5-1-2018

# Full and Dense Cyclic Gray codes in Mixed Radices

Devina Kumar
*Dartmouth College*

Dartmouth College Computer Science
Technical Report TR2018-851

# Full and Dense Cyclic Gray Codes in Mixed Radices

Devina Kumar

**Abstract**

The Gray code is a sequence of $n$ consecutive binary numbers arranged so that adjacent numbers in the sequence differ by 1 and that each number appears in the sequence exactly once. A Gray code is considered cyclic if the first and last numbers in the sequence differ in only one digit position, and those digits have a difference of exactly 1. In this thesis, numbers in a Gray-code sequence may be modular cyclic, meaning that in radix $r$, two consecutive numbers may vary in a digit with values 0 and $r-1$. This thesis focuses on methods to generate mixed-radix Gray codes. Mixed-radix representation refers to any $k$-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$. In a mixed-radix representation, each digit position can correspond to a different numerical base. In this thesis, we examine methods to generate two types of cyclic mixed-radix Gray codes: full and dense. Given $p_i = \prod_{j=0}^{i} r_j$ and $p_{-1} = 1$, a Gray code is considered full if $n = p_{k-1}$, and a Gray code is considered dense if $n$ is strictly less than $p_{k-1}$. Any full or dense Gray code sequence is a permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$, with each number appearing exactly once. We give a constant-time algorithm to generate each digit of each number in a cyclic mixed-radix full Gray code for any set of radices $r$. This thesis also provides a new case in which it is impossible to compute a cyclic mixed-radix dense Gray code. For all $r$ and $n$ for which we do not have proofs that a cyclic Gray code cannot exist and where $r_{k-1} \geq 3$ or $n$ is even, we show a linear-time method to find a cyclic mixed-radix dense Gray code. When $r_{k-1} = 2$ and $n$ is odd, we provide a conjecture regarding the possibility of a cyclic mixed-radix dense Gray code for a mixed-radix tuple $r$.

# 1 Introduction

The Gray code, named after its creator Frank Gray, is a sequence of $n$ consecutive binary numbers arranged so that adjacent numbers in the sequence differ in a single digit and that each number appears in the sequence exactly once; these adjacent numbers are said to have the Gray-code property. The Gray code allows only for values of $n$ that are powers of 2. Figure 1 shows the Gray code for $n = 4$ and $n = 8$. The numbers in the $n = 4$ sequence are represented with 2 bits while the numbers in the $n = 8$ sequence are represented with 3 bits. The number of bits in each number of the sequence, $\lg n$, corresponds to the number of bits needed to represent the largest number in the sequence, which is 3 in the $n = 4$ sequence and 7 in the $n = 8$ sequence. Observe that for both sequences, the first number of the sequence differs by 1 in only one digit from the last number of the sequence. Because the first and the last numbers of the sequences differ in just one bit, the sequence is *cyclic*.

Although the Gray code originally applies to binary numbers, the Gray-code property can be expanded beyond binary radices. In fact, the Gray-code property can be applied to sequences with any fixed-radix representation. For example, suppose we choose to represent numbers in quaternary radices—base 4. Observe that with this fixed-radix representation $(4, 4, 4, 4)$, 0302 and 0312 (which correspond to numbers 50 and 54, respectively) would still have the Gray-code property because the two numbers differ by 1 in a single digit. For the purposes of this thesis, we will say that numbers in a Gray-code sequence may be *modular cyclic*, meaning that in radix $r$, two consecutive numbers may vary in a digit with values 0 and $r - 1$. For example, in the case of a fixed-radix representation in base 4, 0000 and 0003 could be adjacent in a Gray sequence because 0 is modular cyclic with 3.

The Gray-code property can also be applied to mixed-radix representations. Mixed-radix representation refers to any $k$-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$. In a mixed-radix representation, each digit position can correspond to a different numerical base. For example, time is often written in mixed-radix notation. Time is made of hours, which are in base 24; minutes, which are in base 60; and seconds, which are also in base 60. Thus, for time, $r = (24, 60, 60)$. If we define $p_i = \prod_{j=0}^{i} r_j$ and $p_{-1} = 1$, the value of a number in radix notation $(x_{k-1}, x_{k-2}, \ldots, x_0)$ has the value $\sum_{i=0}^{k-1} x_i p_{i-1}$.

In this thesis, we will examine methods to generate two types of mixed-radix Gray codes: *full* and *dense*. A Gray code is considered full if $n = p_{k-1}$, and a Gray code is considered dense if $n$ is strictly less than $p_{k-1}$. Any full or dense Gray code sequence is a

| Gray code for $n = 4$ | Gray code for $n = 8$ |
|---|---|
| 00 | 000 |
| 01 | 001 |
| 11 | 011 |
| 10 | 010 |
| | 110 |
| | 111 |
| | 101 |
| | 100 |

**Figure 1:** Standard reflected binary Gray code for $n = 4$ and $n = 8$.

permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$, with each number appearing exactly once. Note that $n$ can also be written as a $k$-tuple of radices $n = (n_{k-1}, n_{k-2}, \ldots, n_0)$. For the purposes of this thesis, we require that $n_{k-1} = r_{k-1} - 1$.

We can conceptualize finding a cyclic Gray code as a graph problem. We refer to the numbers in the Gray-code sequence as *codewords*. Suppose that each possible codeword $0, 1, \ldots, n-1$ in the sequence is a vertex and that codewords are adjacent if and only if they can be consecutive in a Gray-code sequence. Thus, finding a cyclic Gray code is equivalent to finding a Hamiltonian cycle in the graph. We will use this idea and graph construction to find cyclic mixed-radix Gray codes.

In her 2017 honors thesis [2], Jessica Fan examined cyclic mixed-radix full Gray codes and cyclic mixed-radix dense Gray codes. Her research resulted in a recursive algorithm to generate a cyclic mixed-radix full Gray code for a $k$-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$. She also found two situations in which a cyclic mixed-radix dense Gray code could not occur. The first occurs when every radix in $r$ is even and $n$ is odd; the second occurs when $n = (1, 0, 0, \ldots, 0, 1)$. Fan's research resulted in two remaining questions, which we will address in this thesis.

1. Is there a method to compute each digit of each codeword in a cyclic mixed-radix full Gray code in constant time?

2. Under what conditions do cyclic mixed-radix dense Gray codes exist, and when they do, how can we find them?

This thesis contributes the following:

- A constant-time algorithm to generate each digit of each codeword in a cyclic mixed-radix full Gray code.

- A new case in which it is impossible to compute a cyclic mixed-radix dense Gray code.

- A linear-time method to find a cyclic mixed-radix dense Gray code for all $r$ and $n$ for which we do not have proofs that a cyclic Gray code cannot exist and where $r_{k-1} \geq 3$ or $n$ is even.

- A conjecture regarding the possibility of a cyclic mixed-radix dense Gray code for a mixed-radix tuple $r$, where $r_{k-1} = 2$ and $n$ is odd.

Part I of this thesis discusses cyclic mixed-radix full Gray codes and presents a formula to generate each digit of each codeword in the sequence in constant time. In this part of the thesis, we examine Er's method of constructing mixed-radix Gray codes and modify it so that it is able to find a cyclic mixed-radix full Gray code for any $r$.

Part II of this thesis examines cyclic mixed-radix dense Gray codes and how to find them. We first present a third case in which a cyclic mixed-radix dense Gray code cannot be found. We then present a linear-time method to find a cyclic mixed-radix dense Gray code when $r_{k-1} \geq 3$. The method involves dividing the codewords in the sequence into smaller groups, finding Hamiltonian cycles within these smaller groups, and then combining these smaller cycles into a larger cycle that includes all codewords in the sequence.

# Part I
# Cyclic Gray codes in mixed radices

We present a non-recursive method to generate cyclic mixed-radix full Gray codes for any $k$-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and integer $n = p_{k-1}$, where $p_i = \prod_{j=0}^{i} r_j$, so that the Gray code produced is a permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$. We can write codewords as $(a_{k-1}, a_{k-2}, ..., a_0)$, where $a_i$ represents the digit in the $i$th position of the codeword.

Section 2 discusses an existing non-recursive method for determining cyclic mixed-radix full Gray codes. Although this method computes a cyclic mixed-radix full Gray code, it restricts the values of the radix tuple; the method requires that at least one of the radices be even. In Section 3, we provide a non-recursive method to generate mixed-radix Gray codes for a radix tuple in which all of the radices are odd.

## 2 Existing method of computing cyclic mixed-radix full Gray codes for even numbers

Here, we will examine an existing method to generate cyclic mixed-radix full Gray codes: Er's reflected mixed Gray code. We know that Er's method generates a cyclic mixed-radix full Gray code if $r_{k-1}$ is even [2]. We will use this information, as well as Gray-code properties, to adopt Er's method for cases in which any of the radices in $r$ are even.

**Lemma 1** *Given a radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, if any radix in $r$ is even, there exists a cyclic full Gray code sequence for $r$.*

*Proof:* Consider a radix tuple $r$ such that $r_{k-1}$ is an even number. We are given that there exists a modular cyclic sequence for any $r$ that satisfies this condition. Now, consider a radix tuple $q$ with an even radix $q_m$, where $m \in \{k-1, k-2, \ldots, 0\}$. Suppose that we create a new radix tuple $q'$ such that $q'$ differs from $q$ in only two radices; in $q'$, the radices corresponding to $q_{k-1}$ and $q_m$ are switched, so that $q'_{k-1} = q_m$ and $q'_m = q_{k-1}$. Because $q'_{k-1}$ is even, we can find a modular cyclic Gray code for $q'$. After we find the modular cyclic Gray code for $q'$, we permute the digits such that for each codeword in the modular cyclic Gray code, we switch the digits in positions $m$ and $k-1$. Doing so does not alter the modular cyclic property of $q'$. We also note that switching the digits in positions $m$ and $k-1$ results in a radix tuple $(q'_m, q'_{k-2}, \ldots, q'_{k-1}, \ldots, q'_0)$. We can rewrite this tuple as $(q_{k-1}, q_{k-2}, \ldots, q_m, \ldots, q_0)$, which is our original radix tuple $q$. Thus, there exists a modular cyclic sequence for $q$. ∎

## 3 A non-recursive method for a cyclic mixed-radix full Gray code

Now that we have seen a way to compute cyclic mixed-radix full Gray codes for $k$-tuples of radices where at least one radix is even, we propose a way to compute a cyclic mixed-radix Gray code for the case in which all radices are odd. The method draws from Er's reflected mixed-radix Gray code, which produces a full Gray code for any radices [1]. Although

| | | | | |
|---|---|---|---|---|
| 000 | 142 | 200 | 342 | 400 |
| 001 | 141 | 201 | 341 | 401 |
| 002 | 140 | 202 | 340 | 402 |
| 012 | 130 | 212 | 330 | 412 |
| 011 | 131 | 211 | 331 | 411 |
| 010 | 132 | 210 | 332 | 410 |
| 020 | 122 | 220 | 322 | 420 |
| 021 | 121 | 221 | 321 | 421 |
| 022 | 120 | 222 | 320 | 422 |
| 032 | 110 | 232 | 310 | 432 |
| 031 | 111 | 231 | 311 | 431 |
| 030 | 112 | 230 | 312 | 430 |
| 040 | 102 | 240 | 302 | 440 |
| 041 | 101 | 241 | 301 | 441 |
| 042 | 100 | 242 | 300 | 442 |

**Figure 2:** Er's full Gray code sequence for $r = (5, 5, 3)$. Notice how each column alternates between ascending and descending sequences of numbers in digit positions $k-2, k-3, \ldots, 0$.

Er's reflected mixed-radix Gray code does not produce a modular cyclic sequence, we introduce a new process called *stitching* to arrange the numbers in the sequence so that they possess the modular cyclic property.

## 3.1 Setting up the algorithm

We set our sequence up using alternating ascending and descending patterns for each radix. Consider the digits 0 through $r_{k-1} - 1$. First, we write each digit $p_{k-2}$ times in ascending order, starting a new column each time the digit changes. For example, as we see in Figure 2, where $p_{k-2} = 15$, we write the digit 0 fifteen times, the digit 1 fifteen times, the digit 2 fifteen times, the digit 3 fifteen times, and the digit 4 fifteen times; each of these digits makes up its own column. Then, we move on to digit $r_{k-2}$. We write each digit $p_{k-3}$ times in ascending and then descending order, as shown in the figure, with $p_{k-3} = 3$. Generally, for $i = 1, 2, \ldots, k$, at radix $r_{k-i}$, we write each digit $p_{k-i-1}$ times until we switch to the next digit in the sequence; when we have written the maximum or minimum possible value of the digit $p_{k-i-1}$ times, we switch to descending digits if we were ascending, and to ascending digits if we were descending. We continue this pattern all the way through radix $r_0$. As we see in Figure 2, although this pattern creates a full Gray code when all radices are odd, it is not cyclic.

**Lemma 2** *Given a radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, if all radices in $r$ are odd, Er's method does not produce a full cyclic mixed-radix Gray code.*

*Proof:* When we set up the algorithm as described above, we have an odd number of columns with an odd number of items per column. This way of setting up the numbers will not produce a cyclic Gray code. Here's why. Let us refer to each column by the value of the digit in the $k-1$ position of the code word. The first number of each column varies, so that for the first code word in the sequence (top of column 0) to align with the last code word in the sequence (bottom of column $r_{k-1} - 1$), every digit except the first one would need to be the same. For $i = 2, 3, \ldots, k$, consider the digits at position $k - i$. The digits at position $k - i$ ascend through all of their possible values at the start of column 0 and then descend through all of their values at the end of column 1. This alternating ascending-descending pattern continues through all of the columns, and we

can generalize the pattern into a rule: for even-numbered columns $l = 0, 2, 4, \ldots, r_{k-1} - 1$, column $l$ contains an ascending sequence whose codewords start at $(l, 0, \ldots, 0)$ and end at $(l, r_{k-i} - 1, \ldots, r_0 - 1)$; for odd-numbered columns $l = 1, 3, \ldots, r_{k-1} - 2$, column $l$ contains a descending sequence whose codewords start at $(l, r_{k-i} - 1, \ldots, r_0 - 1)$ and end at $(l, 0, \ldots, 0)$.

If we take an even number of adjacent columns starting at column 0, then column 0 starts by ascending through the possible values of the digit at position $k - i$, and the last column ends by descending through the possible values at position $k - i$. Therefore, column 0 starts with 0 in the $k - i$ position, and the last column ends with 0 in the $k - i$ position. If we choose an odd number of adjacent columns, however, we always end with an ascending sequence (because we alternate between descending and ascending). Therefore, the column 0 would begin with 0 in the $k - i$ position, and the last column would end with $r_{k-i} - 1$ in the $k - i$ position. We know $0 \neq r_{k-i} - 1$ and we also know that the digit in the $r_{k-1}$ position of each column is different in each column; since more than one digit differs between the first value and the last value, this sequence of numbers cannot be cyclic. Given that we have an odd number of columns, we know that we will have two ascending columns book-ending our sequence. Thus, this sequence does not generate a full mixed-radix cyclic Gray code when all radices are odd. ∎

## 3.2 Description of the stitching method and its application

Fortunately, we can modify some of the columns of this sequence to generate a full mixed-radix cyclic Gray code when all radices are odd. We start by setting up the sequence in the same column structure as we just described. Because both the first and last columns are even-numbered, they are both ascending, which prohibits a cyclic Gray code. Observe, however, that we can pair codewords across columns so that they differ only in the leftmost digit. We refer to codewords that can be adjacent to each other but are in different columns as *friendly*. For example, in Figure 3, 000 and 100 are friendly. In turn, codewords that are adjacent to each other and are in the same column are *neighbors*. For example, in Figure 3, 000 and 001 are neighbors in column 0. We refer a contiguous subsequence of a possible Gray code as a *neighborhood*. If we examine the first two columns, we see that for $i = 0, \ldots, p_{k-2} - 1$, the codeword at index $i$ in column 0 is friendly with the codeword at index $p_{k-2} - i - 1$ in column 1. For reference, Figure 3 depicts some of these friendly relationships among codewords in columns 0 and 1 for $r = (5, 5, 3)$. Given this information about friendly relationships, we can create a new Gray-code sequence that is modular cyclic. First, we reverse every column but column 0, as we see in Figure 4. Observe that columns 2 through $r_{k-1} - 1$ preserve the Gray-code property within each column after the reversal.

To generate a cyclic Gray-code sequence, we will reorder the Gray-code sequence through the 2 leftmost columns to start at $0, \ldots, 0)$ and end at $(1, r_{k-2} - 1, \ldots, r_0 - 1)$. Then, we can just go through the remaining columns in column-major order to complete the rest of the sequence. We apply a method called *stitching* to column 0 and column 1 to generate a Gray code sequence that is cyclic with the remaining columns. To stitch our new sequence, we follow a simple pattern: friendly, neighbor, friendly, neighbor, repeat. For example, using the radices $r = (5, 5, 3)$, as shown in Figure 5, we start at the first number 000 in column 0. Then, we cross over to its friendly counterpart 100 in column 1. From there, we go to 101, the neighbor of 100 in column 1. Then, we move back across to column 0 to 001, the friendly counterpart of 101. We can say that the sequence

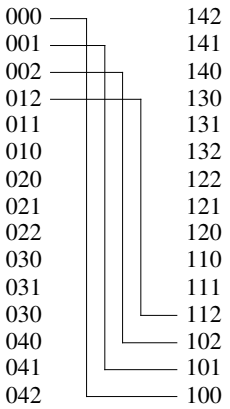


**Figure 3:** The first four friendly pairs in columns 0 and 1 of $r = (5, 5, 3)$. Each of the codewords differs from its friendly counterpart in just one digit.

$000, 100, 101, 001$ forms a neighborhood. Following the same stitching pattern, the next neighborhood would be $002, 102, 112, 012$, followed by $011, 111, 110, 010$, and so on. We repeat the stitching process until we have put almost all of the code words into their new four-codeword neighborhoods. All of the neighborhoods start and end with codewords from the same column, so that the codewords that form the boundaries between adjacent neighborhoods differ in one digit. Thus, piecing together the neighborhoods in this fashion will preserve the Gray-code property.

After we have formed as many four-codeword neighborhoods as possible, we are left with one codeword in each column, so we end our stitched sequence of columns 0 and 1 with the last codeword in column 1, which is $(1, r_{k-2} - 1, \ldots, r_0 - 1)$. Because we reversed the columns before stitching, column 2 is descending and the first codeword of column 2 is $(2, r_{k-2} - 1, \ldots, r_0 - 1)$. Therefore, our stitching of columns 0 and 1 preserves the Gray-code property through the columns. In addition, note that after the reversal, column $r_{k-1} - 1$ is descending and ends in $(r_{k-1} - 1, 0, \ldots, 0)$. Since the first number of our sequence is $(0, 0, \ldots, 0)$, the first and last numbers of our sequence differ in only one digit and have the modular property. Therefore, as Figure 6 shows, our new sequence is a full mixed-radix modular cyclic Gray code.

## 3.3 The full mixed-radix cyclic Gray code

The method to generate the modular cyclic mixed-radix full Gray code given an index $x = 0, 1, \ldots, n - 1$ in the stitched Gray-code sequence and a digit position $i = 0, 1, \ldots, k - 1$ is a two-step process that can be accomplished in constant time. First, we define a function $f(x)$ that determines the index corresponding to $x$ in the full mixed-radix sequence generated by Er's Gray code (shown in Figure 2). For example, for $r = (5, 5, 3)$, we have $f(1) = 29$, $f(2) = 28$, and $f(3) = 1$. The codeword at index 1 in the stitched Gray-code sequence is the codeword at index 29 in Er's Gray-code sequence; the codeword at index 2 in the stitched Gray-code sequence is the codeword at index 28 in Er's Gray-code sequence; and the codeword at index 3 in the stitched Gray-code sequence is the codeword at index 1 in Er's Gray-code sequence. Generally, we map the indices of the codewords in Figure 6 back to the index from which they originated in Figure 2. Then, letting $x' = f(x)$, we compute the $i$th digit of the codeword at index $x'$.

| | | | | |
|---|---|---|---|---|
| 000 | 100 | 242 | 300 | 442 |
| 001 | 101 | 241 | 301 | 441 |
| 002 | 102 | 240 | 302 | 440 |
| 012 | 112 | 230 | 312 | 430 |
| 011 | 111 | 231 | 311 | 431 |
| 010 | 110 | 232 | 310 | 432 |
| 020 | 120 | 222 | 320 | 422 |
| 021 | 121 | 221 | 321 | 421 |
| 022 | 122 | 220 | 322 | 420 |
| 032 | 132 | 210 | 332 | 410 |
| 031 | 131 | 211 | 331 | 411 |
| 030 | 130 | 212 | 330 | 412 |
| 040 | 140 | 202 | 340 | 402 |
| 041 | 141 | 201 | 341 | 401 |
| 042 | 142 | 200 | 342 | 400 |

**Figure 4:** Er's method for $r = (5, 5, 3)$ with columns 1 through 4 reversed. Observe that even after reversing the columns, the Gray code property is still preserved within columns 1 through 4.

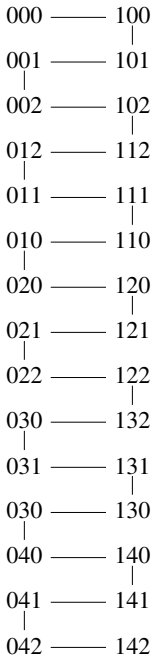


**Figure 5:** Stitching the friendly pairs in columns 0 and 1 for $r = (5, 5, 3)$. Note that this is after column 1 has been reversed. To stitch the sequence, follow the lines across and between the friendly codewords in each column. For example, the first four codewords of the stitched sequence are 000, 100, 101, 001.

| | | | | |
|---|---|---|---|---|
| 000 | 021 | 242 | 300 | 442 |
| 100 | 022 | 241 | 301 | 441 |
| 101 | 122 | 240 | 302 | 440 |
| 001 | 132 | 230 | 312 | 430 |
| 002 | 032 | 231 | 311 | 431 |
| 102 | 031 | 232 | 310 | 432 |
| 112 | 131 | 222 | 320 | 422 |
| 012 | 130 | 221 | 321 | 421 |
| 011 | 030 | 220 | 322 | 420 |
| 111 | 040 | 210 | 332 | 410 |
| 110 | 140 | 211 | 331 | 411 |
| 010 | 141 | 212 | 330 | 412 |
| 020 | 041 | 202 | 340 | 402 |
| 120 | 042 | 201 | 341 | 401 |
| 121 | 142 | 200 | 342 | 400 |

**Figure 6:** The completed stitched sequence for $r = (5, 5, 3)$. This sequence forms a modular cyclic mixed-radix full Gray code.

The function $f(x)$ is as follows:

$$f(x) = \begin{cases} p_{k-2}\left(1 + 2\left\lfloor \dfrac{x}{p_{k-2}} \right\rfloor\right) - x - 1 & \text{if } \left\lfloor \dfrac{x}{p_{k-2}} \right\rfloor \geq 2 \ , \\ \left\lfloor \dfrac{x}{2} \right\rfloor & \text{if } \left\lfloor \dfrac{x}{p_{k-2}} \right\rfloor < 2 \text{ and } x \bmod 4 \in \{0, 3\} \ , \\ 2p_{k-2} - 1 - \left\lfloor \dfrac{x}{2} \right\rfloor & \text{if } \left\lfloor \dfrac{x}{p_{k-2}} \right\rfloor < 2 \text{ and } x \bmod 4 \in \{1, 2\} \ . \end{cases}$$

The first case applies to columns 2 and greater. The formula $f(x)$ in this case is a simplified form of

$$(p_{k-2} - 1) - \left(x - \left\lfloor \frac{x}{p_{k-2}} \right\rfloor p_{k-2}\right) + \left\lfloor \frac{x}{p_{k-2}} \right\rfloor p_{k-2} \ . \tag{1}$$

Here's how to understand equation (1). The term $\lfloor x/p_{k-2} \rfloor$ gives the column number for $x$, and $p_{k-2}\lfloor x/p_{k-2} \rfloor$ gives the number of items preceding $x$'s column. Observe that the term $(x - \lfloor x/p_{k-2} \rfloor\, p_{k-2})$ subtracts the column offset from $x$, thus finding the index of $x$ within its respective column. Subtracting this quantity from $p_{k-2} - 1$ computes the reflected index of $x$ within its column. Finally, to adjust for the appropriate column, add $\lfloor x/p_{k-2} \rfloor\, p_{k-2}$.

The second and third cases apply to columns 0 and 1. Both of these cases rely on the idea that the stitching method returns to the same column every four codewords. To find $x' = f(x)$, we simply find the place that $x$ would occupy in this sequence of four. If $x$ is first or last in the sequence of four, then $x'$ comes from column 0, and if $x$ is second or third, then $x'$ comes from column 1. Then, we compute the index of $x'$ in its respective column.

Once we have completed the first step of finding $x' = f(x)$, we move on to the next step: finding the number in digit position $i$. We create a new function $g(x'_i, i)$, where $x'_i$ is the $i$th digit of the codeword at index $x'$ in the mixed-radix representation:

$$x'_i = \left\lfloor \frac{x'}{p_{i-1}} \right\rfloor \bmod r_i \ . \tag{2}$$

The $i$th digit of the $x$th codeword in the stitched Gray code sequence will be given by $g(f(x), i)$. We define $g(x', i)$ as follows:

$$g(x', i) = \begin{cases} \left\lfloor \dfrac{x'}{p_{k-2}} \right\rfloor & \text{if } i = k-1 \ , \\ x'_i & \text{if } i < k-1 \text{ and } \left\lfloor \dfrac{x' - \lfloor x'/p_{k-2} \rfloor \, p_{k-2}}{p_i} \right\rfloor \bmod 2 = \left\lfloor \dfrac{x'}{p_{k-2}} \right\rfloor \bmod 2 \ , \\ r_i - 1 - x_i & \text{if } i < k-1 \text{ and } \left\lfloor \dfrac{x' - \lfloor x'/p_{k-2} \rfloor \, p_{k-2}}{p_i} \right\rfloor \bmod 2 \neq \left\lfloor \dfrac{x'}{p_{k-2}} \right\rfloor \bmod 2 \ . \end{cases}$$

The first case applies when digit position $i$ is the leftmost digit of the codeword. Since the first digit corresponds to the column number, we simply find the column number of $x'$. The second and third cases apply to digit positions $k-2, k-3, \ldots, 0$. Recall that the column set-up of Er's original Gray code designated columns $0, 2, \ldots, r_{k-1}-1$ as ascending and columns $1, 3, \ldots, r_{k-1}-2$ as descending. Thus, in the second and third cases, the term $\lfloor x'/p_{k-2} \rfloor \bmod 2$ denotes whether the column is ascending or descending. Also recall that within the columns, the digits at each digit position alternate between ascending and descending sequences. The term $\lfloor (x' - \lfloor x'/p_{k-2} \rfloor \, p_{k-2})/p_i \rfloor \bmod 2$ denotes whether the digit at position $i$ is part of an ascending or descending sequence. If $x'_i$ is part of an ascending column with an ascending sequence for digit position $i$ or a descending column with a descending sequence for digit position $i$, then $g(x', i)$ is the same as the mixed-radix representation of $x'_i$ in equation (2). If the column and sequence of digit position $i$ differ for $x'_i$, then we simply take the reflection of the digit. This formula gives us alternating ascending and descending sequences for each digit position $i$, as dictated by Er's method. Thus, computing $g(x', i)$ gives us the $i$th digit of the $x'$th codeword in Er's Gray code sequence, which in turn gives us the $i$th digit of the $x$th codeword in the stitched Gray code sequence. Therefore, $g(f(x), i)$ gives us a way to compute each digit of each number of the stitched Gray code sequence in constant time.

# Part II
# Dense Cyclic Gray codes in mixed radices

In this part of the thesis, we present methods to determine whether a cyclic mixed-radix dense Gray code exists for any $k$-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and any positive integer $n < p_{k-1}$, where $p_i = \prod_{j=0}^{i} r_j$. We also provide methods to find a cyclic mixed-radix dense Gray code when one exists.

We are aware of two prior cases in which cyclic mixed-radix dense Gray codes cannot be found [2]. The first case applies when $n$ has the digit representation $n = (1, 0, 0, \ldots, 0, 1)$. The second case applies when every radix in $r$ is even and $n$ is odd. Section 4 introduces a new case for a particular set of radices and a positive integer $n$ for which a cyclic mixed-radix Gray code cannot be found. For any set of radices and any positive integer $n$ that does not fall under these three cases, we can find a cyclic mixed-radix dense Gray code. In Section 5, we provide a method to create a cyclic mixed-radix

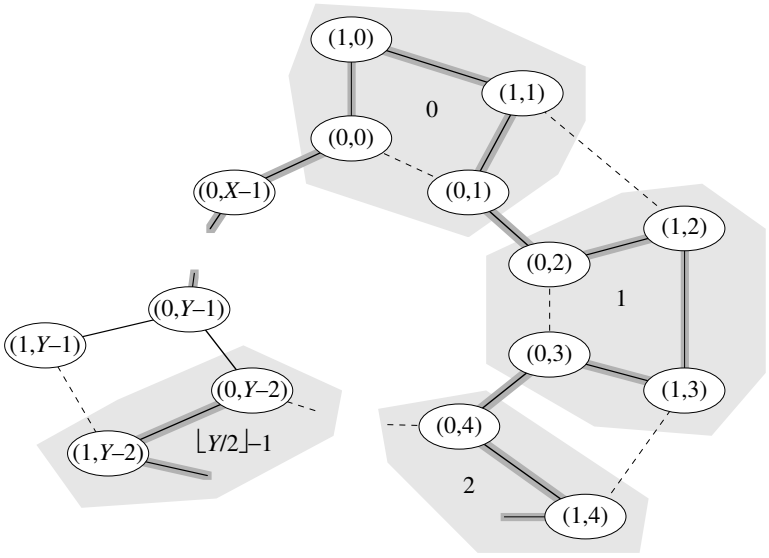


**Figure 7:** A graph of $n$ codewords we wish to arrange into a Gray-code sequence. The edges between the codewords indicate that the two numbers may be adjacent to each other in a Gray-code sequence. The shaded edges represent mandatory connections in the cycle. The dotted edges represent edges that cannot be in the cycle because they are incident on vertices that already have two incident edges in the cycle. The shaded portions consisting of four codewords represent a way to divide the codewords into groups that show a repeating pattern among edges included in the cycle. The numbers within the shaded portions correspond to the groups to which we are referring. For example, the shaded portion with the 0 in its center is group 0.

dense Gray code for any $r$ and $n$ where $r_{k-1} \geq 3$. Although we do not have an exact method to determine a cyclic mixed-radix dense Gray code when $r_{k-1} = 2$ and $n$ is odd, we conjecture that a cyclic mixed-radix dense Gray code exists when $r_{k-1} = 2$, provided that $r$ and $n$ do not fall under the cases for which we have shown a cyclic mixed-radix dense Gray code cannot exist.

# 4 A new case in which cyclic mixed-radix dense Gray codes cannot be found

**Lemma 3** *Consider a radix tuple $r = (2, X)$, where $X \geq 2$. If $n = (1, Y)$, where $Y < X$ and $Y$ is odd, a cyclic mixed-radix dense Gray code does not exist.*

*Proof:* Consider a graph of the $n$ codewords we wish to put into a cyclic Gray-code sequence. Finding a cyclic Gray code is equivalent to finding a Hamiltonian cycle within the graph. We set up the graph as shown in Figure 7: the numbers with the radix representation $(0, i)$ for $i = 0, \ldots, X - 1$ are arranged in a ring, and the codewords with the radix representation $(1, j)$ for $j = 0, \ldots, Y - 1$ are arranged in a ring outside of the codewords in the $(0, i)$ ring. In Figure 7, if two codewords can be adjacent in the Gray-code sequence, there is an edge between them. For any codeword with degree 2, its incident edges must be in the Hamiltonian cycle. These mandatory edges are indicated by the shaded edges.

Note that all of the codewords in the inner ring that cannot be connected to any codewords in the outer ring have two edges; these edges must be in the cycle. In addition, the first codeword of the outer ring has only two edges; these edges must also be in the cycle.

The remaining codewords have edges to codewords in both rings. After selecting the mandatory edges to put into the cycle (the edges between codewords in the inner ring that are not connected to any numbers in the outer ring and the edges from the first codeword of the outer ring), we are left with codewords from the inner ring that are connected to codewords in both rings. Observe that the edges $(0,0)$ — $(1,0)$ and $(0,0)$ — $(0,X-1)$ must be in the cycle, so that edge $(0,0)$—$(0,1)$ must not be in the cycle. Thus, $(0,1)$ is left with only two possible edges: one to $(1,1)$ and one to $(0,2)$. Both of these edges must be in the cycle. After these edges are added to the cycle, two edges in the cycle are incident on $(1,1)$, so the edge $(1,1)$ — $(1,2)$ cannot be in the cycle. Since the edge $(1,1)$ — $(1,2)$ cannot be in the cycle, $(1,2)$ is left with two edges: one to $(0,2)$ and one to $(1,3)$. Adding in these edges means that $(0,2)$ has two edges remaining, and thus the edge between $(0,2)$ and $(0,3)$ cannot be in the cycle. If the edge between $(0,2)$ and $(0,3)$ cannot be in the cycle, then $(0,3)$ is left with only two edges, both of which must be in the cycle. This pattern continues for the rest of the cycle.

Suppose we divide the codewords with edges connecting to codewords in both the inner and outer rings into groups of four—two codewords from the outer ring and two codewords from the inner ring. In Figure 7, these groups are indicated by shading around four codewords; groups 0 and 1 are labeled. Note that each group follows the same pattern of edges that must be included in the cycle. We can apply the logic used to determine the included edges in groups 0 and 1 to other groups of four codewords. Each group $i = 0, 1, \ldots, \lfloor Y/2 \rfloor - 1$, follows the same pattern regarding which of its edges must be in the cycle and which edges must not be in the cycle:

| Edges in cycle | Edges not in cycle |
|---|---|
| $(0,2i)$ — $(1,2i)$ | $(0,2i)$ — $(0,2i+1)$ |
| $(1,2i)$ — $(1,2i+1)$ | $(1,2i+1)$ — $(1,2i+2)$ |
| $(1,2i+1)$ — $(0,2i+1)$ | |
| $(0,2i+1)$ — $(0,2i+2)$ | |

Since $Y$ is odd, we know that codewords $(1,Y-1)$ and $(0,Y-1)$ are in a different group from codewords $(1,Y-2)$ and $(0,Y-2)$. Because the $(0,2i+1)$ — $(0,2i+2)$ edge is mandatory, there must be an edge between $(0,Y-2)$ and $(0,Y-1)$. As shown in Figure 7, however, this edge cannot be in the cycle: otherwise $(0,Y-1)$ would have three incident edges in the cycle. Since we must have the $(0,2i+1)$ — $(0,2i+2)$ edge to make a cyclic Gray code sequence, but $(0,Y-2)$ and $(0,Y-1)$ cannot have an edge between them, we cannot make a cyclic mixed-radix dense Gray code. ∎

When $Y$ is even, a cyclic mixed-radix dense Gray code exists. Figure 8 shows the cycle when $X = 7$ and $Y = 6$. Observe the same pattern of edges among groups of four codewords. Only the mandatory edges are in the cycle, thus forming a cyclic mixed-radix dense Gray code.

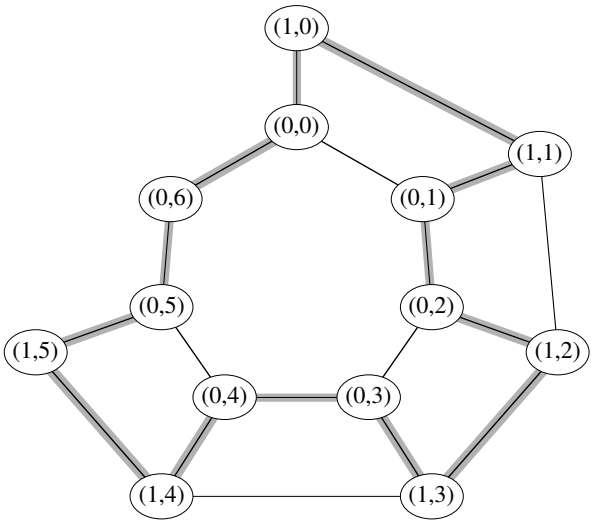


**Figure 8:** Graph when $X = 7$ and $Y = 6$.

# 5 A method to create cyclic mixed-radix dense Gray codes

Now that we have examined the cases in which a cyclic mixed-radix dense Gray code cannot be found, we propose a way to find a cyclic mixed-radix dense Gray code for the remaining combinations of $r$ and $n$ where $r_{k-1} \geq 3$. The method involves arranging the numbers into columns, dividing the entire columnar structure into subunits called *blocks*, which consist of $2r_j$ rows, where $j$ is the position at which the digit varies with the most frequency in the columnar structure. Then, we find a modular cyclic sequence for each block and recombine these intercolumn blocks to create the overall modular cyclic sequence.

## 5.1 Setting up the algorithm

We set up our sequence for the cyclic mixed-radix dense Gray code slightly differently from how we did for the cyclic mixed-radix full Gray code. Although the columnar structure stays the same, the frequency with which we vary the radices is slightly different. In the algorithm for the cyclic mixed-radix full Gray code, we varied the digit at $r_{k-1}$ the slowest, the digit at $r_{k-2}$ the second slowest, and so on, until we reached the digit at $r_0$, which we varied the fastest. If every radix in $r$ is even or if $r_{k-1}$ is odd, then we follow the same pattern of variation that we used for the cyclic mixed-radix full Gray code—vary the digit at $r_{k-1}$ the slowest and the digit at $r_0$ the fastest. If $r$ does not satisfy these requirements, then we have a slightly different method of varying the digits at each radix position. The radix position whose digit we vary fastest is the most significant odd radix from $r_{k-2}, \ldots, r_0$. For example, suppose $r = (4, 8, 8, 5, 6, 3)$. The most significant odd radix is $r_{k-4} = 5$, which is the digit we will vary the fastest. Figure 9 shows the first $2r_{k-4}$ rows of the columnar structure for $r = (4, 8, 8, 5, 6, 3)$. Observe that the digit at position $k-4$, which corresponds to the position of the most significant odd radix, varies fastest.

Now that we have our basic columnar structure, we do two things do it. First, we

| | | | |
|---|---|---|---|
| 100000 | 200000 | 300000 | 000000 |
| 100100 | 200100 | 300100 | 000100 |
| 100200 | 200200 | 300200 | 000200 |
| 100300 | 200300 | 300300 | 000300 |
| 100400 | 200400 | 300400 | 000400 |
| 100401 | 200401 | 300401 | 000401 |
| 100301 | 200301 | 300301 | 000301 |
| 100201 | 200201 | 300201 | 000201 |
| 100101 | 200101 | 300101 | 000101 |
| 100001 | 200001 | 300001 | 000001 |

**Figure 9:** The first block of the columnar structure for $r = (4, 8, 8, 5, 6, 3)$. The column with 0 as the most significant digit is the rightmost column, and the digit at position $k - 4$ is the one that varies fastest.

shift the very first column to the very right; it is now visually the last column. As we can see in Figure 9, visually, the first column is now the column with all the codewords whose first digit is 1, the second-to-last column the one with codewords whose first digit is $r_{k-1} - 1$, and the last column the one with codewords whose first digit is 0 (the column we just shifted). Second, now that we have our new columnar structure, we split it into smaller sections called *blocks* and *half-blocks*. Consider the value $r_j$ of the radix at the digit position that we vary with the most frequency in our column. We create a half-block by taking the first $r_j$ numbers in each column. Note that these numbers are ascending in the most frequently varied digit. We refer to these ascending half-blocks as *ascending* halves. For example, in Figure 9, the first five rows form an ascending half. We can create a descending half-block by taking the next $r_j$ numbers in each column; we refer to these descending half-blocks as *descending* halves. In Figure 9, the second five rows form a descending half. To create a block, we combine an ascending half with the descending half that follows. Figure 9 shows the first block for $r = (4, 8, 8, 5, 6, 3)$. For our columnar structure, we create the first block by taking the first $2r_j$ numbers in each column. We create the second block by taking the next $2r_j$ numbers in each column. We continue in this manner until we created as many blocks of size $2r_j$ as possible. If there are remaining numbers, they will form an ascending half-block of height $r_j$.

**Lemma 4** *Given a radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, suppose that $j$ is the position at which the digit varies with the most frequency in the columnar structure. Then, after dividing the columns into blocks, we will have at most one half-block of height $r_j$ left.*

*Proof:* Each column has a height $p_{k-2} = \prod_{i=0}^{k-2} r_i$. Now consider dividing the columns into groups with height $r_j$. Then, we would have $b = p_{k-2}/r_j$ groups of height $r_j$. We know that $b$ is a positive integer because $r_j$ is a factor of $p_{k-2}$. Now, suppose we want to combine these groups of height $r_j$ into groups of height $2r_j$. If $b$ is even, then we know $b/2$ is a positive integer and we must have $b/2$ groups of height $2r_j$. If $b$ is odd, then we know $b - 1$ must be even, which means we have $(b - 1)/2$ groups of height $2r_j$ and one remaining group of height $r_j$. This group of height $r_j$ would be an ascending half because it would follow a complete block ending in a descending half. Thus, no matter the radices, we will always have at most one ascending half of height $r_j$ left. ∎

Now that we have divided our structure into blocks, let us knock out the codewords in each block that correspond to numbers whose value is greater than $n$. We notice that the knocked out groups in each block are made up of one or more consecutive codewords.

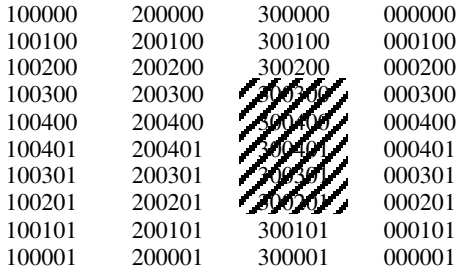


**Figure 10:** The first block of the columnar structure for $r = (4, 8, 8, 5, 6, 3)$. The shaded portion in the third column represents the knockout group when the largest number in the Gray-code sequence is 300200.

We refer to these groups of consecutively knocked out codewords as *knockout groups*. For example, in Figure 10, we see a block for $r = (4, 8, 8, 5, 6, 3)$. Suppose that 300200 is the largest codeword allowed in our Gray-code sequence. Then, all of the numbers larger than this codeword must be knocked out of the sequence. In Figure 10, we see a knockout group clustered at the center of a block.

**Lemma 5** *There is no block or half-block with more than one knockout group.*

*Proof:* We also know that the blocks comprise an ascending half followed by a descending half. Note that in the ascending half, the codewords are increasing in value. Now, suppose one of the codewords $c$ in the ascending half is the smallest codeword in the ascending sequence that can be knocked out. Then, all of the numbers after $c$ in the ascending half of the block would have to be knocked out because they are necessarily greater than $c$; the numbers that come before $c$ in the ascending half cannot be knocked out because they are necessarily less than $c$. Thus, we see than an ascending half cannot have more than one group of consecutively knocked out codewords and that this group of knocked out codewords always starts from the bottom of the ascending half and extends upward.

Let us now examine the descending half of the block. We note that in the descending half, the codewords are decreasing in value. Using logic similar to what we used to analyze the ascending half, we can say that a descending half also has at most one group of consecutively knocked out codewords. Because the descending half is decreasing in value, however, this group of knocked out codewords always starts from the top of the descending half and extends downward.

Now, consider a block, which comprises an ascending half followed by a descending half. We know that the knockout group for the ascending half starts at the bottom of the half and extends upward, while the knockout group for the descending half starts at the top of the half and extends downward. Combining the knockout groups from each half, we observe that the knockout group for the overall block starts from the center of the block and extends either upward, downward, or in both directions. Since combining the knockout groups of the halves yields one knockout group in the middle of the block and since we know each of the halves cannot have more than one knockout group, the knockout group extending out from middle of the block is the only one that can exist. Thus, each block has at most one knockout group. ∎

## 5.2 Finding a cyclic mixed-radix dense Gray code

We now propose a method to connect the blocks and half-blocks to create a cyclic mixed-radix dense Gray code when $r_{k-1} \geq 3$. The method involves finding cyclic Gray codes within each block or half-block and then connecting these blocks and half-blocks together to form a larger cyclic Gray code. After connecting all of the blocks and half-blocks, we end up with a cyclic mixed-radix dense Gray code for the values of $r$ and $n$ with which we started.

We begin by examining the blocks. Given that each block has at most one knockout group, we can think of the block as having three sections: the rows above the knockout group, the rows containing the knockout group, and the rows below the knockout group. We note that some of these sections might be empty if the knockout group is present in either all the rows of the block or none of them. Consider the parity of each of the sections: it is either odd or even. Thus, there are eight possible combinations of these section parities. However, since the height of a block is an even number, we cannot have certain combinations of section parities: all odd sections or one odd section and two even sections. For the remaining combinations of parities, we can create cyclic Gray codes out of the codewords in the block. The methods of creating these intrablock cyclic Gray codes vary according to parities of the sections as well as the parity of the number of columns.

Given that we cannot have sections with all odd parities or two even parities and one odd parity, we propose methods of creating cyclic Gray codes for blocks with all remaining combinations of section parities. Going forward, we will denote these parity combinations with "E" representing even parity for a section and "O" representing odd parity for a section. The order of the parity combination is the following: the parity of the section with rows above the knockout group, then the parity of the section with rows containing the knockout group, and then the parity of the section with rows below the knockout group. For example, OEO denotes a block with an odd number of rows above the rows containing the knockout group, followed by an even number of rows that contain the knockout group, followed by an odd number of rows below the knockout group. Blocks with combinations OOO, EEO, EOE, and OEE are not possible. Thus, the possible combinations for the blocks are the following: EOO, OEO, OOE, EEE, all rows containing knocked out numbers, and no rows containing knocked out numbers. Note that the last two are special cases of EEE.

Let us start with EOO. Figure 11 shows how to create an intrablock cycle for EOO if we have an odd number of columns. The knockout group is represented by the diagonally-lined section. The solid and dotted lines indicate the cycle connecting the codewords in the block. The solid lines represent connections between the blocks, while the dotted lines represent the jump taken from one codeword to another. Figure 12 has three columns and thus falls under the category of Figure 11. Note that in Figure 12, however, instead of weaving up and down the rows in the columns left of the column with the knockout group, we simply progress straight up the column to the left of the column with the knockout group. All of the diagrams showing how to create intrablock cycles with an odd number of columns can be adapted for use with three columns simply by going straight up or down a column instead of weaving up and down among columns. Figure 13 shows how to create an intrablock cycle for EOO if we have an even number of columns.

The diagrams for the remaining parities also show how to create intrablock cycles given an odd number of columns and an even number of columns. Figures 14 and 15

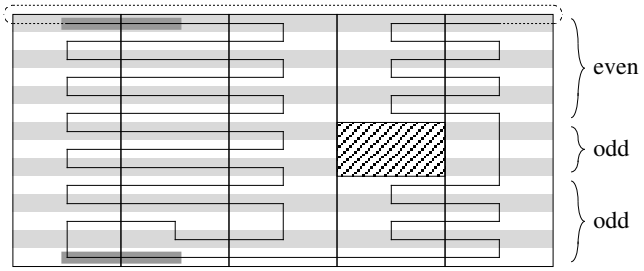


**Figure 11:** Method to create an intrablock cycle for EOO with an odd number of columns. Each of the alternating shaded/unshaded bands in the background of the arrows corresponds to a row. The block in this diagram has 14 rows. The section with the diagonal lines in the fourth column represents the knocked out numbers. The cycle is depicted with the solid and dotted lines. The solid lines represent connections between adjacent codewords. The dotted lines do not indicate connections but rather where the cycle jumps between columns or within columns. Notice that the cycle includes two shaded connections in the cycle: one in the upper left corner and one in the lower left corner. These shaded connections are the ones we use to connect this block to the blocks before and after it.

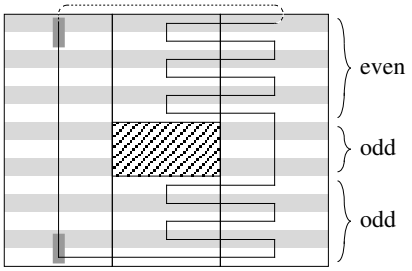


**Figure 12:** Method to create an intrablock cycle for EOO with three columns.

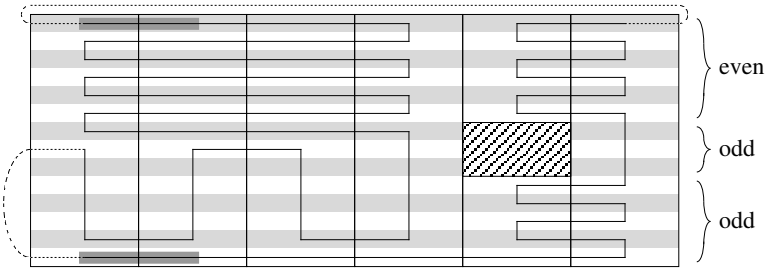


**Figure 13:** Method to create an intrablock cycle for EOO with an even number of columns. The dotted line represents a jump between the bottom of the descending half and the top of the descending half. The cycle progresses in a ladder-like motion, going through all the codewords of the column in the descending block then moving to the next column and repeating the process until all of the codewords in the descending half have been included in the cycle.

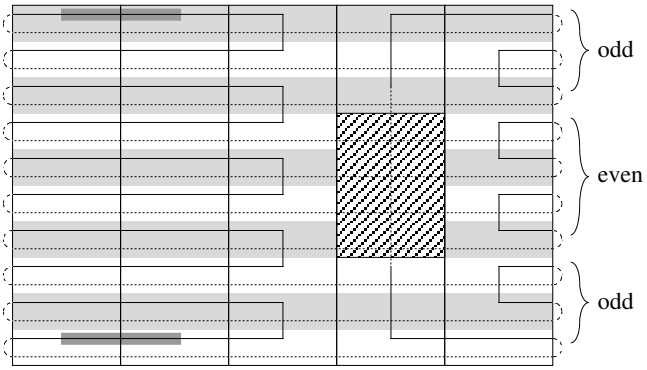


**Figure 14:** Method to create an intrablock cycle for OEO with an odd number of columns.

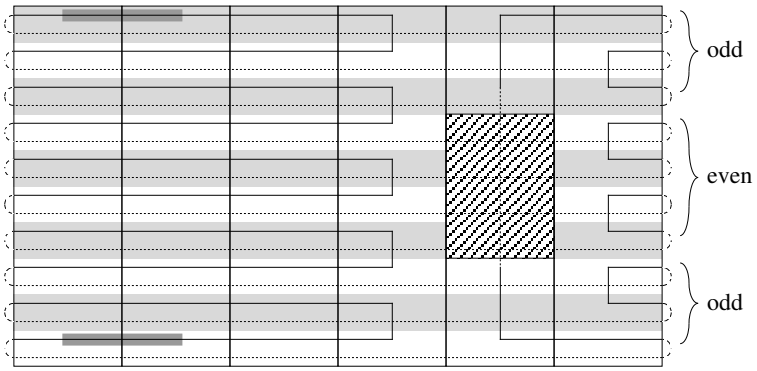


**Figure 15:** Method to create an intrablock cycle for OEO with an even number of columns.

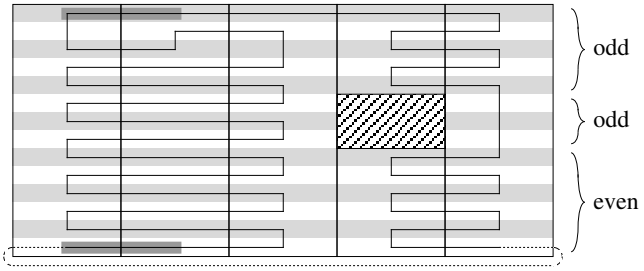


**Figure 16:** Method to create an intrablock cycle for OOE with an odd number of columns.

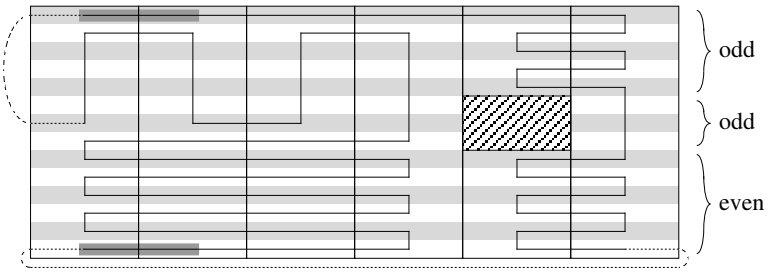


**Figure 17:** Method to create an intrablock cycle for OOE with an even number of columns.

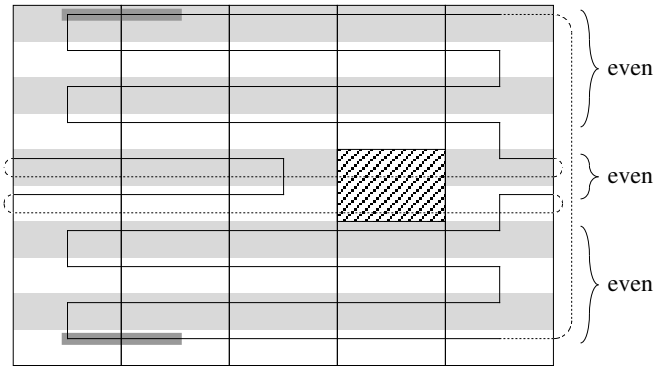


**Figure 18:** Method to create an intrablock cycle for EEE with an odd number of columns.

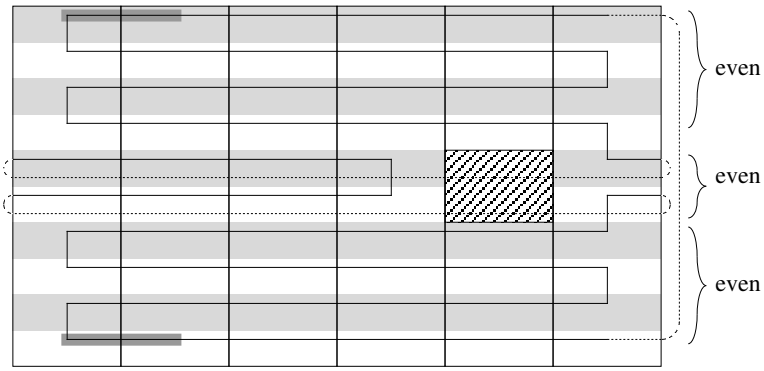


**Figure 19:** Method to create an intrablock cycle for EEE with an even number of columns.

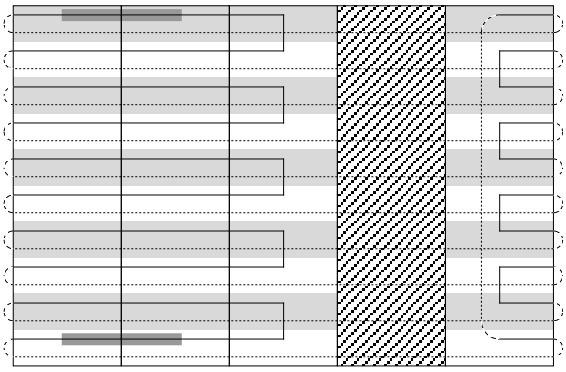

**Figure 20:** Method to create an intrablock cycle for an odd number of columns with every row containing a knocked out number.

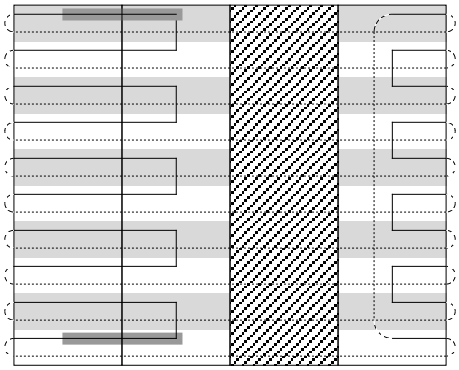

**Figure 21:** Method to create an intrablock cycle for an even number of columns with every row containing a knocked out number.

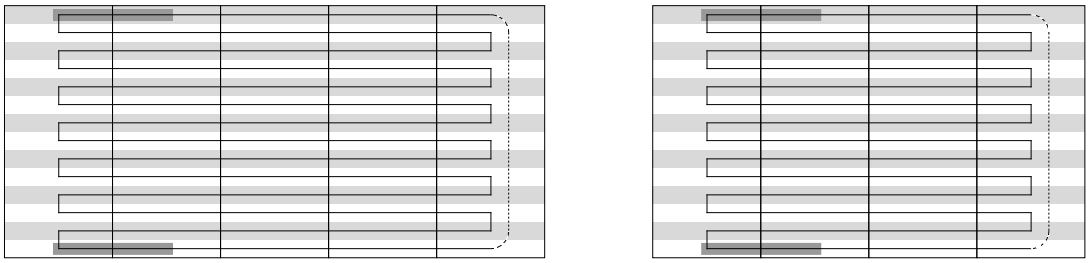

**Figure 22:** Method to create an intrablock cycle for an odd number of columns (left) and even number of columns (right) given that no numbers are knocked out in the block.

show how to create intrablock cycles for parity combination OEO. Figures 16 and 17 show how to create intrablock cycles for parity combination OOE; note that these cycles are reflections of the cycles shown in EOO. Figures 18 and 19 show the methods to create a cycle within an EEE block. The method to create a cycle for EEE blocks relies exclusively on cross-column stitching. Cross-column stitching can also be applied to create cycles in blocks with every row containing a knocked out number, as well as blocks with no knocked out numbers. The former is shown in Figures 20 and 21; the latter is shown in Figure 22.

Observe that for every block, no matter the parities, there exists the same type of link in the upper left corner. For blocks with four or more columns, this link is a stitch between the two leftmost columns; for blocks with three columns, this link is a stitch between the top two codewords of the leftmost column. We observe the same pattern in the bottom left corner of the block. These links are highlighted in gray in all of the diagrams. We note that between two blocks, there exist two gray highlighted links; one link at the bottom left corner of one block and one link at the top left corner of one block. Observe that the codewords of one link differ in one digit from the codewords in the other link. Thus, if we break the links and connect the codeword of each block to its corresponding codeword in the other block, we will have connected the two intrablock cycles to create a cycle among the two blocks. Figure 23 shows the process of breaking these upper and lower left links in an EEE block and an EOE block to create a cycle that spans both blocks. The process shown in Figure 23 translates to links between blocks other than EEE and EOE blocks. Since we are guaranteed the presence of the highlighted links in our blocks, we can simply keep connecting the blocks by breaking the links highlighted in gray and connecting them to links in the preceding or following block, eventually forming a cyclic Gray code among all of the blocks.

What about half-blocks? We know that after dividing the columnar structure into blocks, we will have at most one half-block left at the end of the structure. We have shown how to combine all of the blocks such that they form a cyclic Gray code. Now, we show how to create a cyclic intra-half-block sequence and how to link this half-block to the block that precedes it.

Since these blocks do not have a section that follows the rows containing the knockout group, the order of the parity combination is the following: the parity of the section with rows above the knockout group, then the parity of the section with rows containing the knockout group. We know that the number of rows in the half-block must be odd (if there were an even radix among $r_{k-2}, \ldots, r_0$, we would not have a half block; see Lemma 4). Thus, we have two possible parity combinations for the block: OE and EO.

Figures 24 and 25 show how to create a cycle within an EO half-block with odd and even columns. Figures 26 and 27 show how to create a cycle within an OE half-block with odd and even columns. Observe that no matter the parity combinations, each half-block has the same link as the blocks in the upper left corner. Since we know that half-blocks will always be at the very end of a columnar sequence and will always be preceded by a block, in the same way that we connected the blocks, we can use the top left link of the half-block and the bottom left link of the block preceding it to connect the half-block cycle to the Gray code cycle formed by all the preceding blocks.

Given that we can connect blocks together and append a half-block to the end of a group of blocks, we have shown a way to connect the numbers in the columnar structure in a cycle so that adjacent numbers vary only in one digit. Thus, we have shown a way to find a cyclic mixed-radix Gray code for the values of $r$ and $n$ with which we started.

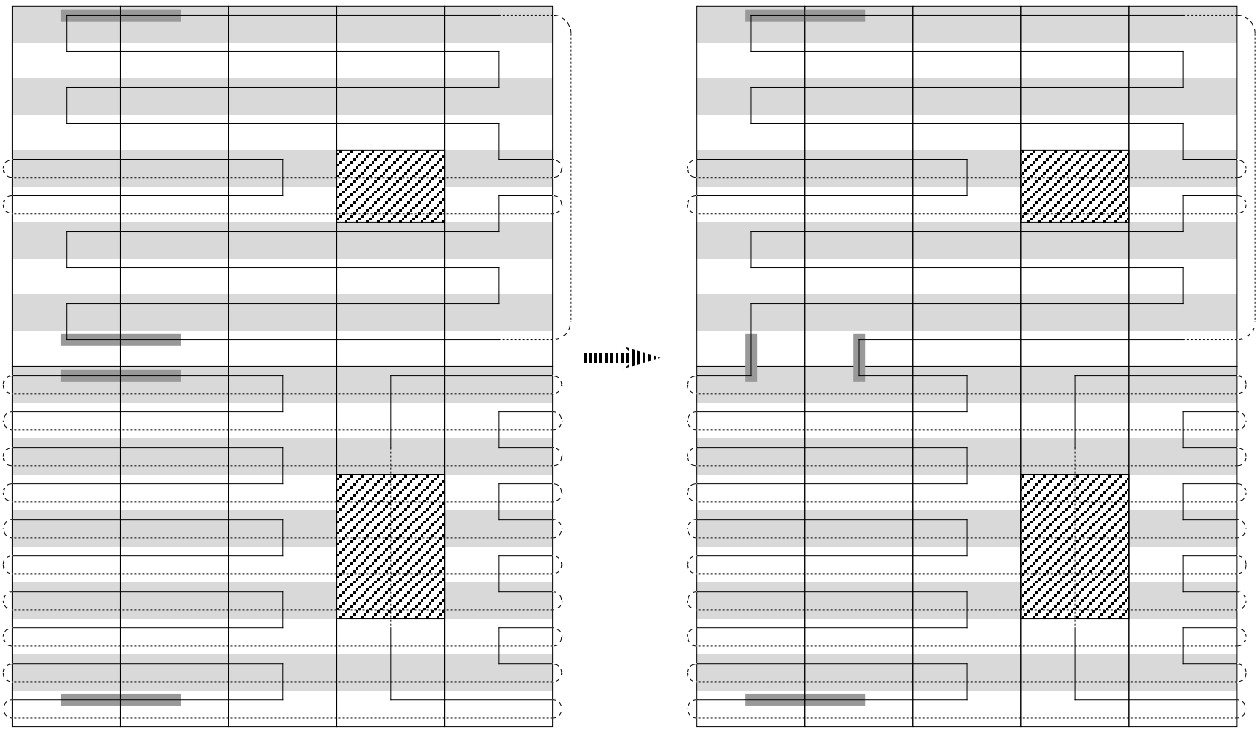

**Figure 23:** Method of linking two blocks to form an interblock cycle. On the left side of the figure are EEE (top) and EOE (bottom). Observe that the link highlighted in gray in the lower left corner of EEE and the link highlighted in gray in the upper left corner of EOE are broken to form an interblock cycle (shown on the right side).

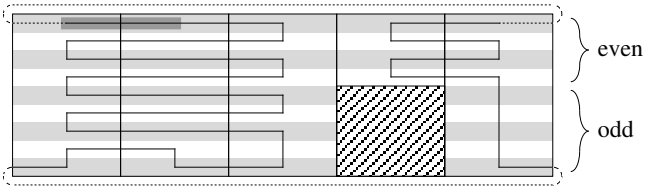


**Figure 24:** Method to create an intra-half-block cycle for EO with an odd number of columns.

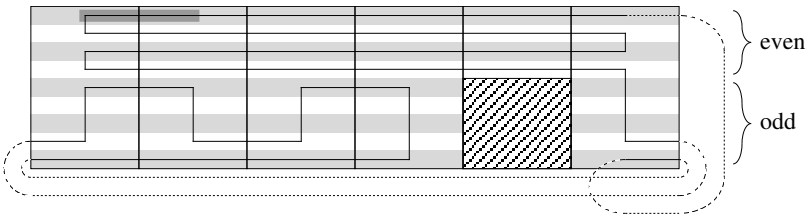


**Figure 25:** Method to create an intra-half-block cycle for EO with an even number of columns.

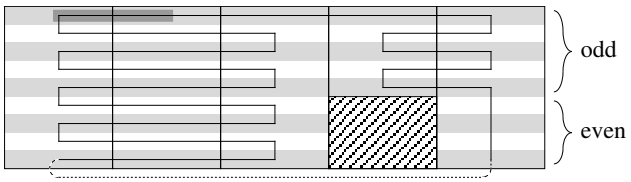


**Figure 26:** Method to create an intra-half-block cycle for OE with an odd number of columns.

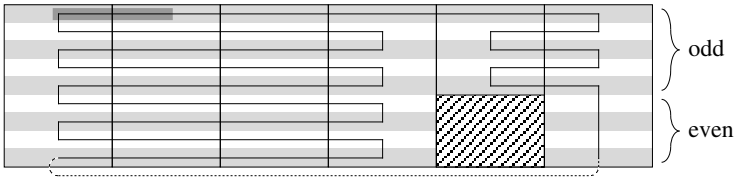


**Figure 27:** Method to create an intra-half-block cycle for OE with an even number of columns.

## 5.3 Finding a cyclic mixed-radix dense Gray code when $r_{k-1} = 2$ and $n$ is even

The method to find a cyclic mixed-radix Gray code if $r_{k-1} = 2$ and $n$ is even is similar to the method used when $r_{k-1} \geq 3$. We set up the columnar sequence, divide it into blocks, and then connect the blocks. Figures 28, 29, 30, and 31 show the block structures that occur when $n$ is even. Observe that much like the method of breaking links that we used to combine blocks for $r_{k-1} \geq 3$, we can combine blocks in two ways: either using the topmost and bottom links of each block or using the intracolumn (vertical) links within each block.

Observe in Figures 28, 29, and 31 that there exists a link between the codewords at the top of the both columns and a link between the codewords at the bottom of both columns. We can use the links at the top and the bottom of a block to connect one block to another. The bottom link of one block can be linked to the top link of the following block. Thus, we can link these three types of blocks. As we see in Figure 30, however, these top and bottom links do not exist in blocks with a knocked out number in every row. Observe, however, that although these top and bottom links are not present, there are multiple intracolumn links in the block. Also note that there exist intracolumn links in the right column of every type of block. There must be at least two intracolumn links in the right column of each block. For the blocks with knockout groups, there exists one intracolumn link between the codeword in the first row of the knockout group and the codeword preceding it in the right column; there exists another intracolumn link between the codeword in the last row of the knockout group and the codeword following it in the right column. For the block with no knockout group, since there are at least four rows in a block (because the lowest possible radix in $r$ is 2), there must be at least two intracolumn links in the rightmost column. Thus, every block has at least two intracolumn links in the rightmost column. We can use these intracolumn links in the rightmost column to link blocks with an entire column knocked out to other blocks; since there are at least two of these intracolumn links, if a block is both preceded and followed by a block with an entire column knocked out, we can use one link to connect to the preceding block and one link to connect to the following block. Figure 32 shows the process of linking an OEO to two blocks in which every row contains a knocked out number. Observe in

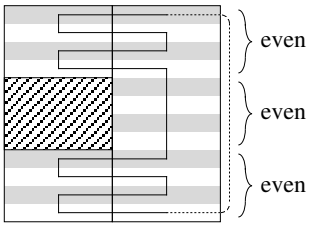


**Figure 28:** Method to create an intrablock cycle for EEE with two columns.

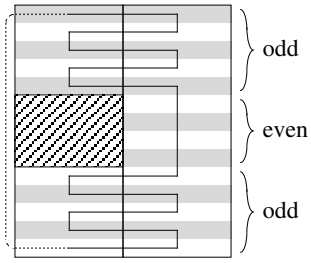


**Figure 29:** Method to create an intrablock cycle for OEO with two columns.

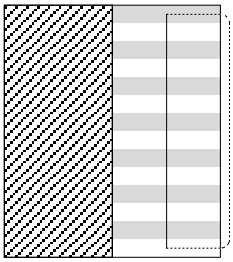

**Figure 30:** Method to create an intrablock cycle for two columns with every row containing a knocked out number.

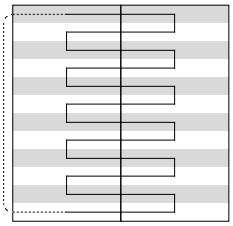

**Figure 31:** Method to create an intrablock cycle for two columns given that no numbers are knocked out in the block.

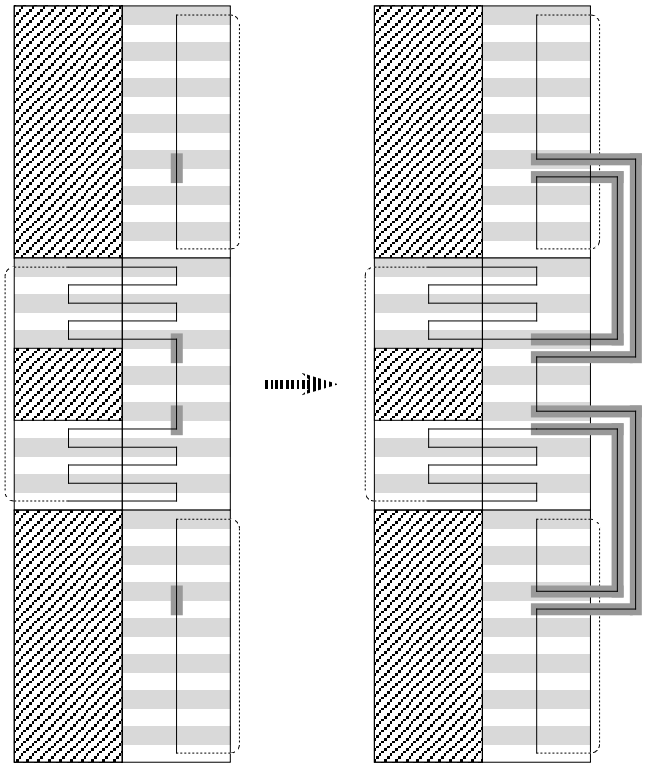

**Figure 32:** Method of linking blocks with two columns to form an interblock cycle. On the left side of the figure are three blocks: two with a knocked out codeword in every row and one OEO block. The OEO block is between the two other blocks. The links highlighted in gray are the intracolumn links used to connect the blocks to create an interblock cycle spanning all three blocks (shown on the right side of the figure).

Figure 32 that we create an interblock cycle than spans all three blocks by breaking and recombining intracolumn links in each block.

Thus, we can link blocks with knockout groups and blocks with no knockout groups using the top and bottom links of each block, and we can link blocks with an entire column knocked out to any other block using intracolumn links in the rightmost column. Like the method we used when $r_{k-1} \geq 3$, we link these intrablock cycles together to create a cyclic mixed-radix dense Gray code.

### 5.4 Conjecture: provided that $r$ and $n$ do not fall under the cases in which we know a cyclic mixed-radix dense Gray code does not exist, when $r_{k-1} = 2$ and $n$ is odd, a cyclic mixed-radix dense Gray code exists

We have not yet found a consistent method to find a cyclic mixed-radix dense Gray code when $r_{k-1} = 2$ and $n$ is odd. This combination of $r$ and $n$ results in one or more blocks for which it is impossible to find an intrablock cycle, thus making the method we used for all other combinations of $r$ and $n$ inapplicable. After running code to find a cyclic mixed-radix Gray code when $r_{k-1} = 2$ and $n$ is odd, however, we have not found an instance (other than the cases for which we know $r$ and $n$ cannot generate a cyclic mixed-radix dense Gray code) in which $r_{k-1} = 2$ and $n$ odd does not work. Thus, we conjecture that a Gray code exists in such cases.

## 6 Conclusion

We have now shown the contributions of this thesis:

- A constant-time algorithm to generate each digit of each codeword in a cyclic mixed-radix full Gray code.
- A new case in which it is impossible to compute a cyclic mixed-radix dense Gray code.
- A linear-time method to find a cyclic mixed-radix dense Gray code for a mixed-radix tuple $r$, where $r_{k-1} \geq 3$ or $n$ is even.
- A conjecture regarding the possibility of a cyclic mixed-radix dense Gray code for a mixed-radix tuple $r$, where $r_{k-1} = 2$ and $n$ is odd.

We have not yet found a method to determine a cyclic mixed-radix dense Gray code for a mixed-radix tuple $r$, where $r_{k-1} = 2$, and $n$, where $n$ is odd. Finding a method that generates a cyclic mixed-radix dense Gray code when $r_{k-1} = 2$ and $n$ is odd would be a part of our future research.

## Acknowledgments

I would like to thank my thesis advisor, Professor Thomas H. Cormen of the Dartmouth Computer Science Department. Professor Cormen has been a mentor to me ever since I took his algorithms class in Fall 2016. It is because of Professor Cormen that I engaged in

undergraduate research at all, and I will forever be grateful to him for supporting me and guiding me through what was one of the most rewarding experiences of my Dartmouth career. It's a privilege to have had the opportunity to work with him.

I would also like to thank the members of my thesis committee: Professor Prasad Jayanti, of the Dartmouth Computer Science Department, and Professor Peter Winkler, of the Dartmouth Computer Science and Mathematics Departments. Their expertise and support, as well as the general support from the Dartmouth Computer Science Department, have made this thesis possible.

Finally, I would like to thank all the people, places, and things whose contributions to this thesis are intangible but profoundly important: my family, Shashwat Kala, Kent Sutton, sisters of Alpha Phi, CS50 Code Masters, members of the Dartmouth Programming Collaborative, Flamin' Hot Cheetos, T.N.C., Balch Hill, Stinson's, Susan Miller's monthly horoscopes, and everyone who heard me talk about this thesis over the past year. None of this would have been possible without you.

# References

[1] M. C. Er. On generating the $n$-ary reflected Gray codes. *IEEE Transactions on Computers*, C-33(8):739–741, August 1984.

[2] Jessica C. Fan. Dense Gray codes in mixed radices. Technical Report TR2017-818, Dartmouth College Department of Computer Science, 2017. Senior honors thesis.