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.
A covering space approach to \((d,k)\) constrained codes

Perry, Patrick Neil, Ph.D.

University of Hawai'i, 1992
A COVERING SPACE APPROACH
TO $(d, k)$ CONSTRAINED CODES

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF
THE UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MATHEMATICS

MAY 1992

By

Patrick Neil Perry

Dissertation Committee:

H. M. Hilden, Chairman
K. H. Dovermann
R. D. Little
L. J. Wallen
E. J. Weldon
Abstract

The capacity of the $(d, k)$ constrained codes and of the $(d, k)L$ level charge constrained codes is considered. The case of rational capacity is examined, and an error in the literature is corrected. This leads to an interesting $(0, 3)L = 4$ level charge constrained code. The error control for this code is done using the finite field $GF(3)$. A table of the ternary convolutional codes of greatest free distance is given for possible applications.

The topological properties of the $(d, k)$ constraint graphs are examined. The fundamental group of a constraint graph and covering spaces of a constraint graph are discussed. A constructive process for building a covering space of a constraint graph is given.

The construction of $(d, k)$ constrained block codes from covering spaces of the $(d, k)$ constraint graph is examined. Several types of block codes are introduced. The base point codes consist of the $(d, k)$ constrained sequences whose associated edge paths in the covering space are loops at a specified vertex. The parity point codes consist of the $(d, k)$ constrained sequences whose associated edge paths in the covering space connect two specified vertices. It is shown that an $[n, k]$ cyclic code can be constructed as a base point code for a $2^{n-k}$ fold covering of the $(0, \infty)$ constraint graph.
Systematic \((d, k)\) constrained block codes are constructed for detecting all single shift errors, drop in errors, and drop out errors. The average probability of an undetected error for the systematic \((d, k)\) constrained block codes is shown to decrease exponentially with the parity length times the capacity of a \((d, k)\) constrained code.
# Table of Contents

Abstract ......................................................... iii

List of Tables ................................................. vii

List of Figures ................................................ viii

1. Capacity for the \((d, k)\) Constrained Codes ......................... 1
   1.1 Introduction ............................................. 1
   1.2 Capacity .................................................. 1
   1.3 \((d, k)L\) Level Charge Constrained Codes .................... 7
   1.4 Rational Capacity ......................................... 13
   1.5 A \((0, 3)L = 4\) Level Charge Constrained Coding Scheme .... 26
   1.6 Conclusion ................................................. 31

2. Topological Properties of the \((d, k)\) Constraint Graphs .......... 32
   2.1 Introduction ................................................ 32
   2.2 The Fundamental Group of a \((d, k)\) Constraint Graph ............. 32
   2.3 Covering Spaces for the \((d, k)\) Constraint Graphs .............. 43
   2.4 Construction of Covering Spaces
       for the \((d, k)\) Constraint Graphs .......................... 50
   2.5 Conclusion ................................................. 60

3. \((d, k)\) Constrained Block Codes From Covering Spaces .......... 61
   3.1 Introduction ................................................. 61
3.2 Introduction to $(d, k)$ Constrained Block Codes .................. 61
3.3 Base Point Codes .................................................. 64
3.4 Parity Point Codes .................................................. 76
3.5 Error Check Codes .................................................. 82
3.6 Probability of an Undetected Error for the Systematic $(d, k)$ Constrained Block Codes .................. 89
3.7 Conclusion ......................................................... 96

Appendix: Algorithm to find the Free Distance of a Ternary Convolutional Codes .................. 98

Bibliography ......................................................... 102
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Encoding Table for (0, 1) Constrained Code</td>
<td>19</td>
</tr>
<tr>
<td>1.2 Next State Table for H Code</td>
<td>22</td>
</tr>
<tr>
<td>1.3 Output Table for H Code</td>
<td>23</td>
</tr>
<tr>
<td>1.4 Decoding Table for H Code</td>
<td>23</td>
</tr>
<tr>
<td>1.5 Possible Bit/Trit Converters</td>
<td>28</td>
</tr>
<tr>
<td>1.6 Possible Coding Rates Using Block Codes</td>
<td>29</td>
</tr>
<tr>
<td>1.7 Rate 1/2 Ternary Convolutional Codes With Largest $d_{\text{free}}$</td>
<td>30</td>
</tr>
<tr>
<td>1.8 Coding Rates When Using Rate 1/2 Ternary Convolutional Code</td>
<td>30</td>
</tr>
<tr>
<td>2.1 Maximal Trees for the (1, 3) Constraint Graph</td>
<td>37</td>
</tr>
<tr>
<td>2.2 Maximal Trees for the (2, $\infty$) Constraint Graph</td>
<td>37</td>
</tr>
<tr>
<td>2.3 Splitting Complexes for the (1, 3) Constraint Graph</td>
<td>39</td>
</tr>
<tr>
<td>2.4 Splitting Complexes for the (2, $\infty$) Constraint Graph</td>
<td>40</td>
</tr>
<tr>
<td>3.1 Parity Point Codes for $E$ Covering of the (0, 2) Constraint Graph</td>
<td>78</td>
</tr>
<tr>
<td>3.2 Error Check Encoder for the (1, 3) Constraints</td>
<td>87</td>
</tr>
<tr>
<td>3.3 Error Check Encoder for the (2, 7) Constraints</td>
<td>88</td>
</tr>
<tr>
<td>3.4 Error Check Encoder for the (1, 7) Constraints</td>
<td>89</td>
</tr>
</tbody>
</table>
List Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(1,3) Constraint Graph</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>(2,∞) Constraint Graph</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Diagram of Rectangular Wave Form</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>(1,3)L = 5 Level Charge Constraint Graph</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>(0,1)L = 2 Level Charge Constraint Graph</td>
<td>18</td>
</tr>
<tr>
<td>1.6</td>
<td>Paths of Length Two Graph</td>
<td>18</td>
</tr>
<tr>
<td>1.7</td>
<td>(0,3)L = 4 Level Charge Constraint Graph</td>
<td>20</td>
</tr>
<tr>
<td>1.8</td>
<td>Paths of Length Two Graph</td>
<td>21</td>
</tr>
<tr>
<td>1.9</td>
<td>Reduced Paths of Length Two Graph</td>
<td>22</td>
</tr>
<tr>
<td>1.10</td>
<td>Coding Scheme Using H Code</td>
<td>26</td>
</tr>
<tr>
<td>2.1</td>
<td>Covering Space of (0,∞) Constraint Graph</td>
<td>45</td>
</tr>
<tr>
<td>2.2</td>
<td>Covering Space of (1,3) Constraint Graph</td>
<td>45</td>
</tr>
<tr>
<td>2.3</td>
<td>Split Figure $G^K$</td>
<td>57</td>
</tr>
<tr>
<td>2.4</td>
<td>Four Fold Covering of (1,3) Constraint Graph</td>
<td>59</td>
</tr>
<tr>
<td>3.1</td>
<td>Conventional Coding Scheme</td>
<td>62</td>
</tr>
<tr>
<td>3.2</td>
<td>Three Fold Covering of (1,2) Constraint Graph</td>
<td>67</td>
</tr>
<tr>
<td>3.3</td>
<td>Eight Fold Covering of (0,2) Constraint Graph</td>
<td>69</td>
</tr>
<tr>
<td>3.4</td>
<td>Eight Fold Covering of (0,∞) Constraint Graph</td>
<td>71</td>
</tr>
</tbody>
</table>
3.5 Shift Register for Dividing by $g(x)$ ........................................ 75

3.6 Error Check Covering for (1,3) Constraint Graph ...................... 86

A.1 Algorithm for Finding $d_{\text{free}}$ of a Ternary Convolutional Code .......... 100
Chapter 1

Capacity for the \((d, k)\) Constrained Codes

1.1 Introduction

In this chapter, the capacity of the \((d, k)\) constrained codes is discussed. We derive an equation for the capacity of the \((d, k)\) \(L\) level charge constrained codes. The case of rational capacity is considered and we point out an error in the literature. This leads to an interesting new code for the \((0.3)\) \(L = 4\) level charge constraints. The last part of the chapter deals with possible implementations for this code.

1.2 Capacity

A \((d, k)\) constrained sequence is a binary sequence in which two consecutive ones are separated by at least \(d\) zeros and at most \(k\) zeros.

A \((d, k)\) constrained code is a mapping of binary sequences to the set of \((d, k)\) constrained sequences. These codes are also referred to in the literature as run-length limited codes. They have attracted much recent interest as they are important for magnetic and optical recording. The paper of Siegel provides a good introduction to the \((d, k)\) constrained codes. Other references are given in the bibliography.

Shannon showed that there is an upper bound for the rate of a constrained code. This bound is called the capacity of the code. In the literature, the term capacity has been used with some ambiguity. In [ZW], the capacity of the \((d, k)\) constrained
codes is found, in [S] and [TB], the capacity of the channel is found when using a
(d, k) constrained code, and in [AS], the capacity of the (d, k) constrained systems
is found. In each instance, they are talking about the same thing. We will use the
description "capacity of the (d, k) constrained codes" in this paper.

From Shannon, we see the capacity is given by \( c = \lim_{n \to \infty} \frac{\log N(n)}{n} \) where \( N(n) \)
is the number of constrained sequences of length \( n \). If the base of the logarithm is
taken to be \( b \), then we say that \( c \) is the base \( b \) capacity of the constrained code.

The capacity of the (d, k) constrained codes has been derived several times in
the literature. In [TB], the capacity of the (d, k) constrained codes is found by
counting the number of constrained sequences. In [ZW], the capacity is found
using a probabilistic approach. And in [ACH] it is found using the state transition
matrix of the (d, k) constraint graph. We follow the later approach to rederive the
capacity of the (d, k) constrained codes.

Associated with the (d, k) constrained sequences is a (d, k) constraint graph.
The (d, k) constraint graph has \( k + 1 \) vertices and \( 2k - d + 1 \) edges. Labeling the
vertices \( a_i \), \( 1 \leq i \leq k + 1 \), we get that the edges are:

\[
\begin{align*}
(1) & \quad \text{for } 1 \leq i \leq k, \quad a_i \to a_{i+1} \\
& \quad \text{for } d + 1 \leq i \leq k + 1, \quad a_i \to a_1
\end{align*}
\]

The edges of the graph that enter vertex \( a_1 \) are labeled with a one. All other edges
are labeled with zero. For example, the (1, 3) constraint graph is shown below. The
study of the \((d, k)\) constrained sequences using the associated constraint graph was introduced by Shannon.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{constraint_graph}
\caption{(1, 3) Constraint Graph}
\end{figure}

Let \(T\) be the state transition matrix for a \((d, k)\) constraint graph. The entry \(t_{ij}\) is one if there is an edge in the graph from the \(i^{th}\) vertex to the \(j^{th}\) vertex. Otherwise the entry is zero. The state transition matrix is a square matrix whose entries are nonnegative. For example, the state transition matrix for the \((1, 3)\) constraint graph is shown below.

\begin{equation}
T = \begin{pmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0
\end{pmatrix}
\end{equation}

We digress now to the Perron-Frobenius theory of nonnegative matrices. The necessary facts can be found in [V].

A nonnegative square matrix \(T\) is irreducible if for all \(i\) and \(j\), there is a positive integer \(n\) such that \(t_{ij}^{(n)} > 0\) where \(t_{ij}^{(n)}\) is the \(ij^{th}\) entry of \(T^n\). The state transition matrix for a \((d, k)\) constraint graph is an irreducible matrix.
The Perron-Frobenius theorem for nonnegative matrices, as found in [V], is shown below.

**Theorem 1. (Perron-Frobenius).** Let $T$ be an irreducible nonnegative square matrix. Then

1. $T$ has a positive real eigenvalue $\lambda$ with $\lambda \geq |\mu|$ for all other eigenvalues $\mu$ of $T$.

2. To $\lambda$ there corresponds an eigenvector $w$ whose entries are positive reals.

3. $\lambda$ is a simple eigenvalue of $T$.

The dominant real eigenvalue will be referred to as the Perron eigenvalue.

The base $b$ capacity of the $(d, k)$ constrained codes, according to Shannon, is equal to $\log_b \lambda$ where $\lambda$ is the Perron eigenvalue of the state transition matrix for the $(d, k)$ constraint graph.

We rederive the capacity for the $(d, k)$ constrained codes starting from the eigenvector equation $Tw = \lambda w$. We call the entry $w_i$ of the eigenvector, the weight of the state $i$. The eigenvector equation then says that the sum of the weights of the successors for a state will equal $\lambda$ times the weight of the state. Or alternatively, $\lambda^{-1}$ times the weight of the successors is equal to the weight of the state.

Starting from the first vertex of the $(d, k)$ constraint graph, and following all possible paths until they return to the first vertex gives the equation:

$$w_1 = \lambda^{-(d+1)}w_1 + \lambda^{-(d+2)}w_1 + \cdots + \lambda^{-(k+1)}w_1$$
From the Perron-Frobenius theorem, we know that $w_1$ is nonzero. Dividing by $w_1$ and rearranging gives the equation:

\begin{equation}
\lambda^{k+1} - \frac{\lambda^{k-d+1} - 1}{\lambda - 1} = 0
\end{equation}

The Perron eigenvalue is the largest real solution of this equation and the capacity of the $(d, k)$ constrained codes is the logarithm of this eigenvalue.

The same expression for the capacity of the $(d, k)$ constrained codes has appeared many times in the literature, for instance [ACH], [TB], [AS], and [NB].

If no $k$ constraint is imposed, we get the $(d, \infty)$ constraints. A binary sequence satisfies the $(d, \infty)$ constraints if two consecutive ones are separated by at least $d$ zeros. No limit is imposed on the number of consecutive zeros. According to Shannon, the capacity for the $(d, \infty)$ constrained codes can be calculated in the same way as above for the $(d, k)$ constrained codes. We start with the $(d, \infty)$ constraint graph and use the eigenvector equation for the state transition matrix to find the Perron eigenvalue of the state transition matrix. The $(d, \infty)$ constraint graph has $d+1$ vertices and $d+2$ edges. If we index the vertices from one to $d+1$, the edges will be:

\begin{equation}
\text{for } 1 \leq i \leq d, \quad a_i \rightarrow a_{i+1}
\end{equation}

\begin{equation}
\text{for } i = d+1, \quad a_{d+1} \rightarrow a_1 \text{ and } a_{d+1}
\end{equation}
The edge in the graph that enters vertex $a_1$ is labeled with a one. All other edges are labeled with zero. As an example of the $(d, \infty)$ constraint graph, the $(2, \infty)$ constraint graph is shown below.

![Figure 1.2: (2, \infty) Constraint Graph](image)

To solve the eigenvector equation, we use the fact that the weight of a state is equal to $\lambda^{-1}$ times the weight of its successors. Starting from the last state of the $(d, \infty)$ constraint graph, this becomes $w_{d+1} = \lambda^{-1}w_{d+1} + \lambda^{-(d+1)}w_{d+1}$. Rearranging gives the equation $\lambda^{d+1} - \lambda^d - 1 = 0$. The capacity of the $(d, \infty)$ constrained code is equal to the logarithm of the largest real solution of the equation. This has appeared before in [ACH] and [TB].

We note here that the capacity of the $(d, \infty)$ constrained codes is larger than the capacity of any $(d, k)$ constrained code for the same $d$. This follows as there are more sequences if no $k$ constraint is imposed than if there is a $k$ constraint imposed. We will use this fact later in the next section.
An important subclass of the $(d, k)$ constrained codes is the $(d, k)$ charge constrained codes. These codes have been previously considered in [AS] and [NB].

To motivate the definition of a charge constrained sequence, consider a $(d, k)$ constrained sequence $v$. Associated with $v$ is a rectangular wave form $I_v(t)$ which represents the current when transmitting the sequence $v$. The ones in $v$ represent the transitions of $I_v(t)$ and the zeros represent no transition. A diagram of this process is shown below. The clock is assumed to tick at integer intervals and the transitions to occur at times $k + \frac{1}{2}, k \in \mathbb{Z}$.

The charge that a capacitor accumulates from the wave form $I_v(t)$ is given by $\int_{t_0}^{t_1} I_v(t)dt$. It is very desirable for the accumulated charge to be bounded. As
noted in [AS], the accumulated charge being bounded ensures that the code has a spectral null at $dc$.

A $(d,k)$ constrained sequence satisfies the $L$ level charge constraint if there is a nonnegative integer $\tau \leq L$ such that for all $t_0, t_1$, the accumulated charge satisfies:

$$-\tau \leq \int_{t_0}^{t_1} I_v(t) dt \leq L + d - \tau$$

A $(d,k)$ constrained code is said to be a $(d,k)L$ level charge constrained code if there is a nonnegative integer $\tau \leq L$ such that all code sequences satisfy the $L$ level charge constraint for the integer $\tau$.

We can calculate the charge accumulation for a $(d,k)$ constrained sequence by calculating its running digital sum. This is noted in [AS]. To understand the running digital sum, consider $v$ a $(d,k)$ constrained sequence. Parse $v$ into blocks of length between $d+1$ and $k+1$, each block ending in a one. Let $b_i$ be the number of zeros in the $i^{th}$ block of the parsed sequence. The running digital sum for the sequence $v$ is then $\sum_{i=1}^{n} (-1)^i b_i$. This is useful, as for the rectangular wave form $I_v(t)$, we have that $\int_0^t I_v(t) dt = \sum_{i=1}^{n} (-1)^i b_i$ where $t_n$ is the number of digits in the first $n$ blocks of the parsed sequence. Thus, an equivalent condition for a sequence to satisfy the $L$ level charge constraint is for its running digital sum to be bounded between $-\tau$ and $L + d - \tau$.

We want to derive an equation for the capacity of the $(d,k)L$ level charge constrained codes. We use a directed graph for the $(d,k)L$ level charge constrained
codes, and follow the method used in the previous section for finding the capacity of the \((d, k)\) constrained codes.

The \((d, k)L\) level charge constraint graph will consist of vertices labeled with a double index \(i\) and \(j\). The index \(i\) corresponds to the state the sequence would be in for the \((d, k)\) constraint graph, and the \(j\) corresponds to the charge level. The indices are required to satisfy \(1 \leq i \leq k + 1\), \(1 \leq j \leq L + d\), and \(0 \leq j - i < L\). The edges for the graph are then:

\[
(7) \begin{align*}
    &\text{for } 1 \leq i \leq k, \quad a_{i,j} \rightarrow a_{i+1,j+1} \\
    &\text{for } d + 1 \leq i \leq k + 1 \quad a_{i,j} \rightarrow a_{1,L+d+1-j}
\end{align*}
\]

The edges of the graph that enter a state whose \(i\) index is one are labeled with a one. All other edges are labeled with a zero. As an example, the \((1, 3)L = 5\) level charge constraint graph is shown below.
Figure 1.4: \((1, 3)L = 5\) Level Charge Constraint Graph
It is clear that a sequence generated from the \((d, k)L\) level charge constraint graph, which starts in vertex \(a_{ij}\), will have its running digital sum bounded between \(-j\) and \(L + d - j\). Thus, the sequence will be a \((d, k)L\) level charge constrained sequence.

The capacity of the \((d, k)L\) level charge constrained codes, as it was for the \((d, k)\) constrained codes, will be the logarithm of the Perron eigenvalue for the state transition matrix of the constraint graph. An equation for the capacity of the \((d, k)L\) level charge constrained codes when \(L + d\) is even is derived in [NB]. We derive a similar equation for the \((d, k)L\) level charge constrained codes.

To find the Perron eigenvalue, we start with the eigenvector equation \(Tw = \lambda w\). This says that the weight of a state is equal to \(\lambda^{-1}\) times the weight of its successors. Using the \(L\) states whose \(i\) index is one, we get the set of \(L\) equations shown below.

Only the \(j\) index is used to denote the weight of the state \(w_{1,j}\):

\[
\begin{align*}
w_1 &= \lambda^{-(d+1)}w_L + \lambda^{-(d+2)}w_{L-1} + \cdots + \lambda^{-(k+1)}w_{L+d-k} \\
w_2 &= \lambda^{-(d+1)}w_{L-1} + \lambda^{-(d+2)}w_{L-2} + \cdots + \lambda^{-(k+1)}w_{L+d-k-1} \\
\vdots \\
w_{L-1} &= \lambda^{-(d+1)}w_2 + \lambda^{-(d+2)}w_1 \\
w_L &= \lambda^{-(d+1)}w_1
\end{align*}
\]

(8)

Writing these as a matrix equation gives \(w = A_\lambda w\), or \((A_\lambda - I)w = 0\). For this to occur, the determinant of \(A_\lambda - I\) must equal zero. We state this as a theorem.
Theorem 2. The capacity of the \((d, k)L\) level charge constrained codes will be equal to the logarithm of the largest real solution of the equation \(\det A = 0\). where the matrix \(A\) has its \(ij\)th entry as \(\lambda^{-(i+j+1-L)}f(i+j+1-L) - \delta_{ij}\) with \(\delta_{ij}\) the Kroenicker delta function and \(f(p) = \begin{cases} 1 & \text{for } d+1 \leq p \leq k+1 \\ 0 & \text{otherwise} \end{cases}\).

To illustrate the theorem, we work out as an example the capacity for the \((1, 3)L = 5\) level charge constrained codes. The constraint graph for the \((1, 3)L = 5\) level charge constraints is shown above. Using the five vertices whose \(i\) index is one and the fact that the weight of a state is equal to \(\lambda^{-1}\) times the weight of its successors, we get the five equations shown below:

\[
\begin{align*}
    w_1 &= \lambda^{-2}w_5 + \lambda^{-3}w_4 + \lambda^{-4}w_3 \\
    w_2 &= \lambda^{-2}w_4 + \lambda^{-3}w_3 + \lambda^{-4}w_2 \\
    w_3 &= \lambda^{-2}w_3 + \lambda^{-3}w_2 + \lambda^{-4}w_1 \\
    w_4 &= \lambda^{-2}w_2 + \lambda^{-3}w_1 \\
    w_5 &= \lambda^{-2}w_1
\end{align*}
\]

(9)

Combining these into a matrix equation, we get the equation \((A_\lambda - I)w = 0\).

This equation has a solution only when \(\det(A_\lambda - I) = 0\). This equation is shown below:

\[
\begin{vmatrix}
-1 & 0 & \lambda^{-4} & \lambda^{-3} & \lambda^{-2} \\
0 & \lambda^{-4} - 1 & \lambda^{-3} & \lambda^{-2} & 0 \\
\lambda^{-4} & \lambda^{-3} & \lambda^{-2} - 1 & 0 & 0 \\
\lambda^{-3} & \lambda^{-2} & 0 & -1 & 0 \\
\lambda^{-2} & 0 & 0 & 0 & -1
\end{vmatrix} = 0
\]

(10)
Working out the determinant gives us the equation \((1-2\lambda^{-2})(1+\lambda^{-2})(1-\lambda^{-4}) = 0\). The largest real solution of this is clearly \(\sqrt{2}\). The binary capacity for the \((1,3)L = 5\) level charge constrained codes is then \(c = \log_2 \sqrt{2} = \frac{1}{2}\).

We point out that for the cases where \(L + d\) is even, the equation for the Perron eigenvalue in [NB] reduces to the same equation as in the theorem.

1.4 Rational Capacity

We want to determine which \((d,k)L\) level charge constrained codes have rational capacity. This was considered in [AS], where the codes with \(L + d\) even are examined. We extend their results to all the \((d,k)L\) level charge constrained codes and point out an error in a statement of theirs. The development of this section follows that of [AS].

The key to determining if the \((d,k)L\) level charge constrained code has rational capacity is the relationship between the capacity of the constrained code and the period of the constraint graph. The period of a graph is equal to the greatest common divisor of the lengths of all the loops in the graph. We first determine the period of the \((d,k)L\) level charge constraint graphs.

Lemma 3. The period of the \((d,k)L\) level charge constraint graph divides two.

proof. Starting from the vertex \(a_{11}\), the path determined by the sequence of \(d\) zeros followed by a one, repeated twice, will be a loop in the graph.
Similarly, the sequence of \( d + 1 \) zeros followed by a one, repeated twice, will be a loop in the graph.

Since the period of the graph divides the length of each loop, we see that the period must divide \( 2(d + 1) \) and \( 2(d + 2) \). This occurs only if the period divides two. \( \square \)

**Theorem 4.** The \((d, k)L\) level charge constraint graph has period one if \( L + d \) is odd.

**proof.** Suppose \( L + d \) is odd. We will show that there is a loop in the constraint graph of odd length.

If \( L \) is odd, then consider the path in the graph starting from vertex \( a_{1,\lfloor \frac{L}{2} \rfloor + 1} \) and determined by the binary sequence of \( d \) zeros followed by a one. The path will be in vertex \( a_{d+1,\lfloor \frac{L}{2} \rfloor + d+1} \) after the \( d \) zeros. The one will send the path to the vertex \( a_{1,m} \) where \( m = L + d + 1 - (\lfloor \frac{L}{2} \rfloor + d + 1) = L - \lfloor \frac{L}{2} \rfloor = \lfloor \frac{L}{2} \rfloor + 1 \). Thus the path is a loop in the graph and its length is \( d + 1 \). The period of the graph must divide the length of any loop, so the period will divide \( d + 1 \). Since \( L \) is odd and \( L + d \) is odd, \( d + 1 \) will be odd. The lemma says that the period divides two, and we have shown that the period divides an odd number. Thus the period is one.

If \( L \) is even, then consider the path in the graph starting from the vertex \( a_{1,\frac{L}{2}} \) and determined by the binary sequence of \( d + 1 \) zeros followed by a one. The path will be in vertex \( a_{d+2,d+1+\frac{L}{2}} \) after the zeros and in vertex \( a_{1,m} \) after the one, where
Thus the path is a loop in the graph of length \( d + 2 \). Since \( L + d \) is odd and \( L \) is even, \( d + 2 \) is odd. Thus by the same reasoning as above, the period of the graph is one. \( \square \)

**Theorem 5.** The \((d, k)L\) level charge constraint graph has period two if \( L + d \) is even.

**proof.** Suppose that \( L + d \) is even. We show that all loops have even length.

Let \( V_0 \) be the set of vertices in the graph whose \( j \) index is even, and let \( V_1 \) be the set of vertices whose \( j \) index is odd. An edge in the graph connects two vertices in opposite sets. A path in the graph which starts from a vertex in \( V_0 \) will end in a vertex from \( V_1 \) if the path is odd length. Similarly, a path which starts in a vertex in \( V_1 \) will end in vertex from \( V_0 \) if the path is odd length. Thus, any loop must be of even length.

The lemma showed that the period divides two and we have shown that all loops have even length. Therefore, the period for the graph is two. \( \square \)

We digress again for a moment to the Perron-Frobenius theory for nonnegative matrices. The necessary facts are found in Varga. We condense them into the following theorem.

**Theorem 6.** The state transition matrix for a graph whose period is \( \rho \) will have \( \rho \) eigenvalues of maximum modulus. These eigenvalues will be of the form \( \lambda e^{\frac{2\pi i l}{\rho}} \).

for \( 0 \leq l \leq \rho - 1 \), where \( \lambda \) is the Perron eigenvalue of the state transition matrix.
We use this theorem to show the possible rational capacities for the $(d,k)L$ level charge constrained codes. The key lemma from Ashley-Siegel shows how the period of the constraint graph is related to the capacity of the constrained codes.

**Lemma 7. (Ashley-Siegel).** If the $(d,k)L$ level charge constrained code has rational capacity $\frac{k}{s}$ for any base $b$, then the period of the $(d,k)L$ level charge graph is a multiple of $s$.

**Proof.** Since $c = \frac{k}{s}$, the Perron eigenvalue of the state transition matrix $T$ will be $b^{\frac{k}{s}}$, where $b$ is the base of the capacity. This eigenvalue is a root of the polynomial $m(x) = x^s - b^t$, which is irreducible. So $m(x)$ must divide the characteristic polynomial of $T$. Thus, the roots of $m(x)$ are eigenvalues for $T$, where the roots are $b^{\frac{k}{s}} e^{i\frac{2\pi l}{s}}$ for $0 \leq l \leq s - 1$.

Theorem six shows that for a constraint graph with period $\rho$, there are $\rho$ eigenvalues of maximum modulus for the state transition matrix $T$. These eigenvalues being of the form $\lambda e^{i\frac{2\pi l}{\rho}}$, for $0 \leq l \leq \rho - 1$.

Equating these two facts, we see that for some $l_0$, $e^{i\frac{2\pi l_0}{s}} = e^{i\frac{2\pi l_0}{\rho}}$. This occurs only if $\frac{\rho}{l_0} = s$, thus showing the lemma. \(\square\)

We combine this lemma with the theorems for the period of the $(d,k)L$ level charge constraint graphs to show which rational capacities are possible.

**Theorem 8.** The only possible rational capacities for the $(d,k)L$ level charge constrained codes are binary rate $1/2$ and ternary rate $1/2$ when $L + d$ is even.
proof. If \( L + d \) is odd, the period of the \((d, k)L\) level charge constraint graph is one. So by the lemma, the only possible rational capacities are of the form \( m/1 \). Since this is greater than one, and the capacity of a constrained code is less than one, we see that no such codes with rational capacity can exist.

If \( L + d \) is even, the period of the \((d, k)L\) level charge constraint graph is two. So by the lemma, the possible rational capacities are of the form \( m/2 \). Only the case with \( m = 1 \) will give a capacity less than one.

If the base \( b \) capacity were \( 1/2 \), then the binary capacity will be \( \log_2 b^{1/2} \). Only the cases for \( b = 2 \) and \( b = 3 \) will have binary capacities less than one. \( \square \)

We work some examples of \((d, k)L\) level charge constrained codes which have rational capacity.

The first case we consider is for the \((0, 1)L = 2\) level charge constraints. The equation for determining the capacity of the \((0, 1)L = 2\) level charge constrained codes is found from theorem two to be:

\[
\text{det}\begin{vmatrix}
\lambda^{-2} - 1 & \lambda^{-1} \\
\lambda^{-1} & -1 \\
\end{vmatrix} = 0
\]  

(11)

The determinant in the equation reduces to \( 1 - 2\lambda^{-2} \) which has largest real root equal to \( \sqrt{2} \). The binary capacity is then equal to \( \log_2 \sqrt{2} = 1/2 \).

To get a \((0, 1)L = 2\) level charge constrained code, we examine the \((0, 1)L = 2\) level charge constraint graph. The constraint graph and the graph of paths of length two are shown below.
We can design an encoder by choosing a connected component of the paths of length two graph. Taking the component consisting of the single vertex, we get an encoder by labeling the 01 edge with 0 and labeling the 11 edge with 1. The encoding/decoding table for this code are shown below.
Table 1.1: Encoding Table for (0, 1) Constrained Code

<table>
<thead>
<tr>
<th>input bit</th>
<th>output block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

This code is called frequency modulation or phase encoding. As noted by [AS], it was an early standard in magnetic recording.

As a second example, we consider the (1, 3)L = 5 level charge constrained codes. As we showed earlier, the (1, 3)L = 5 level charge constrained codes will have binary capacity rate 1/2. Unfortunately, the method for designing an encoder used for the first example does not work for the (1, 3)L = 5 level charge constraints. In fact, there does not appear to be a simple way to encode for these constraints at rate 1/2.

As a final example, we consider the (0, 3)L = 4 level charge constrained codes. The equation for determining the capacity of the (0, 3)L = 4 level charge constrained codes is found from theorem two to be:

\[
\begin{vmatrix}
\lambda^{-4} - 1 & \lambda^{-3} & \lambda^{-2} & \lambda^{-1} \\
\lambda^{-3} & \lambda^{-2} - 1 & \lambda^{-1} & 0 \\
\lambda^{-2} & \lambda^{-1} & -1 & 0 \\
\lambda^{-1} & 0 & 0 & -1
\end{vmatrix} = 0
\]

(12)

The determinant in the equation reduces to \((1 - 3\lambda^{-2})(1 + \lambda^{-2})\), which has largest real root equal to \(\sqrt{3}\). The ternary capacity of the (0, 3)L = 4 level charge constrained codes is equal to \(\log_3 \sqrt{3} = \frac{1}{2}\).
The method used in the first example does work for the $(0, 3)L = 4$ level charge constrained codes to give an encoder. We start with the $(0, 3)L = 4$ level charge constraint graph and look at the graph of paths of length two. The constraint graph and the paths of length two graph are shown below.

Figure 1.7: $(0, 3)L = 4$ Level Charge Constraint Graph
Now notice that the vertices $a_{2,3}$ and $a_{3,3}$ have the same successors in the graph of paths of length two. Also note that each has the same binary pattern for the edges with the same terminal vertex. So, we merge the two vertices together. This gives the reduced paths of length two graph shown below.
This graph has three edges outgoing from each vertex and three edges incoming to each vertex. By using the ternary field $GF(3)$ we can design an encoder from the reduced graph. The encoder will be obtained by labeling the 01 edges with 1, labeling the 11 edges with 2, and labeling the 00 and 10 edges with 0. The resulting encoding tables are shown below. We will call this $(0, 3)L = 4$ level charge constrained code the $H$ code.

**Table 1.2: Next State Table for H Code**

<table>
<thead>
<tr>
<th>present state</th>
<th>input trit</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{11}$</td>
<td>$x$</td>
<td>$a_{13}$</td>
<td>$a_{11}$</td>
<td></td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$x$</td>
<td>$a_{11}$</td>
<td>$a_{13}$</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$x$</td>
<td>$a_{11}$</td>
<td>$a_{13}$</td>
<td></td>
</tr>
</tbody>
</table>
Table 1.3: Output Table for H Code

<table>
<thead>
<tr>
<th>input trit</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>present</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>00</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>10</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>$x$</td>
<td>10</td>
<td>01</td>
<td>11</td>
</tr>
</tbody>
</table>

By the choice of the labels, the decoder will be independent of the state that the sequence is in. This property will ensure that errors induced by the channel will not be propagated by the decoder. This is a very desirable property for practical applications. The decoder table for the code is shown below:

Table 1.4: Decoding Table for H Code

<table>
<thead>
<tr>
<th>Input Ternary</th>
<th>Ternary Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

In the final section of this chapter, we will examine possible applications of the H code.

We note that the (0, 3) charge constrained codes have been previously considered in the literature. The article of Patel describes a (0, 3) charge constrained code constructed in a manner different from the H code. The code of Patel turns out to be a (0, 3)$L = 10$ level charge constrained code.
These three codes are the only \((d, k)L\) level charge constrained codes which have rational capacity for any base. This generalizes the main result from Ashley-Siegel.

**Theorem 9.** The only \((d, k)L\) level charge constrained codes which have rational capacity are the \((0, 1)L = 2\) and the \((1, 3)L = 5\) level charge constrained codes at binary capacity rate \(1/2\) and the \((0, 3)L = 4\) level charge constrained code at ternary rate \(1/2\).

**proof.** The examples above show that each code has the capacity claimed by the theorem.

Recall that the capacity of the \((d, \infty)\) constrained codes will always be larger than the capacity of any \((d, k)L\) level charge constrained code with the same \(d\). We use this to show that no other \((d, k)L\) level charge constrained codes with rational capacity exist.

First consider the capacity of the \((1, \infty)\) constrained codes. The polynomial for the Perron eigenvalue of the \((1, \infty)\) constraint graph state transition matrix is \(\lambda^2 - \lambda - 1\). It is easily checked that the largest real root of this polynomial is less than \(\sqrt{3}\). This implies that no \((d, k)L\) level charge constrained code for \(d \geq 1\) can have ternary capacity rate \(1/2\). The case where \(d = 0\) gives the \((0, 3)L = 4\) level charge constrained code shown above. This shows that no other \((d, k)L\) level charge constrained codes exist with ternary capacity rate \(1/2\).
Now consider the capacity of the \((3, \infty)\) constrained codes. The polynomial for the Perron eigenvalue of the \((3, \infty)\) constraint graph state transition matrix is 
\[
\lambda^4 - \lambda^3 - 1.
\]
It is easily checked that the largest real root of this polynomial is less than \(\sqrt{2}\). This implies that no \((d, k)\) level charge constrained code for \(d \geq 3\) can have binary capacity rate \(1/2\). The cases where \(d = 0\) and \(d = 1\) give the \((0, 1)\) level charge constrained code mentioned above.

The case where \(d = 2\) must be considered separately. If \(L\) is odd, then by the theorem eight there will be no \((2, k)\) level charge constrained code with rational capacity. If \(L\) is even, then by examining the table from [NB], we see that no \((2, k)\) level charge constrained code with binary capacity rate \(1/2\) exists. \(\square\)

This completes showing all the \((d, k)\) level charge constrained codes which have rational capacity. In [AS], they consider the \((d, k)\) level charge constrained codes for which \(L + d\) is even. They use the notation \("(d, k; c) charge constrained system\)" to represent what we are calling the \((d, k)\) level charge constrained code. The theorem corrects a statement in the article [AS], where it is claimed that "the only \((d, k; c)\) charge constrained systems with rational base \(b\) capacity are the \((0, 1; 1)\) and \((1, 3; 3)\) charge constrained systems where the base is a power of two." The additional code for the \((0, 3)\) level charge constraints corrects their statement.
We examine possible applications using the H code. The encoding and decoding tables for this code were shown above. The encoder takes a ternary sequence and outputs a binary sequence that satisfies the \((0, 3)L = 4\) level charge constraints. In typical applications, the information to be encoded is binary, not ternary. Thus to use the H code, we must first convert the binary sequence into a ternary sequence. Then the H code can be implemented to give a binary \((0, 3)L = 4\) level charge constrained sequence. A coding scheme for this process is shown below.

Notice that the error-control code can be put between the bit-trit converter and the H code encoder. The error-control code will be a code using the ternary field \(GF(3)\). Recall that we earlier showed that the H code does not propagate errors.
Thus, errors in the channel will reach the error-control code without effecting the adjacent trits in the ternary sequence. This is a very desirable property.

First we consider the bit-trit converter. We want an arbitrary binary sequence to have a representation as a ternary sequence. This will be possible at a rate of \( \frac{p}{q} \) as long as there are more binary sequences of length \( p \) than there are ternary sequences of length \( q \). The limiting rate, as we prove below, for the bit-trit converter will be \( \log_2 3 \).

**Lemma 10.** A rate \( \frac{p}{q} \) bit-trit converter exists for all \( \frac{p}{q} \) less than \( \log_2 3 \).

**Proof.** If \( \frac{p}{q} \leq \log_2 3 \), then \( 2^p \leq 3^q \). This says that there are more ternary sequences of length \( q \) than there are binary sequences of length \( p \). This is exactly what is needed for there to be a bit-trit converter of rate \( \frac{p}{q} \). \( \square \)

So, the limiting rate for a bit-trit converter will be \( \log_2 3 = 1.58496 \). For practical purposes, it is necessary to keep \( p \) and \( q \) small. Consider if we use a bit-trit converter that converts binary sequences of length \( p \) into ternary sequences of length \( q \). If an uncorrected error occurs in the ternary sequence, then this error will affect the entire length \( p \) binary sequence. Thus the \( p \) represents the length an uncorrected error can propagate in the output sequence.
Possibilities for the bit-trit converter with small $p$ and $q$ are:

<table>
<thead>
<tr>
<th>$p/q$</th>
<th>$p$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3/2 = 1.5$</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$11/7 = 1.57$</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>$19/12 = 1.58$</td>
<td>19</td>
<td>12</td>
</tr>
</tbody>
</table>

We now consider possible ternary error-control codes. There are two well-developed types of error-control codes, block codes and convolutional codes. We examine the possibilities for both types of codes.

A block code partitions the information sequence into blocks of length $k$. Each block is encoded independently into a block of length $n$. See [SM], [LC], or [PW] for an introduction to block codes.

The rate using the coding scheme will always be less than the capacity of the $(0, 3)L = 4$ level charge constrained code, which in binary terms is $\log_2 \sqrt{3} = .7925$.

Suppose that in the coding scheme, we use a rate $\frac{p}{q}$ bit-trit converter and a rate $\frac{k}{n}$ block code. For $pk$ input bits, the bit-trit converter outputs $qk$ trits. The ternary error-control code takes the $qk$ trits, and outputs $qn$ trits. The H code encoder takes the $qn$ trits, and sends $2qn$ bits to the channel. The rate of the coding scheme is then $\frac{pk}{2qn}$.

For comparison, we present the rate for the coding scheme using various combinations of bit-trit converters and ternary block codes. The rates were found using the formula above.
Table 1.6: Possible Coding Rates Using Block Codes

<table>
<thead>
<tr>
<th>Ternary Block Code</th>
<th>Bit/Trit Converter</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Golay [11, 6, 5]</td>
<td>3/2</td>
<td>.409</td>
</tr>
<tr>
<td></td>
<td>11/7</td>
<td>.428</td>
</tr>
<tr>
<td></td>
<td>19/12</td>
<td>.431</td>
</tr>
<tr>
<td>Hamming [26, 23, 3]</td>
<td>3/2</td>
<td>.663</td>
</tr>
<tr>
<td></td>
<td>11/7</td>
<td>.695</td>
</tr>
<tr>
<td></td>
<td>19/12</td>
<td>.700</td>
</tr>
</tbody>
</table>

The other type of commonly used error-control codes are the convolutional codes. Convolutional codes are popular because simple encoding and decoding techniques exist. A rate \( \frac{k}{n} \) convolutional code partitions the input sequence into blocks of length \( k \) and outputs a block of length \( n \), formed as a linear combination of the input block and the \( m \) previous input blocks. For an introduction to convolutional codes, see [LC] or [PW].

There does not appear to have been any published list of ternary convolutional codes in the literature. This can be explained since most practical situations use binary codes. For possible applications of the H code, it desirable to have a table of ternary convolutional codes. The table below shows the rate 1/2 noncatastrophic ternary convolutional codes of largest free distance. These codes were found using a computer search that implements a modified version of the algorithm in [L]. In the appendix, the algorithm is discussed.
Table 1.7: Rate 1/2 Ternary Convolutional Codes With Largest \( d_{\text{free}} \)

<table>
<thead>
<tr>
<th>Memory</th>
<th>( g_1 )</th>
<th>( g_2 )</th>
<th>( d_{\text{free}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(1, 2, 2)</td>
<td>(1, 1, 1)</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>(1, 1, 1, 2)</td>
<td>(1, 0, 1, 1)</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>(1, 1, 1, 2, 1)</td>
<td>(1, 0, 2, 1, 1)</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1, 2, 2,1)</td>
<td>(1, 0, 1, 1, 2,2)</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>(1, 1, 1, 0, 2,1, 1)</td>
<td>(1, 0, 2, 1, 2,2,1)</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>(1, 0, 2, 1, 1,0, 2,2)</td>
<td>(1, 0, 1, 2,1,1)</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>(1, 1, 0, 2,2,1,1,1,2)</td>
<td>(1, 0, 2, 1,2,0,2,1,1)</td>
<td>15</td>
</tr>
</tbody>
</table>

Suppose that in the coding scheme, we use a rate \( \frac{p}{q} \) bit-trit converter and a rate 1/2 ternary convolutional code. For \( p \) input bits, the bit-trit converter outputs \( q \) trits. The ternary convolutional code takes the \( q \) trits and outputs \( 2q \) trits. The H code takes the \( 2q \) trits, and sends \( 4q \) bits to the channel. The rate of the coding scheme using a rate \( \frac{p}{q} \) bit-trit converter and a rate 1/2 ternary convolutional code is then \( \frac{p}{4q} \).

Possible rates for the coding scheme using a rate \( \frac{p}{q} \) bit-trit converter and a rate 1/2 ternary convolutional code are presented in the table below:

Table 1.8: Coding Rates When Using Rate 1/2 Ternary Convolutional Code

<table>
<thead>
<tr>
<th>Bit/Trit Converter</th>
<th>Rate of Coding Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/2</td>
<td>.375</td>
</tr>
<tr>
<td>11/7</td>
<td>.392</td>
</tr>
<tr>
<td>19/12</td>
<td>.395</td>
</tr>
</tbody>
</table>
1.6 CONCLUSION

In this chapter, we have introduced the \((d, k)\) constrained codes and the \((d, k)L\) level charge constrained codes. An equation is derived to compute the capacity of the \((d, k)L\) level charge constrained codes. It is shown that the only \((d, k)L\) level charge constrained codes with rational capacity are the \((0, 1)L = 2\) and \((1, 3)L = 5\) level charge constrained codes at binary capacity rate \(\frac{1}{2}\) and the \((0, 3)L = 4\) level charge constrained code at ternary capacity rate \(\frac{1}{2}\). The last section of the chapter examines possible applications of the \((0.3)L = 4\) level charge constrained code.
Chapter 2

Topological Properties of
the \((d, k)\) Constraint Graphs

2.1 INTRODUCTION

In this chapter, we discuss the topological properties of the \((d, k)\) constraint graphs. The chapter is intended as an introduction, for the nonexpert, to some basic topological facts. In the first section, the fundamental group is defined. In the second section, a covering space is defined and some basic properties of covering spaces are examined. In the last section of the chapter, a constructive proof is given for building a covering space for a \((d, k)\) constraint graph.

2.2 THE FUNDAMENTAL GROUP OF A \((d, k)\) CONSTRAINT GRAPH

The \((d, k)\) constraint graph has \(k + 1\) vertices and \(2k - d + 1\) edges. Labeling the vertices, \(a_i, 1 \leq i \leq k + 1\), the edges are:

\[
\begin{align*}
\text{for } 1 \leq i \leq k, & \quad a_i \rightarrow a_{i+1} \\
\text{for } d + 1 \leq i \leq k + 1, & \quad a_i \rightarrow a_1
\end{align*}
\]

The edges in the graph that enter vertex \(a_1\) are labeled with a one. All other edges are labeled with zero. The \((d, k)\) constraint graphs were introduced in the previous chapter.
We also want to consider the \((d, \infty)\) constraint graphs. The \((d, \infty)\) constraint graph has \(d + 1\) vertices and \(d + 2\) edges. Labeling the vertices \(a_i, 1 \leq i \leq d + 1\), the edges are:

\[
\begin{align*}
\text{for } 1 \leq i \leq d, & \quad a_i \rightarrow a_{i+1} \\
\text{for } i = d + 1, & \quad a_{d+1} \rightarrow a_1 \text{ and } a_{d+1}
\end{align*}
\]

(2)

The edge in the graph that enters vertex \(a_1\) is labeled with a one. All other edges are labeled with zero. The \((d, \infty)\) constraint graphs were introduced in the previous chapter.

We will use the term "constraint graph" to mean a \((d, k)\) constraint graph or a \((d, \infty)\) constraint graph. We will work only with constraint graphs in this chapter, but we note that the development works for finite graphs.

For an edge in a constraint graph, we use \(e^{-1}\) to represent the reversed edge of the edge \(e\). That is, if \(e\) is an edge from vertex \(a_i\) to vertex \(a_j\), then the reversed edge is the edge from vertex \(a_j\) to vertex \(a_i\) obtained by reversing the direction of the edge \(e\). We call \(e\) a proper edge and \(e^{-1}\) a reversed edge.

An edge path in a constraint graph is a sequence of edges, \(\rho = (\rho_1, \rho_2, \ldots, \rho_n)\), where each \(\rho_i\) is a proper edge or a reversed edge in the constraint graph \(G\), and for each \(i\), the terminal vertex of \(\rho_i\) is the initial vertex of \(\rho_{i+1}\). If \(u\) is the initial vertex of \(\rho_1\) and \(v\) is the terminal vertex of \(\rho_n\), then we say \(\rho\) is an edge path from \(u\) to \(v\). An edge path from a vertex \(u\) to itself will be called a loop at the vertex \(u\).
A reduced edge path will be an edge path such that for each $i$, the edge $p_i$ is not the reversed edge of the edge $p_{i+1}$. An edge path $p = (p_1, p_2, \ldots, p_n)$ which is not a reduced edge path has an associated reduced edge path. If $p_i$ is the reversed edge of the edge $p_{i+1}$, then a reduction of $p$ can be done by removing the edges $p_i$ and $p_{i+1}$ from the edge path. This reduction process is repeated on the resulting edge path until a reduced edge path is reached. The edge path obtained will be called the reduced edge path associated with the edge path $p$. We note that the reduced edge path associated with an edge path is unique.

If each edge $p_i$ from the edge path $p$ is a proper edge from the constraint graph, then we say that $p$ is a proper edge path for the constraint graph. For a proper edge path $p = (p_1, p_2, \ldots, p_n)$, there is an associated $(d, k)$ constrained sequence $s = (s_1, s_2, \ldots, s_n)$, where $s_i$ is the binary label of the edge $p_i$ in the constraint graph. In the next chapter, we will construct $(d, k)$ constrained codes using the set of proper edge paths for the $(d, k)$ constraint graph. Note that a proper edge path for a constraint graph will be a reduced edge path.

The length of an edge path $p = (p_1, p_2, \ldots, p_n)$ will be the number of edges in the edge path. For instance, the length of $p$ is $n$.

Let $u$ be a vertex from the constraint graph $G$. Let $\omega_u$ be the set of all finite length reduced edge paths which are loops at the vertex $u$. We define a product on $\omega_u$ which makes $\omega_u$ into a group.
Suppose $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$ and $\tau = (\tau_1, \tau_2, \ldots, \tau_m)$ are elements of $\omega_u$. To define the product $\rho\tau$, we find the smallest integer $i$ such that the edge $\tau_i$ is not the reversed edge of the edge $\rho_{n+1-i}$. The product $\rho\tau$ is then defined to be $(\rho_1, \ldots, \rho_{n+1-i}, \tau_i, \ldots, \tau_m)$. It can be verified that the product $\rho\tau$ is a reduced edge path which is a loop at $u$. Thus $\rho\tau$ is an element of $\omega_u$. To show that $\omega_u$ is a group with this product, we have to show that there is an identity element and that for each $\rho$ in $\omega_u$, there is an inverse element $\rho^{-1}$ in $\omega_u$.

Consider the empty sequence. This represents the loop at $u$ which never leaves the vertex $u$. We use $1$ to represent the empty sequence. It is easily verified that $1$ is an identity for the set $\omega_u$ with the product defined.

For an element $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$, consider the element $\rho^{-1} = (\rho_n^{-1}, \ldots, \rho_1^{-1})$, where $\rho_i^{-1}$ is the reversed edge of the edge $\rho_i$. The product of $\rho$ and $\rho^{-1}$ will then be equal to empty sequence $1$, showing that $\rho^{-1}$ is an inverse for $\rho$ in $\omega_u$.

The edge paths $(\rho\tau)\sigma$ and $\rho(\tau\sigma)$ will both be equal to the reduced edge path associated with the edge path obtained by concatenating $\tau$ onto $\rho$ and then concatenating $\sigma$ onto the resulting edge path. Since the reduced edge path is unique, the associative property holds.

This suffices to show that $\omega_u$ is a group with the product defined. This group is called the fundamental group of the constraint graph at the vertex $u$. The fundamental group of the constraint graph at the vertex $u$ will be denoted as $\pi(G, u)$. The fundamental group of a space is well known in topology. For an
introduction to the fundamental group presented in greater generality, the reader is referred to [M] or [GH].

We want to show that the fundamental group of a constraint graph is a free group. It is a well known fact that the fundamental group of any graph is a free group. See [M] for a proof of this. We rederive the fundamental group of a \((d, k)\) constraint graph. The development of this section follows that in [M].

We start out by determining what a maximal tree for a constraint graph is. A tree is a connected graph with one more vertex than edge. A maximal tree for a constraint graph is a subgraph which is a tree and is contained in no other subgraph that is a tree. It turns out to be easy to determine if a tree is a maximal tree for a constraint graph.

**Theorem 1.** Let \(G\) be a constraint graph and let \(t\) a subgraph of \(G\) which is a tree. Then \(t\) is a maximal tree for \(G\) if and only if \(t\) contains all the vertices of \(G\).

**proof.** If \(t\) contains all the vertices of \(G\), then any additional edge would connect two vertices of \(t\). Thus the additional edge would give a graph which is not a tree. So, the subgraph \(t\) is a maximal tree for \(G\).

If \(t\) does not contain a vertex \(v\) in \(G\), then let \(P\) be the set of edge paths in \(G\) that connect a vertex of \(t\) to the vertex \(v\). Since \(G\) is connected, we know that \(P\) is a nonempty set. Let \(\rho\) be the edge path in \(P\) of shortest length. Let \(X\) be the subgraph of \(G\) whose edges are the edges from the edge path \(\rho\). It is easily verified
that the subgraph \( X \cup t \) will be a tree which contains \( t \). Thus \( t \) is contained in another subgraph which is a tree. So \( t \) is not a maximal tree for \( G \). This completes the proof of the theorem. \( \square \)

There are typically several different maximal trees for a constraint graph. The maximal trees for the \((1, 3)\) constraint graph and for the \((2, \infty)\) constraint graph are shown below as examples:

### Table 2.1: Maximal Trees for the \((1, 3)\) Constraint Graph

<table>
<thead>
<tr>
<th>Tree</th>
<th>Maximal Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( {a_1a_2, a_2a_3, a_3a_4} )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( {a_1a_2, a_2a_3, a_4a_1} )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( {a_1a_2, a_3a_4, a_4a_1} )</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( {a_2a_3, a_3a_4, a_4a_1} )</td>
</tr>
</tbody>
</table>

### Table 2.2: Maximal Trees for the \((2, \infty)\) Constraint Graph

<table>
<thead>
<tr>
<th>Tree</th>
<th>Maximal Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( {a_1a_2, a_2a_3} )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( {a_1a_2, a_3a_1} )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( {a_2a_3, a_3a_1} )</td>
</tr>
</tbody>
</table>

For the \((1, 3)\) constraint graph there are four maximal trees and for the \((2, \infty)\) constraint graph there are three maximal trees. It is easily verified that the \((d, k)\) constraint graph will have \( k+1 \) maximal trees and that the \((d, \infty)\) constraint graph will have \( d + 1 \) maximal trees.

We consider the fundamental group of a tree. Let \( t \) be a tree and \( u \) a vertex of \( t \). Consider a nonempty loop \( \rho \) at the vertex \( u \). We show that the edge path
\( \rho \) can be reduced. Let \( v \) be the vertex in \( t \) incident to the edge path \( \rho \) which is the largest number of edges away from the vertex \( u \) in \( t \). Then the sequence of edges \( (\rho_1, \rho_2, \ldots, \rho_n) \) that make up the edge path \( \rho \) will have some \( i \) such that the edge \( \rho_i \) has \( v \) as its terminal vertex, the edge \( \rho_{i+1} \) has \( v \) as its initial vertex, and the edge \( \rho_{i+1} \) is the reversed edge of the edge \( \rho_i \). By the reduction process, we can remove these two edges from the original edge path \( \rho \). Since no assumptions were made about the original edge path \( \rho \), and we showed that it can be reduced, we can continue the reduction process until the empty sequence is reached. This shows that the only reduced edge path which is a loop at the vertex \( u \) is the empty sequence. Recall that the fundamental group \( \pi(t, u) \) is the set of reduced edge paths which are loops at the vertex \( u \). So, we have shown that the fundamental group \( \pi(t, u) \) is the trivial group with one element. We summarize this below.

**Theorem 2.** The fundamental group \( \pi(t, u) \) for a tree \( t \) is the trivial group with one element.

Let \( t \) be a maximal tree for the constraint graph \( G \). The edges of \( G \) which are not contained in \( t \) are important in determining the fundamental group of the constraint graph. We use the term “splitting complex” to represent the edges of \( G \) which are not contained in the maximal tree \( t \). The motivation for this term will become clear in the last section of this chapter. We formalize the definition of a splitting complex with the definition below.
Definition 1. Let $M$ be the set of midpoints of the edges in the constraint graph $G$. A splitting complex $K$ for $G$ will be a subset of $M$ such that the subgraph of $G$ consisting of the edges which do not intersect $K$ is a maximal tree for $G$.

For each point $\lambda$ of the splitting complex $K$, there is an edge $e_\lambda$ which contains the point $\lambda$ as its midpoint. The edge $e_\lambda$ is not contained in the maximal tree associated with the splitting complex $K$ that contains $\lambda$. The points of the splitting complex index the edges of the constraint graph which are not contained in the maximal tree associated with the splitting complex $K$.

As an example, all the splitting complexes for the $(1,3)$ constraint graph and for the $(2,\infty)$ constraint graph are shown below:

Table 2.3: Splitting Complexes for the $(1,3)$ Constraint Graph

\[
\begin{align*}
K_1 &= \{\delta, \epsilon, \psi\} \\
K_2 &= \{\delta, \epsilon, \gamma\} \\
K_3 &= \{\delta, \epsilon, \alpha\} \\
K_4 &= \{\delta, \epsilon, \beta\}
\end{align*}
\]

The point $\alpha$ is the midpoint of the edge $a_1a_2$, the point $\beta$ is the midpoint of the edge $a_2a_3$, the point $\gamma$ is the midpoint of the edge $a_3a_4$, the point $\delta$ is the midpoint of the edge $a_2a_1$, the point $\epsilon$ is the midpoint of the edge $a_3a_1$, and the point $\psi$ is the midpoint of the edge $a_4a_1$. 

39
Table 2.4: Splitting Complexes For the $(2, \infty)$ Constraint Graph

- $K_1 = \{\gamma, \delta\}$
- $K_2 = \{\gamma, \beta\}$
- $K_3 = \{\gamma, \alpha\}$

The point $\alpha$ is the midpoint of the edge $a_1a_2$, the point $\beta$ is the midpoint of the edge $a_2a_3$, the point $\delta$ is the midpoint of the edge $a_3a_1$, and the point $\gamma$ is the midpoint of the loop at $a_3$.

There are four splitting complexes for the $(1, 3)$ constraint graph and there are three splitting complexes for the $(2, \infty)$ constraint graph. Since each splitting complex is associated with a maximal tree, we have from the results above for the maximal trees in a constraint graph that there are $k + 1$ splitting complexes for the $(d, k)$ constraint graph and that there are $d + 1$ splitting complexes for the $(d, \infty)$ constraint graph.

From theorem one, we see that a maximal tree for a $(d, k)$ constraint graph has $k$ edges and that a maximal tree for a $(d, \infty)$ constraint graph has $d$ edges. Since the $(d, k)$ constraint graph has $2k - d + 1$ edges, we see that a splitting complex for the $(d, k)$ constraint graph has $(2k - d + 1) - k = k - d + 1$ points. Since the $(d, \infty)$ constraint graph has $d + 2$ edges, we see that a splitting complex for the $(d, \infty)$ constraint graph has $(d + 2) - d = 2$ points.
We summarize the results of the two previous paragraphs in the theorem below.

**Theorem 3.** The \((d, k)\) constraint graph has \(k + 1\) splitting complexes, with each splitting complex containing \(k - d + 1\) points. The \((d, \infty)\) constraint graph has \(d + 1\) splitting complexes, each splitting complex containing two points.

We want to show how the fundamental group of a constraint graph is related to a splitting complex for the graph. Let \(K\) be a splitting complex for the constraint graph \(G\) and let \(t\) be the maximal tree associated with \(K\). For each point \(\lambda\) in \(K\), let \(e_\lambda\) be the edge in \(G\) that contains \(\lambda\) as its midpoint. Let \(A_\lambda\) be the edge path in the maximal tree \(t\) of smallest length from the vertex \(u\) to the initial vertex of \(e_\lambda\). Similarly, let \(B_\lambda\) be the edge path in \(t\) of smallest length from the terminal vertex of \(e_\lambda\) to the vertex \(u\). We define the loop \(\alpha_\lambda\) at \(u\) to be edge path consisting of following \(A_\lambda\), then following \(e_\lambda\) and then following \(B_\lambda\). The loop \(\alpha_\lambda\) will be called the loop at \(u\) associated with the point \(\lambda\) of the splitting complex \(K\). By the construction of \(\alpha_\lambda\), the loop will be a reduced edge path for each point of the splitting complex.

The theorem below shows how the fundamental group is related to a splitting complex. The fundamental group turns out to be a free group.

A free group on a set \(\{x_i\}\) is the set of formal products \(x_{i_1}^{e_1} \ldots x_{i_m}^{e_m}\) where each \(e_i\) is 1 or \(-1\) and for all \(j\) with \(i_j = i_{j+1}\), we have \(e_j = e_{j+1}\). Such a product is called a reduced word. The product of two elements \(x\) and \(y\) is obtained by placing
the formal products together and then removing from the expression all pairs of
the form $x_i x_i^{-1}$ and $x_i^{-1} x_i$. The reduced word obtained is unique and is called the
product of $x$ and $y$. Free groups are discussed in [M] and in [J].

**Theorem 4.** The fundamental group $\pi(G, u)$ for a constraint graph $G$ is a free
group. If $K$ is a splitting complex for $G$, then the generators of $\pi(G, u)$ can be
taken to be $\{\alpha_\lambda | \lambda \in K\}$ where $\alpha_\lambda$ is the loop at $u$ associated with the point $\lambda$ in
the splitting complex $K$.

**Sketch of proof.** A detailed proof of this theorem can be found in [M]. For the reader
who is not familiar with topology, their proof may be difficult to understand. We
present a sketch of the proof here.

The basic idea is to show that the fundamental group $\pi(G, u)$ is isomorphic to
the free group generated by the set $\{\alpha_\lambda | \lambda \in K\}$. For any loop $\rho = (\rho_1, \ldots, \rho_n)$
at the vertex $u$, let $\rho_i, \rho_i, \ldots, \rho_i$, with $i_1 < i_2 < \cdots < i_t$, be the edges of $\rho$
which intersect the splitting complex $K$. We define $\epsilon_{ij}$ to be $-1$ if the edge $\rho_{ij}$ is
a reversed edge in the constraint graph and to be $1$ if the edge $\rho_{ij}$ is a proper edge
in the constraint graph. Let $\lambda_j$ be the point of the splitting complex $K$ which is
the midpoint of the edge $\rho_{ij}$.
Now we define the mapping:

\[ \phi : \pi(G, u) \rightarrow \langle \{ \alpha_\lambda \mid \lambda \in K \} \rangle \]

by:

\[ \phi(\rho) = \alpha_\lambda^{e_1} \alpha_\lambda^{e_2} \cdots \alpha_\lambda^{e_n} \]

The details of the proof go on to show that the mapping \( \phi \) is an isomorphism. This shows that the fundamental group is the free group generated by the set \( \{ \alpha_\lambda \mid \lambda \in K \} \). \( \square \)

Theorems three and four imply that the fundamental group of a \( (d, k) \) constraint graph is a free group with \( k - d + 1 \) generators and that the fundamental group of a \( (d, \infty) \) constraint graph is a free group with two generators.

2.3 COVERING SPACES FOR THE \( (d, k) \) CONSTRAINT GRAPHS

A graph mapping \( p \) is a map of a graph \( E \) onto a graph \( G \) such that \( p \) maps the vertices of \( E \) onto the vertices of \( G \), \( p \) maps the edges of \( E \) onto the edges of \( G \), and for each edge \( \bar{e} \) in \( E \), if \( w \) is the initial vertex of \( \bar{e} \) and \( z \) is the terminal vertex of \( \bar{e} \), then \( p(w) \) is the initial vertex of \( p(\bar{e}) \) and \( p(z) \) is the terminal vertex of \( p(\bar{e}) \).

Suppose \( p \) is a graph mapping of the graph \( E \) onto the graph \( G \). If \( u \) is a vertex of \( G \), then the set \( p^{-1}(u) \) is the set of vertices in \( E \) which are mapped by \( p \) onto the vertex \( u \). Similarly, if \( e \) is an edge of \( G \), then the set \( p^{-1}(e) \) is the set of edges in \( E \) which are mapped by \( p \) onto the edge \( e \).
We want to define a covering space of a constraint graph. Covering spaces are used extensively in topology. For an introduction to covering spaces, see [M] or [GH].

Definition 2. Let $G$ be a finite graph. A graph $E$ is a covering space of the graph $G$ if there is a graph mapping $p$ of $E$ onto $G$ that satisfies:

1. for each vertex $u$ in $G$, each vertex in the set $p^{-1}(u)$ has the same number of incoming and outgoing edges as the vertex $u$.

2. for each edge $e$ in $G$, no two edges in the set $p^{-1}(e)$ have the same initial vertex in $E$, and no two edges in the set $p^{-1}(e)$ have the same terminal vertex in $E$.

In this section, we want to examine some properties of covering spaces of constraint graphs. Before proceeding, we consider some examples of covering spaces.

As a first example, consider the graph $E$ shown below. The graph $E$ is a covering space of the $(0, \infty)$ constraint graph. The map $p$ is obtained by dropping the indices from the vertices and the edges.
As a second example, consider the graph $E$ shown below. The graph $E$ is a covering of the $(1,3)$ constraint graph. The map $p$ is obtained by dropping the indices on the vertices and the edges.
A covering space $E$ of a constraint graph $G$ is called an $N$ fold covering if for all points $x \in G$, we have $|p^{-1}(x)| = N$. In the first example above, the graph $E$ is a three fold covering of the $(0, \infty)$ constraint graph. In the second example above, the graph $E$ is a two fold covering of the $(1, 3)$ constraint graph. We show that all finite graphs which are coverings for a constraint graph are $N$ fold coverings.

**Theorem 5.** If $E$ is a finite graph which is a covering space of a constraint graph $G$, then $E$ is an $N$ fold covering for some positive integer $N$.

**proof.** Consider a vertex $u$ of the constraint graph $G$. By counting the total number of outgoing edges from the vertices in the set $p^{-1}(u)$, we get that $|p^{-1}(u)| = |p^{-1}(e)|$ for each edge $e$ whose initial vertex is $u$. Similarly, we get $|p^{-1}(u)| = |p^{-1}(e)|$ for each edge $e$ whose terminal vertex is $u$.

Since the constraint graph is connected, the condition that all edges which are incoming or outgoing to a vertex $v$ satisfy $|p^{-1}(v)| = |p^{-1}(e)|$ implies that all edges $e$ and all vertices $v$ satisfy $|p^{-1}(u)| = |p^{-1}(e)| = |p^{-1}(v)|$.

This is the definition of an $N$ fold covering where $N = |p^{-1}(u)|$. □

We want to examine edge paths in a covering space. The question we want to look at is if $\rho$ is an edge path in a constraint graph $G$, what can we say about the edge paths in an $N$ fold covering of $G$ that are mapped by $p$ onto $\rho$? It turns out that the edge paths in $E$ that are mapped onto $\rho$ are characterized by their initial vertex. This characterization is a special case of the path lifting theorem for
covering spaces. See [M] or [GH] for a proof of the path lifting theorem for covering spaces.

**Theorem 6.** Let $E$ be an $N$ fold covering of a constraint graph $G$. Let $w$ be a vertex in $E$ with $p(w) = u$. Then for any edge path $\rho$ in $G$ with initial vertex $u$, there is a unique edge path $\overline{\rho}$ in $E$ with initial vertex $w$ which satisfies $p(\overline{\rho}) = \rho$.

**proof.** Let $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$ be an edge path in the constraint graph $G$ whose initial vertex is the vertex $u = p(w)$.

Since $\rho_1$ has its initial vertex at $u$, we know that each edge in the set $p^{-1}(\rho_1)$ has its initial vertex in the set $p^{-1}(u)$ and that no two edges in the set $p^{-1}(\rho_1)$ have the same initial vertex. By the proof of theorem five, the sets $p^{-1}(\rho_1)$ and $p^{-1}(u)$ have the same number of members. This shows that for each vertex $z$ in $p^{-1}(u)$, there is an outgoing edge in $p^{-1}(\rho_1)$ whose initial vertex is $z$. Let $\overline{\rho_1}$ be this unique edge in $p^{-1}(\rho_1)$ whose initial vertex is the vertex $w$.

Repeating the reasoning above for the terminal vertex of $\overline{\rho_1}$, we get the unique edge in $p^{-1}(\rho_2)$ whose initial vertex is the terminal vertex of $\overline{\rho_1}$. This edge will be called $\overline{\rho_2}$. By induction on the length $n$, we get the edge path $\overline{\rho} = (\overline{\rho_1}, \ldots, \overline{\rho_n})$.

The initial vertex of $\overline{\rho}$ is $w$, each edge $\overline{\rho_i}$ is the unique edge in $p^{-1}(\rho_i)$ whose initial vertex is the terminal vertex of the edge $\overline{\rho_{i-1}}$, and the edge path $\overline{\rho}$ satisfies $p(\overline{\rho}) = \rho$. □
We will refer to this theorem as the edge path lifting theorem. In the next chapter, the edge path lifting theorem will be important for the construction of 

(d, k) constrained codes from a covering space of the (d, k) constraint graph.

Recall that a proper edge path $\rho = (\rho_1, \ldots, \rho_n)$ in a constraint graph $G$ is an edge path for which each edge $\rho_i$ is a proper edge from $G$. For a proper edge path $\rho$ in a (d, k) constraint graph, there is an associated (d, k) constrained sequence obtained by using the binary labels for the edges in the edge path. The (d, k) constrained sequence associated with the proper edge path $\rho = (\rho_1, \ldots, \rho_n)$ will be $s = (s_1, \ldots, s_n)$ where $s_i$ is the binary label in the constraint graph $G$ for the edge $\rho_i$. By the edge path lifting theorem, if $w$ is a vertex of $E$ with $p(w)$ the initial vertex of the edge $\rho_1$, then there is a unique edge path $\overline{\rho}$ in the covering space $E$ whose initial vertex is $w$ and which satisfies $p(\overline{\rho}) = \rho$. The edge path $\overline{\rho}$ is determined by its initial vertex and by the (d, k) constrained sequence which is associated with the edge path $\rho$. This association of edge paths in a covering space with (d, k) constrained sequences will be a key in the construction of (d, k) constrained codes introduced in the next chapter.

We want to consider how the fundamental group of a covering space of a constraint graph is related to the fundamental group of the constraint graph. Let $E$ be a covering space of the constraint graph $G$ and let $w$ be a vertex in $E$ with $u = p(w)$. Suppose $\overline{\rho} = (\overline{\rho_1}, \ldots, \overline{\rho_n})$ is a reduced edge path in $E$ which is a loop at $w$. Then the loop $\overline{\rho}$ is an element of the fundamental group $\pi(E, w)$. We define
the mapping \( p^*: \pi(E, w) \to \pi(G, u) \) by \( p^*(\rho) = (p(\rho_1), \ldots, p(\rho_n)) \). The mapping \( p^* \) maps the fundamental group of the covering space to the fundamental group of the constraint graph.

We want to show that \( p^* \) is a monomorphism of \( \pi(E, w) \) into \( \pi(G, u) \). If \( \rho \) and \( \tau \) are elements of \( \pi(E, w) \), then their product \( \rho \tau \) is the reduced edge path obtained from the edge path \( \rho \) concatenated with the edge path \( \tau \). That is, \( \rho \tau = (\rho_1, \ldots, \rho_{n+1-i}, \tau_i, \ldots, \tau_m) \) where \( i \) is the smallest positive integer such that \( \tau_i \) is not the reversed edge of the edge \( \rho_{n+1-i} \). The mapping \( p^* \) then satisfies:

\[
\begin{align*}
p^*(\rho \tau) &= (p(\rho_1), \ldots, p(\rho_{n+1-i}), p(\tau_i), \ldots, p(\tau_m)) \\
&= (p(\rho_1), \ldots, p(\rho_n), p(\tau_1), \ldots, p(\tau_m)) \\
&= p^*(\rho)p^*(\tau)
\end{align*}
\]

Thus showing that \( p^* \) is a homomorphism.

To show that \( p^* \) is one-to-one, suppose \( \sigma \) and \( \tau \) are reduced edge paths which are elements of \( \pi(E, w) \) and which satisfy \( p^*(\sigma) = p^*(\tau) = \rho \). By the edge path lifting theorem, there is a unique edge path \( \rho \) in \( E \) with initial vertex \( w \) and with \( p^*(\rho) = \rho \). But the two edge path \( \sigma \) and \( \tau \) have initial vertex \( w \) and satisfy \( \rho = p^*(\sigma) = p^*(\tau) \). Thus the two edge paths \( \sigma \) and \( \tau \) must be equal, showing that \( p^* \) is one-to-one.
We summarize these results in the theorem below.

**Theorem 7.** The mapping $p^* : \pi(E, w) \to \pi(G, u)$ defined by

\begin{align*}
    p^*(\overline{p}) &= (p(\overline{p}_1), \ldots, p(\overline{p}_n)) \tag{6}
\end{align*}

is a monomorphism of the fundamental group $\pi(E, w)$ of the covering space into the fundamental group $\pi(G, u)$ of the constraint graph.

This is a well known result for covering spaces. See [M] or [GH] for a proof of the general case of this theorem.

### 2.4 Construction of Covering Spaces for the $(d, k)$ Constraint Graphs

In this section, a constructive process for building a covering space of a $(d, k)$ constraint graph is given. The topological properties of the $(d, k)$ constraint graphs presented in the previous sections will be used to develop the construction. The construction of covering spaces presented here follows the construction presented in [N].

Let $G$ be a constraint graph, either a $(d, k)$ constraint graph or a $(d, \infty)$ constraint graph. We start by choosing a splitting complex for the constraint graph. Recall that a splitting complex $K$ is a subset of the set of midpoints for the edges in $G$ such that the subgraph of $G$ consisting of the edges in $G$ which do not intersect $K$ is a maximal tree for the constraint graph $G$. In the first section of this chapter.
some properties of splitting complexes are presented. By theorem three, there are $k+1$ splitting complexes for the $(d, k)$ constraint graph and there are $d+1$ splitting complexes for the $(d, \infty)$ constraint graph. We pick $K$ to be a splitting complex for the constraint graph $G$.

Consider the constraint graph $G$. For each point $\lambda$ of the splitting complex $K$, let $e_\lambda$ be the edge in the constraint graph which contains the point $\lambda$ as its midpoint. We form the figure $G^K$ from the constraint graph $G$ by splitting the edge $e_\lambda$ at each point $\lambda$ of the splitting complex $K$. That is, for each point $\lambda$ of the splitting complex $K$, we break the edge $e_\lambda$ at the point $\lambda$. The direction of the break is maintained, so that one side of the break is incoming to the break and the other side is outgoing to the break. The resulting figure, after each point of the splitting complex $K$ is split, will be the split figure $G^K$.

Let $\theta$ be a finite set. For the examples we will consider, the set $\theta$ will be the integers from 1 to $N$, or will be a set of binary polynomials. Let $N$ be the number of elements in the set $\theta$. The covering space to be constructed using the set $\theta$ will be an $N$ fold covering of the constraint graph $G$. The set $\theta$ will be used as labels for the $N$ copies of the split figure $G^K$. The covering space is obtained by gluing the $N$ copies of the split figure $G^K$ together.

To build the covering space from the $N$ copies of the split figure $G^K$, we need a set of permutations on the set $\theta$. For each point $\lambda$ of the splitting complex $K$, we pick a permutation $p_\lambda$ from the symmetric group on the set $\theta$. The permutation $p_\lambda$
will be called the permutation associated with the point $\lambda$ of the splitting complex $K$.

The permutations $p_\lambda$ are chosen so that the subgroup generated by the set

\[ \{ p_\lambda | \lambda \in K \} \]

is a transitive subgroup of the symmetric group on the set $\theta$. A subgroup $H$ of the symmetric group on the set $\theta$ is a transitive subgroup if for each pair $\gamma$ and $\delta$ of elements from $\theta$, there is a permutation in the subgroup $H$ which sends $\gamma$ to $\delta$. See [J] for more on transitive subgroups. The condition that the subgroup generated by the permutations $\{ p_\lambda | \lambda \in K \}$ be a transitive subgroup ensures that the constructed covering space is a connected graph. A set of permutations $\{ p_\lambda | \lambda \in K \}$ from the symmetric group on the set $\theta$, which generates a transitive subgroup, will be called a set of defining permutations for the splitting complex $K$.

We want to build an $N = |\theta|$ fold covering of the constraint graph $G$. The construction process is taken from [N], where it is done in greater generality. To build a covering space of $G$, we start by taking $N$ copies of the split figure $G^K$. We label the copies of the figure $G^K$ using the set $\theta$. The set of defining permutations $\{ p_\lambda | \lambda \in K \}$ is used to reconnect the split figures $G^K$. The permutation $p_\lambda$ is used as a guide for which levels to glue together for the break at the point $\lambda$ of the splitting complex $K$. That is, we identify the incoming side of the break at the point $\lambda$ in the figure labeled by the element $\gamma$ with the outgoing side of the break at the point $\lambda$ in the figure labeled by the element $\delta$, whenever the permutation
\( p_\lambda \) sends the element \( \gamma \) to the element \( \delta \). This process of identifying sides of the breaks at the \( N \) copies of the point \( \lambda \) is continued until the breaks at all the copies of the point \( \lambda \) have been reconnected. We repeat this process for all the points of the splitting complex \( K \). The resulting figure, after all the split edges have been reconnected, is a connected graph, which we call \( E \).

The graph \( E \), constructed in the manner above, is an \( N \) fold covering of the constraint graph \( G \). We summarize the results of the construction in the theorem below.

**Theorem 8.** To every set of defining permutations for a splitting complex \( K \) of the constraint graph \( G \), there corresponds a covering space \( E \) of the constraint graph \( G \).

**Proof.** The construction of the graph \( E \) is described in the paragraphs above. The graph mapping \( p \) of \( E \) onto \( G \) sends each level of the split figure \( G^K \) onto the constraint graph \( G \) in the natural way. It can be verified that \( p \) satisfies the conditions in the definition of a covering space. □

In the covering space \( E \), the defining permutations \( \{p_\lambda | \lambda \in K\} \) are used as a guide for which levels of the split figure \( G^K \) to glue together at the points of the splitting complex \( K \). We want to define a mapping of the fundamental group \( \pi(G,u) \) into the group of permutations generated by the set of defining permutations. To define the mapping, recall that each element \( \rho \) of the fundamental group
\[ \pi(G, u) \] can be written as a product \( \alpha_{\lambda_1}^{\epsilon_1} \alpha_{\lambda_2}^{\epsilon_2} \cdots \alpha_{\lambda_m}^{\epsilon_m} \) where \( \alpha_{\lambda_i} \) is the loop at \( u \) associated with the point \( \lambda_i \) of the splitting complex \( K \) and \( \epsilon_i \) is either \(-1\) or \(1\). We define the mapping \( \psi : \pi(G, u) \to \{ p_\lambda | \lambda \in K \} \) by \( \psi(\rho) = p_{\lambda_m}^{\epsilon_m} p_{\lambda_{m-1}}^{\epsilon_{m-1}} \cdots p_{\lambda_2}^{\epsilon_2} p_{\lambda_1}^{\epsilon_1} \) where \( \rho \) is equal to the product \( \alpha_{\lambda_1}^{\epsilon_1} \alpha_{\lambda_2}^{\epsilon_2} \cdots \alpha_{\lambda_m}^{\epsilon_m} \). The mapping \( \psi \) is a group homomorphism.

We use \( P_\rho \) to represent the permutation \( \psi(\rho) \). The permutation \( P_\rho \) will be called the permutation associated with the edge path \( \rho \).

**Lemma 9.** Let \( \rho \) be an edge path in \( \pi(G, u) \) and let \( \bar{\rho} \) be an edge path lifting of the edge path \( \rho \) to the covering space \( E \). If the initial level of \( \bar{\rho} \) is the level labeled by \( \gamma \), then the terminal level of \( \bar{\rho} \) will be the level labeled by \( P_\rho(\gamma) \), where \( P_\rho \) is the permutation associated with the edge path \( \rho \).

**proof.** Notice that the first time the edge path \( \rho \) crosses a point of the splitting complex \( K \), the edge path \( \bar{\rho} \) goes from the level labeled by \( \gamma \) to the level labeled by \( p_{\lambda_1}^{\epsilon_1}(\gamma) \). This process repeats for each point of the splitting complex \( K \) that the edge path \( \rho \) crosses, giving that the edge path \( \bar{\rho} \) goes from the level labeled by \( \gamma \) to the level labeled by \( p_{\lambda_m}^{\epsilon_m} p_{\lambda_{m-1}}^{\epsilon_{m-1}} \cdots p_{\lambda_2}^{\epsilon_2} p_{\lambda_1}^{\epsilon_1}(\gamma) \). But this is exactly the action of the permutation \( P_\rho \) on \( \gamma \). \( \Box \)

This lemma is useful for finding the fundamental group \( \pi(E, w) \) of the covering space \( E \) constructed above. Suppose \( \gamma \) is the element from the set \( \theta \) used as a label for the level which contains the vertex \( w \). We take \( S \) to be the set of elements
from the fundamental group $\pi(G, u)$ whose associated permutations fix the label $\gamma$. That is, $s$ is in the set $S$ if and only if $P_s(\gamma) = \gamma$ where $P_s$ is the permutation associated with the edge path $s$. The set $S$ is a subgroup of the fundamental group $\pi(G, u)$. We want to show that the fundamental group $\pi(E, w)$ is isomorphic to the subgroup $S$.

Recall that the group homomorphism $p^*$ induced by the graph mapping $p$ is a monomorphism of the fundamental group $\pi(E, w)$ into the fundamental group $\pi(G, u)$. We show now that $p^*$ is an isomorphism between the subgroup $S$ and the fundamental group $\pi(E, w)$.

Consider an edge path $\rho$ in the subgroup $S$. By the edge path lifting theorem, there is a unique edge path $\bar{\rho}$ in $E$ with initial vertex $w$ and with $p^*(\bar{\rho}) = \rho$. By the lemma, $\bar{\rho}$ will end in the level labeled by $P_{\rho}(\gamma)$. Since $\rho$ is in the subgroup $S$, this gives $P_{\rho}(\gamma) = \gamma$. The vertex on the level labeled with $\gamma$ and in the set $p^{-1}(u)$ is the vertex $w$. This shows that $\bar{\rho}$ is a loop at the vertex $w$ and so $S \subseteq p^*(\pi(E, w))$.

Now consider an edge path $\bar{\rho}$ in $\pi(E, w)$. Since $\bar{\rho}$ begins and ends on the level labeled with $\gamma$, the lemma implies that the permutation associated with $p^*(\bar{\rho})$ will fix the label $\gamma$. This is the condition for $p^*(\bar{\rho})$ to be in the subgroup $S$ and so $p^*(\pi(E, w)) \subseteq S$.

This completes showing that the subgroup $S$ is isomorphic to the fundamental group $\pi(E, w)$. We summarize this below.
Theorem 10. The fundamental group $\pi(E, w)$ of the covering space $E$ constructed above is isomorphic to the subgroup $S$ of the fundamental group $\pi(G, u)$ where $S$ consists of the edge paths whose associated permutations fix the label $\gamma$ of the level which contains the vertex $w$.

Suppose $Sg$ is a right coset of the subgroup $S = p^*(\pi(E, w))$ in the fundamental group $\pi(G, u)$. Consider two elements $s_1g$ and $s_2g$ of the right coset, with $P_{s_1g}$ and $P_{s_2g}$ being their associated permutations. If $\gamma$ is the label for the level which contains the vertex $w$, then $P_{s_1g}(\gamma) = P_gP_{s_1}(\gamma) = P_g(\gamma) = P_gP_{s_2}(\gamma) = P_{s_2g}(\gamma)$. This shows that the associated permutation for each edge path in the coset $Sg$ will send the label $\gamma$ to the label $P_g(\gamma)$.

Now consider the two right cosets $Sh$ and $Sg$ of the subgroup $S$. Suppose that the associated permutations for two elements $s_1g$ and $s_2h$ from the respective cosets satisfy $P_{s_1g}(\gamma) = P_{s_2h}(\gamma)$. Then it follows that $P_{s_1g}(\gamma) = P_{s_2h}(\gamma) = P_hP_{s_1}(\gamma) = P_h(\gamma)$. Or by rearranging that $\gamma = P_h^{-1}P_{s_1g}(\gamma) = P_h^{-1}P_{s_1g}(\gamma) = P_{s_1gh^{-1}}(\gamma)$. This shows that the permutation associated with the edge path $s_1gh^{-1}$ fixes the label $\gamma$. Thus, the edge path $s_1gh^{-1}$ is in the subgroup $S$. That is, $Sgh^{-1} = S$ or $Sg = Sh$, showing that the two cosets are equal.

We have shown that the cosets of the subgroup $S$ are determined by the action of their associated permutations on the label $\gamma$. We summarize the two previous paragraphs in the theorem below.
**Theorem 11.** For each $x \in \theta$, the set of edge paths in $\pi(G, u)$ which have associated permutations that send $\gamma$ to the label $x$ will be a right coset of the subgroup $S = p^*(\pi(E, w))$.

These two theorems will be used when we examine the constrained codes constructed from the covering space $E$.

To conclude this section, we present an example of the construction of a covering space. We begin with the $(1, 3)$ constraint graph $G$. The splitting complex $K$ will be the midpoints of the three edges that are incoming to the vertex $a_1$. It is clear that $K$ is a splitting complex for $G$, as the subgraph of $G$ which does not intersect $K$ is a maximal tree for $G$. We label the points of the splitting complex $K$ as $\alpha$, $\beta$, and $\delta$. The edges in the $(1, 3)$ constraint graph are split at the points $\alpha$, $\beta$, and $\delta$ to give the split figure $G^K$. The split figure $G^K$ is shown below.

![Split Figure $G^K$](image-url)
The set $\theta$ used to index the levels of the split figure $G^K$ will be the integers from one to four. The defining permutations will then be from the symmetric group $S_4$. We choose the defining permutations as $p_\alpha = (1.2.3)$, $p_\beta = (1.2)(3.4)$, and $p_\delta = (2,3)$. The defining permutations are used to connect together the four copies of the split figure $G^K$ together.

To reconnect the four copies of the point $\alpha$, we use the permutation $p_\alpha = (1,2,3)$. This is done by identifying the incoming side of the break at $\alpha_1$ with the outgoing side of the break at $\alpha_2$, identifying the incoming side of the break at $\alpha_2$ with the outgoing side of the break at $\alpha_3$, and identifying the incoming side of the break at $\alpha_3$ with the outgoing side of the break at $\alpha_1$. The incoming side and outgoing side of the break at $\alpha_4$ are identified.

This process of reconnecting the split edges is repeated for the points $\beta$ and $\delta$ using the permutations $p_\beta = (1.2)(3.4)$ and $p_\delta = (2,3)$. The resulting graph $E$ is shown below. The graph $E$ is a four fold covering of the $(1,3)$ constraint graph.
Figure 2.4: Four Fold Covering of $(1,3)$ Constraint Graph
2.5 Conclusion

In this chapter, the topological properties of the $(d, k)$ constraint graphs and of the $(d, \infty)$ constraint graphs are examined. The first section deals with the fundamental group of a constraint graph. The second section deals with covering spaces of constraint graphs. The final section contains a construction for building covering spaces of a constraint graph. This chapter is intended as an introduction for the nonexpert to some basic topological facts about constraint graphs.
Chapter 3

(d, k) Constrained Block

Codes From Covering Spaces

3.1 INTRODUCTION

In this chapter, (d, k) constrained codes constructed from covering spaces of the (d, k) constraint graph are examined. Examples of (d, k) constrained codes and (d, ∞) codes are given. Systematic codes for detecting all single errors are given for possible applications. In the final section, the average probability of an undetected error for the family of systematic (d, k) constrained block codes is shown to decrease exponentially.

3.2 INTRODUCTION TO (d, k) CONSTRAINED BLOCK CODES

The (d, k) constrained codes and the (d, ∞) constrained codes were introduced in the first chapter. We will use the term "constrained code" to mean either a (d, k) constrained code or a (d, ∞) constrained code.

A practical method for constructing constrained codes in given in [ACH]. An excellent introduction to these codes is given in [Si]. The constrained codes they construct are called sliding block codes. Other constructions of constrained codes are given in [TB] and [LeC].
The conventional coding scheme in magnetic recording concatenates an error-control code with a constrained code. For instance, see [HHW]. The input binary sequence is first encoded using an error-control code. The encoded sequence is then further encoded using a constrained code to obtain a constrained sequence. The constrained sequence is then sent to the channel. A diagram of this coding scheme is shown below.

![Diagram of Conventional Coding Scheme](image)

Figure 3.1: Conventional Coding Scheme

The sequence received from the channel is first decoded using the constrained code decoder. This gives an unconstrained sequence which is then decoded using the error-control code decoder. The binary sequence obtained from the error-control code is an estimate of the original input sequence.
As noted in [HHW], this coding scheme is not entirely satisfactory. The problem is that for all known constrained codes, errors which occur in the channel are multiplied by the constrained code. The constrained code decoder will turn a single error into a burst of errors. This is an undesirable property. The codes presented in this chapter are an attempt to alleviate this difficulty with the standard coding scheme.

The codes that we will construct are \((d, k)\) constrained block codes. A \((d, k)\) constrained block code is a set of \((d, k)\) constrained sequences of fixed length \(n\) such that for any pair of code sequences \(x\) and \(y\), the length \(2n\) sequence obtained by concatenating \(x\) onto \(y\) will be a \((d, k)\) constrained sequence. See [LW] or [FL] for examples of \((d, k)\) constrained block codes. We present some new constructions for \((d, k)\) constrained block codes.

We will also consider \((d, \infty)\) constrained block codes. The definition will be the same as for the \((d, k)\) constrained block codes. The term "constrained block code" will be used to mean a \((d, k)\) constrained block code or a \((d, \infty)\) constrained block code.

The idea is to construct constrained block codes which provide some error protection against errors which occur in the channel. In [H], the errors which occur in magnetic recording are analyzed. The errors are categorized as either shift errors, drop in errors, or drop out errors. A shift error is said to occur when a one in the \(i^{th}\) position is shifted to the \(i-1^{th}\) position or to the \(i+1^{th}\) position. A drop in/out
error is said to occur when a one is changed to a zero or a zero is changed to a one. In magnetic recording, shift errors and drop in/out errors occur simultaneously.

The shift distance between two constrained sequences will be the minimum number of shift errors required to change one sequence into the other. Similarly, the Hamming distance between two constrained sequences will be the minimum number of drop in/out errors required to change one sequence into the other. The minimum shift distance of a constrained block code will be the minimum shift distance attained by considering all possible pairs of nonequal code sequences. Similarly, the minimum Hamming distance of a constrained block code will be the minimum Hamming distance attained by considering all possible pairs of nonequal code sequences.

For each code presented in the next section, we will give its minimum shift distance and its minimum Hamming distance.

3.3 Base Point Codes

In chapter two, covering spaces for the \((d, k)\) constraint graphs were introduced. We want to use covering spaces of a \((d, k)\) constraint graph to construct \((d, k)\) constrained block codes. Before proceeding, we review the relevant facts about constraint graphs and covering spaces.

To each proper edge path in the \((d, k)\) constraint graph, there is an associated \((d, k)\) constrained sequence obtained by using the binary labels of the edges in the
edge path. Suppose $E$ is a covering space of the constraint graph $G$. By the edge path lifting theorem, to each edge path $\rho$ in the constraint graph whose initial vertex is $u$, there corresponds, for each vertex $w$ in $p^{-1}(u)$, a unique edge path $\overline{\rho}$ in the covering space $E$ such that the initial vertex of $\overline{\rho}$ is $w$ and $p^*(\overline{\rho}) = \rho$. The fundamental group $\pi(G, u)$ is the set of reduced edge paths in the constraint graph $G$ which are loops at the vertex $u$. The set of proper edge paths in the constraint graph which are loops at the vertex $u$ is a subset of the fundamental group $\pi(G, u)$.

For constructing the base point codes from a covering space $E$, we use the fact that to each $(d, k)$ constrained sequence associated with a proper edge path from $\pi(G, u)$, there is a unique edge path in the covering space with a specified initial vertex.

**Definition 1.** Suppose $E$ is a covering space of the $(d, k)$ constraint graph $G$ and let $w$ be a vertex in the $E$ with $p(w) = u$. The length $n$ base point code for the covering space $E$ at the vertex $w$ is the set of $(d, k)$ constrained sequences associated with the set of proper edge paths in the fundamental group $\pi(G, u)$ which lift to edge paths in the covering $E$ that are loops at the vertex $w$.

We will also consider base point codes for covering spaces of the $(d, \infty)$ constraint graphs. The definition of base point codes for covering spaces of the $(d, \infty)$ constraint graph will be the same as the definition of base point codes for covering spaces of the $(d, k)$ constraint graph.
For a base point code, we use $C$ to represent the set of constrained sequences that comprise the code, $C$ to represent the set of proper edge paths from $\pi(G, u)$ whose associated sequences form the code $C$, and $\overline{C}$ to represent the edge paths in the covering space $E$ determined by lifting the edge paths in $C$.

The definition of the base point codes for a covering space says that $C$ is the set of constrained sequences such that $\overline{C}$ is a subset of the fundamental group $\pi(E, w)$.

We want to consider some examples of base point codes for covering spaces of the $(d, k)$ constraint graphs and of the $(d, \infty)$ constraint graphs.

Before proceeding, we review some facts about the construction of covering spaces for constraint graphs.

Recall that a covering space $E$ of a constraint graph $G$ is determined by a set of indices $\theta$ and a set $\{p_{\lambda} | \lambda \in K\}$ of defining permutations for a splitting complex $K$ of the constraint graph. The permutations $p_{\lambda}$ are chosen from the symmetric group on the set $\theta$. The covering space $E$ will be a $|\theta|$ fold covering of the constraint graph $G$. See chapter two for the details of the construction.

The splitting complex used in the construction of a covering space of a $(d, k)$ constraint graph will be taken to be the midpoints of the $k - d + 1$ edges that are incoming to the vertex $a_1$. The splitting complex used in the construction of a covering space of a $(d, \infty)$ constraint graph will be taken to be the midpoints of the two edges that are outgoing to the vertex $a_{d+1}$. These conventions are made to make the analysis of the constructed covering spaces easier.
As a first example, we construct a length twelve (1, 2) constrained block code which is a base point code for a three fold covering of the (1, 2) constraint graph. The set $\theta$ is taken to be the integers from one to three. The set of defining permutations is taken from the symmetric group $S^3$. Let $\alpha$ be the midpoint of the edge from vertex $a_2$ to vertex $a_1$ and let $\beta$ be the midpoint of the edge from vertex $a_3$ to vertex $a_1$. The defining permutations are chosen as $p_\alpha = (1, 2)$ and $p_\beta = (2, 3)$. The set $\{p_\alpha, p_\beta\}$ is a set of defining permutations for the splitting complex $K = \{\alpha, \beta\}$. The three fold covering constructed with the set $\{p_\alpha, p_\beta\}$ is shown below.

![Figure 3.2: Three Fold Covering of (1, 2) Constraint Graph](image)

We consider the base point code at the vertex $w$, where $w$ is the vertex $a_1$ in the level of the covering space indexed by one. The length twelve base point code
at $w$ will consist of the five (1,2) constrained sequences whose corresponding edge paths in the covering space are loops at the vertex $w$. The base point code $C$ is shown below:

$$C = \{010101010101, 010100101001, 001010010010, 001001010010, 001010010010001\}$$

The rate of the base point code $C$ is $R = \frac{\log_2 |C|}{12} = 0.1934$, its minimum shift distance is two, and its minimum Hamming distance is three. For comparison, we note that the capacity of the (1,2) constrained codes is 0.4057.

As a second example, we construct a length eight (0,2) constrained block code which is a base point code for an eight fold covering of the (0,2) constraint graph. The set $\theta$ is taken to be the integers from one to eight. The set of defining permutations is taken from the symmetric group $S^8$. Let $\alpha$ be the midpoint of the edge which is a loop at the vertex $a_1$, let $\beta$ be the midpoint of the edge from vertex $a_2$ to vertex $a_1$, and let $\gamma$ be the midpoint of the edge from vertex $a_3$ to vertex $a_1$. The defining permutations are chosen as $p_\alpha = (1,2,3,4,5,6,7)$, $p_\beta = (1,2,7,3,8,5,4)$, and $p_\gamma = (1,2,6,4,3,5,8)$. The set $\{p_\alpha, p_\beta, p_\gamma\}$ is a set of defining permutations for the splitting complex $K = \{\alpha, \beta, \gamma\}$. The eight fold covering constructed with the set $\{p_\alpha, p_\beta, p_\gamma\}$ is shown below.
Figure 3.3: Eight Fold Covering of (0,2) Constraint Graph
We consider the base point code at the vertex $w$, where $w$ is the vertex $o_1$ in the level of the covering space indexed by a one. The length eight base point code at $w$ will consist of the ten $(0.2)$ constrained sequences whose corresponding edge paths in the covering space are loops at the vertex $w$. The base point code $C$ is shown below:

$$C = \{01111111, 01010011, 01101001, 00100111, 10111011, 11001111, 11011001, 10101101, 10010111, 11110101\}$$

(2)

The rate of the base point code $C$ is $R = \frac{\log_2 |C|}{8} = .4152$, its minimum shift distance is three, and its minimum Hamming distance is three. For comparison, the capacity of a $(0,2)$ constrained code is .8792.

As a third example, we construct a length seven $(0,\infty)$ constrained block code which is a base point code for an eight fold covering of the $(0,\infty)$ constraint graph. The set $\theta$ is taken to be the set of binary polynomials whose degree is less than three. The set of defining permutations are taken from the symmetric group on the set of binary polynomials whose degree is less than three. Let $\alpha$ be the midpoint of the edge labeled with a one and let $\beta$ be the midpoint of the edge labeled with a zero. The defining permutations are chosen as:

$$p_\alpha : h_0 + h_1 x + h_2 x^2 \rightarrow (1 + h_2) + (h_0 + h_2)x + h_1 x^2$$

(3)

$$p_\beta : h_0 + h_1 x + h_2 x^2 \rightarrow h_2 + (h_0 + h_2)x + h_1 x^2$$

The set $\{p_\alpha, p_\beta\}$ is a set of defining permutations for the splitting complex $K = \{\alpha, \beta\}$. The eight fold covering of the $(0,\infty)$ constraint graph is shown below.
We consider the base point code at the vertex $w$, where $w$ is the vertex labeled with the zero polynomial. The length seven base point code at the vertex $w$ will consist of the sixteen binary sequences whose corresponding edge paths in the covering space are loops at the vertex $w$. The base point code $C$ is shown below:

$$C = \{0000000, 0010111, 0101100, 1011000, 0110001, 1100010, 1000101, 1101001, 1010110, 0101100, 1011000, 0110001, 1100010, 1000101, 1101001, 1010110, 0100111, 1101110, 0011101, 0111010, 1110100, 1111111\}$$
The rate of the base point code $C$ is $R = \frac{\log_2 |C|}{7} = 4/7$. Its minimum shift distance is three, and its minimum Hamming distance is three. The base point code $C$ is the well known Hamming $[7,4]$ cyclic code. (See [LC], [PW], or [SM]).

The examples above show that a base point code for a covering space of the $(d,k)$ constraint graph is a $(d,k)$ constrained block code. We prove this now.

**Theorem 1.** Suppose $C$ is a base point code for a covering space of the $(d,k)$ constraint graph. Then $C$ is a $(d,k)$ constrained block code.

**proof.** We need only show that $C$ satisfies the concatenability condition. Suppose $x$ and $y$ be sequences in the base point code $C$ and with $x$ and $y$ their associated edge paths in $\pi(G,u)$. Since the edge paths $x$ and $y$ are proper edge paths, their product $xy$ will also be a proper edge path. The $(d,k)$ constrained sequence associated with the proper edge path $xy$ will be the sequence $x$ concatenated by $y$. This shows that the base point code $C$ is a $(d,k)$ constrained block code. \qed

The third example above showed that the Hamming $[7,4]$ cyclic code is a base point code for an eight fold covering of the $(0,\infty)$ constraint graph. We show in theorem two that an $[n,k]$ cyclic code is a base point code for an $2^{n-k}$ fold covering of the $(0,\infty)$ constraint graph. Before proving the theorem, we review some basic facts concerning cyclic codes.

A binary polynomial has an associated binary sequence obtained by taking the coefficients of the polynomial. An $[n,k]$ cyclic code $C$ has a distinguished binary
polynomial $g(x)$ of degree $n - k$. The polynomial $g(x)$ is called the generator polynomial of the cyclic code. The code $C$ consists of the length $n$ binary sequences whose associated binary polynomials are divisible by $g(x)$. See [LC], [PW], or [SM] for a comprehensive treatment of cyclic codes. A further note is that the generator polynomial $g(x)$ divides the polynomial $x^n + 1$.

**Theorem 2.** An $[n, k]$ cyclic code is a base point code for a $2^{n-k}$ fold covering of the $(0, \infty)$ constraint graph.

**proof.** Suppose $g(x)$ is the generator polynomial of an $[n, k]$ cyclic code $C$. We use $g(x)$ to get a set of defining permutations. The set $\theta$ used to index the levels of the covering space is taken to be the set of binary polynomials of degree less than $2^{n-k}$. The set of defining permutations are taken from the symmetric group on the set of binary polynomials whose degree is less than $2^{n-k}$. Let $\alpha$ be the midpoint of the edge labeled with a one and let $\beta$ be the midpoint of the edge labeled with a zero.

The defining permutations are chosen as:

$$p_\alpha : h(x) \rightarrow x \cdot h(x) + h_{n-k-1} \cdot g(x) + 1 \mod x^{n-k}$$

$$p_\beta : h(x) \rightarrow x \cdot h(x) + h_{n-k-1} \cdot g(x) \mod x^{n-k}$$

(5)

where $h_{n-k-1}$ is the coefficient of $x^{n-k-1}$ in $h(x)$. 

73
To see that these mappings are actually permutations on the set of binary polynomials of degree less than $n-k$, suppose that two polynomials $r(x)$ and $q(x)$ are sent to the same state by $p_\alpha$. That is, $r(x)$ and $q(x)$ satisfy:

$$x \cdot r(x) + r_{n-k-1} \cdot g(x) + 1 = x \cdot q(x) + q_{n-k-1} \cdot g(x) + 1 \text{ modulo } x^{n-k}$$

or rearranging gives:

$$x \cdot (q(x) - r(x)) + (q_{n-k-1} - r_{n-k-1}) \cdot g(x) = 0 \text{ modulo } x^{n-k}$$

Now consider the two cases, whether $q_{n-k-1} = r_{n-k-1}$ or $q_{n-k-1} \neq r_{n-k-1}$.

First, if $q_{n-k-1} = r_{n-k-1}$, then equation seven reduces to $x \cdot (q(x) - r(x)) = 0 \text{ modulo } x^{n-k}$. The polynomial $q(x) - r(x)$ has degree less than $n-k-1$, since their leading coefficients are equal. So, the equation $x \cdot (q(x) - r(x)) = 0 \text{ modulo } x^{n-k}$ implies that the polynomial $q(x) - r(x)$ is zero. Showing that $q(x) = r(x)$.

If $q_{n-k-1} \neq r_{n-k-1}$, then equation seven reduces to $x \cdot (q(x) - r(x)) + g(x) = 0 \text{ modulo } x^{n-k}$. Since $g(x)$ divides $x^{n+1}$, its constant term is one. The polynomial $x \cdot (q(x) - p(x))$ has constant term zero. Thus the left side has constant term one, but the right side has constant term zero. This contradiction shows that this case can not occur.

Hence, we have that $p_\alpha$ is a permutation on the set of binary polynomials whose degree less than $n-k$. The proof that $p_\beta$ is a permutation is the same as the proof for $p_\alpha$. Thus, we have that $\{p_\alpha, p_\beta\}$ is a set of defining permutations for the splitting complex $\{\alpha, \beta\}$. 

74
Let $E$ be the $2^{n-k}$ fold covering of the $(0, \infty)$ constraint graph constructed using the set of defining permutations $\{p_\alpha, p_\beta\}$. Suppose $w$ is the vertex in $E$ that is indexed by the zero polynomial. We want to show that the length $n$ base point code for the covering space $E$ at the vertex $w$ and the $[n,k]$ cyclic code whose generator polynomial is $g(x)$ are the same code.

Consider a code sequence $s$ from the base point code, with $\bar{s}$ the associated edge path in the covering space. Suppose after the $l-1$th edge, the edge path $\bar{s}$ is in the vertex labeled by $c(x)$. Then after the $l$th edge, the edge path $\bar{s}$ will be in the vertex labeled by the polynomial $x \cdot c(x) + c_{n-k-1} \cdot g(x) + s_{n-l}$ modulo $x^{n-k}$.

Figure 3.5: Shift Register for Dividing by $g(x)$

Now consider the shift register shown above for dividing by $g(x)$. See [PW] or [LC] for an introduction to shift registers. The shift register above is designed
to divide bit by bit, the input polynomial by \( g(x) \). After the coefficients of a polynomial \( h(x) \) are input, the contents of the shift register will be the remainder of \( h(x)/g(x) \). Suppose the contents of the shift register, after the \( l - 1 \)th input bit are \( c(x) \). Then, after the \( l \)th input bit, the contents of the shift register will be
\[
x \cdot c(x) + c_{n-k-1} \cdot g(x) + h_{n-l} \text{ modulo } x^{n-k}.
\]

Comparing the two previous paragraphs shows that the contents of the shift register correspond to the state in the covering space that the edge path is in. Since a sequence is in the base point code when its associated edge path is a loop at the vertex \( w \), we see that a sequence from the base point code when inputed into the shift register will put the contents of the register to zero. This is the same condition for a sequence to be in the cyclic code generated by \( g(x) \). Therefore, the two codes are the same. □

The example of the Hamming [7, 4] cyclic code above is derived using this theorem.

The covering space \( E \) constructed in the theorem will be called the cyclic covering generated by \( g(x) \).

### 3.4 Parity Point Codes

A base point code for covering space \( E \) is a set of \((d, k)\) constrained sequences whose associated edge paths in \( E \) are loops at a specified vertex. We get different codes if the edge paths are required to connect two specified vertices. Suppose \( w \)
and $z$ are two vertices in the set $p^{-1}(u)$. We get a $(d, k)$ constrained block code by taking the $(d, k)$ constrained sequences whose associated edge paths have initial vertex $w$ and terminal vertex $z$.

**Definition 2.** Suppose $E$ is a covering space of the $(d, k)$ constraint graph $G$ and let $w$ and $z$ be two vertices in the set $p^{-1}(u)$. The length $n$ parity point code for the covering space $E$ from $w$ to $z$ will be the set of $(d, k)$ constrained sequences whose associated edge paths in $E$ have initial vertex $w$ and terminal vertex $z$.

As usual, the definition for the $(d, \infty)$ constrained sequences will be the same. We note that the base point codes are parity point codes where the initial vertex and the terminal vertex are the same. We now consider some examples of parity point codes.

As a first example, we consider the eight fold covering of the $(0, 2)$ constraint graph that was used in the second example above. The initial vertex is chosen to be the vertex $a_1$ on the level indexed by one. The code length is chosen to be eight. The various parity point codes are listed below by the terminal vertex used to generate the code. We use $a^{(j)}_i$ to represent the vertex $a_i$ on the level $j$. 

77
Table 3.1: Parity Point Codes for $E$ Covering of the (0,2) Constraint Graph

<table>
<thead>
<tr>
<th>Terminal Vertex</th>
<th>Parity Point Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1^{(1)}$</td>
<td>{01111111, 01010011, 01101001, 00100111, 10111011, 11001111, 11011001, 10101101, 10010111, 11110101}</td>
</tr>
<tr>
<td>$a_1^{(2)}$</td>
<td>{00110111, 01111001, 11100101, 11011111, 01101111, 01010101, 11001001, 10101011, 11011111, 10111101}</td>
</tr>
<tr>
<td>$a_1^{(3)}$</td>
<td>{10110111, 11111001, 01100101, 01011111, 00101011, 10011011, 11010101, 01110011, 01101111, 00101001}</td>
</tr>
<tr>
<td>$a_1^{(4)}$</td>
<td>{00101101, 01011001, 11111111, 00111011, 01110101, 11101001, 10100111, 01001111, 10011101}</td>
</tr>
<tr>
<td>$a_1^{(5)}$</td>
<td>{11010111, 11101101, 10011001, 00111111, 01001011, 11111011, 10110101, 00101001, 01100111, 01011101}</td>
</tr>
<tr>
<td>$a_1^{(6)}$</td>
<td>{01110111, 01001101, 00111001, 10100101, 10011111, 11101011, 01011011, 10110011, 00101111, 11111101}</td>
</tr>
<tr>
<td>$a_1^{(7)}$</td>
<td>{11110111, 11001101, 10111001, 00100101, 01101011, 10010101, 00110011, 10101111, 01111101}</td>
</tr>
<tr>
<td>$a_1^{(8)}$</td>
<td>{01010111, 01101101, 10111111, 11001011, 01111011, 00110101, 10101001, 10010011, 11100111, 11011101}</td>
</tr>
</tbody>
</table>
The parity point codes vary in size from nine to eleven code sequences. Each parity point code has shift distance at least three and Hamming distance at least three. The code rates range from .3962 to .4324. For comparison, the capacity of a (0,2) constrained code is .8792.

There are eighty-one sequences of length eight which satisfy the (0, 2) constraints and whose associated edge paths in $G$ are loops at the vertex $a_1$. The parity point codes partition these sequences into eight disjoint subsets.

As a second example, we consider the eight fold covering of the (0, $\infty$) constraint graph that was used in the third example above. The initial vertex is chosen to be the vertex indexed by the zero polynomial. The code length is chosen to be seven. For each vertex in the covering, we get a different parity point code. The parity point code with the terminal vertex being the vertex indexed by the polynomial $x$ is shown below:

$$C = \{0000010, 0001001, 0010100, 0101110, 1011010, 0110011, 1100000, 1000111, 1101011, 1010001, 0100101, 1001100, 0011111, 0111000, 1110110, 1111101\}$$

(8)

The parity point code is a coset of the Hamming [7, 4] cyclic code. The rate of the code is $4/7$, its minimum shift distance is three, and its minimum Hamming distance is three. The covering space of this example is the cyclic covering generated by $g(x) = x^3 + x + 1$.  

79
The parity point codes for the cyclic covering generated by a polynomial \( g(x) \) will always be the cosets of the cyclic code generated by \( g(x) \). As a corollary of theorem two, we prove this fact. The idea is that the index of an edge path’s terminal vertex in the covering space determines the coset the binary sequence will be in.

**Corollary 3.** Suppose \( E \) is the cyclic covering generated by \( g(x) \). Let \( h(x) \) be a polynomial of degree less than \( n - k \) and let \( h \) be the length \( n \) sequence corresponding to \( h(x) \). Then the parity point code from the vertex indexed by the zero polynomial to the vertex indexed by \( h(x) \) will be the coset \( h + C \) of the cyclic code \( C \) generated by the polynomial \( g(x) \).

**proof.** Suppose \( s \) is a sequence from the parity point code. In the proof of theorem two, it was shown that the state polynomial \( h(x) \) of the terminal vertex for the edge path \( \bar{s} \) is the remainder from dividing the polynomial \( s(x) \) by the generator polynomial \( g(x) \).

Consider the coset \( h + C \) of the cyclic code \( C \) generated by \( g(x) \). Suppose \( h + c \) is a sequence from this coset. Since \( g(x) \) divides the polynomial \( c(x) \) and the degree of \( h(x) \) is less than the degree of \( g(x) \), we see that the remainder of the division \( \frac{h(x) + c(x)}{g(x)} \) will be \( h(x) \).

This shows that the parity point code and the coset \( h + C \) are the same. \( \square \)
The examples above show that the parity point code for a covering of the \((d, k)\) constraint graph are \((d, k)\) constrained block codes. The proof of this is the same as the proof of theorem one. We omit repeating the proof.

**Theorem 4.** Suppose \(C\) is a parity point code for a covering of the \((d, k)\) constraint graph. Then \(C\) is a \((d, k)\) constrained block code.

We want consider what conditions on the covering space ensure that an error in transmission will be detected by the parity point codes for the covering. Consider the covering \(E\) of the \((d, k)\) constraint graph \(G\), constructed with the standard splitting complex \(K\) and the set \(\{p_\lambda|\lambda \in K\}\) of defining permutations. In chapter two, we showed that each edge path \(x\) in the fundamental group \(\pi(G, u)\) has an associated permutation \(P_x\), formed as a product of the defining permutations. Suppose \(\bar{x}\) is the edge path in \(E\) which has initial vertex on the level indexed by \(\gamma\) and which satisfies \(p(\bar{x}) = x\). By theorem eleven of chapter two, the edge path \(\bar{x}\) has its terminal vertex on the level indexed by \(P_x(\gamma)\). We use this fact to show which errors are detected by the parity point codes for the covering space \(E\).

**Lemma 5.** Suppose \(x\) and \(y\) are two proper edge paths in \(\pi(G, u)\) such that the permutation \(P_x^{-1}P_y\) is fixed point free. Then the error pattern which changes the sequence \(x\) into the sequence \(y\) will be detected for all parity point codes for the covering space \(E\).
proof. Suppose $axb$ and $ayb$ are two sequences in the same parity point code, where $axb$ is the sequence $a$ concatenated by the sequence $x$, which is further concatenated by the sequence $b$. Then the permutations associated with the edge paths $axb$ and $ayb$ will satisfy $P_{axb}(\gamma) = P_{ayb}(\gamma)$, where $\gamma$ is the index of the initial vertex of the parity point code. This becomes $P_b P_x P_a(\gamma) = P_b P_y P_a(\gamma)$, and rearranging $P_a(\gamma) = P_{x^{-1}} P_y P_a(\gamma)$. Showing that the permutation $P_{x^{-1}} P_y$ has a fixed point at $P_a(\gamma)$.

Thus, if we assume that $P_{x^{-1}} P_y$ is fixed point free, then it follows that no two sequences of the form $axb$ and $ayb$ will be in the same parity point code for the covering $E$. Hence, the error pattern that changes the sequence $x$ into the sequence $y$ will be detected. □

This lemma shows that parity point codes are natural for detecting errors. In the next section, we construct practical codes for detecting all single errors. Lemma five will be used to help choose the defining permutations for constructing the covering space.

3.5 Error Check Codes

In practice, systematic codes are generally preferred. The idea of a systematic code is as follows. An input sequence of length $l$ is encoded by concatenating a length $r$ sequence onto it. The length $l$ input sequence is called the information and the length $r$ sequence used to concatenate is called the parity. To decode
the received sequence, the sequence is checked to see if it is a code sequence. If
the received sequence is a code sequence, then the parity is discarded to give the
estimate of the information. The ease of encoding and decoding makes systematic
codes popular.

We want to construct systematic \((d, k)\) constrained block codes which will detect
all single errors. We want to detect shift errors and to detect drop in/out errors.
The idea is to construct a covering space \(E\) of the \((d, k)\) constraint graph such that
the parity point codes for the covering are single error detecting. The systematic
\((d, k)\) constrained block code will be chosen as a subset of a parity point code for
\(E\).

We take \(\alpha_i\) to be the loop in \(G\) at the vertex \(a_i\) determined by the sequence
of \(i\) zeros followed by a one. We express the single error patterns in terms of the
\(\alpha_i\)'s. The drop out errors which do not violate the \((d, k)\) constraints will change the
sequence associated with the edge path \(\alpha_i \alpha_j\) into the sequence associated with the
edge path \(\alpha_{i+j+1}\). The drop in errors which do not violate the \((d, k)\) constraints
will change the sequence associated with the edge path \(\alpha_{i+j+1}\) into the sequence
associated with the edge path \(\alpha_i \alpha_j\). The shift errors which do not violate the \((d, k)\)
constraints will change the sequence associated with the edge path \(\alpha_i \alpha_j\) into the
sequence associated with the edge path \(\alpha_{i-1} \alpha_{j+1}\) or associated with the edge path
\(\alpha_{i+1} \alpha_{j-1}\).
We use lemma five to find conditions on the defining permutations for the parity point codes to be single error detecting. The drop out errors will be detected if the permutations of the form $P_{\alpha_{i+1}}^{-1} P_{\alpha_i} P_{\alpha_{i+1}}$ are fixed point free. The drop in errors will be detected if the permutations of the form $P_{\alpha_{i+1}}^{-1} P_{\alpha_i} P_{\alpha_{i-1}} P_{\alpha_{i+1}}$ and of the form $P_{\alpha_i}^{-1} P_{\alpha_{i+1}} P_{\alpha_{i-1}}$ are fixed point free.

We can reduce the number of single error detecting conditions by setting $P_i = a$ for even $i$ and $P_i = b$ for odd $i$. The drop out errors will be detected if permutations of the form $a^{-2}b$, $a^{-1}b^{-1}a$, and $b$ are fixed point free. The drop in errors will be detected if same permutations are fixed point free. The shift errors will be detected if permutations of the form $a^{-2}b^2$ and $a^{-1}b^{-1}ab$ are fixed point free.

We want to find permutations $a$ and $b$ which satisfy the five conditions above. Consider $a = (2, 3)$ and $b = (1, 2, 3)$. The permutations desired to be fixed point free are then $a^{-2}b = (1, 2, 3)$, $a^{-1}b^{-1}a = (1, 2, 3)$, $b = (1, 2, 3)$, $a^{-2}b^2 = (1, 3, 2)$, and $a^{-1}b^{-1}ab = (1, 3, 2)$. Thus, setting $P_{\alpha_i} = (2, 3)$ for even $i$ and $P_{\alpha_i} = (1, 2, 3)$ for odd $i$, all the single error detecting conditions will be satisfied.

We use the set $\{P_{\alpha_i}|d \leq i \leq k\}$ to construct a three fold covering of the $(d, k)$ constraint graph $G$. Suppose $K$ is the standard splitting complex of the $(d, k)$ constraint graph. The intersection of the splitting complex $K$ with the edge path $\alpha_i$ will be the midpoint of the edge $a_{i+1} \rightarrow a_1$. This associates a permutation from the set $\{P_{\alpha_i}|d \leq i \leq k\}$ with each point of the splitting complex $K$. So, the set
\{P_{\alpha_i}| d \leq i \leq k\} is a set of defining permutations for the splitting complex K. By the construction process from chapter two, we can build a three fold covering \(E\) of the \((d,k)\) constraint graph using this set of defining permutations. We call this covering the error check covering for the \((d,k)\) constraint graph.

**Theorem 6.** Suppose \(E\) is the error check covering for the \((d,k)\) constraint graph and suppose \(C\) is a parity point code for \(E\). Then the code \(C\) will detect all single errors.

**proof.** By lemma five and the discussion above, the theorem follows. \(\square\)

As noted earlier, we want to construct systematic codes which are a subset of a parity point code for \(E\). We consider some examples for the \((1,3)\) constraint graph, for the \((2,7)\) constraint graph, and for the \((1,7)\) constraint graph.

As a first example, we consider the \((1,3)\) constraint graph. There are three shift errors, one drop in error, and one drop out error which do not violate the \((1,3)\) constraints. The error check covering is constructed using the standard splitting complex and the defining permutation \(P_{\alpha_1} = P_{\alpha_3} = (1,2,3)\) and \(P_{\alpha_2} = (2,3)\). We show the error check covering below.
We want to construct a systematic \((d,k)\) constrained block code using the error check covering. The parity is chosen so that each edge path associated with a code sequence is a loop at the vertex \(a_1\) on the level indexed by a one. The resulting code will then be a subset of the base point code at the vertex \(a_1\) on the level indexed by a one. Thus, by theorem six, the code will be single error detecting.
Suppose \( w \) is the vertex \( a_1 \) on the level indexed by a one. To encode an information sequence, the edge path associated with the sequence is traced out in the covering space to find the terminal vertex. The parity is chosen such that each edge path will be forced back to the vertex \( w \). The parity length is the smallest number \( r \) such that there is an edge path of length \( r \) from every vertex in the covering to the vertex \( w \). The parity length for the (1,3) constrained code is eight. An encoding table using the error check covering is shown below. We use \( a_i^{(j)} \) to represent the vertex \( a_i \) on the level \( j \).

<table>
<thead>
<tr>
<th>Final State</th>
<th>Parity Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1^{(1)}, a_3^{(1)} )</td>
<td>01000101</td>
</tr>
<tr>
<td>( a_2^{(2)}, a_2^{(2)} )</td>
<td>00101001</td>
</tr>
<tr>
<td>( a_1^{(3)}, a_3^{(3)} )</td>
<td>01001001</td>
</tr>
<tr>
<td>( a_2^{(1)}, a_4^{(1)} )</td>
<td>10101001</td>
</tr>
<tr>
<td>( a_2^{(3)}, a_4^{(3)} )</td>
<td>10100101</td>
</tr>
<tr>
<td>( a_3^{(2)}, a_4^{(2)} )</td>
<td>10001001</td>
</tr>
</tbody>
</table>

To decode, the edge path associated with received sequence is traced out in the error check covering. If the edge path is a loop at the vertex \( w \), it is assumed that no error occurred. The parity is dropped and the estimated information is sent to the constrained code decoder. If the edge path is not a loop, an error has been detected. Corrective measures such as retransmission or rereading can be taken.
As other examples, we consider the (2,7) constraint graph and the (1,7) constraint graph. There are twenty-five shift errors, six drop in errors, and six drop out errors which do not violate the (2,7) constraints. There are thirty-six shift errors, fifteen drop in errors, and fifteen drop out errors which do not violate the (1,7) constraints. The parity length for the (2,7) constrained code will be nine and the parity length for the (1,7) constrained code will be six. Encoding tables for the codes are shown below.

Table 3.3: Error Check Encoder for the (2,7) Constraints

<table>
<thead>
<tr>
<th>Final State</th>
<th>Parity Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1^{(1)}, a_2^{(3)}, a_3^{(1)}, a_4^{(3)}, a_5^{(1)}$</td>
<td>001001001</td>
</tr>
<tr>
<td>$a_1^{(2)}, a_2^{(2)}, a_3^{(2)}, a_4^{(2)}$</td>
<td>000010001</td>
</tr>
<tr>
<td>$a_1^{(3)}, a_2^{(1)}, a_3^{(3)}$</td>
<td>000001001</td>
</tr>
<tr>
<td>$a_4^{(1)}, a_6^{(1)}, a_8^{(1)}$</td>
<td>100010001</td>
</tr>
<tr>
<td>$a_5^{(2)}, a_6^{(2)}, a_7^{(2)}, a_8^{(2)}$</td>
<td>100000001</td>
</tr>
<tr>
<td>$a_5^{(3)}, a_7^{(3)}$</td>
<td>010000001</td>
</tr>
<tr>
<td>$a_6^{(3)}, a_7^{(1)}, a_8^{(3)}$</td>
<td>100001001</td>
</tr>
</tbody>
</table>
Table 3.4: Error Check Encoder for the (1,7) Constraints

<table>
<thead>
<tr>
<th>Final State</th>
<th>Parity Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1^{(1)}, a_3^{(1)}, a_5^{(1)}, a_7^{(1)})</td>
<td>010101</td>
</tr>
<tr>
<td>(a_1^{(2)}, a_2^{(2)}, a_3^{(2)}, a_4^{(2)}, a_5^{(2)})</td>
<td>000101</td>
</tr>
<tr>
<td>(a_2^{(3)}, a_3^{(3)})</td>
<td>000001</td>
</tr>
<tr>
<td>(a_1^{(1)}, a_4^{(1)}, a_6^{(1)}, a_8^{(1)}, a_5^{(3)}, a_7^{(3)})</td>
<td>100101</td>
</tr>
<tr>
<td>(a_2^{(3)}, a_4^{(3)}, a_6^{(3)}, a_8^{(3)})</td>
<td>100001</td>
</tr>
<tr>
<td>(a_6^{(2)}, a_7^{(2)}, a_8^{(2)})</td>
<td>101001</td>
</tr>
</tbody>
</table>

Encoding and decoding for these codes is done in the same manner as for the (1,3) constrained error check code.

The codes presented detect single shift errors and single drop in/out errors. The ease of encoding and decoding makes these codes practical for applications.

3.6 Probability of an Undetected Error for the Systematic \((d,k)\) Constrained Block Codes

We want now to compute the average probability of an undetected error for the family of systematic \((d,k)\) constrained block codes. We will get an upper bound on the probability which decreases exponentially and is independent of the channel model chosen. The method for this proof is the standard trick of averaging over a large family of codes.
First we discuss what is meant by systematic \((d, k)\) constrained block codes. The systematic codes were discussed in the previous section. The idea of a systematic code is as follows. An input sequence of length \(l\) is made into a code sequence by concatenating a sequence of length \(r\) onto it. The length \(l\) sequence is called the information and the length \(r\) sequence is called the parity. This gives code sequences of length \(l + r\).

The \((d, k)\) constrained block codes were introduced in the first section of this chapter. All information sequences, for the codes we consider here, are assumed to have an associated edge path in the \((d, k)\) constraint graph whose initial vertex is a specified vertex \(u\). Also, all parity sequences for the code are assumed to have an associated edge path in the \((d, k)\) constraint graph whose terminal vertex is the vertex \(u\). These conditions on the information and parity ensures that the code satisfies the concatenability condition for a \((d, k)\) constrained block code.

The error check codes of the previous section are examples of systematic \((d, k)\) constrained block codes.

We want to compute the average probability of an undetected error for the systematic \((d, k)\) constrained block described above. We take \(u\) to be the initial vertex of the information and the terminal vertex of the parity.

Take \(a_1, a_2, \ldots, a_{k+1}\) to be the vertices of the \((d, k)\) constraint graph.

Now let \(N_i\) be the set of information sequences whose associated edge paths in the constraint graph start in vertex \(u\) and end in vertex \(a_i\). Similarly, let \(iN\) be the
set of parity sequences of length $r$ whose associated edge paths in the constraint
graph start in vertex $a_i$ and end in vertex $u$.

The size of a code will be: $|C| = \sum_{i=1}^{k+1} |N_i|$ 

Let $F$ be the family of all systematic $(d, k)$ constrained block codes with infor-
mation length $l$ and parity length $r$. The size of $F$ is: $|F| = \prod_{i=1}^{k+1} |iN|^{|N_i|}$ 

The number of possible code sequences whose associated edge paths have their
information sequence ending in vertex $a_i$ is: $|N_i||iN|$ 

Now let $x$ and $y$ be two possible code sequences with different information se-
quences. Let $F(x, y)$ be the subset of $F$ which consists of all the codes which
contain both $x$ and $y$. The size of $F(x, y)$ is then:

$|F(x, y)| = \begin{cases} 
\prod_{q=1}^{k+1} |qN||N_q||iN||N_i|^{-2}, & x \text{ and } y \in N_i, \\
\prod_{q=1}^{k+1} |qN||N_q||iN||N_i|^{-1}|jN||N_j|^{-1}, & x \in N_i \text{ and } y \in N_j. 
\end{cases}$

Since the size of $F(x, y)$ depends only upon where the edge paths associated with
$x$ and $y$ have their information sequences ending, we let $F_{ij}$ be the size of $F(x, y)$
whenever $x \in N_i$ and $y \in N_j$.

So, this gives for any $i$ and $j$ that:

$F_{ij} = \frac{1}{|F||iN||jN|}$

This will be used later on.
Now we compute the average probability of an undetected error for the family of codes $F$. Actually, we obtain an upper bound which is independent of the channel model used. We start with the average probability of an undetected error:

\[(11) \quad \overline{P(C, p)} = \sum_{C \in F} P(C) \sum_{x \in C} P(x) \sum_{\text{all received } y} P(y|x)P_{\text{err}}(y, x, C)\]

Now assume that each code is equally likely to occur, and that in a given code, each codeword is equally likely to occur. That is:

\[(12) \quad P(C) = \frac{1}{|F|} \text{ and } P(x) = \frac{1}{|C|}\]

So it follows that:

\[(13) \quad \overline{P(C, p)} = \frac{1}{|F||C|} \sum_{C \in F} \sum_{x \in C} \sum_{\text{all received } y} P(y|x)P_{\text{err}}(y, x, C)\]

We consider that an undetected error occurs when the code sequence sent to the channel is changed into a sequence $y$, and the change is undetected. The error will be undetected only when the received sequence $y$ is a code sequence for the code being used. So, the probability that an undetected error occurs will be:

\[(14) \quad P_{\text{err}}(y, x, C) = \begin{cases} 1 & \text{if } x \text{ and } y \in C \\ 0 & \text{otherwise} \end{cases}\]

This gives the average probability as:

\[(15) \quad \overline{P(C, p)} = \frac{1}{|F||C|} \sum_{C \in F} \sum_{x \in C} \sum_{\text{all } y \in C \text{ such that } y \neq x} P(y|x)\]
Rearranging summations gives:

\[ P(C, p) = \frac{1}{|F||C|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{j=1}^{k+1} \sum_{y \in N_j} \sum_{C: x, y \in C} P(y|x) \]  

By all \( x \), we mean that \( x \) is a length \( l + r \) sequence which is a code sequence for some code \( C \in F \). By info \( y \neq \) info \( x \), we mean that the information portion of \( x \) is not equal to the information portion of \( y \). Breaking the summations up by which vertex the information portion ends in gives:

\[ P(C, p) = \frac{1}{|F||C|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{j=1}^{k+1} \sum_{y \in N_j} \sum_{C: x, y \in C} P(y|x) \]

Now, the inner sum will be counted \( F_{ij} \) times. This gives:

\[ P(C, p) = \frac{1}{|F||C|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{j=1}^{k+1} \sum_{y \in N_j} F_{ij} P(y|x) \]

Now recall by equation ten, that \( \frac{F_{ij}}{|F|} = \frac{1}{|i|N_i|j|N_j} \). So we have:

\[ P(C, p) = \frac{1}{|C|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{j=1}^{k+1} \sum_{y \in N_j} \frac{1}{|i|N_i|j|N_j} P(y|x) \]

which becomes:

\[ P(C, p) = \frac{1}{|C|} \sum_{i=1}^{k+1} \frac{1}{|i|N_i} \sum_{x \in N_i} \sum_{j=1}^{k+1} \frac{1}{|j|N_j} \sum_{y \in N_j} P(y|x) \]
Now the inner term $\frac{1}{|jN|}$ can be taken outside the summations, by maximizing of $j$. This gives:

$$P(C, p) \leq \frac{1}{\min_q |qN| |C|} \frac{1}{|N_i|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{j=1}^{k+1} \sum_{y \in N_j} P(y|x)$$

Up to this point, no assumptions have been made about the channel model. In practice, the errors in transmission appear as a mixture of shift errors, drop in errors, and drop out errors. To give an expression for $P(y|x)$, we would need to chose a model for the error probabilities. This does not appear to be easy to do. However, we can get an upper bound on the probability of an undetected error by observing:

$$\sum_{j=1}^{k+1} \sum_{y \in N_j} P(y|x) \leq \sum_{\text{all } y} P(y|x) = 1$$

So, for any channel we get that:

$$P(C, p) \leq \frac{1}{\min_q |qN| |C|} \frac{1}{|N_i|} \sum_{i=1}^{k+1} \sum_{x \in N_i} \sum_{y \in N_j} 1$$

Now recall that the number of admissible code sequences which have the edge paths associated with their information portion ending in vertex $a_i$ is $|N_i| |iN|$. This gives:

$$P(C, p) \leq \frac{1}{\min_q |qN| |C|} \frac{1}{|N_i|} \sum_{i=1}^{k+1} |N_i||iN|$$

But the summation is equal to $|C|$, so we have:

$$P(C, p) \leq \frac{1}{\min_q |qN|}$$
We summarize the discussion above in the theorem below.

**Theorem 7.** The probability of an undetected error averaged over the family of systematic \((d, k)\) constrained block codes is less than \[
\frac{1}{\min_q |q^N|}.
\]

This theorem shows that systematic \((d, k)\) constrained block codes exist with small probability of an undetected error. The theorem suffers from the standard defect of such theorems, in that it is nonconstructive.

We want to obtain an upper bound in terms of the capacity of the \((d, k)\) constrained codes. In chapter one, the capacity of the \((d, k)\) constrained codes is introduced. The capacity is equal to \(c = \lim_{n \to \infty} \frac{\log_2 N(n)}{n} = \log_2 \lambda\), where \(N(n)\) is the number of \((d, k)\) constrained sequences of length \(n\) and \(\lambda\) is the Perron eigen-value of state transition matrix for the \((d, k)\) constraint graph. So we see that there are constants \(A\) and \(B\) such that for all \(n\), \(A\lambda^n \leq N(n) \leq B\lambda^n\).

Now let \(qN(r)\) be the set of length \(r\) \((d, k)\) constrained sequences whose associated edge paths in the \((d, k)\) constraint graph have initial vertex \(a_q\) and have terminal vertex \(u\). Take \(t\) to be the minimum number such that there are length \(t\) proper edge paths in the \((d, k)\) constraint graph from the vertex \(a_q\) to each of the vertices in the graph. Take \(s\) to be the minimum number such that there are length \(s\) proper edge paths in the \((d, k)\) constraint graph from each of the vertices in the graph to the vertex \(u\). Then \(|qN(r)| \geq N(r - s - t) \geq A\lambda^{n-s-t}$. 

95
Since $q$ was not specified, this shows that there is a constant $\alpha$ such that for all $q$ and all $r$, $\alpha |qN(r)| \geq \lambda^r$. We use this to prove theorem eight.

**Theorem 8.** The probability of an undetected error averaged over the family of systematic $(d, k)$ constrained block codes is less than $\alpha 2^{-cr}$. where $\alpha$ is a constant for all $r$, $c$ is the capacity of the $(d, k)$ constrained codes, and $r$ is the parity length of the codes.

**proof.** The probability of an undetected error satisfies the following string of inequalities:

\[
\frac{1}{P(C,p)} \leq \max_q \frac{1}{|qN|} \leq \frac{\alpha}{\lambda^r} = \frac{\alpha}{2^{cr}}
\]

This shows that the probability of an undetected error decreases exponentially with the capacity times the parity length.

**3.7 Conclusion**

In this chapter, we introduced $(d, k)$ constrained block codes that are constructed using a covering space of the $(d, k)$ constraint graph. The base point codes consist of the $(d, k)$ constrained sequences whose associated edge paths in the covering space are loops at a specified vertex. The parity point codes consist of the $(d, k)$ constrained sequences whose associated edge paths in the covering space connect two specified vertices. The $(d, k)$ constrained error check code is a systematic


$(d, k)$ constrained block code constructed from a three fold covering of the $(d, k)$ constraint graph. The error check codes detect all single shift errors, drop in errors, and drop out errors.

In the final section, an upper bound on the average probability for an undetected error, when using a systematic $(d, k)$ constrained block code is found.
Appendix

Algorithm to find the Free Distance

do  a  Ternary Convolutional Code

The rate 1/2 ternary convolutional codes presented in table 1.7 are the non-catastrophic codes of largest free distance. The codes were found by a computer search. The algorithm used in the search is a slight modification of the algorithm in [L]. In this appendix, we briefly discuss the algorithm.

The algorithm to find the free distance of a convolutional code, uses the state diagram of the convolutional code. See [LC] for an introduction to the state diagram for a convolutional code. The state diagram of a ternary convolutional code with memory $m$, is a directed graph with $3^m$ vertices. Each vertex has three outgoing edges and each vertex has three incoming edges. The three outgoing edges to a vertex correspond to the three possible ternary inputs when the encoder is in the state associated with the vertex. Each edge is labeled with a triple $x/yz$ of ternary elements. The $x$ corresponds to the input and the $yz$ correspond to the output. The Hamming weight of an edge will be the Hamming weight of the pair $yz$.

Each code sequence from the convolutional code has a corresponding proper edge path in the state diagram, with the initial vertex being the zero vertex. The Hamming weight of the code sequence will be the sum of the weights of the edges in the edge path corresponding to the sequence. The algorithm for finding the free
distance searches for the edge path which is a loop at the zero vertex and which has minimum weight for all loops at the zero vertex. The weight of this edge path will be the free distance of the convolutional code.

The algorithm proceeds by examining proper edge paths and reversed edge paths simultaneously. The difference in the algorithm for ternary codes and the algorithm in [L] for binary codes is that each vertex has three edges leaving it in the state diagram. This leads to a modification of the initialization and to an extra edge path to consider when extending each vertex. A diagram of this process is shown below. The notation used is explained below:

- $T_m$: type of edge path used to reach the vertex (proper=1 or reversed=2)
  
  positive if the vertex has not been extended and negative if it has

- $w_m$: minimum weight of all the edge path which have reached the vertex

- $w^*$: present estimate of the free distance
Figure A.1: Algorithm for Finding $d_{\text{free}}$ of a Ternary Convolutional Code
Figure A.1: (continued) Algorithm for Finding $d_{\text{free}}$ of a Ternary Convolutional Code

1. search for state $S_k = S_m$
   - If yes, found same state
     - If no, paths to state are the same type
       - If yes, state is dead
         - $w_k = \min(w_k, w_m)$
       - If no, set $w^* = \min(w^*, w_m + w_k)$
   - If yes, store state $S_m$, type, and weight

2. $w_k = w_m$ and type equal to type of $S_m$
Bibliography


