# Perfect One Error Correcting Codes on Iterated Complete Graphs

Shawn Alspaugh, Nathan Knight, and Kathleen Meloney

Advisor: Paul Cull

Oregon State University

September 17, 2001

**Abstract**

We define how to construct iterated complete graphs and define an iterative method for choosing codevertices on these graphs such that the coded graph is a perfect one error correcting code. These graphs are infinite in two ways: we allow there to be any number, $d$, of vertices in the complete graph and any number, $n$, of iterations of the complete graph. We also present three labeling schemes. The first labeling method provides easy error correction and finite state machines which are independent of $d$ and somewhat independent of $n$. The second method has the gray code property and finite state machines of size $d+1$ that will do codeword recognition and error correction. The third labeling method has finite state machines of size $d+1$ which will recognize codewords and an easy method for coding and decoding.

## 1 Introduction

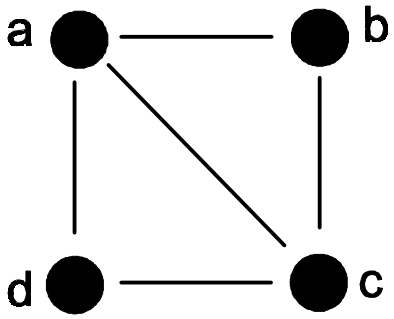
P. Cull and I. Nelson [1] classified a coded graph which defines the Towers of Hanoi puzzle movements and solutions. A legal positioning of disks on the towers is represented by a vertex. Legal moves are represented by an

edge between vertices. A solution to the Towers of Hanoi puzzle is when all the disks are on a single tower, in order from largest to smallest. Edges between vertices represent legal movements of disks. By iterating a basic graph which represents 1 disk, they can represent the puzzle for any number $n$ of disks. Cull and Nelson found a perfect-one error correcting code on the graph and labeling method so that codevertices could be differentiated from noncodevertices, error correcting noncodevertices is trivial, and coding and decoding is equally trivial. By generalizing their results, we define perfect one-error correcting codes on $n$-iterated complete graphs on $d$-vertices for any n and any d and show that labeling and error correction are similarly trivial. We begin with a few definitions.

## 1.1 Background and Definitions

**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**.*

**Example 1.2** *Graph $G$ with vertex set $V(G) = \{a, b, c, d\}$ and edge set $E(G) = \{(a, b), (b, c), (c, d), (a, d)\}$. Edges can also be denoted without the parentheses: $E(G) = \{ab, bc, ac, cd, ad\}$*




**Definition 1.3** *Two vertices $a, b \in V(G)$ are **adjacent** if $ab \in E(G)$.*

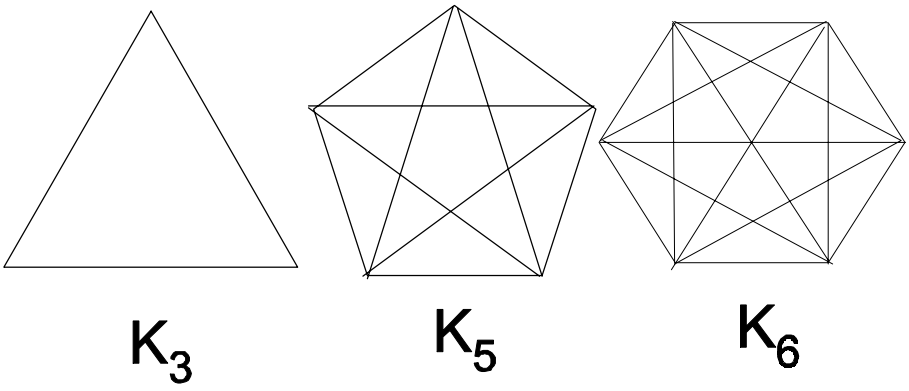
**Definition 1.4** *The **degree** of a vertex $v_0$, denoted $\deg(v_0)$, is the number $n$ of adjacent vertices $v_1, v_2, ..., v_n$. (Note that in the figure above, $\deg(a) = 3$ and $\deg(b) = 2$.)*

**Definition 1.5** *A **subgraph** $H$ of a graph $G$ consists of a subset $V(H) \subset V(G)$ with the associated edges.*

**Definition 1.6** *Two distinct subgraphs $H$ and $K$ of a graph $G$ are **adjacent** if there exists $h \in H$ and $k \in K$ st. $hk \in E(G)$.*

**Definition 1.7** *A **complete graph on** $\mathbf{d}$ **vertices** is a simple graph in which each pair of distinct vertices are adjacent. We denote the complete graph on $d$ vertices by $\mathbf{K}_d$. We call $d$ the **dimension** of the complete graph.*

**Example 1.8** *Complete graphs on $d = 3$, $d = 5$, and $d = 6$ vertices.*

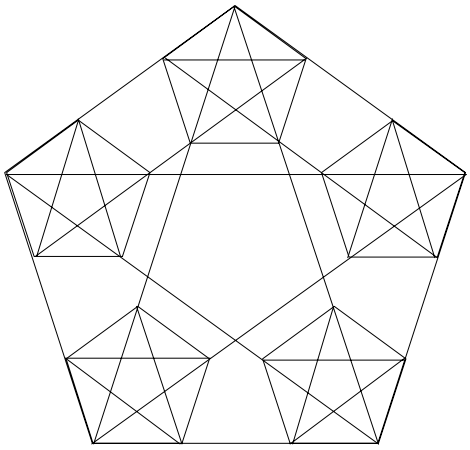



**Definition 1.9** *A **corner vertex** is a vertex of an iterated complete graph $K_d^n$ whose degree is $d - 1$. An **internal vertex** is any vertex that is not a corner vertex.*

**Remark 1.10** *There are exactly $d$ corner vertices on any $K_d^n$.*

We will define the iterative construction for a biinfinite family of graphs, $K_d^n$. The first iteration of the graph, denoted $K_d^1$ is simply the complete graph on $d - vertices$. The second iterate, named $K_d^2$, is made of d copies of $K_d^1$ so that each copy is adjacent to every other copy. The process to construct $K_d^n$ consists of making $d$ copies of $K_d^{n-1}$and forming edges between pairs of subgraphs such that the edge connects a corner vertex of one subgraph with a corner vertex of another subgraph (every subgraph is adjacent to every other subgraph). The family is biinfinite because there is an infinite number of dimensions $d$ and an infinite number of iterations $n$.

**Example 1.11** *Two iterations of a complete graph on 5 vertices, $K_5^2$. Note that there are exactly 5 vertices which have degree $d-1=4$ (these are the corner vertices) and the other vertices have degree $d$.*



$K_5^2$

# 2 Codes on Graphs

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

## 2.1 Definitions

**Definition 2.1** *A **code** on a graph $G$ is any subset of vertices $C(G) \subset V(G)$. Vertices $c \in C(G)$ are called **codevertices**. Vertices $v \in V(G) - C(G)$ are called **noncodevertices**.*

Noncodevertices are those vertices of a graph that are not chosen as codevertices. It is usually desirable to be able to associate each noncodevertex to a codevertex. For example, in one type of code, every vertex is within a distance $d$ of exactly one codevertex. A particular group of codes on graphs uniquely associates each noncodevertex to an adjacent codevertex. These are called perfect one-error correcting codes and are explicitly defined below.

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

*1. No two codevertices are adjacent.*

*2. Every noncodevertex is adjacent to exactly one codevertex.*

For any number of $n - iterations$ of a complete graph on any number of $d - vertices$ $(K_d^n)$, we have discovered that there is exactly one choice of codevertices in $K_d^n$ that produces a perfect one error correcting code. Codevertices are chosen while the graph is iteratively constructed. This construction is made of two different types of codes on iterated complete graphs, called $G - codes$ and $U - codes$, which are, in fact, **weak codes** (as defined below).

## 2.2 Weak Codes and The Existence and Uniqueness of a Perfect One Error Correcting Code on $K_d^n$

Weak codes are almost perfect one error correcting codes with the exception that we allow some of the corner vertices of the graph to not be a codevertex and not be adjacent to any vertex. After proving the existence and uniqueness of a weak code on $K_d^n$, the existence of a unique perfect one error correcting code on $K_d^n$ will follow very easily.

### 2.2.1 Weak Codes and Perfect One Error Correcting Connections

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

*1. No two codevertices are adjacent.*

*2. Every internal vertex is either a codevertex or adjacent to exactly one codevertex.*

The reader should compare the definitions of weak code and perfect one error correcting code to note that a weak code is exactly a perfect one error correcting code with the exception that in a weak code, corner vertices that are noncodevertices are allowed to not be adjacent to any codevertex.

We classify vertices as being one of three types in order to make useful conclusions about connections between vertices. The usefulness of this denotation will soon become apparent.

**Definition 2.4** *If a vertex $v \in K_d^n$ is:*

*1. A codevertex that is not adjacent to any other codevertex, denote it by* **c**

*2. A noncodevertex which is adjacent to exactly one codevertex denote it by an* **a**

*3. A noncodevertex which is NOT adjacent to any codevertices, denote it by an* **x**

Once again, we will make another definition in order to simplify our proof:

**Definition 2.5** *A* ***perfect one error correcting connection*** *(abbreviated* ***p1ecc-c****) is a connection between two corner vertices $v_1$ and $v_2$ of two different $K_d^{n-1}$ subgraphs of $K_d^n$ such that $v_1$ and $v_2$ satisfy the following properties after being connected:*

*1. No two codevertices are adjacent ($v_1$ and $v_2$ are not both codevertices).*

*2. For both $v_1$ and $v_2$, either $v_i$ is a codevertex or is adjacent to exactly one codevertex.*

**Remark 2.6** *NOTE that a* ***p1ecc-c*** *will preserve weak codes and perfect one error correcting codes.*

We will now prove a lemma about perfect one error correcting connections (p1ecc-c).

**Lemma 2.7** *There are exactly two possible p1ecc-c, they are $x-c$ and $a-a$.*

**Proof.** We have three types of corner vertices: $x, a$,and $c$, therefore there are six different connections. Clearly we cannot have $c-c$ because then we have two adjacent codevertices. We cannot have $x-x$ because then both $x$ vertices will not be adjacent to a codevertex. We cannot have $c-a$ because then the $a$ vertex will be adjacent to two codevertices. We cannot have $x-a$ because then the $x$ vertex will not be adjacent to any codevertex. So there is at most two connections: $a-a$ and $x-c$. To see that there are p1ecc-c, note that for $a-a$ clearly no two codevertices are adjacent and each vertex is adjacent to exactly one codevertex (by definition of $a$). Thus this connection is p1ecc-c. Note that each vertex of $x-c$ is adjacent to exactly one codevertex (since $x$ was not adjacent to any, but is now connected to one). It is clear that no two codevertices are adjacent, thus $x-c$ is p1ecc-c. ■

### 2.2.2 Uniqueness of Choice of Codevertices on $K_d^n$

We can now proceed with proving that the construction method for choosing codevertices on $K_d^n$ is unique. The following definitions will specifically describe the two types of weak codes that are used to construct a weak code on $K_d^n$, they are called $G-codes$ and $U-codes$.

**Definition 2.8** *A $G-code$, $G_d^n$, is a weak code on $d-vertices$ of $n-itertaions$ such that for:*

*$n$ even$\rightarrow$ all corner vertices are $c$*

*$n$ odd $\rightarrow$ all exactly one corner vertex is a $c$, and all other corner vertices are $a$*

**Definition 2.9** *A $U-code$, $U_d^n$, is a weak code on $d-vertices$ of $n-itertaions$ such that for:*

*$n$ even$\rightarrow$ exactly one corner vertex is an $x$, and all other corner vertices are $a$*

*$n$ odd $\rightarrow$ all corner vertices are $x$*

The theorem below proves that on any $K_d^n$, there are exactly two weak codes–a $G-code$ and a $U-code$. We use induction to show that if $K_d^n$ has a weak code, then the copies of $K_d^{n-1}$ that are used to construct $K_d^n$ must all have a weak code. Our inductive hypothesis allows us to limit the $K_d^{n-1\prime}$s that we use to only $G_d^{n-1}$ or $U_d^{n-1}$. Consequently, for any choice of $n$ and $d$ we can only construct a $G-code$ or a $U-code$ on $K_d^n$ and that these codes are exactly as we have described them above.

**Theorem 2.10** *There are exactly two weak codes on $K_d^n$, one $G-code$ and one $U-code$.*

**Proof.** For $n=1$, $K_d^1$ has exactly two weak codes–one in which a single vertex is chosen as a codevertex and one in which no vertex is chosen as a codevertex. Clearly, the former is a $G-code$ and the latter is a $U-code$.

Assume for induction that there are exactly two weak codes for $K_d^{n-1}$, a $G-code$, $G_d^{n-1}$, and a $U-code$, $U_d^{n-1}$.

Assume $K_d^n$ has a weak code. We claim that each of the constituent $d$ copies of $K_d^{n-1}$must also have weak codes. Since no codevertices in $K_d^n$ are

adjacent and every vertex in $K_d^n$ is a vertex in some $K_d^{n-1}$, we know that no two codevertices in $K_d^{n-1}$ are adjacent. Similarly, every internal vertex of $K_d^n$ is also an internal vertex of each $K_d^{n-1}$ and so, must be adjacent to exactly one codevertex. Corner vertices of each $K_d^{n-1}$ which are not adjacent to another copy of $K_d^{n-1}$ are corner vertices of $K_d^n$, and thus, may fail to be adjacent to a codevertex. Corner vertices of $K_d^{n-1}$ which are internal vertices of $K_d^n$ may be adjacent to a codevertex in another copy of $K_d^{n-1}$but not adjacent to a codevertex within their own $K_d^{n-1}$. So, clearly if $K_d^n$ has a weak code, then all $d$ $K_d^{n-1}$ subgraphs of $K_d^n$ must also have weak codes.

Note that although having a weak code is downward preserving, the type of weak code ($G-code$ or $U-code$) is not downward preserving. That is, it may be possible for $K_d^n$ to be a $G-code$, while its constituent $K_d^{n-1}$ are $U-codes$.

By the induction step there are only two weak codes for $K_d^{n-1}$, which are $G_d^{n-1}$ and $U_d^{n-1}$.

**Case I: $K_d^n$ has at least one corner vertex that is a $c$**

In this case, the $c$ of $K_d^n$ must be a $c$ corner vertex of some copy of $G_d^{n-1}$, since no corner vertex of any $U_d^{n-1}$ is a $c$.

If $n$ is even, then the other $d-1$ corner vertices of $G_d^{n-1}$ are $a$. Since the only p1ecc-c are $x-c$ and $a-a$, the $a$ corner vertices of $G_d^{n-1}$ must connect to other $a$ corner vertices. However, every corner vertex of $U_d^{n-1}$ is an $x$ so no copy of $U_d^{n-1}$ will appear in the construction. Thus, each $a$ in the copy of $G_d^{n-1}$ must connect to an $a$ of another $G_d^{n-1}$ subgraph. Thus we have $d$ copies of $G_d^{n-1}$. By connecting $G_d^{n-1}$ subgraphs with $a-a$, $K_d^n$ is a weak code. Since the $c$ corner vertex of each $G_d^{n-1}$ remains unconnected, each corner vertex of $K_d^n$ is a $c$ and thus $K_d^n$ is actually $G_d^n$.

If $n$ is odd, then the other $d-1$ corner vertices of $G_d^{n-1}$are $c$. By Lemma 17, these $c$ corner vertices must connect to an $x$. Since every corner vertex of $G_d^{n-1}$ is a $c$, the $x$ must come from a corner vertex of $U_d^{n-1}$. However, only one corner vertex of $U_d^{n-1}$ is an $x$, so the $d-1$ corner vertices of $G_d^{n-1}$ must form an $x-c$ connection with $d-1$ $U_d^{n-1}$ subgraphs. The other corner vertices of $U_d^{n-1}$ are $a$ and form $a-a$ p1ecc-c when connecting with other copies of $U_d^{n-1}$. Thus, we have a weak code on $K_d^n$. Since one corner vertex of $K_d^n$ is a $c$ corner vertex of $G_d^{n-1}$ and the other corner vertices are $a$ corner vertices of $U_d^{n-1}$, $K_d^n$ is actually $G_d^n$.

Therefore, in the case where at least one corner vertex of $K_d^n$ is a $c$, there is exactly one weak code that can be produced, specifically a $G-code$.

Furthermore, in this case, the corner vertices of $K_d^n$ are either all $c$, or there is exactly one corner vertex which is a $c$, just as in our definition of $G-code$.

**Case II: No corner vertex of $K_d^n$ is a $c$**

In the above case, we saw that if at least one corner is a $c$, then either exactly one corner vertex is a $c$ or all corner vertices are $c$. Thus, our only other option is that no corner vertices are $c$. Since all the corner vertices of $G_d^{n-1}$ are either $c$ or $a$ and we cannot have any corner vertex of $K_d^n$ be a $c$, we cannot construct $K_d^n$ of only $G_d^{n-1}$ because those two restrictions would necessarily force two copies of $G_d^{n-1}$ to be connected with an illegal $a-c$ edge. Therefore, $K_d^n$ must be constructed of at least one $U_d^{n-1}$.

If $n$ is even, then all of the corner vertices of $U_d^{n-1}$ are $x$. Since the only possible p1ecc-c for an $x$ vertex is $x-c$ and $G_d^{n-1}$ is our only code that has $c$ as a corner vertex, we must connect $U_d^{n-1}$ to a $G_d^{n-1}$. For $n-1$ odd, $G_d^{n-1}$ has exactly one $c$ as a corner vertex, so we must make an $x-c$ connection between an $x$ vertex of each $U_d^{n-1}$and a $c$ vertex of one of $d-1$ copies of $G_d^{n-1}$. The remaining corner vertices of each $G_d^{n-1}$ are all $a$ and so the connections between $G_d^{n-1}$ subgraphs will all be $a-a$ and thus p1ecc-c. So clearly $K_d^n$ will have a weak code. Furthermore, exactly one corner vertex of $K_d^n$ will be $x$ and the remaining $d-1$ vertices will all be $a$. Clearly $K_d^n$ is actually $U_d^n$.

If $n$ is odd, then $U_d^{n-1}$ has exactly one $x$ as a corner vertex and the remaining corner vertices are all $a$. This $x$ vertex must be a corner vertex of $K_d^n$ since otherwise it would need to be connected to a $c$, which can only belong to a $G_d^{n-1}$. If we had a copy of $G_d^{n-1}$(all of whose corner vertices are $c$), then at least one corner vertex of $K_d^n$ would need to be a $c$, which would contradict our assumption that no corner vertex of $K_d^n$ is a $c$. Thus, we cannot use any $G_d^{n-1}$ and must use only $U_d^{n-1}$. The other corner vertices of $U_d^{n-1}$ are all $a$ and all need to be connected to another $K_d^{n-1}$. Since the only possible p1ecc-c for an $a$ vertex is $a-a$, and $U_d^{n-1}$ is our only code that has $a$ as a corner vertex, we must connect the first $U_d^{n-1}$ to $d-1$ copies of other $U_d^{n-1}$. Clearly all connections between the $d$ copies must be $a-a$, so the $x$ vertex of each $U_d^{n-1}$ will remain unconnected and be a corner vertex of $K_d^n$. Thus $K_d^n$ will clearly be a weak code, namely $U_d^{n-1}$.

In this case, there is exactly one weak code that can be produced on $K_d^n$– a $U-code$ whose corner vertices are exactly as described in the definition. Thus, we have that there are exactly two weak codes on any $K_d^n$ –a $G-code$ and a $U-code$, whose corner vertices are exactly as we have described them.

■

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. The reader may have noticed that some $K_d^n$ are constructed from $d$ copies of the same $K_d^{n-1}$ subgraph and that any corner vertex of $K_d^n$ is the top vertex of some $K_d^{n-1}$. Thus, under rotations of the graph, $K_d^n$ is exactly the same–rotations are isomorphic to one another. As part of the uniqueness, we need to distinguish between these isomorphic graphs. We designate a top vertex of a graph in order to distinguish between isomorphic graphs, in which, the graph is rotationally symmetric no matter which vertex is chosen as the top vertex.

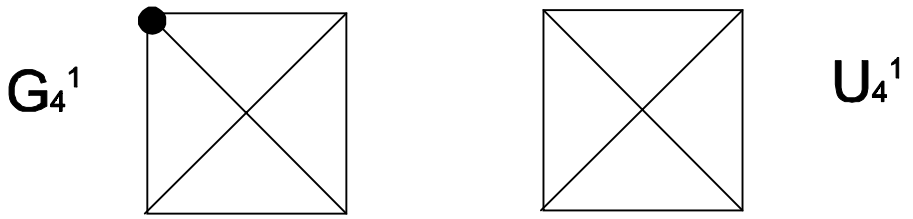
**Theorem 2.11** *There is exactly one (up to isomorphism) perfect one error correcting code on $K_d^n$.*

**Proof.** The $G-code$ is the perfect one error correcting code. When $n$ is even, this $G-code$ is invariant under rotations of the graph and is truly unique. When $n$ is odd, the $G-code$ has only one corner vertex as a codevertex. Thus the perfect one error correcting code is only unique after the single corner code vertex is rotated to the top position. ■

### 2.2.3 The G-U Construction

In this section, we explicitly describe the G-U construction method for any $K_d^n$. In the previous section, we proved that there is exactly one perfect error correcting code on an iterated complete graph $K_d^n$ and the proof of Theorem 2.10 describes its construction. For clarity, however, we will describe the construction of $G-codes$ and $U-codes$.

First, construct two complete graphs on $d-vertices$. To construct $G_d^1$, choose one vertex as a codevertex and rotate this vertex to the top position–it is now referred to as the *top vertex* (corner vertices excluding the top vertex will heretofore be called *nontop corner vertices*). To construct $U_d^1$, do not choose any vertices as codevertices and choose one vertex to be called the top vertex. The figure illustrates the construction for $G_4^1$ and $U_4^1$. The codevertex in $G_4^1$ has been labeled with a black dot.




Now we can describe the construction of $G_d^n$ and $U_d^n$ for any $n > 1$:

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 (i.e. connections between copies will be between nontop corner vertices of each copy).
3. Designate the top vertex of one $G_d^{n-1}$ to be the top vertex of $G_d^n$.

**Remark 2.12** *Note that all of the corner vertices of $G_d^n$, for $n$ even, will be top vertices of some $G_d^{n-1}$, and thus all c.*

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 the top vertex of each $U_d^{n-1}$ to a distinct nontop corner vertex of $G_d^{n-1}$. Connect nontop corner vertices of $U_d^{n-1}$ such that each copy of $U_d^{n-1}$ is adjacent to the other $d-2$ copies of $U_d^{n-1}$ and exactly one nontop corner vertex of each $U_d^{n-1}$ is unconnected. Call the top vertex of $G_d^n$ the top vertex of $G_d^{n-1}$.

**Remark 2.13** *Note that only the top vertex of $G_d^n$ for $n$ odd is a c. The other corner vertices of $G_d^n$ are all a.*

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 the top vertex of each $G_d^{n-1}$ to a distinct nontop corner vertex of $U_d^{n-1}$. Connect non top corner vertices of $G_d^{n-1}$ such that each copy of $G_d^{n-1}$ is adjacent to the other $d-2$ copies of $G_d^{n-1}$ and exactly one corner vertex of each $G_d^{n-1}$ remains unconnected. Call the top vertex of $U_d^n$ the top vertex of $U_d^{n-1}$.

**Remark 2.14** *Note that for $U_d^n$, $n$ even, the top vertex of $U_d^n$ will be an $x$, and the remaining corner vertices will all be $a$.*

To construct $U_d^n$, make $d$ copies of $U_d^{n-1}$. Make connections between copies such that:

1. The top vertex of each copy remains unconnected.
2. Each copy is adjacent (i.e. connections between copies will be between nontop corner vertices of each copy).
3. Designate the top vertex of one $U_d^{n-1}$ to be the top vertex of $U_d^n$.

**Remark 2.15** *Note that for $n$ odd, all of the corner vertices of $U_d^n$ will be top vertices of some $U_d^{n-1}$, and thus all $x$.*

**Remark 2.16** *For $n \geq 1$, none of the corner vertices of $U_d^n$ will be codevertices.*

# 3 Labeling Methods of the G-U construction on $K_d^n$

Cull and Nelson [1] discovered a labeling for the Towers of Hanoi graphs, $K_3^n$, in which recognition of codevertices, error correction, and coding and decoding are trivial. They also discovered that recognition and error correction can be done using relatively small finite state machines. There are other labeling methods of general $K_d^n$ graphs which have some of these properties. We have developed several of these labeling methods which are presented in this section.

## 3.1 The C-R-E-L Labeling

This labeling is actually attributed to B. Birchall and J. Tedor [1] who discovered the method for labeling during a summer REU at Oregon State in 1999. Their conclusions about error correcting, however, were somewhat flawed. We have reviewed and modified their work as well as provided simpler, easier to follow proofs. The C-R-E-L labeling is easy to follow, gives very specific instructions for labeling, and is constructed from two labeling methods–a $G-labeling$ and a $U-labeling$–just as the $K_d^n$ is constructed of a $G-code$ and a $U-code$. The labeling construction exactly follows the original $G-U$ construction. A single finite state machine recognizes codewords and another finite state machine sorts noncodewords so that they

may be error corrected. Although a finite state machine for error correction is not suggested, a small finite state machine exists. Furthermore, both of these machines are independent of $d$. Unfortunately, it does not seem that coding and decoding is easy with this labeling method. Further exploration would be necessary to conclude that coding and decoding is too difficult. We will begin by describing the construction and give an example of the labeling for $K_4^n$.

### 3.1.1 Labeling Construction

The C-R-E-L labeling for any $K_d^n$ graph will consist of $n - length$ strings over the alphabet $\{0, 1, ..., d-1\}$. We will specifically describe how to connect copies $C_0, C_1, ..., C_{d-1}$ of $K_d^{n-1}$ by specifying how to connect a corner vertex $x_i$ of $C_i$ to a corner vertex $x_j$ of $C_j$. Throughout the description of the labeling, we will refer to the label of $x_i$ and the vertex $x_i$ as simply $x_i$ without making a distinction between the two–which one we are talking about should be clear from context. Similarly, when talking about the labeling of a codevertex we will call the label of the codevertex the codeword. As part of the labeling construction, we will prepend a character to the left of a string at each iteration step and we will call this process **prefixing a string**.

For $G_d^1$, label the codevertex (top vertex) 0 and the remaining $d-1$ vertices with a distinct $i \in \{1, 2, ..., d-1\}$. For $U_d^1$, label the top vertex 0 and the remaining $d-1$ vertices with a distinct $i \in \{1, 2, ..., d-1\}$.

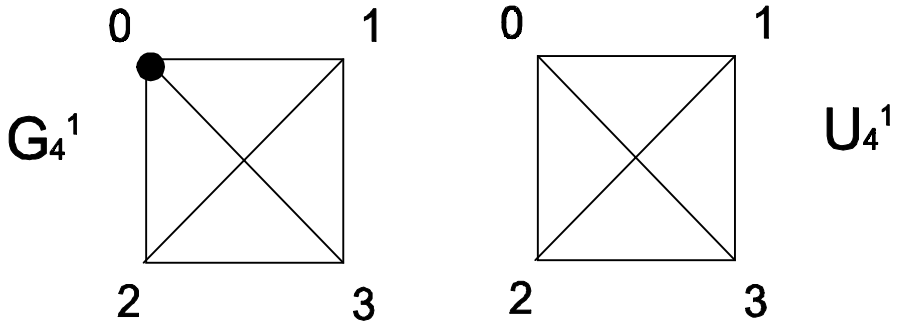



Figure 1: The labeled graphs for $G_4^1$ and $U_4^1$.

For each construction, $K_d^n$ will be constructed from copies of $K_d^{n-1}$, which will be called $C_0, C_1, ..., C_{d-1}$. For each $i \neq j$, we will form exactly one edge between $x_i$ and $x_j$ (recall that these are corner vertices of $C_i$ and $C_j$, respectively) such that one of the following connection rules is satisfied:

The Congruence Connection Rule:

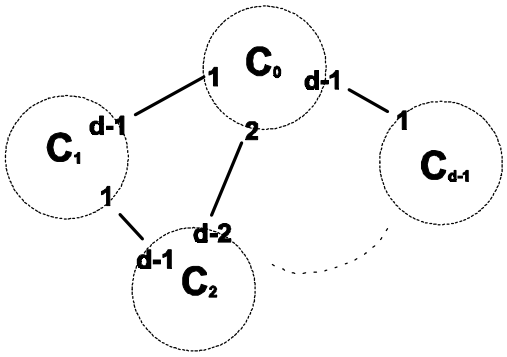
1. The leftmost character of $x_i$ is $k$ where $k \equiv j - i \bmod d$
2. The leftmost character of $x_j$ is $k$ where $k \equiv i - j \bmod d$

The Swap Connection Rule:

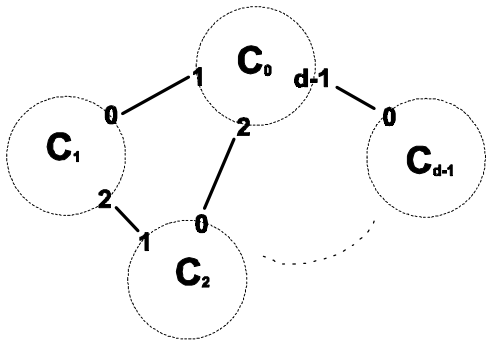
1. The leftmost character of $x_i$ is $j$
2. The leftmost character of $x_j$ is $i$

After making these connections, we will prefix each vertex $v_i \in C_i$ with the index number $i$.

The figures below give an illustration of these two connection rules.




The Congruence Connection Rule.




The Swap Connection Rule.

We will now explicitly define the labeling construction for $G_d^n$ and $U_d^n$.

To construct $G_d^n$ for $n$ even, form $d$ copies of labeled $G_d^{n-1}$ and call these copies $C_0, C_1, ..., C_{d-1}$. Then, for $i \neq j$, form exactly one edge between $x_i$ and $x_j$ (recall that these are corner vertices of $C_i$ and $C_j$, respectively) according to the Congruence Connection Rule above. To complete the labeling of $G_d^n$, for every vertex $v_i \in C_i$, prepend an $i$ to the left of $v_i$. Designate the top vertex of $C_0$ as the top vertex of $G_d^n$.

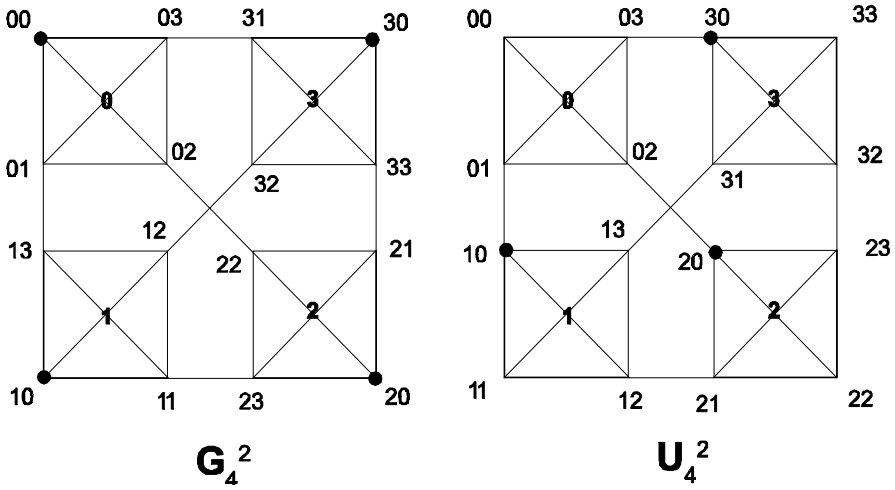
To construct $G_d^n$ for $n$ odd, form 1 copy of labeled $G_d^{n-1}$ and call this copy $C_0$ and $d-1$ copies of labeled $U_d^{n-1}$ called $C_1, C_2, ..., C_{d-1}$. Then, for $i \neq j$, form exactly one edge between $x_i$ and $x_j$ according to the Swap Connection Rule. To complete the labeling of $G_d^n$, for every vertex $v_i \in C_i$, prepend an $i$ to the left of $v_i$. Designate the top vertex of $C_0$ as the top vertex of $G_d^n$.

To construct $U_d^n$ for $n$ even, form 1 copy of labeled $U_d^{n-1}$ and call this copy $C_0$ and $d-1$ copies of labeled $G_d^{n-1}$ called $C_1, C_2, ..., C_{d-1}$. Then, for $i \neq j$, form exactly one edge between $x_i$ and $x_j$ according to the Swap Connection Rule. To complete the labeling of $U_d^n$, for every vertex $v_i \in C_i$, prepend an $i$ to the left of $v_i$. Designate the top vertex of $C_0$ as the top vertex of $U_d^n$.

To construct $U_d^n$ for $n$ odd, form $d$ copies of labeled $U_d^{n-1}$ and call these copies $C_0, C_1, ..., C_{d-1}$. Then, for $i \neq j$, form exactly one edge between $x_i$ and $x_j$ according to the Congruence Connection Rule. To complete the labeling of $U_d^n$, for every vertex $v_i \in C_i$, prepend an $i$ to the left of $v_i$. Designate the top vertex of $C_0$ as the top vertex of $U_d^n$.

**Remark 3.1** *This labeling construction is exactly the same construction method and will produce the same $K_d^n$ as the G-U construction described in the previous section.*

**Example 3.2** *Labeling for the $G_4^2$ and $U_4^2$. The figure below shows labeled graphs for $G_4^2$, and $U_4^2$ so that the reader can better grasp the labeling construction. The two figures represent both of the two connection rules. The $G_4^2$ graph is labeled with the congruence rule, while the $U_4^2$ graph is labeled with the swap connection rule. Each copy $C_0, C_1, C_2$, and $C_3$ of $K_4^1$ is labeled with its index number in the middle of the copy so that the reader can clearly see how copies are connected. These two figures give good representations of how to construct the labeling.*




Labeled graphs for $G_4^2$ and $U_4^2$.

### 3.1.2 Results of the Labeling Construction

We will now state and prove some results of the C-R-E-L labeling construction.

**Codeword Recognition** In this section, we explore the labels of a codeword contained within any $K_d^n$ and present two finite state machines which will recognize codewords contained within a complete $U_d^n$ or a complete $G_d^n$. It is important to understand what it means for a vertex to be contained within a complete graph.

**Notation 3.3** *When we say that a vertex $v$ is a $K_d^n$ string, we mean that $v$ is some labeled vertex of a complete $K_d^n$, which is the complete graph and $K_d^n$ is not a subgraph of any larger complete graph. Furthermore, during our discussion of codeword recognition, noncodeword classification, and error correction for some vertex $v$, we will never consider $v$ as an element of $H_d^m$, where $m \prec n$, i.e. where $H_d^m$ is a subgraph of $K_d^n$.*

**Notation 3.4** *It is also important to understand what how we differentiate between string types. We only say that a string $v$ ends in an odd (or even) number of zeros if there is at least one nonzero digit a that precedes the zeros*

*at the end of the string. For example, the string* $v = ...ija\underbrace{0...0}_{odd}$ *where* $a \neq 0$ *and* $i, j \in \{0, 1, ..., d-1\}$ *ends in an odd number of zeros, but the string* $w = \underbrace{0....0}_{odd}$ *does NOT end in an odd number of zeros since it is made of all zeros (even though there are an odd number of them).*

**Lemma 3.5** *1. For any vertex* $v$ *of* $G_d^n$*,* $v$ *is a codeword iff* $v$ *ends in an odd number of zeros or* $v = \underbrace{0....0}_{n}$*.*

*2. For any vertex* $v$ *of* $U_d^n$*,* $v$ *is a codeword iff* $v$ *ends in an odd number of zeros (and* $v \neq \underbrace{0...0}_{n}$*).*

**Proof.** For $n = 1$, the lemma is obviously true for $G_d^1$ and $U_d^1$. Assume the result is true for $n - 1$.

**Case I:** $G_d^n$ for $n$ even.

$G_d^n$ is composed of d copies of $G_d^{n-1}$. By the inductive hypothesis, the codewords in each $G_d^{n-1}$ are either $\underbrace{0...0}_{n}$ or end in an odd number of zeros. If a codeword in $G_d^{n-1}$ is $\underbrace{0...0}_{n-1}$, then in $G_d^n$ it will be prefixed with $i \in \{0, 1, ..., d-1\}$ and be labeled $i\underbrace{0...0}_{n-1}$. Since $n-1$ is odd, it will either end in an odd number of zeros or, in the case where $i = 0$, be $\underbrace{0...0}_{n}$. If a codeword in $G_d^{n-1}$ ends in an odd number of zeros then prefixing the string with an $i \in \{0, 1, ..., d-1\}$ will clearly not affect the fact that the string ends in an odd number of zeros. Thus the result is true.

**Case II:** $G_d^n$ for $n$ odd.

$G_d^n$ is composed of 1 copy of $G_d^{n-1}$ labeled $C_0$ and $d-1$ copies of $U_d^{n-1}$ labeled $C_1, C_2, ...C_{d-1}$. If a codeword is a member of $G_d^{n-1}$, then it will be prefixed with a zero. Thus, the result for these codewords is obvious. If a codeword is a member of one of the $C_1, C_2, ..., C_{d-1}$ copies of $U_d^{n-1}$, by the induction hypothesis for statement 2 of the theorem, it ends in an odd number of zeros. Clearly prefixing one of these codewords with a number will not change that fact. Thus, the desired result is true.

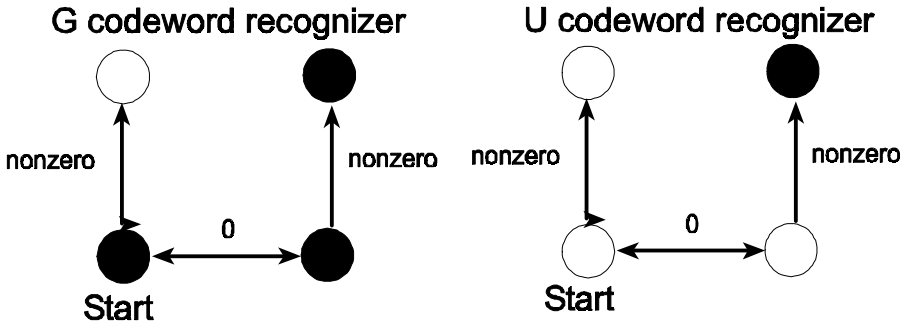
**Case III:** $U_d^n$ for $n$ even.

$U_d^n$ is composed of 1 copy of $U_d^{n-1}$ labeled $C_0$ and $d-1$ copies of $G_d^{n-1}$ labeled $C_1, C_2, ...C_{d-1}$. If a codeword is a member of $U_d^{n-1}$, then it ends in an odd number of zeros and so prefixing a zero will not affect that fact. If a codeword is a member of one of the $C_1, C_2, ..., C_{d-1}$ copies of $G_d^{n-1}$, by the induction hypothesis for statement 1 of the theorem, it either ends in an odd number of zeros or is of the form $\underbrace{0...0}_{n-1}$ where $n-1$ is odd. If it ends in an odd number of zeros, then prefixing the string with a number will not affect that. If it is of the form $\underbrace{0...0}_{n-1}$, then when the string is prefixed with one of $1, 2, ..., d-1$, it will clearly end in an odd number of zeros. Thus, the desired result is true.

**Case IV:** $U_d^n$ for $n$ odd.

$U_d^n$ is composed of $d$ copies of $U_d^{n-1}$. By the inductive hypothesis, the codewords in each $U_d^{n-1}$ end in an odd number of zeros. Clearly prefixing a string with any number will not change the fact that it ends in an odd number of zeros. Thus the result is true. ■

It is clear to see that the finite state machine in the figure below is a codeword recognizer for strings in $G_d^n$ and $U_d^n$, respectively. The recognizer reads strings from right to left and begins in the position labeled "Start". In both recognizers, the black circles designate codeword states.




Codeword Recognizers for strings in $G_d^n$ and $U_d^n$, respectively.

### Labels of Corner Vertices of $K_d^n$

**Lemma 3.6** *For vertices contained within the same $K_d^1$ subgraph of $K_d^n$, the label for each will be the same except for the last character.*

**Proof.** Clear from the construction. ■

**Lemma 3.7** *The top vertex of $G_d^n$ and $U_d^n$ for $n \geq 1$ is labeled $\underbrace{0...0}_{n}$.*

**Proof.** The result for $n = 1$ is trivial. Assume the result is true for $n-1$. By the construction, the top vertex of each $K_d^n$ belongs to the top vertex of $K_d^{n-1}$, which is the $C_0$ copy. Thus since the top vertex of each $K_d^{n-1}$ is $\underbrace{0...0}_{n-1}$ by the inductive hypothesis, the top vertex of $K_d^n$ will clearly be $\underbrace{0...0}_{n}$. ■

**Lemma 3.8** *For $G_d^n$ with $n$ odd and $U_d^m$ with $m$ even, the top vertex of each $C_1, ..., C_{d-1}$ will be connected to a nontop vertex of $C_0$.*

**Proof.** Note that $G_d^n$ and $U_d^m$ have the same construction rules. Since the top vertex of each copy $C_1, ..., C_{d-1}$ is $\underbrace{0...0}_{n-1}$ by the previous lemma, this vertex will clearly be connected to the $C_0$ copy by the construction rules. Furthermore, since the top vertex of $C_0$ will remain unconnected, the top vertices of each $C_1, ..., C_{d-1}$ will be connected to a nontop vertex of $C_0$. ■

**Lemma 3.9** *For $G_d^n$ with $n$ even and $U_d^m$ with $m$ odd, the top vertex of each $C_0, C_1, ..., C_{d-1}$ will become a corner vertex of $K_d^n$.*

**Proof.** Note that both of these graphs are constructed using the congruence connection rule. Thus, when we connect copies to one another, the top vertex of each $K_d^{n-1}$ (which is labeled $\underbrace{0...0}_{n-1}$ by Lemma 3.7) will remain unconnected. Thus, the top vertex of each $K_d^{n-1}$ will become a corner vertex of $K_d^n$. ■

**Corollary 3.10** *The corner vertices of $G_d^n$ for $n$ even and $U_d^m$ for $m$ odd are all labeled $i\underbrace{0...0}_{n-1}$, $i \in \{0, 1, ..., d-1\}$.*

**Proof.** Every corner vertex of $K_d^n$ is a top vertex of some $K_d^{n-1}$ by Lemma 3.9. After prepending the top vertex $\underbrace{0...0}_{n-1}$ of each $K_d^{n-1}$, we see that the corner vertices of $K_d^n$ will be labeled $i\underbrace{0...0}_{n-1}$, $i \in \{0, 1, ..., d-1\}$. ■

**Noncodeword Recognition, Classification, and Error Correction** We now characterize the labels of non-codewords and give a method for error correcting these vertices to the appropriate codeword. We classify non-codewords into one of three types: R, E, or L. These classifications are based on how these noncodewords will be error corrected and what these vertices get connected to in $K_d^2$. Thus, these types will not change when they go from $K_d^n$ to $K_d^{n+1}$. Note that non-codewords must first be put through the codeword recognizer and be classified as a noncodeword before classification into one of these three types is possible.

For non-codewords, it is of type:

1. **R** if it ends in an even number of zeros. (Of the non-codewords, however, it is the only type that ends in a zero.) $...a\underbrace{0...0}_{even}$

2. **E** if it ends in a nonzero preceded by an even number of zeros. $...b\underbrace{0...0}_{even}a$

   or $\underbrace{0...0}_{n-1}a$

3. **L** if it ends in a nonzero preceded by an odd number of zeros. $...b\underbrace{0...0}_{odd}a$

   or $\underbrace{0...0}_{n-1}a$

Note that $a, b \neq 0$.

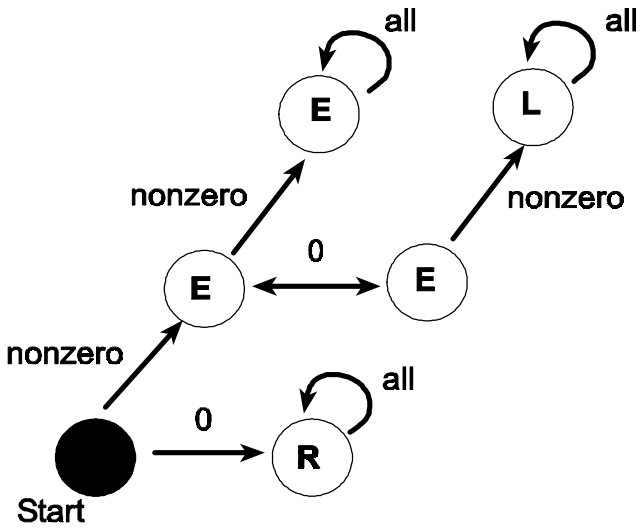
**EXCEPTIONS TO THE ABOVE RULES:**

1. For $n$ **even,** $G_d^n$ strings $\underbrace{\mathbf{0...0a}}_{n-1}$ should be **E** strings, not L.

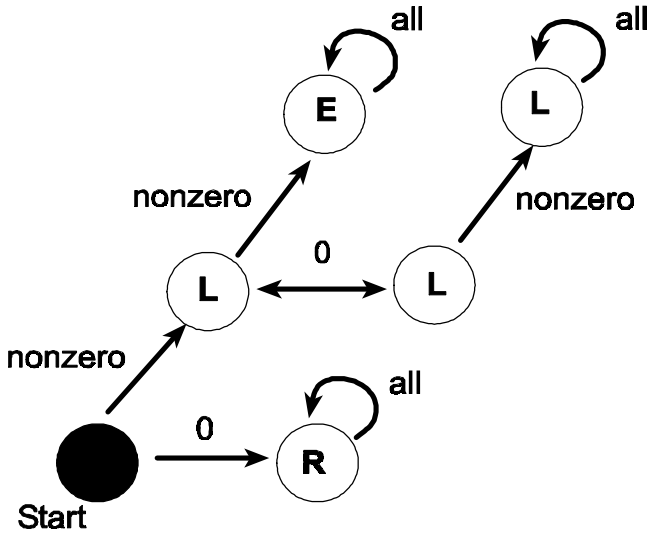
2. For $n$ **odd**, $U_d^n$ strings $\underbrace{\mathbf{0...0a}}_{n-1}$ should be **L** strings, not E.

**Recall:** *When we say that a string is a $G_d^n$ string or a $U_d^n$ string, we mean that string is a current member of the complete graph $G_d^n$ or $U_d^n$ and NOT a member of the subgraph $G_d^n$ or $U_d^n$, of a larger complete graph $K_d^m$, $m > n$.*

It is easy to verify that the finite state machines in the figures will sort noncodeword strings into the correct type. Strings must first be passed through the codeword recognizer and determined to be a noncodeword before being put through any of these sorter machines. All three string sorters will read noncodeword strings from right to left and stop in the correct state for that string type. It is easy to verify that these string sorters will sort strings into the correct types, based on the rules above. The first figure will sort noncodewords contained in $G_d^n$. The second figure will sort noncodewords in $U_d^n$.




The finite state machine for noncodewords in $G_d^n$.




The finite state machine for noncodewords in $U_d^n$.

**Lemma 3.11** *Type E vertices originate as noncodewords in $G_d^1$. Type L vertices originate as a nontop corner vertex of $U_d^1$.*

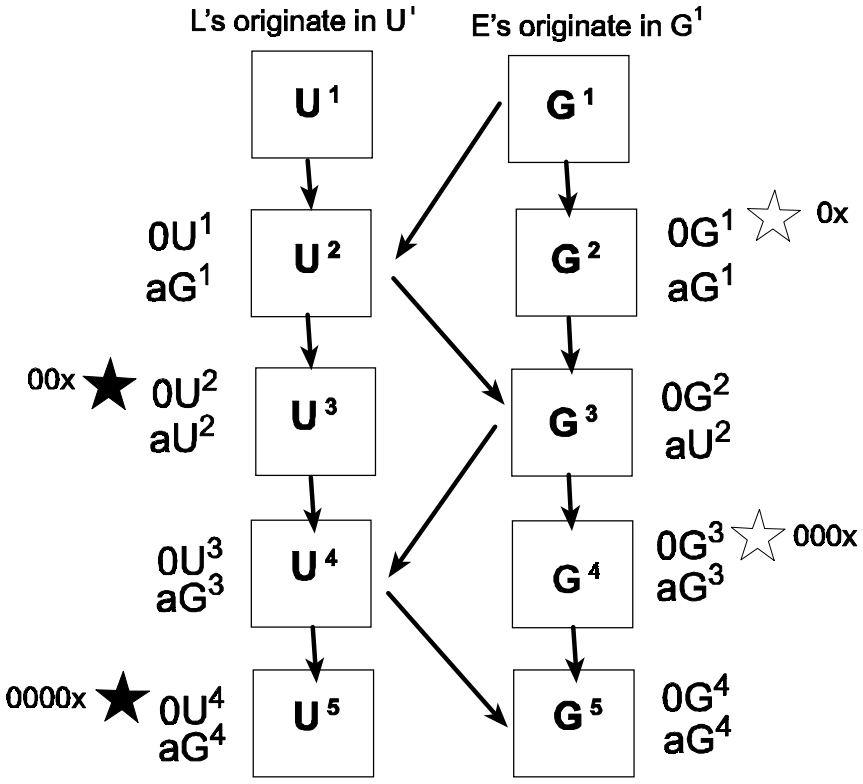
**Proof.** The strings both type E and type L end in a nonzero. Thus, since strings are built from right to left, it must be that this nonzero came from a nonzero vertex in $K_d^1$. Nonzero vertices of $G_d^1$ and $U_d^1$ are those vertices which are not codewords and not the top vertex.

The vertices in question which originate in $G_d^1$ have a path of fates as outlined in the figure labeled "Type E and L Fate Paths". It is clear to see that these vertices will either be $...b\underbrace{0...0}_{even}a$ or $\underbrace{0...0}_{n-1}a$ where $n$ is even, both of which are type E.

The vertices which originate in $U_d^1$ have a path of fates as outlined the figure labeled "Type E and L Fate Paths". It is clear to see that these vertices will either be $...b\underbrace{0...0}_{odd}a$ or $\underbrace{0...0}_{n-1}a$ where $n$ is odd, both of which are type L. ■

**Remark 3.12** *In the next figure, the large boxes indicate the complete graph at that stage. Since the labeling process is independent of d, d has been unspecified in this figure. The box labeled $G^2$ is meant to represent $G_d^2$ and*

*so forth. The arrows between boxes indicate that the subgraph as the base of the arrow is used in the construction of the complete graph at the head of the arrow. The labels to the left and right of each box ($0U^1$ and $aG^1$, for the $U^2$ box, for example), give the subgraphs that the particular $K_d^n$ is constructed from. These subgraphs are prefixed with their copy index number (0 for the $C_0$ copy, a for the $C_a$ copy where a is some nonzero number) to indicate the number that will be prefixed onto all vertices contained within that subgraph. Stars are labeled with some string $0...0x$ where $x$ is the rightmost nonzero character. These strings are picked out because they will be vertices which fall into one of the exception rules and will help to explain the reason for the exception to the rule. Black stars indicate vertices contained within their $K_d^n$ subgraph that should be classified as E strings, but will be classified as L strings based on the simple labeling rules (without the exceptions). White stars indicate those strings which are truly L strings, but will be classified as E strings based on the simple labeling rules (without the exceptions).*



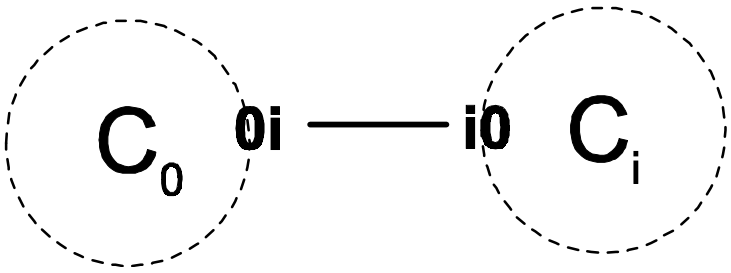

Type E and L Fate Paths

**Corollary 3.13** *Error Correction for type E vertices involves changing the rightmost character to a zero.*

**Proof.** Since type E vertices originate as a member of $G_d^1$, we know that they belong to a subgraph of $G_d^1$ within $K_d^n$. Clearly we want to correct a type E vertex to the codeword contained within the same $G_d^1$ subgraph. By Lemma 3.6, we know that vertices contained within the same $G_d^1$ subgraph will have the same string except for the last character. Since the rightmost character of the codeword in $G_d^1$ is zero, clearly we need only change the last character of the string to a zero in order to error correct a type E noncodeword. ■

**Lemma 3.14** *Error correction for type L vertices is: Swap the rightmost character with the character to its left.*

**Proof.** Since L's originate as the nonzero (nontop) vertices of a $U_d^1$ subgraph, it seems natural to see what these vertices will be connected to in $G_d^2$ and $U_d^2$. Since $G_d^2$ is not constructed from any copies of $U_d^1$, $G_d^2$ will not contain any type L vertices. However, $U_d^2$ is constructed of one copy, $C_0$, of $U_d^1$ and $d-1$ copies, $C_1, ..., C_{d-1}$, of $G_d^1$. When connecting $C_1, ..., C_{d-1}$ to $C_0$, the 0 vertex (top vertex and codeword) of each $C_i$, $i \in \{1, 2, ..., d-1\}$, is connected to the $i$ vertex of $C_0$. Thus the 0 vertex of $C_0$ will be the only unconnected vertex of $U_d^1$. This means that all the type L vertices of $U_d^1$ will be connected to a 0 vertex of some $C_i$, $i \in \{1, 2, ..., d-1\}$ where the 0 vertex is, in fact, a codeword. So we want to error correct the $i^{th}$ vertex of $C_0$ to the 0 vertex of $C_i$. After prepending the appropriate index numbers to the left of each string, we see that we want to error correct $0i$ of $C_0$ to $i0$ of $C_i$. (Please refer to the figure below.) Thus, a simple swap of the rightmost character with the character to its left will suffice to error correct the type L vertex $0i$ to the codeword $i0$. Since this step $[K_d^2]$ is the step in the construction of $K_d^n$ that determines the two rightmost characters of L, this error correction will remain constant throughout $n \geq 2$ for $K_d^n$. ■



Type L vertex in $C_0$ connected to codevertex in $C_i$.

The remainder of this section deals with type R vertices, their recognition and error correction. Since type R vertices will consistently be the top vertex of some $U_d^n$ graph, we do not know at which $n$ they will be connected to a codeword and be able to be error corrected. For this reason, error correcting type R vertices is more complex than error correcting type E or L vertices. In addition, type R vertices will be error corrected in one of two ways, based on what their strings look like. However, error correction still involves a simple swapping of neighboring characters, just as type E and L error correction.

**Lemma 3.15** *Every type R vertex is the top vertex of a $U_d^1$ subgraph and originate as 0 in $U_d^1$.*

**Proof.** Type R vertices are noncodewords whose string ends in a zero. Since its last character is a zero it must be the zero vertex of some $K_d^1$. The zero vertex of $G_d^1$ is a codeword, so it cannot be that vertex. Thus, it must be exactly the top (zero) vertex of $U_d^1$. ■

**Lemma 3.16** *The top vertex of each $U_d^n$ is type R.*

**Proof.** For $n = 1$, the top vertex of $U_d^1$ is a noncodeword that ends in 0, thus it is type R. Assume the result is true for $n - 1$.

**Case I:** $n$ even

For $n$ even, $U_d^n$ is constructed of 1 copy of $U_d^{n-1}$ and $d-1$ copies of $G_d^{n-1}$. Since the top vertex of $U_d^n$ is the top vertex of $U_d^{n-1}$ by Lemma 3.8, which is type R by the inductive hypothesis, we know that the top vertex of $U_d^n$ is type R.

**Case II:** $n$ odd

For $n$ odd, $U_d^n$ is constructed of d copies of $U_d^{n-1}$ oriented so that the top vertex of $U_d^n$ is the top vertex of one copy of $U_d^{n-1}$, which is type R by hypothesis. In addition, the corner vertices of $U_d^n$ will also be type R since they are top vertices of some $U_d^{n-1}$. Thus, it is clear that the top vertex of $U_d^n$ is also type R. ■

**Corollary 3.17** *For $U_d^n$, with $n$ odd, the corner vertices of $U_d^n$ are type R.*

**Proof.** Follows from the Case II of the proof of the above lemma. ■

Finally, we will describe error correction for type R vertices and prove that this error correction is, indeed, correct. Type R vertices are error corrected in one of two ways, depending on what the string looks like.

**Error Correction for Type R vertices**

1. If R is of the form $...b\underbrace{0...0}_{m}a\underbrace{0...0}_{even}$, $a, b \neq 0$, $m = 0$ or $m$ is even, then swap the $a$ with the zero on its RIGHT.
2. If R is of the form $...b\underbrace{0...0}_{odd}a\underbrace{0...0}_{even}$, $a, b \neq 0$, then swap the $a$ with the zero on its LEFT.
3. If R is of the form $\underbrace{0...0}_{m}a\underbrace{0..0}_{even}$ where $m \in \mathbb{N} \cup \{0\}$, then if:
   i) R is in $U_d^n$, swap the $a$ with the zero to its LEFT.
   ii) R is in $G_d^n$, swap the $a$ with the zero to its RIGHT.

**Lemma 3.18** *1. In $U_d^n$ for $n$ even, error correcting type R vertices is possible for all R except $\underbrace{0...0}_{n}$ which cannot be corrected.*

*2. In $U_d^n$ for $n$ odd, error correcting type R vertices is possible and correct for all R except those of the form $i\underbrace{0...0}_{n-1}$, $i \in \{0, 1, ..., d-1\}$ which cannot be corrected.*

*3. In $G_d^n$ $n \geq 1$, all type R vertices can be corrected.*

**Proof.** For $n = 1$, the theorem is trivially true since the type R vertex in $U_d^1$, namely 0, cannot be corrected and there are no type R vertices in $G_d^1$. Assume the result is true for $n - 1$.

**Case I:** $G_d^n$, $n$ even

$G_d^n$ is constructed of $d$ copies of $G_d^{n-1}$. By the inductive hypothesis, all the type R vertices in each subgraph can be error corrected as instructed, so the result follows.

**Case II:** $G_d^n$, $n$ odd

$G_d^n$ is constructed of one copy ($C_0$) of $G_d^{n-1}$ and $d-1$ copies ($C_1, ..., C_{d-1}$) of $U_d^{n-1}$. By the inductive hypothesis, all the type R vertices of the $G_d^{n-1}$ subgraph can be error corrected and all the R vertices of each $U_d^{n-1}$ can be error corrected except those of the form $\underbrace{0...0}_{n-1}$. Thus, we must verify that within $G_d^n$, these vertices can be error corrected as instructed. The rules require us to connect $R = \underbrace{0...0}_{n-1}$ of $C_i$, for $i \in \{1, ..., d-1\}$ to $i\underbrace{0...0}_{n-2}$ of $C_0$. Recall that $i\underbrace{0...0}_{n-2}$ of $C_0$ ($G_d^{n-1}$) is a codeword by the G-U construction and Corollary 3.10, so connecting these will allow us to error correct R of $C_i$. After prefixing our two vertices with the appropriate indices, we see that $R = \underbrace{0...0}_{n-1}$ of $C_i$ becomes $R = i\underbrace{0...0}_{n-1}$ and $i\underbrace{0...0}_{n-2}$ of $C_0$ becomes $0i\underbrace{0...0}_{n-2}$. Thus, to error correct $R = i\underbrace{0...0}_{n-1}$ to our codeword $0i\underbrace{0...0}_{n-2}$, it is clear that we want to swap the rightmost nonzero of R with the zero to its right, thus satisfying error correcting rule #3.

**Case III:** $U_d^n$, $n$ even

We construct $U_d^n$ from 1 copy, $C_0$, of $U_d^{n-1}$ and $d-1$ copies, $C_1, C_2, ..., C_{d-1}$ of $G_d^{n-1}$. By the inductive hypothesis, all the type R vertices of each $G_d^{n-1}$ can be error corrected as instructed and all the type R vertices of $U_d^{n-1}$ can be error corrected as instructed except those of the form $i\underbrace{0...0}_{n-2}$, $i \in \{0, 1, ..., d-1\}$ which cannot be error corrected. Since $\underbrace{0...0}_{n-1}$ of $U_d^{n-1}$ will become the top vertex of $U_d^n$, be labeled $\underbrace{0...0}_{n}$, and be unable to be error corrected, we need only verify that those of the form $j\underbrace{0...0}_{n-2}$, $j \in \{1, ..., d-1\}$ can be error corrected as instructed. These vertices are all corner vertices of $U_d^{n-1}$ (by Corollary 3.10) and will be connected to a top vertex of some $G_d^{n-1}$ subgraph, labeled $\underbrace{0...0}_{n-1}$. The top vertex of each $G_d^{n-1}$ is a codeword. We will connect $R = j\underbrace{0...0}_{n-2}$ of $C_0$ to $\underbrace{0...0}_{n-1}$ of $C_j$, $j \in \{1, ..., d-1\}$. After prepending the index number, we see that we want to error correct $R = 0j\underbrace{0...0}_{n-2}$ to the codeword

$j\underbrace{0...0}_{n-1}$. Clearly this can be done by swapping the rightmost nonzero character of $R$ with the zero to its left, thus satisfying error correcting rule #3.

**Case IV:** $U_d^n$, $n$ odd

$U_d^n$ is constructed of $d$ copies of $U_d^{n-1}$ all of whose $R$ vertices can be error corrected except those labeled $\underbrace{0...0}_{n-1}$. But by Corollary 3.10, these vertices will remain unconnected and become the corner vertices of $U_d^n$. After prepending indices, we see that these vertices of $U_d^n$ are labeled $i\underbrace{0...0}_{n-1}$ for $i \in \{0, 1, ..., d-1\}$. Thus, the result follows. ■

**Lemma 3.19** *Error correction of type R vertices is correct for all $K_d^n$.*

**Proof.** In the previous lemma, we verified that error correction works for vertices that are newly connected in $K_d^n$ (i.e. they were corner vertices of some $K_d^{n-1}$). What happens to these vertices in $K_d^{n+1}$? We need to verify that those type $R$ vertices that need to be left corrected in $K_d^n$ are still left corrected in $K_d^{n+1}$ (and the same for right connected types).

It is clear that once a type $R$ vertex is of the form 1 or 2, it won't change type with successive iterations. We need only verify that type 3 vertices remain either right correcting or left correcting. Since newly connected vertices are always type 3, this will complete the proof that error correction is correct.

**Case I:** $R$ is type 3 contained in $G_d^n$ (i.e. it needs to be right corrected)

Since $G_d^n$ is included in $G_d^{n+1}$ for $n+1$ even or odd and $U_d^{n+1}$ for $n+1$ even, we need to need to verify that $R$ will be right corrected in these cases.

**Subcase i:** $G_d^n$ is included in $G_d^{n+1}$ for $n+1$ even

Since $n+1$ is even, $n$ is odd and so $R$ will have the form $\underbrace{0...0}_{even}a\underbrace{0...0}_{even}$ since there will be an odd number of characters in the string. According to the labeling rules, any vertex of a copy of $G_d^n$ will be prefixed with $i \in \{0, 1, ..., d-1\}$. If $R$ is prefixed with 0, then it will be of the form $0...0a\underbrace{0...0}_{even}$ within $G_d^{n+1}$ and still be right corrected by rule 3. If it is prefixed with a nonzero, then it will be of the form $...b\underbrace{0...0}_{even}a\underbrace{0...0}_{even}$ and will be right corrected by rule 1. Thus, the result is true.

**Subcase ii:** $G_d^n$ is included in $G_d^{n+1}$ for $n+1$ odd

If $n+1$, is odd, then $R$ is of the form $\underbrace{0...0}_{odd}a\underbrace{0...0}_{even}$ in $G_d^n$ . By the labeling rules, $R$ will be prefixed with a 0 and so will be of the form $0...0a\underbrace{0...0}_{even}$ in $G_d^{n+1}$ and still be right corrected by rule 3.

**Subcase iii:** $G_d^n$ is included in $U_d^{n+1}$ for $n+1$ even

In this case, $R$ will be of the form $\underbrace{0...0}_{even}a\underbrace{0...0}_{even}$ in $G_d^n$. In $U_d^{n+1}$, $R$ will be prefixed with a nonzero and thus be of the form $..b\underbrace{0...0}_{even}a\underbrace{0...0}_{even}$ in $U_d^{n+1}$ and forevermore be right corrected by rule 1.

**Case II:** $R$ is type 3 contained in $U_d^n$ (i.e. it needs to be left corrected)

The proof is very similar to that for Case I and is omitted. ■

Thus, we have shown that the C-R-E-L labeling method allows for easy codeword recognition with small finite state machines that are independent of $d$. We have also shown that error correction for noncodewords is easy and the finite state machines required to do so are not only small, but are independent of $d$ as well. This labeling method is the only method presented in this paper that has finite state machines for codeword recognition and error correction that are independent of $d$. Unfortunately, coding and decoding does not seem to be a nice result of this labeling method. Other labeling methods presented in this paper have easy coding and decoding and finite state machines that recognize codewords and error correct noncodewords that are dependent on $d$.

## 3.2 Gray Code Labeling

### 3.2.1 The Labeling Method

**Definition 3.20** *A **gray code** is a labeling with the property that there is only one character change between the labels of two adjacent vertices.*

For our labeling purposes, we will project each complete graph $K_d^n$ onto two-dimensional space. This will cause the graph of $K_d^1$ to appear as an

$d$-gon. (note: a 2-gon is simply a line segment, and a 1-gon does not apply to this setting.)

In our analysis of a gray code labeling of $K_d^n$, it is necessary to discuss the ordering of vertices. The reader should note that ordering is different from labeling, and should be treated differently. Ordering is explained below.

The top vertex of $K_d^1$, as described during the construction of a p1ecc, is ordered zero, and each following corner vertex (moving counter-clockwise along the $d$-gon) with successive integers through $d-1$. Given the $K_d^{n-1}$ construction, $K_d^n$ is made by connecting $d$ copies of $K_d^{n-1}$ subgraphs (ordered zero through $d-1$) such that the $j^{th}$ corner vertex of the $k^{th}$ copy of $K_d^{n-1}$ connects to the $k^{th}$ corner vertex of the $j^{th}$ copy of $K_d^{n-1}$.

Note: if j=k, the vertex is a corner vertex and does not connect to another vertex

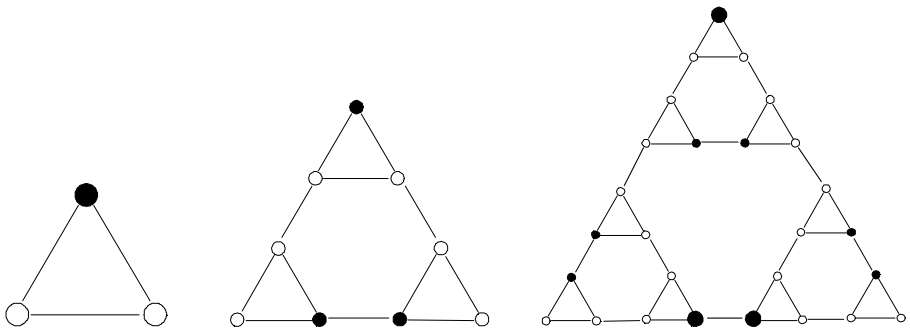
Note: if a label is in position zero, it is in the vertex ordered zero

The $\alpha$-method for labeling graphs is as follows:

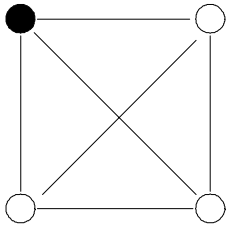
The vertices of complete graph $K_d^1$ are labeled identically as they are ordered.

Given the $K_d^{n-1}$ $\alpha$-method labeling, $K_d^n$ is labeled by rotating the $k^{th}$ subgraph of a labeled $K_d^{n-1}$ $\frac{360*k}{d}$ degrees, or $\frac{2*\pi*k}{d}$ radians clockwise. The character "$k$" is then prepended, or prefixed, to the front of the labeling of $K_d^{n-1}$ for each vertex in that subgraph.

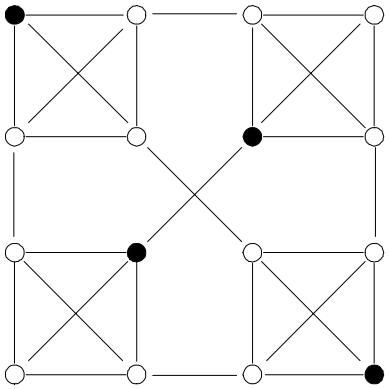
The following pictures give a visual description of the rotation involved. The solid ball will stay fixed within each subgraph. Examples for $d=3$ and $d=4$ are given. For $K_3^3$, the larger ball helps show the rotation involved.



$K_3^1$ $K_3^2$ $K_3^3$



$K_4^1$



$K_4^2$

**Theorem 3.21** *The $\alpha$-method labeling of $K_d^n$ is a gray code.*

**Proof.** The $\alpha$-method labeling of $K_d^1$ is obviously gray code.

Assume the $\alpha$-method labeling of the $K_d^{n-1}$ subgraphs are gray code. Since each graph is gray code, we must make sure the connection between subgraphs involve the change of only one character.

In the $j^{th}$ subgraph, the labeling in the $k^{th}$ position is incremented $j$ times (due to the rotation of the subgraph), becoming the $k + j \bmod d$ character. Similarly, in the $i^{th}$ subgraph, the labeling in the $j^{th}$ position is incremented $k$ times (due to the rotation of the subgraph), becoming the $j + k \bmod d$ character. Since these two vertices connect and are identical (excluding the initial, prepended character), the $\alpha$-method holds the gray code property for $K_d^n$. ■

### 3.2.2 The $\alpha$-method Labeling Codeword Recognizer

The nature of the gray code labeling allows us to describe the effect each character has on the position of a vertex. For example, by observing the $i^{th}$ character (from the left) of a string, we can determine its position in both the $K_d^{n-i+1}$ and the $K_d^{n-i}$ subgraphs of $K_d^n$ (where $i$ can be zero). In $K_d^{n-i+1}$, an $x$ in the $i^{th}$ position increments the position of the vertex $x$ positions ($\bmod d$). In $K_d^{n-i}$, an $x$ in the $i^{th}$ position decrements the position of the vertex $x$

positions (mod $d$, which is the same as incrementing $-x$ positions (mod $d$)). This idea will help us in the construction and proof of the recognizer for this labeling.

**The Description** We will now describe the recognizer for the $\alpha$-method labeling of $K_d^n$. The recognizer will contain $d+1$ states, called state 0, 1, . . ., $d-1$, and $E$. Strings will be read from left to right (front to back). If a string has an even number of characters, the recognizer will start in state $E$; if a string has an odd number of characters, the recognizer will start in the zero state. It may be easiest to visualize the recognizer as a $d+1$-gon, where the vertices of the polygon are the states, and the bottom two vertices are the zero state and state $E$ (see figure below for a partial example). The transitions among the states can be classified by the two variable function $\delta$:

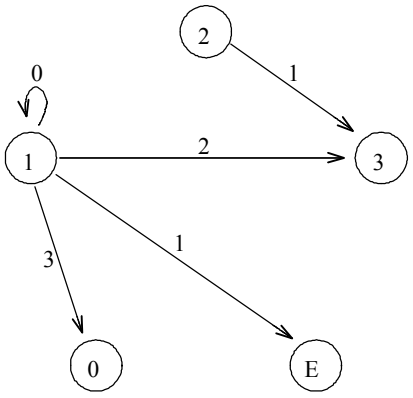
$\delta(x,i) = x + i \bmod d$, where $x \in \{0,1,...,d-1\}$ states and
$i \in \{0,1,...,x-1,x+1,...d-1\}$ input symbols;

$\delta(x,x) = E$, where $x \in \{0,1,...,d-1\}$ states and input symbols;
$\delta(E,i) = 2i \bmod d$, where $i \in \{0,1,...d-1\}$ input symbols.

note: input symbols are the characters that move one state to another state

For example, for $d = 4$,
$\delta(1,0) = 1$
$\delta(1,1) = E$
$\delta(1,2) = 3$
$\delta(1,3) = 0$
$\delta(2,1) = 3$

Part of the $d = 4$ recognizer is shown below, using the previous $\delta$-function values.




**The Proof** We will now prove that the recognizer described by the function $\delta$ and the other requirements listed above does act as the recognizer for all $K_d^n$ labeled using the $\alpha$-method labeling. We start by giving the specific case of the an $x$ character moving the $x^{th}$ state. At this point, we do not know where the codeword state is, but we will later show that this is the $E$ state. We will not introduce the $E$ state until Lemma 5.

**Lemma 3.22** *An $x$ character ($0 \leq x \leq d-1$) sends the $x^{th}$ state to a codeword state. In other words, $\delta(x, x) =$ a codeword state.*

**Proof.** A vertex in state $x$ is at the $d - x \equiv -x \bmod d$ position in some respective subgraph.

Moving $i = x$ positions along that subgraph will put the vertex in position zero of that subgraph, which may be a codeword.

$\therefore$ The an $x$ sends the $x^{th}$ state to a codeword state. ■

**Remark 3.23** *We will later see that some words in position zero of subgraphs are **not** codewords.*

**Lemma 3.24** *An $i$ ($0 \leq i \leq d-1$) character sends the $x^{th}$ state ($0 \leq x \leq d-1$) to the $x + i \bmod d$ state (note: $i \neq x$). In other words, $\delta(x, i) = x + i \bmod d$.*

**Proof.** A vertex in state $x$ is at the $d - x \equiv -x \bmod d$ position in some $H_d^m$, with $m < n$.

Moving a vertex to the $i^{th}$ subgraph of the above subgraph will put the vertex in the $-x + (d - i) \equiv -(x + i) \bmod d$ position of $H_d^{m-1}$, which implies that this vertex should be in the $x + i \bmod d$ state. ■

If we used the zero state as the starting state for all codewords, the above proofs would describe the recognizer if all words of $K_d^n$ in the zero position of their respective $K_d^1$ were the only codewords. Obviously, this makes the zero state the only potential codeword state (out of the states zero through $d - 1$): looking at $K_d^1$ will show that zero would in fact be a codeword, but the one through $d - 1$ states are all not codewords, if the previous statement were true.

However, all zero position words are not codewords. Additionally, all codewords are not zero position words. For $K_d^n$ with $n$ even, $K_d^n$ is made from d copies of $K_d^{n-1}$ which are rotated according to the $\alpha$-method labeling. This forces us to have another starting point in the recognizer for even length strings to compensate for this positional change in codeword vertices. We will call this state "$E$".

**Lemma 3.25** *An $i$ character sends state $E$ to the $2i \bmod d$ state. In other words, $\delta(E, i) = 2i$.*

**Proof.** $K_d^n$ with $n$ even is composed of $d$ copies of rotated $\mathrm{K}_d^{n-1}$. Therefore, for the $k^{th}$ subgraph, $K_d^{n-1}$ is rotated $k$ positions.

The corner position of $K_d^n$ occur at the $k^{th}$ position. These vertices are codewords.

The effective rotation of these codeword vertices (from the zero position of the subgraph) is $2k$. This implies that an $i$ should move a vertex to the $2i \bmod d$ state. ■

We have already argued that states one through $d-1$ cannot be codeword states; we now show that the zero state is not a codeword state, and that state $E$ is a codeword state.

**Lemma 3.26** *The zero state is not a codeword state for $d \geq 3$.*

**Proof.** For $\mathrm{K}_d^3$ with $d \geq 3$, lemmas 3 and 4 show that the vertex labeled "101" is a codeword. This implies that the vertex labeled "$10(d - 1)$", which is in the same $K_d^1$ as "101", is not a codeword (for example, the vertex

"$10(d-1)$" is "104" if $d = 5$). However, for all $d \geq 3$, "$10(d-1)$" finishes in the zero state.

$\therefore$ The zero state is not a codeword state. ■

**Lemma 3.27** *State $E$ is a codeword state.*

**Proof.** By lemmas 3 and 5, the vertex labeled $(d-1)(d-2)$ is a codeword for all $\mathrm{K}_d^2$ (for example, if $d = 3$, the vertex is "21"). Obviously, the vertex labeled "$0(d-1)(d-2)$" is also a codeword. By lemmas two and three, the only vertex that sends a $d-1$ to the $d-2$ state is state $E$. This implies that the zero state sends a 0 to state $E$.

However, for $\mathrm{K}_d^1$, the vertex labeled 0 is a codeword by lemma 3.

$\therefore$ State $E$ is a codeword state. ■

**Corollary 3.28** *The zero state is not a codeword state for $d = 2$.*

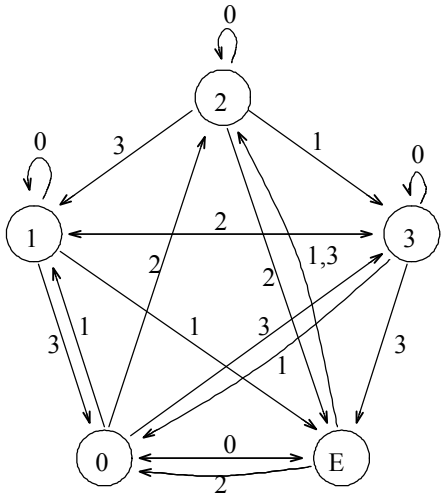
**Proof.** Assuming state $E$ is the codeword state referred to in lemma 1, the words "110" and "111" end in the zero state. However, these words are not codewords.

$\therefore$ The zero state is not a codeword state for $d = 2$. ■

**Theorem 3.29** *There exists a generalized codeword recognizer for the $\alpha$-method labeling of $K_d^n$.*

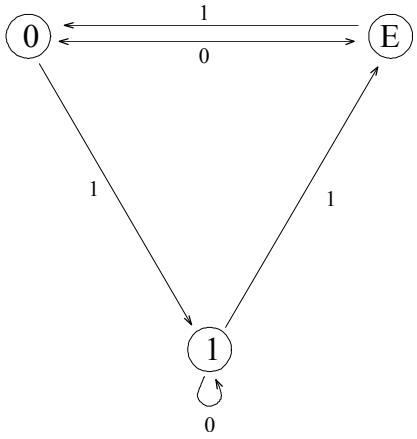
**Proof.** The first three lemmas of this section show that the $\delta$-function is necessary, while the last two lemmas and the corollary show that the $\delta$-function is sufficient to describe the codeword recognizer if state $E$ is chosen as the codeword state.

$\therefore$ A recognizer for the $\alpha$-method labeling of $K_d^n$ can be produced. ■

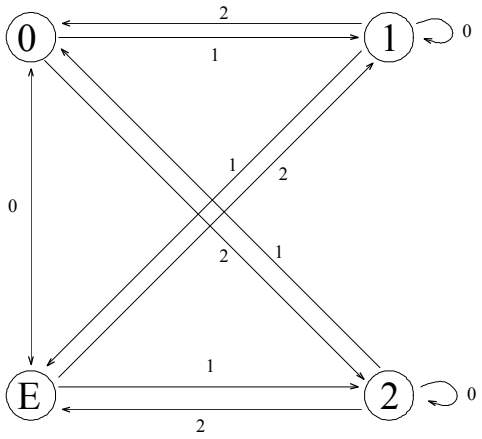



$K_4^n$ Codeword Recognizer

The completed $d = 4$ recognizer is shown above, while those for $d = 2$ and $d = 3$ are below.




$K_2^n$ Recognizer




$K_3^n$ Recognizer

### 3.2.3 The $\alpha$-method Labeling Error Corrector

Now that we have established a codeword recognizer for the $\alpha$-method labeling, we naturally wonder if there is a way to correct a word if the word

received is not a codeword. Put another way, can a finite state machine be constructed that will read a given string and change the appropriate character(s) to convert each noncodeword to the adjacent codeword while "correcting" the codewords to themselves?

**The Description** Fortunately, the answer to this question is yes. The construction of the error corrector follows directly from the recognizer, and so has $d + 1$ states. The construction will first be explained, followed by a proof showing that the construction creates an accurate, working error corrector.

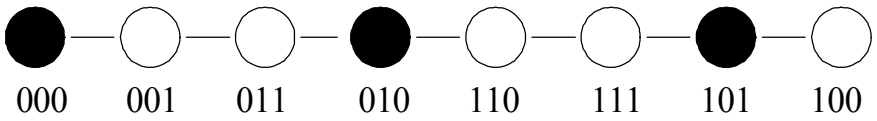
Each of the $d + 1$ states in the error corrector are starting states. The word to be corrected starts in the state that corresponds to the state in which the word finished when run through the codeword recognizer. For that reason, we will label the states of the error corrector as elements of $\{0,\ 1,\ ...,\ d-1,\ E\}$, each representing the identically labeled state of the recognizer. For example, when $d = 3$, the word "100" finishes in the first state, so it will enter the error corrector in the first state. The strings will be read from the right to the left (back to the front).

Once again, the transitions among the states can be classified using a two variable function. This time, however, the function output will have two components: one which determines to which state the current state is sent, and another which shows to what character the character being processed should be corrected. We will use $\rho$ to represent this function.

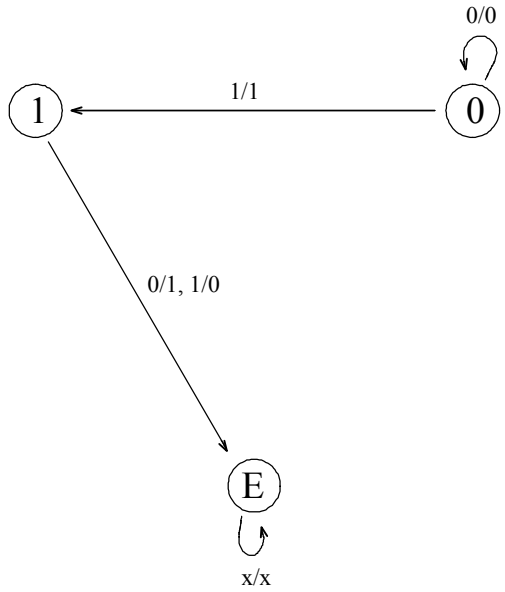
$$\begin{aligned}
&\rho(E, i) = (E, i), \text{ where } i \in \{0, 1, ...d-1\} \text{ input symbols;}\\
&\rho(x, i) = (E, x - i \bmod d), \text{ where } x \in \{0, 1, ...d-1\} \text{ states,}\\
&\qquad\qquad i \in \{0, 1, ...d-1\} \text{ input symbols, and } i \neq x - i \bmod d;\\
&\rho(x, i) = (i, i), \text{ where } x \in \{0, 1, ...d-1\} \text{ states,}\\
&\qquad\qquad i \in \{0, 1, ...d-1\} \text{ input symbols, and } i \equiv x - i \bmod d.
\end{aligned}$$

For example, when $d = 2$, "111" ends in the zero state. To error correct, we take the rightmost "1" and error correct to a "1" while moving to the one state. At the one state, we then error correct to the second "1" from the right to a "0" and move to state E. The last "1" is not changed, and

the error corrected word is "101." As expected and hoped for, this is the codeword adjacent to "111." The diagram below shows the labeled graph, $K_2^3$.




When following a diagram of an error corrector, the first character is the input signal each state reads, and the second character listed is the character that input signal is changed to. The $d = 2$ error corrector is shown below. The reader is encouraged to run the signal "111" through the machine to verify that the above process actually occurs as described.




$K_2^n$ Error Corrector

**The Proof** The error correction of a grade code is quite simple since only one character needs to be changed. A vertex in a gray code is only different in one character with all the vertices to which it is adjacent. This shows that each noncodeword requires the changing of only one character to be corrected to its adjacent codeword. Moreover, in most cases, the codeword is in the same $K_d^1$ subgraph, which implies that only the last character needs to be

changed. Difficulties arise only on the rare occasions that a noncodeword needs to change a character *not* at the end of the labeling. However, since the last character does not change, we can work backwards through the recognizer to determine what state the word had come from and error correct from there. The process can repeat itself until a character changes. The proof follows.

**Theorem 3.30** *The $\rho$-function gives a finite state machine description of the error corrector of an $\alpha$-method labeled graph of $K_d^n$.*

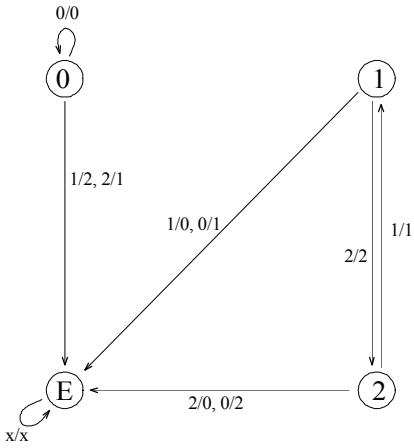
**Proof.** If the string is in the $E$ state of the error corrector, it is a codeword and should not be changed.

When a word finishes in the $x$ state of the recognizer, it was sent by the $i$ character from a previous state. This state is the $x - i \bmod d$ state by definition of $\delta$. By changing the $i$ character to $x - i \bmod d$, the $x - i$ state will be sent to state $E$, the codeword state. A word in the $x - i \bmod d$ state will send an $x - i$ character to the codeword state.

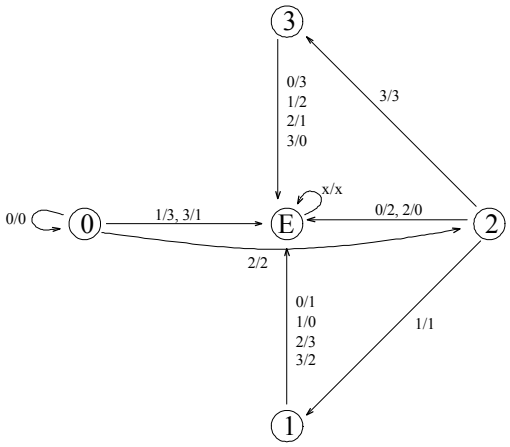
However, if $i \equiv x - i \bmod d$, changing the last character has no effect, and the above argument does not hold because the $\delta$ function acts differently when the character matches the state. Therefore, the last character must be held constant. Also, to continue to error correct, we must move to the $i^{th}$ state. This is because the $i^{th}$ state sends an $i$ character to the codeword state. The $i^{th}$ state will then error correct the next digit. This process is iterative. At the $i^{th}$ state, if the input signal is a $j$, the character will be corrected to $i-j \bmod d$ as above, assuming $j \neq i-j \bmod d$. If $j \equiv i-j \bmod d$, the character is not changed and the $i^{th}$ state is taken to the $j^{th}$ state. This process continues until a character is changed.

$\therefore$ The $\rho$-function characterizes the error corrector for $\mathrm{K}_d^n$. ■

The $K_3^n$ and $K_4^n$ error correctors are shown below.




$K_3^n$ Error Corrector




$K_4^n$ Error Corrector

### 3.2.4 Comparison to Tower of Hanoi Labeling

The Tower of Hanoi labeling of a $K_3^n$ graph is described in a paper by Paul Cull and Ingrid Nelson [1] and mentioned at the beginning of this paper. This labeling method also produces a gray code. There are some similarities and some differences between this method and the $\alpha$-method labeling which are displayed below.

| Labeling | **Towers of Hanoi** | **$\alpha$-method** |
|---|---|---|
| Gray Code | *Yes* | *Yes* |
| FSM Recognizer | *4 states* | *4 states* |
| FSM Error Corrector | *4 states* | *4 states* |
| Coding/Decoding | *Easy* | *Unknown* |
| Generalize to $d$ vertices | *Unknown* | *Yes* |
| Based on Fun Puzzle | *Yes* | *Unknown* |

Note: FSM stands for finite state machine

While the Towers of Hanoi labeling allows for easy coding and decoding while representing a fun and enjoyable game, the method is not known to be generalizable to any other dimension of complete graphs. Unfortunately, effective coding and decoding methods for the $\alpha$-method labeling are not

known at this time. Even more unfortunate, the $\alpha$-method labeled graphs do not represent any known puzzle. Even with these unfortunate consequences, the $\alpha$-method labeling does provide a generalized construction of $K_d^n$, as well as for recognizers and error correctors. Whether these advantages outweigh the disadvantages is left to the reader.

## 3.3 The *Multiples of* $d+1$ Code.

Herein is defined a labeling that works for all ToH type iterated complete graphs (all $K_d^n$) in general, and in which the labels are all $n$ digit words (or numbers, or strings, as I may refer to them) in base-$d$ (that is, each digit is an element of the set $\{0, 1, ..., d-1\}$), so that, since each graph of this type has $d^n$ vertices, every string of this form will be a label of a vertex of the graph. Further, I will define my labeling such that every label in any $K_d^m$ subgraph ($1 \leq m < n$) has the same number in the $m+1^{\text{th}}$ digit from the right. Note that this means that every number to the left of the $m+1^{\text{th}}$ digit will also be the same for every label in this subgraph, since the subgraph is a subset of one $K_d^{m-1}$ subgraph, and one $K_d^{m-2}$ subgraph, etc. (by the construction of the graphs). And this means that given a codeword, $d-1$ of the words that correct to it will be the $d-1$ words that differ from it in only their right-most digit (as these will be all of its neighbors within its $K_d^1$ base-graph).

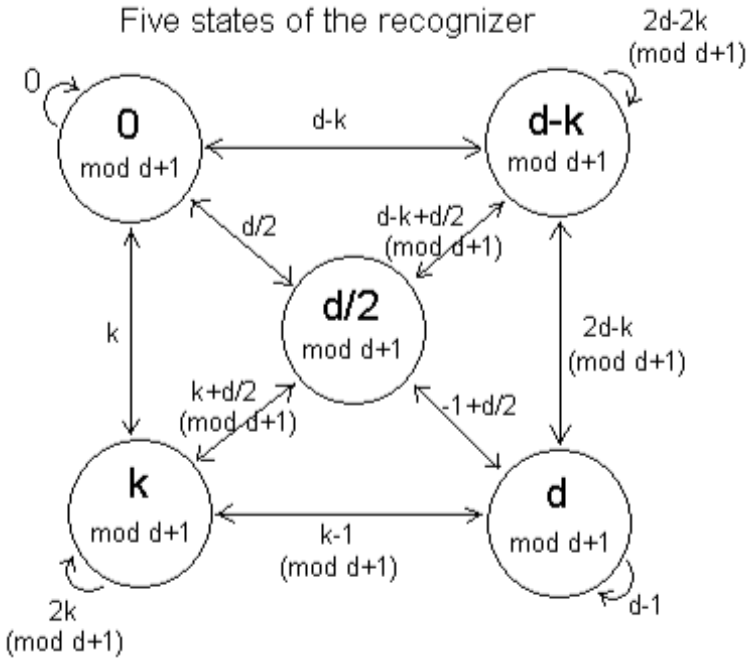
The final main parameter I want to give this labeling is simplicity of encoding/decoding, that is, the $k^{\text{th}}$ codeword will simply be the number $k \times (d+1)$ written out in base-$d$. (Note that this satisfies the requirement that all codewords be in different base-graphs (a weaker restatement of the condition that they can't be adjacent), since the smallest the difference between two codewords can be is 00...011, they will always differ in more than just the right-most digit.)

### 3.3.1 The Finite State Recognizer.

To say that codewords are all the $n$ digit multiples of $d+1$ is one thing, but can they be recognized quickly and efficiently? A simple, finite-state recognizer can be designed for this task as follows. Any base-$d$ number has a certain value $k \bmod d+1$. Adding another digit ($j$) onto the right end of this number is equivalent to multiplying it by $d$ and then adding to that the value of the new digit.

$$h = k(d) + j \equiv k(-1) + j \bmod d+1 \equiv -k + j \bmod d+1$$
$$h + k \equiv j \bmod d+1$$

Thus a $d+1$ state machine can be described, where each state represents a different value $k \bmod d+1$, where each state $k$ of the machine will have arrows from it pointing to all states $-k+j \bmod d+1$ for all $j \in \{0, 1, ..., d-1\}$, and where the arrow from state $k$ to $h$ will be followed (given that the machine is at state $k$) if the digit $k+h \bmod d+1$ is input. In other words, each state points to all but one of the $d+1$ total states, specifically, state $k$ doesn't point to state $d-k \bmod d+1$.




Thus, when this machine reads a base-$d$ string from left to right, one digit at a time, it keeps track of the value mod $d+1$ (going from state to state) of the entire string it has read so far. Note that all arrows in the machine are bi-directional (as $k+h = h+k$). If $d$ is even, then state $\frac{d}{2}$ does not point to state $d - \frac{d}{2} = \frac{d}{2}$ which is itself. Moreover, this is the only state that doesn't point to itself. All other states point to themselves and all but one of the

other states. Note that state $d-k$, which is not pointed to by state $k$, does not point to state $d-(d-k)=k$, as can also be concluded from the fact that all arrows are bi-directional (and thus lack of an arrow in one direction implies the lack of an arrow in the opposite direction).

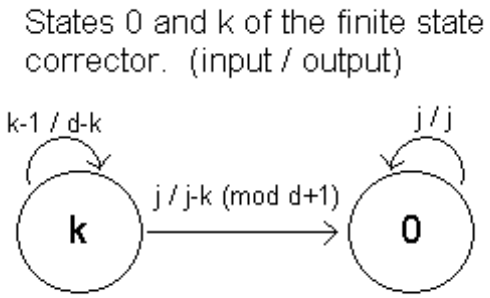
A string of one zero is congruent to 0 mod $d+1$, thus state 0 will be the starting state of the machine, since $h+k=j$ implies $k=0$, when $h=0$ and $j=0$. We shall have this machine output the number of the state it is left in (an element of $\{0,1,...,d\}$) after the entire $n$-digit string has been read in. An output of 0 means the string is a codeword. As we shall see later, the other outputs will be useful in determining how to error-correct non-codeword words, making this machine a classifier as well as a recognizer.

### 3.3.2 The Finite State Corrector.

The code wouldn't be perfect-one-error-correcting if there weren't a single codeword for each non-codeword to be corrected to. The machine that takes any word as an input and then outputs the nearest codeword to that word is called the error-corrector, and is constructed as follows.

The corrector will be a $d+1$ state machine, each state labeled uniquely from $\{0,1,...,d\}$. A base-$d$ string that leaves the recognizer in state $k$ mod $d+1$ will start at state $k$ of the corrector. In the corrector, all strings are read from right to left, and after each digit is read, a digit is output onto the left side of a new string, which will be the word the input string is corrected to.

The $k^{\text{th}}$ state of this machine (when $k \neq 0$) has one arrow that points to itself. The input needed to follow this arrow is $k-1$, and the output that following this arrow gives is $d-k$.



The remaining $d-1$ arrows from state $k$ all point to state 0 ($\forall k$ this time, though when $k=0$ there will be $d$ of these arrows). Their inputs are $0, 1, ..., k-2, k, ..., d-1$ and their outputs are, for input $j$, $j-k \pmod{d+1}$. (Note that had $j = k-1$, this would have given $-1 \bmod d+1 \equiv d$, which is not a valid base-$d$ digit, and that since $j$ cannot equal $d$ or $-1$, none of these outputs will be $d-k$, so no two inputs from any one state will give the same output.) (Also note something special about state 0 of the corrector. All $d$ arrows from it point back to itself, and for each arrow with input $j$, the output is also $j$, so a string that starts at state 0 of the corrector (a codeword) does not get changed, confirming that the closest codeword to any given codeword is itself.)

**To What Any Word Corrects.** Let $\alpha = \alpha_{n-1} \cdots \alpha_1 \alpha_0$ be a base-$d$ string with the value of $\alpha = \sum_{i=0}^{n-1} d^i \alpha_i \equiv k \bmod d+1$. If $k = 0$ then the output from the error-corrector, given input $\alpha$, will just be $\alpha$ and will thus (since $k = 0$) be a multiple of $d+1$. Otherwise, write out the string $\alpha$ as $\alpha_{n-1} \cdots \alpha_{m+l} j_{(m)} (k-1)_{m-1} (k-1)_{m-2} \cdots (k-1)_0$ where $\alpha_i \in \{0, 1, ..., d-1\}$, $j$ is the first digit not equal to $k-1$, and $m \in \{0, 1, ...n-1\}$. The reason $m$ is not allowed to equal $n$, is that, when inputting it into the classifier/recognizer, if $\alpha = (k-1)_{n-1} \cdots (k-1)_1 (k-1)_0$ then it would oscillate back and forth between state 0 and state $k-1$, and so its value mod $d+1$ would be one of these, and not $k$, a contradiction.

Let $\beta = EC(\alpha)$ (i.e. $\alpha$ error-corrects to $\beta$ ). If $m = 0$, $\alpha = \alpha_{n-1} \cdots \alpha_2 \alpha_1 j_{(0)} \equiv k \bmod d+1$, and so $\beta = \alpha_1 \alpha_2 \cdots \alpha_{n-1} (j-k \bmod d+1) \equiv k-k \bmod d+1 \equiv 0 \bmod d+1$. For all $m$,

$$
\begin{aligned}
\beta &= \alpha_{n-1}\cdots\alpha_{m+1}(j-k \bmod d+1)_m(d-k)_{m-1}\cdots(d-k)_1(d-k)_0 \\
&\equiv \sum_{i=m+1}^{n-1} d^i\alpha_i + d^m(j-k) + \sum_{i=0}^{m-1} d^i(d-k)) \qquad \bmod d+1 \\
&\equiv \sum_{i=m+1}^{n-1} d^i\alpha_i + d^m(j) + d^m(-k) + \sum_{i=0}^{m-1} d^i(k-1) - \sum_{i=0}^{m-1} d^i(k-1) + \sum_{i=0}^{m-1} d^i(d-k) \\
&\equiv \sum_{i=0}^{n-1} d^i\alpha_i + d^m(-k) + \sum_{i=0}^{m-1} d^i((d-k)-(k-1)) \\
&\equiv k + (-1)^m(-k) + \sum_{i=0}^{m-1}(-1)^i(-1-2k+1) \qquad \bmod d+1 \\
&\equiv k(1+(-1)^{m-1}) - 2k\sum_{i=0}^{m-1}(-1)^i \\
&\equiv k(1+(-1)^{m-1} - 2(\frac{1-(-1)^m}{1-(-1)})) \\
&\equiv k(1+(-1)^{m-1} - 1 + (-1)^m) \\
&\equiv 0
\end{aligned}
$$

Thus, given any input string, the corrector will output a codeword.

**Which Words Correct to Any Given Codeword.**

**Theorem 3.31** *Given any $n$ digit, base-$d$ string $c$ where $c = \sum_{i=0}^{n-1} d^i c_i \equiv 0 \bmod d+1$, there exist exactly $d$ other strings that the corrector corrects to $c$, unless $c$ consists of the same digit repeated $n$ times, in which case exactly $d-1$ other strings correct to $c$.*

**Proof:** If $c$ is a string of only one repeated digit, for example $c = j_{(n-1)}\cdots j_{(1)}j_{(0)}$, then any $\alpha = j_{(n-1)}\cdots j_{(1)}(j+k \bmod d+1)_0$ will be equivalent to $0+k \equiv k \bmod d+1$ for all $k \in \{1,2,...,d\}\backslash\{d-j\}$, ($k = d-j$ does not give a digit in base-$d$). Note that $j \in \{0,1,...,d-1\} \neq -1 \bmod d+1$, ever, which implies that $j+k \neq k-1$. Thus $\alpha$ will correct to $j_{(n-1)}\cdots j_{(1)}(j+k-k \bmod d+1)_0 = j_{(n-1)}\cdots j_{(1)}j_{(0)} = c$. Thus at least $d-1$ strings $\neq c$ correct to $c$.

The only other strings that could correct to $c$ must be of the form $\alpha = j_{(n-1)} \cdots j_{(l)} h_{(m)} (k-1)_{m-1} \cdots (k-1)_1 (k-1)_0$ where $m \in \{1, 2, ..., n\}$, $\alpha \equiv k \bmod d+1$, and $h$ and $k-1$ both correct to $j$. But since different digits don't correct to the same digit from any one state, $h = k-1$, so $\alpha = j_{(n-1)} \cdots j_{(m)} (k-1)_{m-1} \cdots (k-1)_1 (k-1)_0$, but similarly, $j$ and $k-1$ can't correct to the same thing unless $k-1 = j$, so $\alpha$ must equal $(k-1)_{n-1} \cdots (k-1)_1 (k-1)_0$, but notice from the recognizer that this string must equal either 0 or $k-1 \bmod d+1$, and not $k$, a contradiction. Therefore a $c$ of this form has exactly $d-1$ other strings that correct to it (all different from it in only the right-most digit).

For $c = c_{n-1} ... c_1 c_0$ where the $c_i$ are not all equal, if $\alpha = c_{n-1} \cdots c_1 (c_0 + k \bmod d+1)$ for $k \in \{1, 2, ..., d\} \backslash \{d - c_n\}$, then $\alpha \equiv k \bmod d+1$ and will be corrected to $EC(\alpha) = c_{n-1} \cdots c_1 (c_0 + k - k \bmod d+1) = c$, since $c_0 + k \bmod d+1 \neq k-1$ for $c_0 \in \{0, 1, ..., d-1\}$. Thus at least $d-1$ words $\neq c$ correct to $c$, and note that these are all the strings that differ from $c$ in only their right-most digit.

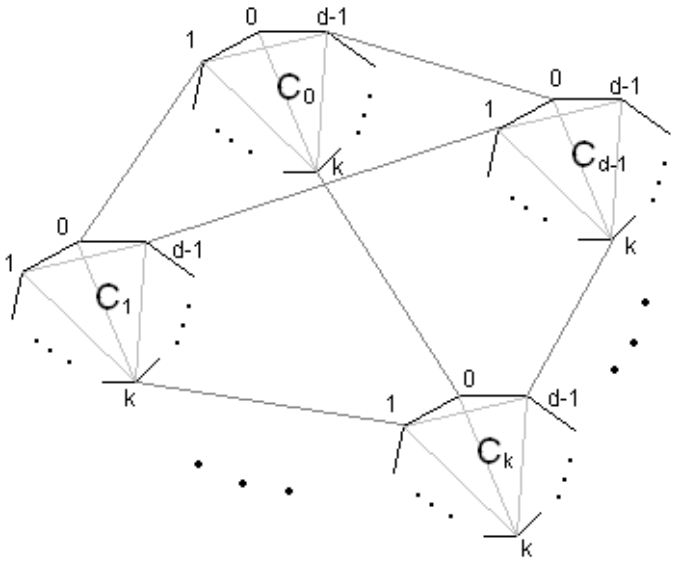
The only other strings that could correct to $c$ must be of the form $\alpha = c_{n-1} \cdots c_{m+1} (c_m + k)(k-1)_{m-1} \cdots (k-1)_0 \equiv k \bmod d+1$, which will only correct to $c$ if $c = c_{n-1} \cdots c_{m+1} c_m (d-k)_{m-1} \cdots (d-k)_0 \equiv 0 \bmod d+1$, where $c_m \neq (d-k)$. Consequently, when $c = c_{n-1} \cdots c_m j_{(m-1)} \cdots j_{(0)}$, $(m \in \{1, 2, ..., n-1\})$, $\alpha = c_{n-1} \cdots c_{m+1} (c_m + d - j \bmod d+1)(d-j-1)_{m-1} \cdots (d-j-1)_0$ (by the error-corrector, since only $d-j-1$ corrects to $j$ without taking the machine to state 0, but only at state $k = d-j$). So we check,

$$
\begin{aligned}
\alpha &= \sum_{i=m+1}^{n-1} d^i c_i + d^m(c_m + d - j \bmod d + 1) + \sum_{i=0}^{m-1} d^i(d-j-1) \\
&\equiv \sum_{i=m+1}^{n-1} d^i c_i + d^m(c_m) + d^m(d-j) + \sum_{i=0}^{m-1} d^i j - \sum_{i=0}^{m-1} d^i j + \sum_{i=0}^{m-1} d^i(d-j-1) \\
&\equiv \sum_{i=0}^{n-1} d^i c_i + d^m(d-j) + \sum_{i=0}^{m-1} d^i(d-j-1-j) \\
&\equiv 0 + (-1)^m(d-j) + \frac{1-d^m}{1-d}(d-2j+d) \qquad \bmod d+1 \\
&\equiv (-1)^m(d-j) + \frac{1-(-1)^m}{1-(-1)}(2d-2j) \\
&\equiv (-1)^m(d-j) + (d-j) - (-1)^m(d-j) \\
&\equiv d-j
\end{aligned}
$$

Which means that there does exist exactly one more word that corrects to any given codeword (other than all the words that differ from the codeword in only the right-most digit), and that word is $\alpha = c_{n-1}\cdots c_{m+1}(c_m + d - j \bmod d+1)(d-j-1)_{m-1}\cdots(d-j-1)_0$, when $c = c_{n-1}\cdots c_m j_{(m-1)}\cdots j_{(0)}$, for $m \in \{0, 1, ..., n-1\}$ and $c_m \neq j$. ■

### 3.3.3 Constructing the Labelings of the Graphs.

Now that a coding, decoding, recognizing, classifying, and error-correction process for this system of labeling is known, we need to actually describe where to put these labels onto a ToH type iterated complete graph in such a way that, 1) each label has its own vertex (and vice-versa), 2) no two codewords are adjacent, and 3) each non-codeword label is adjacent to exactly one codeword, the codeword it error-corrects to.




Labeling Construction

For $n = 1$, each vertex of the $d$-point complete graph ($K_d^1$) is labeled distinctly from $\{0, 1, ..., d-1\}$. Given the $n-1^{\text{th}}$ labeling, the $n^{\text{th}}$ labeling is constructed by connecting $d$ copies of the $n-1^{\text{th}}$ labeling, called $C_0, C_1 ... C_{d-1}$, as shown in figure, such that the $j^{\text{th}}$ corner of $C_k$ is connected to the $k^{\text{th}}$ corner of $C_j$. (The $j^{\text{th}}$ corner of the $n = 1$ labeling is the vertex labeled with a $j$. It can be seen shortly that the $j^{\text{th}}$ corner of the $n-1^{\text{th}}$ labeling will be labeled with a string of $n-1$ $j$'s.) The $k^{\text{th}}$ corner of $C_k$ does not connect to anything, and becomes the $k^{\text{th}}$ corner of the $n^{\text{th}}$ labeling. Next, every label in $C_k$ gets appended on the left with the digit $k$, for all $k$. Then $C_k$ gets *position-swapped* according to the rule:

$$i \rightleftarrows j; \ \forall\, i, j \in \{0, 1, ..., d-1\} \text{ such that } i + j \equiv d + k \bmod d + 1$$

(Note that any given $j \in \{0, 1, ..., d-1\}$ satisfies the condition with exactly one $i$, unless $j = k$, in which case it satisfies it with no $i$ (since $i$ can't equal $d$ or $-1$).)

**The Position-Swapping Isomorphism.** What position-swapping $i$ and $j$ ($i \rightleftarrows j$) means, when applied to a graph $K_d^n$, is that the two $K_d^{n-1}$ subgraphs in the $i^{\text{th}}$ and $j^{\text{th}}$ positions of the $d^{n-1}$-point subgraph switch places, and that

the $K_d^{n-2}$ sub-subgraphs in the $i^{\text{th}}$ and $j^{\text{th}}$ positions of each $K_d^{n-1}$ subgraph switch places, etc., all the way down to the label in the $i^{\text{th}}$ position of every $K_d^1$ base-graph is switched with the label in the $j^{\text{th}}$ position of the same base-graph.

In other words, in the standard (ordered) labeling of ToH type iterated complete graphs, if $x$ is a label on $K_d^n$, then $x = x_{n-1}x_{n-2}\cdots x_1x_0$ ($x_k \in \{0, 1, ..., d-1\}$) means that $x$ is the $x_0^{\text{th}}$ point on the $x_1^{\text{th}}$ $K_d^1$ sub-$\cdots$-graph of the $x_2^{\text{th}}$ $K_d^2$ sub-$\cdots$-graph of the... of the $x_{n-2}^{\text{th}}$ $K_d^{n-2}$ sub-subgraph of the $x_{n-1}^{\text{th}}$ $K_d^{n-1}$ subgraph of $K_d^n$. So position-swapping $x$ $i \rightleftarrows j$ gives $y$, that is $PS_{i\rightleftarrows j}(x) = y = y_{n-1}y_{n-2}\cdots y_1y_0$, where if $i \neq x_k \neq j$ then $y_k = x_k$, if $x_k = i$ then $y_k = j$, and if $x_k = j$ then $y_k = i$.

### 3.3.4 Proving the Labelings.

The proof that the above labeling scheme corresponds to the mechanical recognizing and error-correcting processes that have been described, requires some definitions and a stack of lemmas and theorems.

**Definitions.**

**Definition 3.32** *1: G-graphs and U-graphs (or $G^n$ and $U^n$) are defined as in definitions 2.8 and 2.9.*

**Definition 3.33** *2: The operators $\uparrow$ and $\downarrow$, when applied to the above graph-symbols (as $\uparrow U^n$ or $\downarrow G^n$ for example), mean, respectively, clockwise and counter-clockwise rotations of the graphs they're applied to. Where each rotation is $\frac{1}{d}$ of a complete revolution (i.e. $\frac{360}{d}$ degrees), and $\downarrow^k$ means $\frac{k}{d}$ of a complete revolution. Note that $\downarrow^k = \uparrow^{d-k}$ and $\uparrow^k = \downarrow^{d-k}$.*

**Starting Lemmas.**

**Lemma 3.34** *Appending a digit $k \in \{0, 1, ..., d-1\}$ on the left side of an $n-1$ digit base-$d$ string increments the value of that string by $k \cdot d^{n-1} \equiv k(-1)^{n-1} \bmod d+1$.*

**Proof:** Adding a digit $k$ to the left side of a base-$d$ string of length $n-1$ increases the value of the string by $k \cdot d^{n-1}$. If the value of this string is taken mod $d+1$ then, since $d \equiv -1 \bmod d+1$, the value of the string is increased by $k(-1)^{n-1} \bmod d+1$. ■

**Lemma 3.35** *Starting Lemma 2:*

*1) Position-swapping $i \rightleftarrows j$ a $\downarrow^k G^n$ for $n$ odd or a $\downarrow^k U^n$ for $n$ even, when $i \neq k \neq j$, does not change the graphs.*

*2) Position-swapping $j \rightleftarrows k$ a $\downarrow^k G^n$ for $n$ odd or a $\downarrow^k U^n$ for $n$ even, changes them to a $\downarrow^j G^n$ or a $\downarrow^j U^n$, respectively.*

*3) Position-swapping $i \rightleftarrows j$ a $G^n$ for $n$ even or a $U^n$ for $n$ odd does not change the graphs, $\forall j, i$.*

**Proof:** When $n = 1$ the lemma is clearly true, by the construction of the graphs and the definitions of the functions. Now assume the lemma is true for $n = m - 1$.

When $m$ is odd, $\downarrow^k G^m$ consists of an ordered complete graph of $d$ subgraphs (as in the labeling construction), the $k^{\text{th}}$ of which is $\downarrow^k \mathrm{G}^{m-1} = \mathrm{G}^{m-1}$, while all $d-1$ others are $\downarrow^k \mathrm{U}^{m-1}$.

Position-swapping $i \rightleftarrows j$ $(i \neq k \neq j)$ swaps the $i^{\text{th}}$ and $j^{\text{th}}$ subgraphs, both of which are $\downarrow^k \mathrm{U}^{m-1}$ (thus changing nothing) and then position-swaps $i \rightleftarrows j$ the $\mathrm{G}^{m-1}$ subgraph and all the $\downarrow^k \mathrm{U}^{m-1}$ subgraphs, which, by the assumption on $n = m - 1$, also changes nothing. Therefore, position-swapping $i \rightleftarrows j$ a $\downarrow^k \mathrm{G}^m$ for $i \neq k \neq j$ and $m$ odd does not change the graph if the lemma holds for $m - 1$.

Position-swapping $j \rightleftarrows k$ swaps the $j^{\text{th}}$ and $k^{\text{th}}$ subgraphs, so that the $k^{\text{th}}$ subgraph is then $\downarrow^k \mathrm{U}^{m-1}$ and the $j^{\text{th}}$ is $\mathrm{G}^{m-1}$. Next, every subgraph gets position-swapped, which, by the assumption, changes the $\downarrow^k \mathrm{U}^{m-1}$ to $\downarrow^j \mathrm{U}^{m-1}$ but doesn't affect the $\mathrm{G}^{m-1}$, so what is left is a graph with $\mathrm{G}^{m-1}$ in the $j^{\text{th}}$ position and $\downarrow^j \mathrm{U}^{m-1}$'s in all others, which is the construction of $\downarrow^j \mathrm{G}^m$ for $m$ odd. Thus $PS_{j \rightleftarrows k}(\downarrow^k \mathrm{G}^m) = \downarrow^j \mathrm{G}^m$ (given the assumption and $m$ odd).

When $m$ is even, $\downarrow^k \mathrm{U}^m$ consists of an ordered complete graph of $d$ subgraphs, the $k^{\text{th}}$ of which is $\mathrm{U}^{m-1}$, while all $d-1$ others are $\downarrow^k \mathrm{G}^{m-1}$.

Position-swapping $i \rightleftarrows j$ $(i \neq k \neq j)$ swaps the $i^{\text{th}}$ and $j^{\text{th}}$ subgraphs, both of which are $\downarrow^k \mathrm{G}^{m-1}$ (thus changing nothing) and then position-swaps $i \rightleftarrows j$ the $\mathrm{U}^{m-1}$ subgraph and all the $\downarrow^k \mathrm{G}^{m-1}$ subgraphs, which, by the assumption on $n = m - 1$, also changes nothing. Therefore, position-swapping $i \rightleftarrows j$ a $\downarrow^k \mathrm{U}^m$ for $i \neq k \neq j$ and $m$ even does not change the graph if the lemma holds for $m - 1$.

Position-swapping $j \rightleftarrows k$ swaps the $j^{\text{th}}$ and $k^{\text{th}}$ subgraphs, so that the $k^{\text{th}}$ subgraph is then $\downarrow^k \mathrm{G}^{m-1}$ and the $j^{\text{th}}$ is $\mathrm{U}^{m-1}$. Next, every subgraph gets

position-swapped, which, by the assumption, changes the $\downarrow^k \mathrm{G}^{m-1}$ to $\downarrow^j \mathrm{G}^{m-1}$ but doesn't affect the $\mathrm{U}^{m-1}$, so what is left is a graph with $\mathrm{U}^{m-1}$ in the $j^{\text{th}}$ position and $\downarrow^j \mathrm{G}^{m-1}$'s in all others, which is the construction of $\downarrow^j \mathrm{U}^m$ for $m$ even. Thus $PS_{j \rightleftarrows k}(\downarrow^k \mathrm{U}^m) = \downarrow^j \mathrm{U}^m$ (given the assumption and $m$ even).

When $m$ is even, $\mathrm{G}^m$ consists of an ordered complete graph of $d$ subgraphs, the $k^{\text{th}}$ of which is $\downarrow^k \mathrm{G}^{m-1}$, $\forall k \in \{0, 1, ..., d-1\}$.

Position-swapping $i \rightleftarrows j$ swaps the $i^{\text{th}}$ and $j^{\text{th}}$ subgraphs, so that the $i^{\text{th}}$ subgraph is then $\downarrow^j \mathrm{G}^{m-1}$ and the $j^{\text{th}}$ is $\downarrow^i \mathrm{G}^{m-1}$. Applying the position-swapping to each subgraph then changes $\downarrow^j \mathrm{G}^{m-1}$ to $\downarrow^i \mathrm{G}^{m-1}$ (this in the $i^{\text{th}}$ position), $\downarrow^i \mathrm{G}^{m-1}$ to $\downarrow^j \mathrm{G}^{m-1}$ (this in the $j^{\text{th}}$ position), and leaves unchanged all other subgraphs for which $i \neq k \neq j$. Thus $\mathrm{G}^m$ is the same as it was before the position-swapping, when $m$ is even.

When $m$ is odd, $\mathrm{U}^m$ consists of an ordered complete graph of $d$ subgraphs, the $k^{\text{th}}$ of which is $\downarrow^k \mathrm{U}^{m-1}$, $\forall k \in \{0, 1, ..., d-1\}$.

Position-swapping $i \rightleftarrows j$ swaps the $i^{\text{th}}$ and $j^{\text{th}}$ subgraphs, so that the $i^{\text{th}}$ subgraph is then $\downarrow^j \mathrm{U}^{m-1}$ and the $j^{\text{th}}$ is $\downarrow^i \mathrm{U}^{m-1}$. Applying the position-swapping to each subgraph then changes $\downarrow^j \mathrm{U}^{m-1}$ to $\downarrow^i \mathrm{U}^{m-1}$ (this in the $i^{\text{th}}$ position), $\downarrow^i \mathrm{U}^{m-1}$ to $\downarrow^j \mathrm{U}^{m-1}$ (this in the $j^{\text{th}}$ position), and leaves unchanged all other subgraphs for which $i \neq k \neq j$. Thus $\mathrm{U}^m$ is the same as it was before the position-swapping, when $m$ is odd.

The entire lemma is thus proven for $m$, given it is true for $m-1$, and since it is true for $n = 1$, it is thus proven for all $n$. ■

**Mod $d+1$ Arrangement Theorem.**

**Theorem 3.36** *When reading the previously constructed labelling, the labels, read mod $d+1$, form the following patterns on the graph:*

*When $n$ is odd, the $k$'s mod $d+1$ ($k \in \{0, 1, ..., d-1\}$) are in the codeword positions of $\downarrow^k G^n$, and the $d$'s mod $d+1$ are in the codeword positions of $U^n$.*

*When $n$ is even, the $0$'s mod $d+1$ are in the codeword positions of $G^n$, and the $k$'s mod $d+1$ ($k \in \{1, 2, ..., d\}$) are in the codeword positions of $\uparrow^k U^n = \downarrow^{d-k} U^n$.*

**Proof:** When $n = 1$ the lemma is clearly true, by construction. Now assume the lemma is true for $m = n - 1$.

When $n$ is even, $n-1$ is odd. So the 0's in the constructed $n^{\text{th}}$ labeling mod $d+1$ will be where the $-k(-1)^{n-1} = k$'s are (since $-k(-1)^{n-1} + k(-1)^{n-1} = 0$) in each $k^{\text{th}}$ position-swapped $n-1^{\text{th}}$ labeling ($k \in \{0, 1, ..., d-1\}$), which are $\downarrow^k \mathrm{G}^{n-1}$ (by the assumption), but since the position-swapping doesn't touch the $k^{\text{th}}$ position in the $k^{\text{th}}$ graph (by the labeling construction position-swapping rule), and by starting lemma 2, the locations of the $k$'s in all subgraphs are left unchanged by the position-swapping. Thus the locations of the 0's on the $n^{\text{th}}$ graph (well, the labelling of $K_d^n$, really) are given by combining $d$ $n-1$ graphs, the one in the $k^{\text{th}}$ position being $\downarrow^k \mathrm{G}^{n-1}$, for all $k$. Which is just the definition of $\mathrm{G}^n$ when $n$ is even.

When $n$ is odd, $n-1$ is even. So the $d$'s in the constructed $n^{\text{th}}$ labeling mod $d+1$ will be where the $d - k(-1)^{n-1} = d - k$'s are in each $k^{\text{th}}$ position-swapped $n-1^{\text{th}}$ labeling ($k \in \{0, 1, ..., d-1\}$), which are where the codewords of $\uparrow^{d-k} \mathrm{U}^{n-1} = \downarrow^k \mathrm{U}^{n-1}$ are ($d - k \in \{1, 2, ..., d\}$), but since the position-swapping doesn't touch the $k^{\text{th}}$ position in the $k^{\text{th}}$ graph, and by starting lemma 2, the locations of the $d-k$'s are left unchanged by the position-swapping. Thus the locations of the $d$'s on the $n^{\text{th}}$ graph are given by combining $d$ $n-1$ graphs, the one in the $k^{\text{th}}$ position being $\downarrow^k \mathrm{U}^{n-1}$, for all $k$. Which is the definition of $\mathrm{U}^n$ when $n$ is odd.

When $n$ is even, $n-1$ is odd. So the $j$'s ($j \in \{1, 2, ..., d\}$) in the constructed $n^{\text{th}}$ labeling mod $d+1$ will be where the $j - k(-1)^{n-1} = j + k$'s mod $d+1$ are in each $k^{\text{th}}$ position-swapped $n-1^{\text{th}}$ labeling ($k \in \{0, 1, ..., d-1\}$), which, for $k = d - j$, are the codeword positions of $\mathrm{U}^{n-1}$ which is not changed by position-swapping. For all other $k$, the $j + k$'s will be in the codeword positions of $\downarrow^{j+k} \mathrm{G}^{n-1}$, which position-swapping changes to $\downarrow^{d-j} \mathrm{G}^{n-1}$, since $(j+k)+(d-j) = d+k \bmod d+1$ and thus $j+k \bmod d+1 \rightleftarrows d-j$ in the $k^{\text{th}}$ subgraph. Thus the locations of the $j$'s on the $n^{\text{th}}$ graph are given by combining $d$ $n-1^{\text{th}}$ graphs, the one in the $d-j^{\text{th}}$ position being $\mathrm{U}^{n-1}$, and all others being $\downarrow^{d-j} \mathrm{G}^{n-1}$. Which is just $\downarrow^{d-j} \mathrm{U}^n = \uparrow^j \mathrm{U}^n$ for all $j \in \{1, 2, ..., d\}$ when $n$ is even.

When $n$ is odd, $n-1$ is even. So the $j$'s ($j \in \{0, 1, ..., d-1\}$) in the constructed $n^{\text{th}}$ labeling mod $d+1$ will be where the $j - k(-1)^{n-1} = j - k$'s mod $d+1$ are in each $k^{\text{th}}$ position-swapped $n-1^{\text{th}}$ labeling ($k \in \{0, 1, ..., d-1\}$), which, for $k = j$, are the codeword positions of $\mathrm{G}^{n-1}$ which is not changed by position-swapping. For all other $k$, the $j-k$'s will be in

the codeword positions of $\uparrow^{j-k}\mathrm{U}^{n-1} = \downarrow^{d+k-j}\mathrm{U}^{n-1}$, which position-swapping changes to $\downarrow^{j}\mathrm{U}^{n-1}$, since $(d+k-j)+(j) = d+k \bmod d+1$ and thus $j+k-j \bmod d+1 \rightleftarrows j$ in the $k^{\text{th}}$ subgraph. Thus the locations of the $j$'s on the $n^{\text{th}}$ graph are given by combining $d$ $n-1^{\text{th}}$ graphs, the one in the $j^{\text{th}}$ position being $\mathrm{G}^{n-1}$, and all others being $\downarrow^{j}\mathrm{U}^{n-1}$. Which is just $\downarrow^{j}\mathrm{G}^{n}$ for all $j \in \{0, 1, ..., d-1\}$ when $n$ is odd.

Therefore, since the lemma is true for $n$, given that it is true for $n-1$, and since it is true for $n = 1$, inductively, it is true for all $n$. ■

**Corollary 3.37** *Codeword location corollary: The given labelling construction puts base-d labels that are multiples of* $d+1$ *in the codeword positions of* $G^n$ *for all* $d \times n$.

**Proof:** This is just a subset of what was proved in the theorem. Multiples of $d+1 \equiv 0 \bmod d+1$. ■

**Corollary 3.38** *Weak codes corollary: The given labelling construction puts base-d strings that equal* $k \bmod d+1$ *into the codeword positions of a weak-code on the graph* $\forall k, d, n$.

**Proof:** Again, this is a subset of the theorem, since rotations of G and U are always weak codes. ■

**Corrector Correspondence.** It has been shown (Codeword location corollary) that the finite-state recognizer recognizes as codewords exactly those words that are at the codeword positions of $\mathrm{G}^n$ on the graph. What follows will show a similar correspondence between the graph labelings and the finite-state error corrector.

Right-most Only Error-Correction Lemma:

In the iterative method developed for labeling all $K_d^n$, any codeword is at a vertex of one $K_d^1$ complete subgraph (since the entire graph is made up of $d^{n-1}$ connected $K_d^1$ graphs, and thus there is nowhere else for a codeword to be). The other $d-1$ words in the codeword's $K_d^1$ subgraph differ from it in only their single right-most digits (by construction of the labelings, a $K_d^1$ subgraph gets copied and position-swapped many times, but while position-swapping may change the location of a $K_d^1$ subgraph and the order of the labels within that graph, it never moves a label out of its $K_d^1$ subgraph and

away from the other $d-1$ labels in it, and, since, after the right-most digit, all the labels in one $K_d^1$ subgraph always get prepended (in the construction) by the same thing.) Therefore, those $d-1$ adjacent labels within the base-graph are the words that error-correct to the codeword by changing only the right-most digit.

**Lemma 3.39** *Corner labels: If a label is on a corner, and thus not adjacent to any points other than those in the base-graph it's in, then its string consists of the same digit repeated n times.*

**Proof:** This comes from the construction of the labeling. Assume that the $k^{\text{th}}$ corner $K_d^{m-1}$ is labeled with $m-1$ $k$'s (this is clearly true for all $k$'s on the $n=1$ graph). The $k^{\text{th}}$ corner of $K_d^m$ is the $k^{\text{th}}$ corner of the $k^{\text{th}}$ copy of the position-swapped $K_d^{m-1}$. But since position-swapping doesn't touch the $k^{\text{th}}$ corner of the $k^{\text{th}}$ subgraph, and all the labels on that graph get prepended with a $k$, the $k^{\text{th}}$ corner of the $m^{\text{th}}$ graph consists of $m$ $k$'s. Thus by induction this is true for all $n$. ■

Therefore the error-correction machine has been validated for all words adjacent to codewords consisting of one repeated digit (in that only those $d-1$ words that differ from the codeword in the right-most digit correct to it), and all labels in the same $K_d^1$ subgraph as any codeword. What remains to be proven is the error-correction of labels adjacent to codewords but not in the same $K_d^1$ subgraph as them.

**More and More Lemmas.** Lemma $X$:

Any $K_d^m$ labeling of this construction has a string of $m$ $k$'s labeling the $k^{\text{th}}$ corner. When constructing the labeling of $K_d^{m+1}$ from this, the $j^{\text{th}}$ corner of $C_k$ ($k \neq j$) gets swapped with the $d+k-j^{\text{th}} \bmod d+1$ corner (by the position-swapping isomorphism, a corner must go to a corner) and the $k^{\text{th}}$ corner of $C_j$ gets swapped with the $d+j-k^{\text{th}} \bmod d+1$ corner. This means that, after prepending $k$'s and $j$'s and connecting the corners of the graphs, the string

$$k_{(m)}(d+k-j \bmod d+1)_{m-1}(d+k-j \bmod d+1)_{m-2}\cdots(d+k-j \bmod d+1)_0$$

is adjacent to the string

$j_{(m)}(d+j-k \bmod d+1)_{m-1}(d+j-k \bmod d+1)_{m-2}\cdots(d+j-k \bmod d+1)_0$.

And since

$$(d+k-j \bmod d+1)+(d+j-k \bmod d+1)=2d \bmod d+1=d-1$$

and this is true for all $j$,$k$,$d$, and $m$ ($j \neq k$), then in all $K_d^{m+1}$ labelings the label

$$k_{(m)}j_{(m-1)}j_{(m-2)}\cdots j_{(0)}$$

is adjacent to the label

$$(d+k-j \bmod d+1)_m(d-1-j)_{m-1}(d-1-j)_{m-2}\cdots(d-1-j)_0.$$

**Lemma 3.40** Y*: Position-swapping a labelled graph does not change which labels are adjacent within that graph.*

**Proof:** In the standard (ordered) labeling (see section on position-swapping), the labeling is generated much the same way as the 'multiples-of-$d+1$-codewords' labeling, except that no position swapping takes place. That is the $K_d^n$ labeling is made up of $d$ copies of the $K_d^{n-1}$ labeling, called $C_0, C_1, ..., C_{d-1}$, with an edge drawn between the $j^{\text{th}}$ corner of $C_k$ and the $k^{\text{th}}$ corner of $C_j$, with $C_k$ getting all of its labels prepended with a $k$. Thus it is clear that label $kjj\cdots j$ is adjacent to label $jkk\cdots k$, and, as the iteration continues, label $x_{n-1}x_{n-2}\cdots x_mkjj\cdots j$ will be adjacent to $x_{n-1}x_{n-2}\cdots x_mjkk\cdots k$ for all $x_i \in \{0,1,...,d-1\}$.

Thus for label $x = x_{n-1}x_{n-2}\cdots x_mkjj\cdots j$ ($k \neq j$) adjacent to $w = x_{n-1}x_{n-2}\cdots x_mjkk\cdots k$, given that $h \neq k$, $h \neq j$, $i \neq k$, and $i \neq j$, then $PS_{h\rightleftarrows i}(x) = y_{n-1}y_{n-2}\cdots y_mkjj\cdots j$ and $PS_{h\rightleftarrows i}(x) = y_{n-1}y_{n-2}\cdots y_mjkk\cdots k$, (where $y_l = x_l$ if $h \neq x_l \neq i$, $y_l = h$ if $x_l = i$, and $y_l = i$ if $x_l = h$, in this case) which are adjacent. $PS_{k\rightleftarrows h}(x) = y_{n-1}y_{n-2}\cdots y_mhjj\cdots j$ and $PS_{k\rightleftarrows h}(w) = y_{n-1}y_{n-2}\cdots y_mjhh\cdots h$, which are adjacent. $PS_{j\rightleftarrows k}(x) = y_{n-1}y_{n-2}\cdots y_mjk\cdots k$ and $PS_{k\rightleftarrows h}(w) = y_{n-1}y_{n-2}\cdots y_mkjj\cdots j$, which are adjacent. WLoG this is all possible cases, further position-swapping will similarly leave the still adjacent labels adjacent. Thus lemma $Y$ is true. ■

By lemma $Y$ and lemma $X$, further labeling-construction-iterations (of the multiples-of-$d+1$-codewords labeling) will not change the fact that label $\alpha_1\alpha_2\cdots\alpha_lkj_{(1)}j_{(2)}\cdots j_{(m)}$ is adjacent to $\alpha_1\alpha_2\cdots\alpha_l(d+k-j \bmod d+1)(d-j-1)_1(d-j-1)_2\cdots(d-j-1)_m$.

**The Finale.** Now that we know which pairs of labels are adjacent in the labeling (that differ from one another in more than just the right-most digit), we can see (from the 'Which words correct to any given codeword' section) that if $c = c_{n-1} \cdots c_{m+1} k_{(m)} j_{(m-1)} \cdots j_{(0)}$ is a codeword, then $\alpha = c_{n-1} \cdots c_{m+1} (d + k - j \bmod d + 1)_m (d - j - 1)_{m-1} \cdots (d - j - 1)_0$ (the word adjacent to it) is exactly that one more word which error-corrects to it. Q.E.D.

Thus the simple multiples-of-$d+1$-codewords code is defined for all $K_d^n$ ToH type iterated complete graphs and has a $d+1$ state recognizer and a $d+1$ state error-corrector. (Note that the binary labeling (see Bode 98) is the multiples-of-$d+1$-codewords code for $d = 2$.)

# 4 Conclusion

We have presented a construction which will produce a perfect one error correcting code on any iterated complete graph $K_d^n$. We have proven that this construction is unique up to isomorphic rotations of the graph. While the choice of codewords on any iterated complete graph is unique, certainly the labeling scheme is not. Three labeling methods have been presented, each with its own set of pros and cons. All three methods use iterative constructions to create the labelings. The first of these, the C-R-E-L method, makes codeword recognition and error correction easy with small finite state machines that are independent of $d$. However, coding and decoding for this method does not seem to be easy. The second, the $\alpha$-method, is a Gray Code on the graph with easy codeword recognition and error correction. The finite state machines for this method are all dependent on $d$, in fact, they are all size $d+1$. The third method, the "Mod $d+1$ Labeling" method provides extrememly easy coding and decoding with finite state machines of size $d+1$ for both codeword recognition and error correction. Thus, we have shown that labeling methods are certainly not unique, nor neccessarily dependent on $d$. Furthermore, it seems that different labeling methods have different positive and negative characteristics.

# References

[1] 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.

[2] 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.

[3] Cull, Paul and Ingrid Nelson. *Perfect Codes, NP-Completeness, and Towers of Hanoi Graphs.* Technical Report NSF-Grant DMS 93-00-281. Department of Computer Science, Oregon State University. Corvallis, Oregon. August, 1995.