dimensions of ternary greedy LT codes and lexicodes for 2–burst errors

<table>
<thead>
<tr>
<th>n</th>
<th>dim of LT code</th>
<th>dim of lexicode</th>
<th>size of lexicode</th>
<th>n</th>
<th>dim of LT code</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>1.00</td>
<td>3</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2.00</td>
<td>9</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3.00</td>
<td>27</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4.00</td>
<td>81</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4.00</td>
<td>81</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5.17</td>
<td>294</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES


APPENDIX. FORTRAN SOURCE CODE FOR GREEDY LOOP TRANSVERSAL ALGORITHM

******************************************************
* This program computes a syndrom function using as input* 
* values for q, n, and the number of error patterns* 
*(ndomsyn). The error patterns are stored lexicograph- 
* ically in errfile, and are read into the array err.* 
* The resulting syndrome appears in the array syn.* 
* 
******************************************************

program LTcode
byte err, syn, z, w, x, s, q
integer temp, n
integer answer
integer red, ndomsyn, loc, count, ptition
integer locoferr, eint, sint
integer graphopt
parameter(nerrs=5200, nsymbml=126, nsymb=127, nsymbpl=128)
dimension loc(nsymbpl), count(0:nsymb), ptition(0:nsymb)
dimension err(0:nerrs,0:nsymbml), red(nsymb)
dimension s(0:nsymbml), x(0:nsymbml), z(0:nsymbml)
dimension w(0:nsymbml)
dimension syn(0:nerrs,0:nsymbml)
common syn, err
open(unit=1, file='errfile', status='unknown')
open(unit=2, file='synfile', status='unknown')
open(unit=3, file='LTcode.dat', status='unknown')
open(unit=4, file='errcheck', status='unknown')
950 format(70(i1))
print*, 'What is the number of error patterns?'
read *, ndomsyn
nd=ndomsyn
print *, 'What are q and n?'
read *, q
read *, n
print*, 'type 1 if you want to generate a syndrome file'
print*, 'for use by the xgraph program.'
read*, graphopt
k=1
******************************************************************************
* Read in the set of errors to be corrected
* by the code.
******************************************************************************

if (n.le.69) then
  do 100 i=0, ndomsyn-1
  read (unit=1, fmt=950) (err(i,j), j=n-1,0,-1)
    if (err(i,k-1).ne.0) then
      loc(k)=i
      k=k+1
    endif
  100 continue
else
  do 105 i=0, ndomsyn-1
  read (unit=1, fmt=950) (err(i,j), j=n-1,n-69,-1)
  read (unit=1, fmt=950) (err(i,j), j=n-70,0,-1)
    if (err(i,k-1).ne.0) then
      loc(k)=i
      k=k+1
    endif
  105 continue
endif
loc(n+1)=ndomsyn
red(1)=1
******************************************************************************
* Compute the syndrome
*
******************************************************************************
call syndrome(red,nd,loc,count,ptition,n,q)
******************************************************************************
* Output the syndrome in either
* q-ary vector or integer format
******************************************************************************

if (graphopt.ne.1) then
  if (n.le.30) then
    k=1
    do 120 i=0,ndomsyn-1
write(unit=2,fmt=920) (err(i,j),j=29,0,-1),
   + (syn(i,j),j=29,0,-1)
if (i.eq.loc(k)-1) then
   write(unit=2,fmt=925) ' ',
   k=k+1
endif
120 continue
endif
else
920 format(30(i1),4x,30(i1))
921 format(i20,3x,i20)
k=l
   do 125 i=0,ndomsyn-1
call geterr(i,z,n)
call getsyn(i,w,n)
nfirst=n
mmm=red(n)+1
call numerate(z,nfirst,eint,n,q)
call numerate(w,mmm,sint,n,q)
write(unit=2,fmt=921) eint,sint
if (i.eq.loc(k)-1) then
   write(unit=2,fmt=925) ' ',
   k=k+1
endif
125 continue
endif
write(unit=3,fmt=925) 'wordlength', 'redundancy','dimension'
925 format(a10,3x,a10,3x,a9)
929 format(4x,i4,10x,i4,10x,i4)
do 150 k=1,n
   write(unit=3,fmt=929) k, red(k), k-red(k)
150 continue
end

**********************************************************************
* This subroutine is called from the main program
* to compute the syndrome, stored in the array syn
**********************************************************************
subroutine syndrome(red,nd,loc,count,ptition,n,q)
byte err,syn,s,x,y,z,w,nb,q

integer count, ptition, p, n
integer nd, red, temp
integer loc, answer, whnoise

parameter(nerrs=5200, nsymbml=126, nsymb=127, nsymbpl=128)
dimension loc(nsymbpl), count(0:nsymb), ptition(0:nsymb)
dimension err(0:nerrs, 0:nsymbml), red(nsymb)
dimension s(0:nsymbml), x(0:nsymbml), z(0:nsymbml)
dimension w(0:nsymbml), y(0:nsymbml), nz(0:nsymbml)
dimension syn(0:nerrs, 0:nsymbml)

common syn, err
open(unit=3, file='LTcode.dat', status='unknown')
print*, 'type "1" if this is a white noise case.'
read*, whnoise

*********************************************
* syn(0)=0, and syn(1)=1, and linearity
* automatically determines the syndrome
* on Q**1
*********************************************

call setword(0, 0, s, n)
call setsyn(0, s, n)
call update(count, s, ptition, n)
call setword(1, 0, s, n)
call setsyn(1, s, n)
call update(count, s, ptition, n)

do 188 nc=2, q-1
   call setword(nc, 0, s, n)
call locoferr(s, nd, p, n, q)
   call setsyn(p, s, n)
call update(count, s, ptition, n)
188   continue
820   format(a10, 2x, i3)

*********************************************
* For k=2 to n (for errors less than 2**(n+1)
* compute the syndrome on Q**k
*********************************************
do 190 k=2,n+1
   write(unit=3, fmt=820) 'k= ', k-1
   call reccount(count, ptition, n, q, k, red)
   write(unit=3, fmt=820) ' '*
   print*, 'k=', k, ' out of ', n+1
   call setword(1, k-1, nz, n)
   if (k.lt.n+1) then
     *****************************************************
     * If we are in a white-noise case then anathema is
     * increasing on the set of errors containing a single
     * non-zero 1-digit.
     *****************************************************
     if (whnoise.eq.1) then
       call getsyndoc(k-1), s, n)
     else
       call setword(0, 0, s, n)
     endif
     *********************************************
     * Try the next value s as a possible syndrome value
     *
     *********************************************
  158   call increm(s, n, q)
   call initial(s, answer, n)
   if (answer.gt.k) then
     goto 167
   endif
   do 222 i=0, loc(k)-1
      call geterr(i, z, n)
      call getsyn(i, y, n)
      call nim(nz, z, w, n, q)
      call errstat(w, nd, answer, n, q)
      if (answer.eq.1) then
        do 223 j=0, loc(k)-1
           call getsyn(j, x, n)
           call nimdiff(x, y, w, n, q)
           call compare(w, s, nswer, n)
           if (nswer.eq.1) then
             goto 158
           endif
endif
223 continue
endif
222 continue
goto 168
167 print*, 'syndrome not found for error ', loc(k)
goto 191
168 call setsyn(loc(k), s, n)
call update(count, s, ptition, n)
call initial(s, answer, n)
*********************************************
* Compute the redundancy of C intersect Q**k
*********************************************
red(k)=int(max(answer, red(k-1)))
print*, 'k=', k, 'red=', red(k)
do 225 nc=2,q-1
call nimprod(s, nc, w, n, q)
call nimprod(nz, nc, z, n, q)
call locoferr(z, nd, p, n, q)
call setsyn(p, w, n)
*********************************************
* Extend the syndrome by linearity
*********************************************
225 continue
do 230 i=loc(k)+1, loc(k+1)-1
call setword(0, 0, s, n)
do 240 j=n-1, 0, -1
nb=err(i, j)
if(nb.ne.0) then
call setword(nb, j, w, n)
call locoferr(w, nd, p, n, q)
call getsyn(p, z, n)
call nim(s, z, x, n, q)
call eqt(s, x, n)
endif
240 continue
call setsyn(i, s, n)
call update(count, s, ptition, n)
230 continue
   endif
190 continue
191 end

*******************************************************************************
* This subroutine computes the q-ary scalar product nc times x.
*******************************************************************************
subroutine nimprod(x, nc, z, n, q)
  byte x, y, z, q
  integer n
  parameter(nerrs=10000, nsymbml=126, nsymb=127, nsymbpl=128)
dimension x(0:nsymbml), y(0:nsymbml), z(0:nsymbml)
do 100 i=n-1, 0, -1
   z(i)=x(i)*nc
   if (z(i).ge.q) then
      z(i)=z(i)-(z(i)/q)*q
   endif
100 continue
end

*******************************************************************************
* This subroutine computes the q-ary vector sum of x+y=z.
*******************************************************************************
subroutine nim(x, y, z, n, q)
  byte x, y, z, q
  integer n
  parameter(nerrs=10000, nsymbml=126, nsymb=127, nsymbpl=128)
dimension x(0:nsymbml), y(0:nsymbml), z(0:nsymbml)
do 100 i=n-1, 0, -1
   z(i)=x(i)+y(i)
   if (z(i).ge.q) then
      z(i)=z(i)-q
   endif
100 continue
end
* This subroutine computes the q-ary vector difference \( x-y=z \).

```
subroutine nimdiff(x,y,z,n,q)
  byte x,y,z,q
  integer n
  parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
  dimension x(0:nsymbml),y(0:nsymbml),z(0:nsymbml)
  do 100 i=n-1,0,-1
    z(i)=x(i)-y(i)
    if (z(i).lt.0) then
      z(i)=z(i)+q
    endif
  100 continue
end
```

* This subroutine returns the value of the \( i \)-th syndrome (\( \text{syn}(i,*) \)) which is the syndrome of the \( i \)-th error (stored in \( \text{err}(i,*) \)).

```
subroutine getsyn(i,z,n)
  byte x,y,z,q,syn,err
  integer n
  parameter(nerrs=5200,nsymbml=126,nsymb=127,nsymbpl=128)
  dimension z(0:nsymbml)
  dimension syn(0:nerrs,0:nsymbml)
  dimension err(0:nerrs,0:nsymbml)
  common syn,err
  do 100 j=n-1,0,-1
    z(j)=syn(i,j)
  100 continue
end
```

* This subroutine returns the value of the \( i \)-th error, stored in \( \text{err}(i,*) \).

```
subroutine geterr(i,z,n)
  byte x,y,z,q
  byte err,syn
```
integer n
parameter(nerrs=5200,nsymbml=126,nsymb=127,nsymbpl=128)
dimension z(0:nsymbml)
dimension syn(0:nerrs,0:nsymbml)
dimension err(0:nerrs,0:nsymbml)
common syn,err
do 100 j=n-1,0,-1
   z(j)=err(i,j)
100 continue
end

**************************************************************************
*  This subroutine sets the value of syn(i,*)
*  to be equal to the vector z.
**************************************************************************
subroutine setsyn(i,z,n)
byte x,y,z,q
byte syn,err
integer n
parameter(nerrs=5200,nsymbml=126,nsymb=127,nsymbpl=128)
dimension z(0:nsymbml)
dimension syn(0:nerrs,0:nsymbml)
dimension err(0:nerrs,0:nsymbml)
common syn,err
do 100 j=n-1,0,-1
   syn(i,j)=z(j)
100 continue
end

**************************************************************************
*  This subroutine sets the vector z so as to
*  have the value (an element of Q) appear
*  as the j-th coordinate of z, with zeros
*  elsewhere.
**************************************************************************
subroutine setword(value,j,z,n)
byte z,q
integer value
integer n
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension z(0:nsymbml)
do 100 i=n-1,0,-1
   z(i)=0
100 continue
z(j)=value
end

*****************************************************************************
* This subroutine sets the vector s equal to
* the vector x
*****************************************************************************
subroutine eqt(s,x,n)
  byte x,s,q
  integer n
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension x(0:nsymbml)
dimension s(0:nsymbml)
do 100 j=n-1,0,-1
   s(j)=x(j)
100 continue
end

*****************************************************************************
* This subroutine treats
* integer for which it is
* incrementing it by one.
*****************************************************************************
subroutine increm(s,n,q)
  byte s,q
  integer n
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension s(0:nsymbml)
s(0)=s(0)+1
do 100 j=0,n-1
   if(s(j).ge.q) then
      s(j)=s(j)-q
      s(j+1)=s(j+1)+1
   endif
100 continue
end

*****************************************************************************
* This subroutine determines whether or not (1 or 0)
* the vectors x and y are equal.

***************************************************************
subroutine compare(x,y,answer,n)
byte x,y,q
integer n
integer answer
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension x(0:nsymbml)
dimension y(0:nsymbml)
answer=1
do 100 j=0,n-1
  if(x(j).ne.y(j)) then
    answer=0
    goto 101
  endif
100 continue
101 end
***************************************************************

* This subroutine determines the (lexicographic)
* order relation between z and w
***************************************************************

subroutine comp(z,w,comparison,n)
byte z,w,q
integer n
character*4 comparison
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension z(0:nsymbml)
dimension w(0:nsymbml)
do 100 i=n-1,0,-1
  if(z(i).ne.w(i)) then
    index=i
    goto 199
  endif
100 continue
  comparison='same'
goto 299
199 if (z(index).gt.w(index)) then
  comparison='more'
else
  comparison='less'

subroutine initial(z,answer,n)
   integer answer
   integer n
   byte z,q
   parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
   dimension z(0:nsymbml)
   answer=0
   do 100 j=n-1,0,-1
      if (z(j).ne.0) then
         answer=j+1
         goto 193
      endif
   100 continue
   193 end

subroutine update(count,x,partition,n)
   byte x,q
   integer count,partition,answer
   integer n
   parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
   dimension x(0:nsymbml)
   dimension count(0:nsymb),partition(0:nsymb)
   call initial(x,answer,n)
   do 100 i=0,n
      if (answer.le.i) then
count(i)=count(i)+1
if (answer.eq.i) then
  ptition(i)=ptition(i)+1
endif
eendif
100 continue
end

******************************************************************************
* This subroutine provides output as to the
* values in count and ptition, as calculated by
* the subrouting update.
******************************************************************************
subroutine reccount(count,ptition,n,q,k,red)
integer count,ptition,red
byte q
integer n
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension count(0:nsymb),ptition(0:nsymb),red(nsymb)
open(unit=3,file='LTcode.dat',status='unknown')
939 format(4x,i4,2x,2(il0,2x,f7.5,2x))
do 151 i=l,red(k-1)+2
  zzz=(q*1.0)**(i*1.0)
  www=zzz/(q*1.0)
  xxx=(count(i)*1.0)/zzz
  yyy=(ptition(i)*1.0)/(zzz-www)
  write(unit=3,fmt=939) i,count(i),xxx,ptition(i),yyy
151 continue
end

******************************************************************************
* This subroutine computes the integer equivalent
* of the q-ary vector w, within the storage
* limits of the computer memory.
******************************************************************************
subroutine numerate(w,firstbit,value,n,q)
byte q,w
integer firstbit,value,chunk,temp,lastbit
integer n
parameter(nerrs=10000,nsymbml=126,nsymb=127,nsymbpl=128)
dimension w(0:nsymbml)
chunk=\text{int}(26 \times \frac{\log(2.0)}{0.01+\log(q \times 1.0)})
\text{temp}=0
\text{if (firstbit-chunk lt 0) then}
\quad \text{lastbit}=0
\text{else}
\quad \text{lastbit}=\text{firstbit-chunk}
\text{endif}
\text{do 100 } j=\text{firstbit-1, lastbit-1, -1}
\quad \text{temp}=(w(j) \times (q^{(j-lastbit)})+\text{temp}
100 \text{ continue}
\text{value}=\text{temp}
\text{end}

**************************************************
* This subroutine determines whether or not
* the vector w is included among the errors
* to be corrected by the code.
**************************************************

subroutine errstat(w,nd,answer,n,q)
integer high,low,lookup,answer
byte w,q,z
integer n
byte syn,err
character*4 comparison
parameter(nerrs=5200,nsymbml=126,nsymb=127,nsymbpl=128)
dimension w(0:nsymbml),z(0:nsymbml)
dimension err(0:nerrs,0 :nsymbml)
dimension syn(0:nerrs,0 :nsymbml)
common syn,err
\text{low}=0
\text{high}=\text{nd-1}
127 \text{ if (abs(high-low).gt.1) then}
\quad \text{lookup}=((\text{high-low})/2)+\text{low}
\text{call geterr(lookup,z,n)}
\text{call comp(z,w,comparison,n)}
\quad \text{if (comparison.eq.'same') then}
\quad \text{answer}=1
\quad \text{goto 227}
\quad \text{elseif (comparison.eq.'less') then}
\quad \text{low}=\text{lookup}
\quad \text{goto 127}
elseif (comparison.eq.'more') then
  high=lookup
  goto 127
endif
else
  do 100 i2=-1,1
  call geterr(lookup+i2,z,n)
call comp(z,w,comparison,n)
    if (comparison.eq.'same') then
      answer=1
      goto 227
    endif
  100 continue
endif
answer=0
227 end

*******************************************************************************
* This subroutine determines the (first)index
* within the array err of a particular error
* vector w.
*******************************************************************************

subroutine locoferr(w,nd,answer,n,q)
  integer high, low, lookup, answer
  byte w, q, z
  byte syn, err
  integer n
  character*4 comparison
parameter(nerrs=5200, nsymbml=126, nsymb=127, nsymbpl=128)
dimension w(0:nsymbml), z(0:nsymbml)
dimension err(0:nerrs, 0:nsymbml)
dimension syn(0:nerrs, 0:nsymbml)
common syn, err
  low=0
high=nd-1
127 if (abs(high-low).gt.1) then
   lookup=((high-low)/2)+low
call geterr(lookup, z, n)
call comp(z,w,comparison,n)
    if (comparison.eq.'same') then
answer=lookup
goto 227
    elseif (comparison.eq.'less') then
    low=lookup
    goto 127
    elseif (comparison.eq.'more') then
    high=lookup
    goto 127
    endif
else
    do 100 i2=-1,1
    call geterr(lookup+i2,z,n)
call comp(z,w,comparison,n)
    if (comparison.eq.'same') then
    answer=lookup+i2
    goto 227
    endif
100 continue
endif
print*, 'error in locoferr',low,high,lookup,nd
227     end