Perfect Codes on Odd Dimension Serpinski Graphs

Stephanie Kleven
Advisor: Paul Cull
Central College and Oregon State University

August 15, 2003

Abstract

Serpinski graphs K_d^n are built by an iterative connection from a complete graph K_d. These graphs have essentially one perfect one error correcting code defined on their vertices. We investigate assigning strings strings over $\{0,...,d-1\}$ to vertices, so that the resulting code on strings has several desirable properties. Our SF labeling is defined when d is odd and it becomes the Towers of Hanoi Labeling when $d = 3$. The SF labeling preserves the gray code property observed in the Towers of Hanoi labeling and it has a $2d+2$ state machine for error detection and a finite state machine for error correction.

1 Introduction

Codes have long been used to transfer messages. However, sometimes the encoded message may be distorted, either through electronic or magnetic interference, as in the case with computers, or simply through human error. Error correcting codes have been developed to correct the more common errors. These error correcting codes take substrings of the coded string and associate each with a set of strings which does not contain any other substring of the coded string, but contains the most common distortions of the desired substring. Thus, when an incorrect substring is encountered, the error may be corrected by changing the incorrect substring to the appropriate substring associated with it. An error correcting code can be represented by a graph, with the substrings, both correct and incorrect, represented by vertices on the graph. A perfect one error correcting code associates each non-codevertex with its adacent codevertex. Once a perfect one error correcting code is established on a graph, a regular labeling of the vertices is desired that allows easy codeword recognition, error correction, encoding, decoding, and less variation between a codeword and its associated non-codewords than between two distinct codewords. The best two-dimensional labeling of a perfect one error correcting code on iterated complete graphs is the Towers of Hanoi labeling on K_3^n. Labelings with one or two desirable properties have been found, but it seems difficult to unite all desirable properties in one labeling system that works for all K_d^n. Our SF labeling attempts to maintain as many desirable properties of the Towers of Hanoi labeling as possible on the graphs K_d^n, where d is odd.

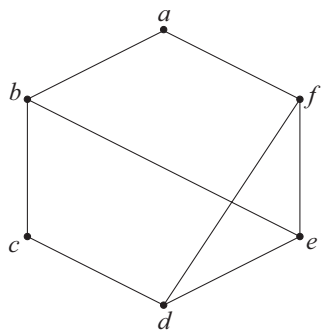

Figure 1: An example of a graph

1.1 Definitions

This section gives definitions from graph theory that we will use throughout the paper.

Definition 1.1 *A **simple graph**, or more generally, a **graph** G consists of a nonempty finite set $V(G)$ of elements called **vertices** and a finite set $E(G)$ of distinct unordered pairs of distinct elements of $V(G)$ called **edges**. Two vertices $v_i, v_j \in V(G)$ are **adjacent** if $\{v_i, v_j\} \in E(G)$. A **subgraph** S of G consists of a subset $V(S) \subset V(G)$ together with the edges connecting adjacent vertices $v_i, v_j \in V(S)$.*

Example 1.2 *Figure 1 shows Graph G with vertex set $V(G) = \{a,b,c,d,e,f\}$ and edge set $E(G) = \{(a,b),(b,c),(c,d),(d,e),(e,f),(a,f),(b,e),(d,f)\}$.*

Definition 1.3 *The **degree** of a vertex v, denoted $deg(v)$, is the number of vertices connected to v by an edge. (In the example above $deg(a) = deg(c) = 2$).*

Definition 1.4 *Two disjoint subgraphs K and M of a graph G are **adjacent** if there exists $k \in V(K)$ and $m \in V(M)$ such that $\{k,m\} \in E(G)$.*

Definition 1.5 *A **complete graph on d vertices** is a graph in which each pair of distinct vertices are adjacent. We will denote the complete graph on d vertices by K_d.*

Example 1.6 *Figure 2 shows complete graphs on $d = 4$, 6, and 9 vertices.*

The following is a description of the iterative construction for a bi-infinite family of graphs, K_d^n. The first iteration of the graph K_d^1 is simply the complete graph on d vertices. The second iteration, K_d^2, connects d copies of K_d^1 so that each copy is adjacent. In general, when constructing K_d^n, we create d copies of $K_d^{n-1}$and form edges between the corner vertices of distinct copies of K_d^{n-1} such that any one of the d copies of K_d^{n-1} is adjacent to each of the other $d-1$ copies. The family is bi-infinite because there is an infinite number of dimensions d and an infinite number of iterations n.

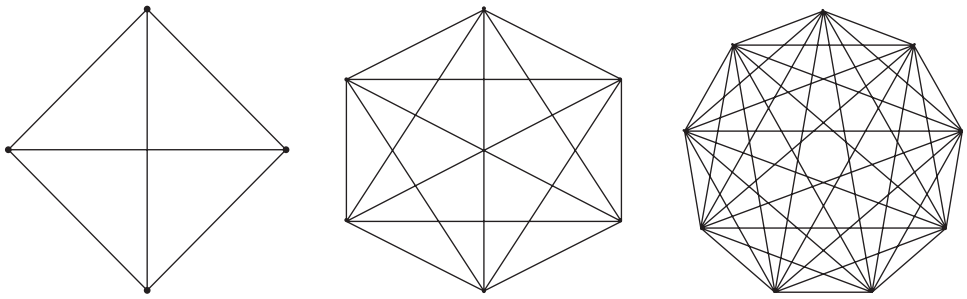

Figure 2: The complete graphs K_4, K_6, and K_9.

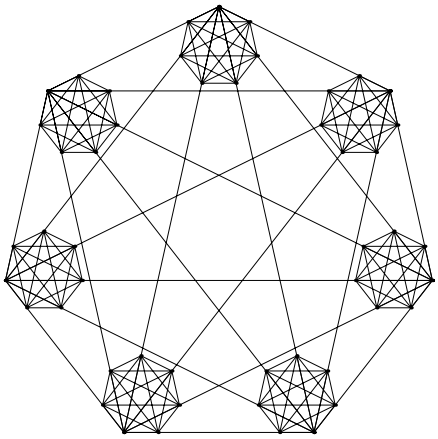

Figure 3: The graph of K_7^2

Definition 1.7 *A **corner vertex** is a vertex of an iterated complete graph K_d^n whose degree is $d-1$. An **internal vertex** is a non-corner vertex and has degree d.*

Example 1.8 *Figure 3 shows two iterations of a complete graph on 7 vertices, K_7^2.*

2 Codes on Graphs

We are interested in perfect one-error correcting codes on iterated complete graphs. We will begin with a few definitions.

2.1 Definitions

Definition 2.1 *A **code** on a graph G is any subset of vertices $C(G) \subset V(G)$. A vertex $c \in C(G)$ is called a **codevertex**. A vertex $v \in V(G) - C(G)$ is called a **non-codevertex**.*

When applying a coding scheme to a set of vertices, it is useful to associate each non-codevertex with a distinct codevertex in order to correct errors. A perfect one error correcting code associates each non-codevertex with the adjacent codevertex.

Definition 2.2 *A **perfect one error correcting code** or p1ecc on a graph is a code that satisfies the following properties:*

1. *No two codevertices are adjacent*
2. *Every non-codevertex is adjacent to exactly one codevertex.*

For each iterated complete graph K_d^n there is a unique p1ecc, [1]. It is iteratively made up of two different weak codes, the $G-code$ and the $U-code$, that define the position of the codevertices on K_d^n.

2.2 Weak Codes and Unique Perfect One Error Correcting Codes on K_d^n

Weak codes are nearly p1ecc's. However, some corner vertices are non-codevertices that are not adjacent to a codevertex and therefore are not associated with any vertex. We will prove the existence and uniqueness of weak codes on K_d^n, from which the existence and uniqueness of p1ecc's on K_d^n will follow. All definitions, lemmas, and theorems in this subsection are from the REU paper written by Alspaugh, Knight, and Meloney, [1].

Definition 2.3 *A **weak code** on a graph is a code that satisfies the following properties:*

1. *No two codevertices are adjacent*
2. *Every internal vertex is a codevertex or is adjacent to exactly one codevertex.*

Definition 2.4 *If a vertex $v \in V\left(K_d^n\right)$ is:*

1. *A codevertex that is not adjacent to any other codevertex, denote it by c.*
2. *A non-codevertex that is adjacent to exactly one codevertex, denote it by a.*
3. *A non-codevertex that is not adjacent to any codevertex, denote it by x.*

Definition 2.5 *A **perfect one error correcting connection** or p1ecc-c is a connection between two corner vertices v_1 and v_2 that satisfies the following properties:*

1. *No two codevertices are adjacent (v_1 and v_2 are not both codevertices)*
2. *v_i is a codevertex or is adjacent to exactly one codevertex.*

Note that a p1ecc-c will preserve a weak code and a p1ecc.

Lemma 2.6 *There are exactly two p1ecc-c: $x-c$ and $a-a$.*

Proof. There exist three types of vertices: x, a, and c. Therefore there are six possible connections: $x-a$, $x-c$, $a-c$, $a-a$, $x-x$, and $c-c$. We cannot have $c-c$ because no two codevertices can be adjacent. We cannot have $x-x$ because both x vertices will not be adjacent to a codevertex. We cannot have $c-a$ because a will be adjacent to two codewords. We cannot have $x-a$ because the x vertex will not be adjacent to any codevertex. So there are at most two connections: $a-a$ and $x-c$. To see that these are p1ecc-c's, note that for $a-a$ clearly no two codewords are adjacent and each vertex is adjacent to a codevertex c. Note that with an $x-c$ connection, an x vertex is adjacent to exactly one codevertex. Thus, no two codevertices are adjacent and $x-c$ is a p1ecc-c. ■

The following definitions describe the two types of weak codes on iterated complete graphs. We will use them to prove that the construction method for choosing codevertices on K_d^n is unique.

Definition 2.7 *A $G-code$, G_d^n, is a weak code on d vertices of n iterations such that if*

1. n is even, then all corner vertices are c's.

2. n is odd, then exactly one corner vertex is a c, and all other corner vertices are a's.

Notice that a $G-code$ is actually a p1ecc since no corner vertex is an x.

Definition 2.8 *A $U-code$, U_d^n, is a weak code on d vertices of n iterations such that if*

1. n is even, then exactly one corner vertex is an x, and all other corner vertices are a's.
2. n is odd, then all corner vertices are x's.

Theorem 2.9 *There are exactly two weak codes on K_d^n, the $G-code$ and the $U-code$.*

Proof. We will use induction on n, the number of iterations. For $n=1$, K_d^1 has exactly two weak codes. If no vertex is chosen as a codeword, then K_d^1 is a $U-code$. If a single vertex is chosen as a codeword, then K_d^1 is a $G-code$.

For our induction, we assume that for $n=k-1$ there exist exactly two weak codes, namely G_d^{k-1} and U_d^{k-1}. There are two cases to consider.

Case 1: K_d^k has at least one corner vertex that is a c.

If a corner vertex of K_d^k is a c, then it must come from some copy of G_d^{k-1}, since U_d^{k-1} has no c corner vertices. We must now consider the two subcases of k odd and k even.

If k is even, then the other $d-1$ corner vertices of G_d^{k-1} are a by the definition of a $G-code$. Since the only p1ecc-c are $x-c$ and $a-a$, the a corner vertices can only connect to other a corner vertices. However, by the definition of a $U-code$ every corner vertex of U_d^{k-1} will be an x. Therefore, no copy of U_d^{k-1} will appear in K_d^k. Since each a in G_d^{k-1} can only connect to another a, we will have d copies of G_d^{k-1}. Connecting the G_d^{k-1} subcodes with $a-a$ forces each corner vertex of K_d^k to be a c. Thus, K_d^k is actually G_d^k.

If k is odd, then by definition all the corner vertices of the G_d^{k-1}subgraph are c. In order to make p1ecc-c's, the other $d-1$ subgraphs must each have an x corner vertex to connect to the c

vertex from the G_d^{k-1}subgraph, which means they must all be copies of $U_d^{k-1}$and must form $a-a$ connections among themselves. Since one corner vertex of K_d^k is a c corner vertex from G_d^{k-1}, and the other $d-1$ corner vertices are a corner vertices from U_d^{k-1}, K_d^k is actually G_d^k.

We have just shown that in the case where at least one of the corner vertices of K_d^k is a c, then there is exactly one weak code that is produced, a $G-code$.

Case 2: No corner vertex of K_d^k is a c.

Let no corner vertex of K_d^k be of type c. Then K_d^k cannot be constructed of only G_d^{k-1} subgraphs because the corner vertices of G_d^{k-1} are either c or a, and our assumption therefore forces an illegal $a-c$ connection. Thus, K_d^k must contain at least one U_d^{k-1} subcode. We will now consider the two subcases of k odd and k even.

If k is even, then all corner vertices of U_d^{k-1} are x by the definition of a $U-code$. Since the only p1ecc-c's for an x vertex is an $x-c$ connection and G_d^{k-1} is the only code that has c as a corner vertex, we must connect U_d^{k-1} to a G_d^{k-1}, which has exactly one c as a corner vertex. Therefore $d-1$ of the x vertices in U_d^{k-1} will make $x-c$ connections with the $d-1$ copies of G_d^{k-1}, and the G_d^{k-1} will make $a-a$ connections among themselves. Furthermore, exactly one corner vertex of K_d^k will be an x and the remaining $d-1$ corner vertices will be a. So we see that K_d^k is in fact the weak code U_d^k.

If k is odd, then U_d^{k-1} has exactly one x as a corner vertex, and the remaining corner vertices must be a. Since all corner vertices of G_d^{k-1} are c's, we see that K_d^k must be made up of only U_d^{k-1} subcodes. The x vertex must be a corner vertex of K_d^k, otherwise it would be connected to a c, which belongs only to G_d^{k-1}. The other $d-1$ vertices of $U_d^{k-1}$are all a's and can only be connected to another a corner vertex of U_d^{k-1}. Furthermore, the x vertex in each of the U_d^{k-1} will become the corner vertex of U_d^k and will remain unconnected. Hence K_d^k is in fact the weak code U_d^k.

We have just shown that in the case where none of the corner vertices of K_d^k are a c, then there is exactly one weak code that is produced, the $U-code$.

Thus by induction, there are exactly two weak codes on K_d^n, the $G-code$ and the $U-code$. ■

We have just proven existence and uniqueness for weak codes. Proving the same for a perfect one error correcting code on K_d^n is very easy.

Theorem 2.10 *There is exactly one (up to isomorphism) perfect one error correcting code on K_d^n.*

Proof. The perfect one error correcting code on K_d^n is the $G-code$. When n is even, the $G-code$ is invariant under rotations of the graph and is truly unique. When n is odd, one corner vertex is always a codevertex. Fixing the codevertex as the top vertex will ensure that the p1ecc is unique. Therefore there is exactly one (up to isomorphism) perfect one error correcting code on K_d^n. ■

2.3 The $G-U$ Construction

Throughout this explanation the process is illustrated with examples from the $d=5$ case (pentagons). However, the general construction is described for any d.

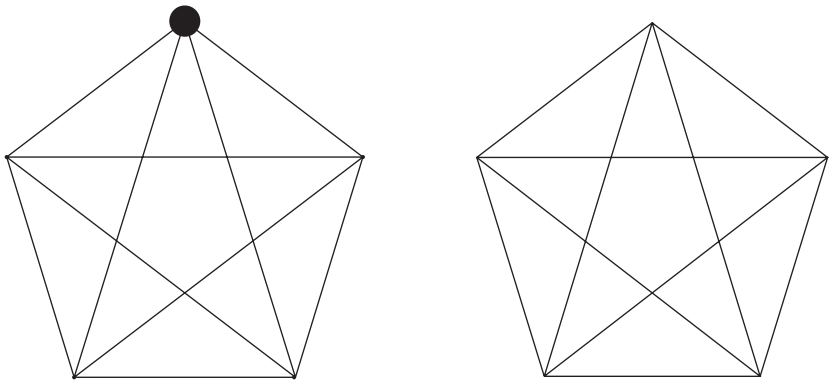

Figure 4: G_5^1 and U_5^1

To construct G_d^1, one vertex of K_d^1 is designated as the codevertex and it is rotated to the top position. This vertex is now referred to as the **top vertex**. All of the other corner vertices are referred to as **non-top corner vertices**. In the construction of U_d^1, no vertex of K_d^1 is chosen as a codevertex; instead, one vertex is designated as the top vertex.

Example 2.11 *Figure 4 shows G_5^1 and U_5^1. The codevertex is represented by a dot.*

The iterative construction of G_d^n and U_d^n for any $n > 1$ is as follows:

To construct G_d^n when n is even, create d copies of G_d^{n-1}. Make connections between these copies such that:

1. The top vertex of each copy of G_d^{n-1} remains unconnected.
2. Each copy is adjacent to all other $d-1$ copies.
3. Designate the top vertex of some G_d^{n-1} to be the top vertex of G_d^n.

Note that all corner vertices will be c vertices.

To construct U_d^n when n is even, create 1 copy of U_d^{n-1} and $d-1$ copies of G_d^{n-1}. Connect these copies so that:

1. The top vertex of U_d^{n-1} remains unconnected and non-top vertices of $U_d^{n-1}$are connected to top vertices of distinct copies of G_d^{n-1}.
2. Connect non-top corner vertices of G_d^{n-1} to the other $d-2$ copies of G_d^{n-1} such that they are all adjacent and one corner vertex remains unconnected.
3. The top vertex of U_d^{n-1} becomes the top vertex of U_d^n.

Note that the top vertex will be an x, and the remaining $d-1$ vertices will be a's.

To construct G_d^n when n is odd, create 1 copy of G_d^{n-1} and $d-1$ copies of U_d^{n-1}. Connect them such that:

1. The top vertex of G_d^{n-1} remains unconnected and and non-top vertices of $G_d^{n-1}$are connected to top vertices of distinct copies of U_d^{n-1}.

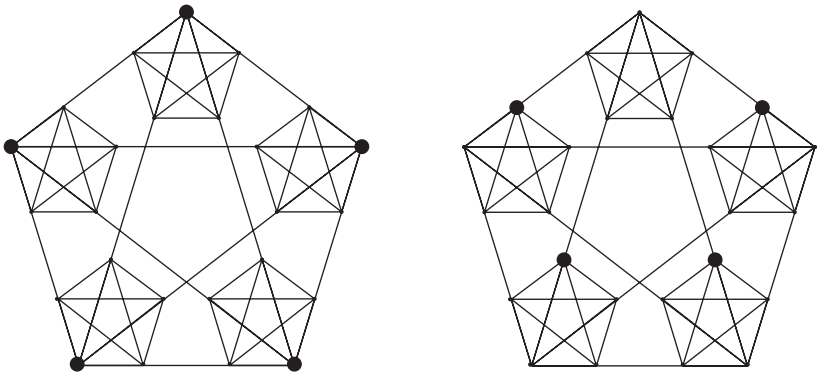

Figure 5: G_5^2 and U_5^2

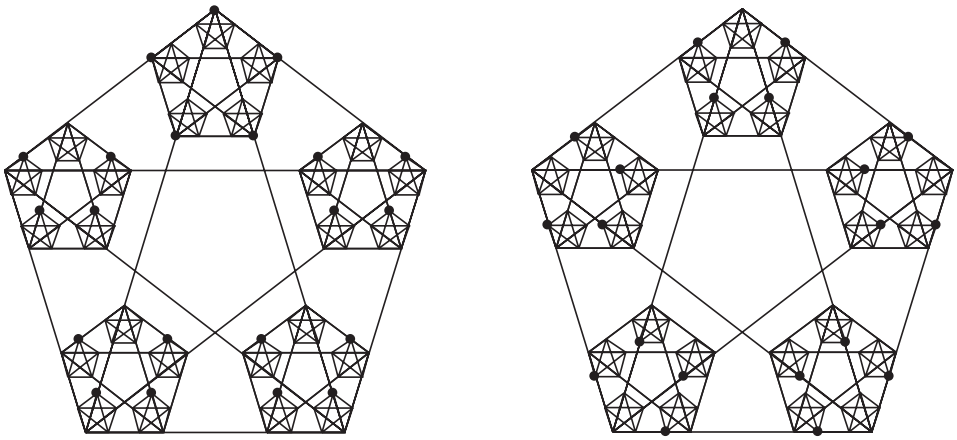

Figure 6: G_5^3 and U_5^3

2. Connect non-top corner vertices of U_d^{n-1} to the other $d-2$ copies of U_d^{n-1} such that they are all adjacent and one corner vertex remains unconnected.

3. The top vertex of G_d^{n-1} becomes the top vertex of G_d^n.

Note that the top vertex will be an x, and the remaining $d-1$ vertices will be a's.

To construct U_d^n when n is odd, create d copies of G_d^{n-1}. Make connections between these copies such that:

1. The top vertex of each copy of U_d^{n-1} remains unconnected.
2. Each copy is adjacent to all other $d-1$ copies.
3. Designate the top vertex of some G_d^{n-1} to be the top vertex of G_d^n.

Note that all corner vertices will be x vertices.

Example 2.12 *Figures 5 and 6 show G_5^2, U_5^2, G_5^3, and U_5^3.*

2.4 The Number of Codewords in a Perfect One Error Correcting Code on K_d^n

Once we have a way of constructing a perfect one error correcting code on a graph, the next theorem allows us to explicitly calculate the number of codevertices on the graph.

Theorem 2.13 *If C_d^n is a p1ecc on K_d^n, then the number of codevertices c_n in C_d^n is:*

$$c_n = \frac{d^n + \frac{d+1}{2} + \frac{d-1}{2}(-1)^n}{d+1} = \begin{cases} (d^n+d)/(d+1), & n \text{ is even} \\ (d^n+1)/(d+1), & n \text{ is odd} \end{cases}$$

Proof. This proof is based on the $G-U$ construction and is a generalized form of a proof given by Cull and Nelson for the $n=3$ case, [4]. For a proof using an alternate construction see Birchall and Tedor, [2].

Let g_n be the number of codevertices in G_n and u_n be the number of codevertices in U_n. Then, from the $G-U$ construction, we have:

$g_n = d\,g_{n-1}$ n even
$g_n = g_{n-1} + (d-1)u_{n-1}$ n odd
$u_n = (d-1)g_{n-1} + u_{n-1}$ n even
$u_n = d\,u_{n-1}$ n odd

with $g_0 = 1$ and $u_0 = 0$. Notice that $g_n - u_n = d\,g_{n-1} - [(d-1)g_{n-1} + u_{n-1}] = g_{n-1} - u_{n-1}$, for n even and $g_n - u_n = g_{n-1} + (d-1)u_{n-1} - d\,u_{n-1} = g_{n-1} - u_{n-1}$, for n odd. Since $g_0 - u_0 = 1$, $g_n - u_n = g_{n-1} - u_{n-1} = 1$, for all n. Then, when n is even, we have $g_n = d\,g_{n-1} = d\,g_{n-1} + a(-1+(-1)^n)$, since $(-1+(-1)^n) = 0$ when n is even. When n is odd, we have $g_n = 1 + u_n = d\,u_{n-1} + u_n = d\,(g_{n-1}-1) + 1 = d\,g_{n-1} - d + 1$. Noticing that this equation is similar to that for g_n when n is even, we set $a(-1+(-1)^n) = -d+1$, with n odd, and find that $a = \frac{d-1}{2}$. So, $g_n = d\,g_{n-1} + \frac{d-1}{2}(-1+(-1)^n)$, for all n, which we see by induction is equivalent to the formula for c_k given above. For the base case, let $n = 0$.

$$c_0 = \frac{d^0 + \frac{d+1}{2} + \frac{d-1}{2}(-1)^0}{d+1} = \frac{1+d}{d+1} = 1 = g_0.$$

Suppose that $c_k = g_k$, or $\dfrac{d^k + \frac{d+1}{2} + \frac{d-1}{2}(-1)^k}{d+1} = d\,g_{k-1} + \frac{d-1}{2}\left(-1+(-1)^k\right)$, for some $k \geq 1$.

Let $n = k+1$. Then

$$g_k = d\,g_k + \tfrac{d-1}{2}\left(-1+(-1)^{k+1}\right)$$
$$= d\left(\frac{d^k + \frac{d+1}{2} + \frac{d-1}{2}(-1)^k}{d+1}\right) + \tfrac{d-1}{2}\left(-1+(-1)^{k+1}\right)$$
$$= \frac{d^{k+1} + \frac{d^2+d}{2} + \frac{-d^2+d}{2}(-1)^{k+1} - \frac{d^2-1}{2} + \frac{d^2-1}{2}(-1)^{k+1}}{d+1}$$
$$= \frac{d^{k+1} + \frac{d+1}{2} + \frac{d-1}{2}(-1)^{k+1}}{d+1}.$$

Therefore, by induction $c_n = \dfrac{d^n + \frac{d+1}{2} + \frac{d-1}{2}(-1)^n}{d+1}$. ■

3 The SF Labeling

The SF labelings are based on single rotations and single permutations. They have finite state machines for codeword recognition and error correction. In the case $n = 3$, the SF labeling corresponds to the Towers of Hanoi labeling found by Cull and Nelson, [4]. It is a Gray code labeling of K_d^n.

Definition 3.1 *A **Gray code** is a labeling in which the labels of two adjacent vertices differ in one digit.*

Definition 3.2 *A **word** is the label of a vertex in a labeled graph K_d^n. A **codeword** is the label of a codevertex and a **non-codeword** is the label of a noncodevertex.*

The following definition will be used when we describe recognizing codewords and correcting non-codewords.

Definition 3.3 *A finite state machine is defined by the quintuple*

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

Q is a finite set of internal states,
Σ is a finite set of symbols called the input alphabet,
$\delta : Q \times \Sigma \to Q$ is a total function called the transition function,
$q_0 \in Q$ is the initial state,
$F \subset Q$ is a set of final states.

If we look at a graphical representation of M, a string $x = x_1 x_2 \cdots x_n$ is accepted by M if and only if starting at q_0 and following the edges labeled in order $x_1, x_2, \cdots, x_n$, we end up in a state in F.

3.1 Construction of the SF Labeling

To construct K_d^n, we permute each digit z in the labeling of K_d^{n-1} by α, where $\alpha(z) = \frac{d+1}{2} z \bmod d$ and then make d copies of $\alpha\left(K_d^{n-1}\right)$. K_d^n is labeled by rotating the k^{th} subgraph of $\alpha\left(K_d^{n-1}\right)$ by $\frac{2\pi k}{d}$ radians clockwise and appending k to each word. For K_d^1, the label of the top vertex is 0 and the labels of the remaining $d-1$ vertices increase by 1 counterclockwise about the $d-$gon, beginning after the top vertex.

Example 3.4 *This is illustrated in Figures 7, 8, and 9 for K_5^1, K_5^2, and K_5^3.*

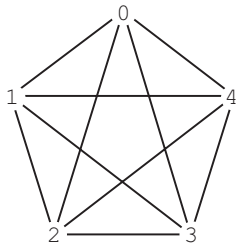

Figure 7: SF labeling of K_5^1

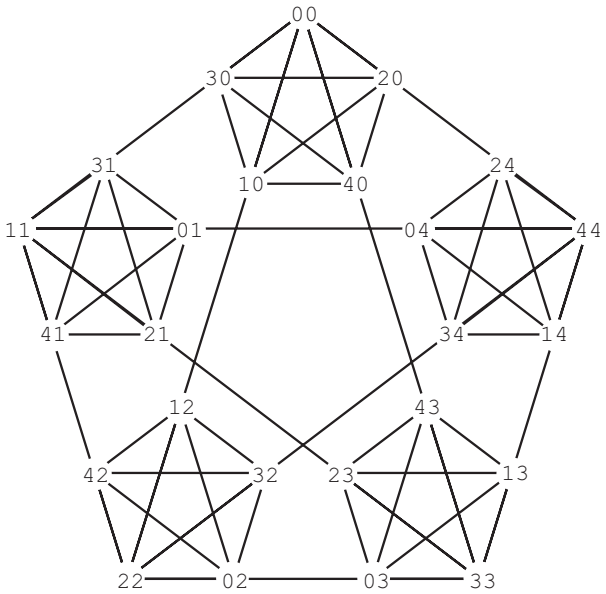

Figure 8: SF labeling of K_5^2

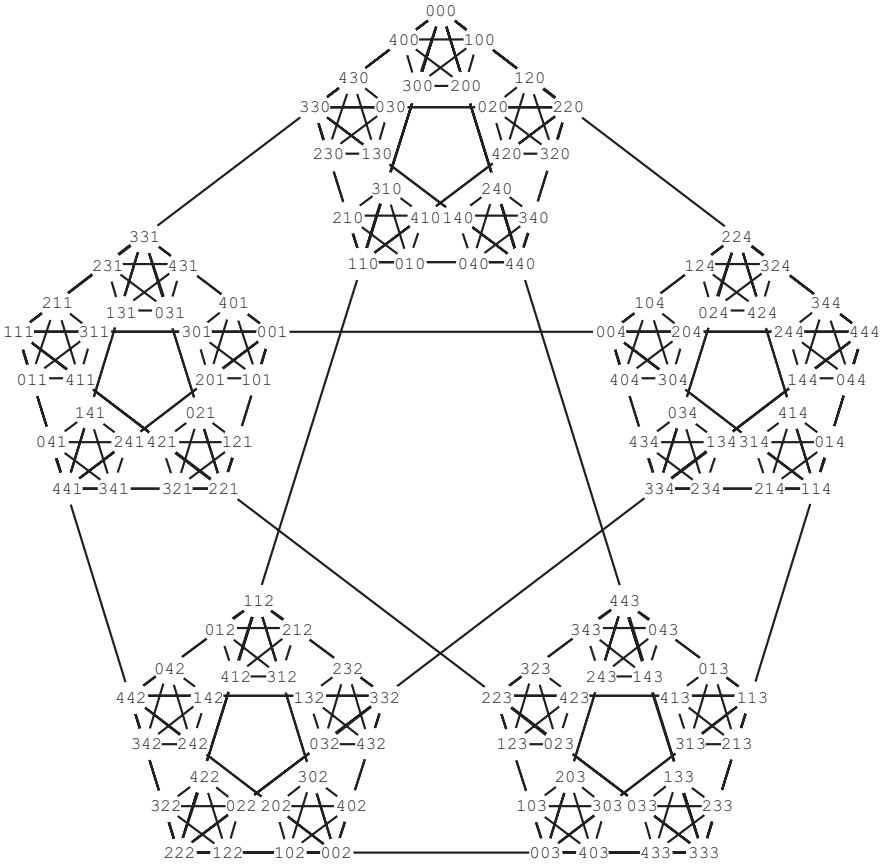

Figure 9: SF labeling of K_5^3

3.2 A Recursive Description of the Codewords

The previous section gave a graphical construction of the codes. In this section, we give a recursive description of the codewords as $d-$ary strings. The SF codewords can be generated by a recursive procedure which follows directly from the construction described above. Here let G_n mean the set of codewords of length n, and let U_n be the auxiliary set of strings of length n. Then, for even n,

$G_n = G_{n-1} \circ 0 \cup T(G_{n-1}) \circ 1 \cup T^2(G_{n-1}) \circ 2 \cup \cdots \cup T^{d-1}(G_{n-1}) \circ (d-1)$

$U_n = U_{n-1} \circ 0 \cup \Gamma_1(G_{n-1}) \circ 1 \cup \Gamma_2(G_{n-1}) \circ 2 \cup \cdots \cup \Gamma_{d-1}(G_{n-1}) \circ (d-1)$, and for odd n,

$G_n = G_{n-1} \circ 0 \cup \Gamma_1(U_{n-1}) \circ 1 \cup \Gamma_2(U_{n-1}) \circ 2 \cup \cdots \cup \Gamma_{d-1}(U_{n-1}) \circ (d-1)$

$U_n = U_{n-1} \circ 0 \cup T(U_{n-1}) \circ 1 \cup T^2(U_{n-1}) \circ 2 \cup \cdots \cup T^{d-1}(U_{n-1}) \circ (d-1)$, where $G_{n-1} \circ 0$ means append a 0 to the right end of each string in G_{n-1}; $\cup$ means set union; T is an operator which replaces each character a in a string by $a+1 \bmod d$; T^m means apply T m times, that is, $T^m(a) = a + m \bmod d$; and Γ_m replaces a character b by $\alpha(m + b \bmod d) = \alpha(T^m(b))$. Of course, when an operation is applied to a set, the operation is applied to each string in the set.

Unlike the Towers of Hanoi labeling, we have so far not been able to find an explicit characterization of the codewords of the SF labelings.

3.3 A Finite State Machine for Codeword Recognition

The FS labeling has a $2d+2$ state machine for codeword recognition when $d > 3$. When $d = 3$, the eight state recognizer simplifies to one with only four states. We have labeled the states of the recognizer as elements of $\left\{ \begin{smallmatrix} E \\ S \end{smallmatrix}, \begin{smallmatrix} E \\ 0 \end{smallmatrix}, \begin{smallmatrix} E \\ 1 \end{smallmatrix}, \cdots, \begin{smallmatrix} E \\ d-1 \end{smallmatrix}, \begin{smallmatrix} O \\ S \end{smallmatrix}, \begin{smallmatrix} O \\ 0 \end{smallmatrix}, \begin{smallmatrix} O \\ 1 \end{smallmatrix}, \cdots, \begin{smallmatrix} O \\ d-1 \end{smallmatrix} \right\}$. Strings will be read from left to right. The recognizer will start in the $\begin{smallmatrix} E \\ S \end{smallmatrix}$ state and strings of even length will be accepted in the $\begin{smallmatrix} E \\ S \end{smallmatrix}$ state and strings of odd length will be accepted in the $\begin{smallmatrix} O \\ 0 \end{smallmatrix}$ state. The transitions among the states can be classified by the function δ:

$\delta\left(\begin{smallmatrix} x \\ y \end{smallmatrix}, z\right) = \begin{pmatrix} \delta_1(x,z) \\ \delta_2(y,z) \end{pmatrix}$

$\delta_1(E,z) = O$

$\delta_1(O,z) = E$

$\delta_2(x,z) = (2x - z) \bmod d$

$\delta_2(x,x) = S$

$\delta_2(S,z) = z$

Proof. We first prove that the codeword recognizer has the minimum number of states necessary. We know that since only the $\begin{smallmatrix} E \\ S \end{smallmatrix}$ state and the $\begin{smallmatrix} O \\ 0 \end{smallmatrix}$ state are accepting states, they must be distinct from the other $2d$ states. By the function δ, we know that $\begin{smallmatrix} O \\ x \end{smallmatrix} \xrightarrow{x} \begin{smallmatrix} E \\ S \end{smallmatrix}$, and since each x is unique, every $\begin{smallmatrix} O \\ i \end{smallmatrix}$, where $i \in \{1, 2, \cdots, d-1\}$, is distinct from every other $\begin{smallmatrix} O \\ i \end{smallmatrix}$. Similarly, since $\begin{smallmatrix} E \\ x \end{smallmatrix} \xrightarrow{2x} \begin{smallmatrix} O \\ 0 \end{smallmatrix}$ and d is odd, we know each $2x$ is distinct mod d , every $\begin{smallmatrix} E \\ i \end{smallmatrix}$, where $i \in \{1, 2, \cdots, d-1\}$, is distinct from every other $\begin{smallmatrix} E \\ i \end{smallmatrix}$. Now we must check if $\begin{smallmatrix} O \\ 2x \end{smallmatrix}$ is distinct from $\begin{smallmatrix} E \\ x \end{smallmatrix}$. We see that they are distinct if and only if $\begin{smallmatrix} E \\ S \end{smallmatrix}$ and $\begin{smallmatrix} O \\ 0 \end{smallmatrix}$ are distinct. We see that given an input x, $\begin{smallmatrix} O \\ 0 \end{smallmatrix} \xrightarrow{x} \begin{smallmatrix} E \\ -x \end{smallmatrix} \xrightarrow{x} \begin{smallmatrix} O \\ -3x \bmod d \end{smallmatrix}$, which is only an accepting state if for all x, $0 = -3x \bmod d$, or $d = 3$. And since $\begin{smallmatrix} E \\ S \end{smallmatrix} \xrightarrow{x} \begin{smallmatrix} O \\ x \end{smallmatrix} \xrightarrow{x} \begin{smallmatrix} E \\ S \end{smallmatrix}$, which is an accepting state, $\begin{smallmatrix} E \\ S \end{smallmatrix}$ and

$\overset{O}{0}$ are distinct if and only if $d \neq 3$. The last two states to check, $\overset{O}{S}$ and $\overset{E}{0}$, are obviously distinct if and only if $\overset{E}{S}$ and $\overset{O}{0}$ are distinct. So the recognizer has the minimum number of states for $d > 3$.

Now we will show that increasing n by 1 causes a certain number of words to flow into and out of each state. The number of codewords in a given state $\#_{\overset{x}{y}}$ is equal to the sum of the number of codewords of length n in each state $\overset{x'}{i}$, where $i \neq y$. In symbols,

$\#_{\overset{E}{y}} = \#_{\overset{E}{0}} + \cdots + \#_{\overset{E}{d-1}} + \#_{\overset{E}{S}} - \#_{\overset{E}{y}}$ and

$\#_{\overset{O}{y}} = \#_{\overset{O}{0}} + \cdots + \#_{\overset{O}{d-1}} + \#_{\overset{O}{S}} - \#_{\overset{O}{y}}$.

The number of words in the $\overset{E}{S}$ state corresponds to the number of codewords with even length, while the number of words in the $\overset{O}{0}$ state corresponds to the number of codewords of odd length. We will proves that $\#_{\overset{E}{S}} = g_n$ when n is even and $\#_{\overset{O}{0}} = g_n$ when n is odd by induction on n. For the base case where $n = 1$, notice that each state $\overset{O}{0}, \overset{O}{1}, \cdots, \overset{O}{d-1}$ contains one word while all other states contain no words, and in particular notice that the $\overset{O}{0}$ state contains the codeword 0. When $n = 2$, each state $\overset{E}{0}, \overset{E}{1}, \cdots, \overset{E}{d-1}$ contains $d-1$ words, while the state $\overset{E}{S}$ contains d words, and there are d codewords in g_n. So now we will suppose that when $n = k$ and k is odd, $\#_{\overset{O}{i}} = \frac{d^k+1}{d+1}$, $i \in \{0, \ldots d-1\}$, and $\#_{\overset{O}{S}} = d^k - d\left(\frac{d^k+1}{d+1}\right) = \frac{d^k-d}{d+1}$, and that when k is even, $\#_{\overset{E}{S}} = \frac{d^k+d}{d+1}$ and $\#_{\overset{E}{i}} = \frac{1}{d}\left(d^k - \frac{d^k+d}{d+1}\right) = \frac{d^k-1}{d+1}$, $i \in \{0, \ldots d-1\}$. Let $n = k+1$. Then we have two cases: either $k+1$ is even or it is odd. Suppose $k+1$ is odd. Then $\#_{\overset{O}{i}} = (d-1)\left(\frac{d^k-1}{d+1}\right) + \frac{d^k+d}{d+1} = \frac{d^{k+1}+1}{d+1}$ and $\#_{\overset{O}{S}} = d\left(\frac{d^k-1}{d+1}\right) = \frac{d^{k+1}-d}{d+1}$, so $\#_{\overset{O}{0}} = g_n$ when $k+1$ is odd. Now suppose $k+1$ is even. Then $\#_{\overset{E}{i}} = (d-1)\left(\frac{d^k+1}{d+1}\right) + \frac{d^k-d}{d+1} = \frac{d^{k+1}-1}{d+1}$ and $\#_{\overset{E}{S}} = d\left(\frac{d^k+1}{d+1}\right) = \frac{d^{k+1}+d}{d+1}$, so $\#_{\overset{E}{S}} = g_n$ when $k+1$ is even. Therefore, since the number of codewords of length n is the same as the number of words that end in a final state of the codeword recognizer and every codeword tested so far (up to length 9) has ended in the accepting state of the recognizer, we conclude that the codeword recognizer accurately distinguishes between noncodewords and codewords. ■

Note also that U_n codewords end in the $\overset{E}{0}$ state if n is even and in the $\overset{O}{S}$ state if n is odd. We will use this property when we correct errors.

Example 3.5 *Figure 10 shows the codeword recognizer for* K_5^n.

Example 3.6 *To get a more intuitive feeling for how the recognizer works, take a label in* K_5^n, *say* 142, *and begin in the* $\overset{E}{S}$ *state. The* 1 *takes us to the* $\overset{0}{1}$ *state, the* 4 *to the* $\overset{E}{3}$ *state, and the* 2 *to the* $\overset{0}{4}$ *state. Since the* $\overset{0}{4}$ *state is not a final state for G codewords,* 142 *is not a codeword in* K_5^3.

3.4 A Finite State Machine for Error Correction

Error correction for the SF labeling consists of first detecting the location of the error, and then fixing the error.

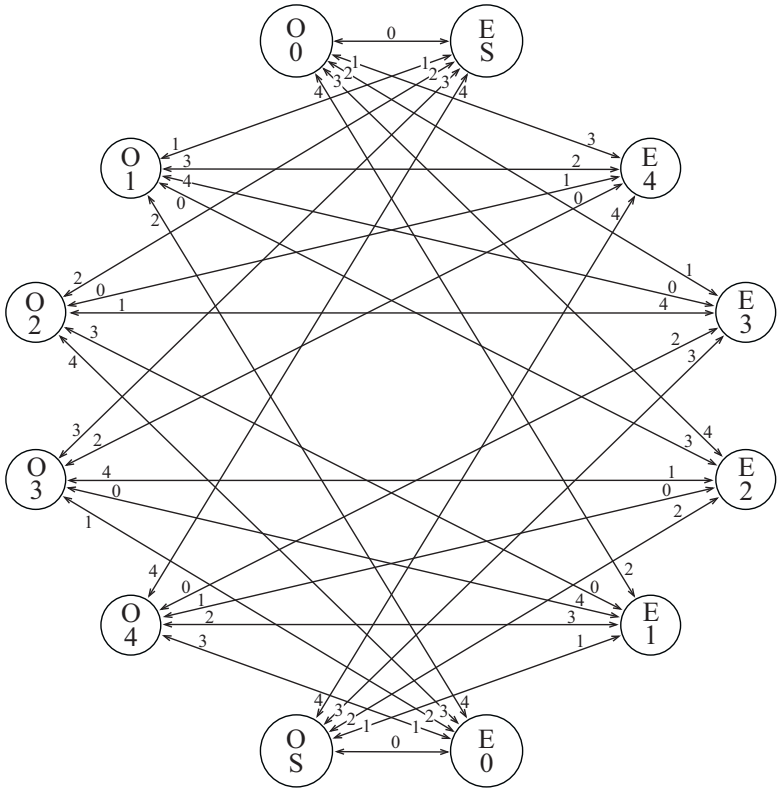

Figure 10: The codeword recognizer for K_5^n.

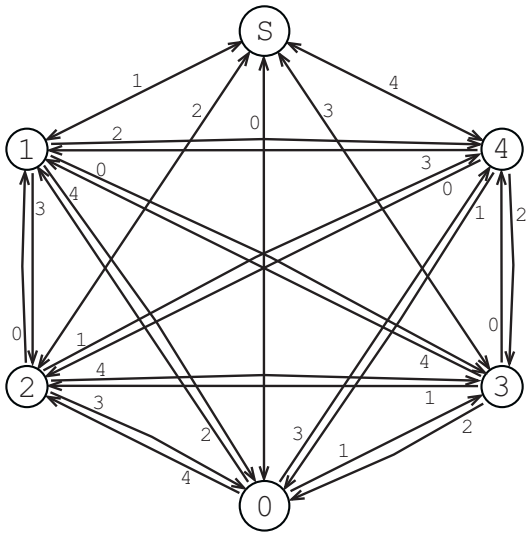

Figure 11: The x_1 correction machine for K_5^n.

The x_1 correction machine is simply the reverse codeword recognizer and it is used if x_1 is the error digit. The states are labeled as elements of $\{S,0,1,\ldots,d-1\}$. The machine starts in the S state if n is even and the 0 state if n is odd. Having multiple starting states is a slight problem as it makes the machine non-deterministic, but we will ignore this because we can always determine which state we need to start in based on whether n is even or odd. Every state is a final state except the S state. The transitions among the states can be classified by the function δ, which is the inverse of δ_2 from the machine used for codeword recognition.

$\delta(x,z) = \left(\frac{x+z}{2}\right) \bmod d$

$\delta(x,x) = S$

$\delta(S,z) = z$

Example 3.7 *Figure 11 shows the x_1 correction machine for K_5^n.*

To error correct $n > 1$, use this algorithm:

IF $x_1x_2\cdots x_n \notin G_n$

THEN IF $x_2\cdots x_n \notin U_{n-1}$

THEN correct $x_1 = q$, where q is the final state after putting $x_n\cdots x_2$ through the x_1 correction machine.

ELSE FOR $i = 2$ TO n

IF $x_i \neq x_1$

THEN IF $i \neq n$

THEN correct $x_i = 2x_1 - x_i \bmod d$

ELSE correct $x_n = 0$

BREAK

ENDFOR

To prove that the error correcting algorithm works, we first need this lemma:

Lemma 3.8 *The labels of any two vertices v_i, v_j in any subgraph K_d^1 of K_d^n, where $n \geq 1$, differ in the first character only.*

Proof. We will prove this using induction on n. The base case, $n = 1$, is trivial by construction. Now suppose that for $n = k$, any two vertices v_i, v_j in any subgraph K_d^1 of K_d^k differ in the first character only. Let $n = k+1$. By the construction of the SF labelings, we know that K_d^{k+1} is made up of rotated $\alpha\left(K_d^k\right)$ which have had an integer from 0 to $d-1$ appended to each label in $\alpha\left(K_d^k\right)$. Since the labels of all K_d^1 in K_d^k differ in only the first character, the labels of all K_d^1 in $\alpha\left(K_d^k\right)$ differ in only the first character because α is applied to every digit of every label in K_d^k. Also, the same digit is appended to every label in $\alpha\left(K_d^k\right)$, so the last $n-1$ digits of a K_d^1 subgraph will be the same and only the first digit in the labels of K_d^1 subgraph will differ. ■

Theorem 3.9 *Error correction for the SF labelings is performed correctly by the algorithm given above.*

Proof. Suppose $x_1x_2\cdots x_n$ is not a G codeword. Then there are two cases to consider: $x_1x_2\cdots x_n$ is the label of an a corner vertex of a subgraph K_d^1 of K_d^n or it is the label of an x corner vertex of a subgraph K_d^1 of K_d^n (We know it cannot be the label of a c corner vertex, because it is not a codeword).

Case 1: $x_1x_2\cdots x_n$ is the label of an a corner vertex of a subgraph K_d^1 of K_d^n, where $1 \leq i < n$. By the definition of an a vertex, we know that it is a non-codevertex which is adjacent to a codevertex, and therefore the subgraph K_d^1 of K_d^n must have a $G-Code$ on it. In other words, one of the vertices in the same subgraph K_d^1 of K_d^n as the a corner vertex is a codevertex. By the previous lemma, we know that the labels of all vertices in the same subgraph K_d^1 of K_d^n as the a corner vertex share the same digits $x_2\cdots x_n$, so changing the first digit will correct the non-codeword. Therefore, the error must be in the first digit.

To correct the error, we simply begin in the appropriate ending state of the codeword recognizer, put the string $x_2\cdots x_n$ backwards through the recognizer, and determine what the first digit should be so that we end up at the $\frac{E}{S}$ state.

Case 2: $x_1x_2\cdots x_n$ is the label of an x corner vertex of a subgraph K_d^1 of K_d^n. By the definition of an x vertex, we know that it is a non-codevertex which is not adjacent to any codevertex, and therefore the subgraph K_d^1 of K_d^n must have a $U-Code$ on it. In other words, none of the vertices in the same subgraph K_d^1 of K_d^n as the x corner vertex is a codevertex. By the previous lemma, we know that the labels of all vertices in the same subgraph K_d^1 of K_d^n as the x corner vertex share the same digits $x_2\cdots x_n$, so changing the first digit will not correct the non-codeword. Also, the word $x_2\cdots x_n$ is a U codeword because of the $G-U$ construction. Therefore, the error must be in one of the digits $x_2,\cdots,x_n$, and an easy way to check for it is to see if $x_2\cdots x_n$ is a U codeword.

Find the largest subgraph K_d^{i-1}, $2 \leq i \leq n$, of K_d^n such that the x vertex is a corner vertex of K_d^{i-1}. Then, we know by the construction of the SF labelings that the first $i-1$ characters are identical in each corner vertex of K_d^{i-1}. Since the SF labeling is a Gray code, the first $i-1$ characters of

the label of the vertex connected to $x_1x_2\cdots x_n$ in the adjacent K_d^{i-1} must be the same as the first $i-1$ characters of $x_1x_2\cdots x_n$. Also, both K_d^{i-1} subgraphs are in the same subgraph K_d^i of K_d^n and therefore cannot differ in character beyond the i^{th} digit. So the error must be in the i^{th} digit.

If $i=n$, then n must be odd since every subgraph K_d^{n-1} of K_d^n is a $G-code$ if n is even and no corner vertex of G_d^n can be an x. By the $G-U$ construction and since $n-1$ is even, the x vertex must be in the top position of a U_d^{n-1} subgraph of G_d^n and must be connected to a corner codevertex of G_d^{n-1}. Since there is only one G_d^{n-1} subgraph of G_d^n and it is in the top position, each label in G_d^{n-1} has a 0 appended to it. Thus, to correct $x_1x_2\cdots x_n$, we must change the last digit to a 0.

Otherwise, if $i\neq n$, then we know that for some subgraph K_d^i of K_d^n, the i^{th} digit of the label of the codevertex adjacent to our non-codevertex was a zero at one point (similar to the $i=n$ case above), but has since been rotated by $\frac{2\pi x_i}{d}$ radians and the characters $x_1x_2\cdots x_{i-1}$, where $x_1=x_2=\cdots=x_{i-1}$, have been permuted by α. To undo this permutation/rotation effect, we multiply the first character x_1 by 2, since 2 is the inverse of $\frac{d+1}{2}$ and subtract x_i.

Therefore, the algorithm is correct and we can use it to correct non-codewords in the SF labeling. ■

Example 3.10 *We know from the previous example that* 142 *is not a codeword in the SF labeling of* K_5^n. *Now we check if* 42 *is an element of* U_{n-1}. *It ends at the* $\overset{E}{1}$ *state, so it is not an element of* U_{n-1} *and the error must be in the first digit. Now we run* 24 *through the* x_1 *correction machine, begining at the* 0 *state since* 142 *has an odd number of digits, and we end up in the* 0 *state. We correct* 142 *to* 042 *and verify that we performed the appropriate correction by looking at Figure 9.*

Example 3.11 *To see how error correction works when a digit other than the first needs to be corrected, we will look at the label* 04431. *It ends in the* $\overset{O}{1}$ *state of the codeword recognizer, so it is obviously not a codeword. Next, we see that* 4431 *is an element of* U_{n-1} *since it ends in the* $\overset{E}{0}$ *state of the codeword recognizer. We check if the second digit is equal to the first, it is not, so the second digit is the incorrect digit. By the algorithm,* $x_2=2(0)-4\bmod 5=1$, *so we correct* 04431 *to* 01431.

4 Conclusion

We have presented a family of labelings, the SF labelings, to which the Towers of Hanoi labeling belongs. The SF labeling is constructed iteratively on iterated complete graphs K_d^n, where d is odd. It has a $2d+2$ state machine for codeword recognition and a $d+1$ state machine for error correction.

Further research could be done on the SF labeling to determine if there is an easy encoding/decoding method and what it is if it exists. Also, a better proof of the codeword recognizer could be found as the current proof is based on a correspondence between the number of G_d^n and U_d^n codewords and the number of words in the accepting states of the codeword recognizer for a given n. Other research could be done to prove the conjecture that no labeling system of all K_d^n exists that unites all the desirable properties of easy codeword recognition, error correction, encoding, decoding, and that is Gray code.

References

[1] Alspaugh, Shawn, Nathan Knight, and Kathleen Meloney. *Perfect One Error Correcting Codes on Iterated Complete Graphs*. Proceedings of the REU Program in Mathematics. NSF and Oregon State University. Corvallis, Oregon. September, 2001.

[2] Birchall, Be and Jason Tedor. *Perfect One Error Correcting Codes on Iterated Complete Graphs*. Proceedings of the REU Program in Mathematics. NSF and Oregon State University. Corvallis, Oregon. August, 1999.

[3] Bode, David. *Alternate Labelings for Graphs Representing Perfect-One-Error-Correcting Codes*. Proceedings of the REU Program in Mathematics. NSF and Oregon State University. September, 1998.

[4] Cull, Paul and Ingrid Nelson. ”Perfect Codes, NP-Completeness, and Towers of Hanoi Graphs.” *Bulletin of the Institute of Combinatorics and its Applications* 26 (1999): 13-38.

[5] Frayer, Christopher and Shalini Reddy. *Perfect One Error Correcting Codes and Complete Iterated Graphs*. Proceedings of the REU Program in Mathematics. NSF and Oregon State University. Corvallis, Oregon. August, 2002.