3-1-2017

Dense Gray Codes in Mixed Radices

Jessica C. Fan
Dartmouth College

Dartmouth College Computer Science
Technical Report TR2017-818

Dense Gray Codes in Mixed Radices

Jessica C. Fan
jessica.c.fan.17@dartmouth.edu

Abstract

The standard binary reflected Gray code describes a sequence of integers 0 to $n-1$, where n is a power of 2, such that the binary representation of each integer in the sequence differs from the binary representation of the preceding integer in exactly one bit. In September 2016, we presented two methods to compute binary dense Gray codes, which extend the possible values of n to the set of all positive integers while preserving both the Gray-code property such that only one bit changes between each pair of consecutive binary numbers, and the density property such that the sequence contains exactly the n integers 0 to $n-1$. The first of the two methods produces a dense Gray code that does not have the cyclic property, meaning that the last integer and the first integer of the sequence do not differ in exactly one bit. The second method, based on the first, produces a cyclic dense Gray code if n is even. This thesis summarizes our previous work and generalizes the methods for binary dense Gray codes to arbitrary radices that may either be a single fixed radix for all digits or mixed radices where each digit may be represented in a different radix. We show how to produce a non-cyclic mixed-radix dense Gray code for any set of radices and any positive integer n—that is, a permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$ such that the digit representation of each number differs from the digit representation of the preceding number in only one digit, and the values of the digits that differ is exactly 1. To this end, we provide a simple formula to compute each digit of each number in the permutation in constant time. Though we do not provide such a formula to generate the digits of a cyclic mixed-radix dense Gray code, we do present, for n equal to the product of the radices, a recursive algorithm that computes the entire cyclic mixed-radix Gray code with the density, strict Gray-code, and modular cyclic properties: given a k-tuple of mixed radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, each of the n integers in the cyclic mixed-radix Gray code differs from its preceding integer—with the first integer differing from the last integer—in only one digit position i, and the values of those digits differ by exactly 1, except for the digits of the first and last numbers, which may also be the integers 0 and $r_i - 1$. For values of n that are less than the product of the radices, we show a list of cases for which we prove it is impossible to generate a mixed-radix dense Gray code that has the modular Gray-code and cyclic properties for a set of mixed radices r and a positive integer n.

1-bit binary reflected Gray code	2-bit binary reflected Gray code	3-bit binary reflected Gray code
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

Table 1: The first 3 binary reflected Gray codes for $n = 2$, $n = 4$, and $n = 8$.

1 Introduction

The standard binary reflected Gray code, patented by Frank Gray in 1953 [5], is a sequence of n binary integers in the range 0 to $n - 1$ (or equivalently, a permutation of the integers $\langle 0, 1, \ldots, n - 1\rangle$) that holds the *Gray-code property*: each integer in the sequence differs from the preceding integer in only one bit. This property gives rise to many powerful applications of the binary Gray code, such as finding Hamiltonian paths along n-dimensional hypercubes [6] and generating Dyck words to compute all n-node binary trees [10]. Though exceptionally useful, Gray's binary code allows only for values of n that are a power of 2; Table 1 shows the first 3 binary reflected Gray codes for $n = 2, 4$, and 8. Notice that each Gray code is also *cyclic*: the first and last numbers of each sequence preserve the Gray-code property, differing in only one bit as the sequence wraps around.

In our 2016 paper [3], we identified three properties of the binary Gray code that are of interest to us: the Gray-code property and the cyclic property as we described before, and the density property, which identifies the sequence as a permutation of $\langle 0, 1, \ldots, n - 1\rangle$. The wide success of the binary reflected Gray code is due in part to its ability to hold all three of these properties. Despite its many uses, however, the binary reflected Gray code constrains n to powers of 2. Our paper expanded upon the possibilities of n while preserving the three properties we listed above.

Our first method for a dense Gray code generates a Gray code that is not cyclic but preserves density for any positive integer n. For example, a non-cyclic dense Gray code for $n = 7$ is the sequence $\langle 011, 010, 000, 001, 101, 100, 110\rangle$, which corresponds to the integers $\langle 3, 2, 0, 1, 5, 4, 6\rangle$. Our second method builds upon the first and produces a cyclic dense Gray code if n is even. Both methods are based on the standard binary reflected Gray code and, as in the binary reflected Gray code, each number in the output sequence can be computed in a constant number of word operations given just its index x in the sequence.

After publishing our paper on binary dense Gray codes, we turned our attention to dense Gray codes for fixed- and mixed-radix numbers. We wondered whether there was a larger mathematical principle behind our methods for binary dense Gray codes that could be extended to compute dense Gray codes for any radices. And sure enough, there was! Not only that, but the method we developed for a non-cyclic mixed-radix dense Gray code shed new light upon our previous work for dense Gray codes in binary, providing a more intuitive way of reasoning about all dense Gray codes and simplifying beautifully to recreate the results of our 2016 paper.

Part I of this thesis discusses Gray codes exclusively for binary numbers. It briefly introduces the standard binary reflected Gray code [5] and discusses its applications. Then, it covers the two methods for a

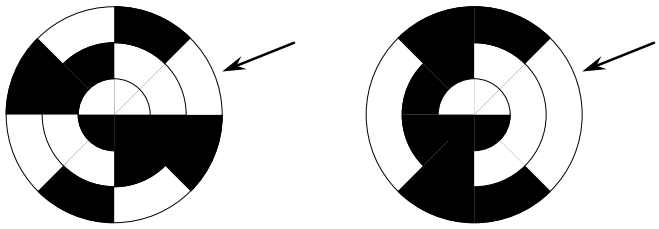
binary dense Gray code that we presented in our 2016 paper, showing how to derive both Gray codes from the standard binary reflected Gray code. Part II begins our discussion for dense Gray codes in mixed radices and builds up to an efficient formula that we can use to compute non-cyclic dense Gray codes for any radices. Within this part of the thesis, we exploit the path we took when discovering the non-cyclic binary dense Gray code to guide us on an analogous approach to the non-cyclic mixed-radix dense Gray code. We find that, like its binary counterpart, the non-cyclic mixed-radix dense Gray code is based on a reflected Gray code, which we can easily produce using Er's recursive methods [4] for fixed- and mixed-radix reflected Gray codes. By closely deconstructing Er's methods, we are able to develop a set of formulas that calculate each integer of the Gray code from just its ordinal index in the sequence. From these equations, we can engineer our simple solution for the non-cyclic mixed-radix dense Gray code. The last section of Part II demonstrates that indeed, a simplified version of our method for a non-cyclic mixed-radix dense Gray code reaffirms our results for the binary dense Gray code as stated in Part I.

Part III discusses cyclic mixed-radix dense Gray codes with the modular Gray-code and cyclic properties, where, given a mixed-radix sequence of integers and a k-tuple of mixed radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, each integer in the sequence differs from its preceding integer—with the first integer differing from the last integer—in only one digit position i, and the values of those digits are either 0 and $r_i - 1$, or they differ by exactly 1. We first present several methods for cyclic mixed-radix full Gray codes, which are where the number n of integers in these sequences is equal to the product of the radices in r. The methods we review from previous literature create restrictions on the radix tuple r in order to guarantee the target sequence, but this thesis introduces a new recursive procedure that generates a cyclic mixed-radix full Gray code for any mixed-radix tuple r. To compute cyclic mixed-radix dense Gray codes with n integers, where n is less than the product of the radices, we introduce a graphical model to represent the modular Gray-code property among integers. Then, we equate the problem of generating a cyclic mixed-radix dense Gray code to the task of finding a Hamiltonian cycle in that graph. This graph-theoretic approach helps us reveal several cases where it is impossible to compute a cyclic mixed-radix dense Gray code for a radix tuple r and sequence length n.

Contributions of this thesis

In summary, the five major contributions of this thesis are as follows:

- A formula for each digit of the non-cyclic binary dense Gray code for any positive integer n, as given in our earlier paper [3]. With this formula, we can generate each number in the non-cyclic binary dense Gray code in constant time.
- An algorithm that generates a cyclic binary dense Gray code for any even number of integers. The algorithm computes each number in constant time.
- A formula for each digit of the non-cyclic mixed-radix dense Gray code for any mixed-radix tuple r and positive integer n less than or equal to the product of the radices in r.
- A recursive algorithm that generates each integer in the cyclic mixed-radix full Gray code for a mixed-radix tuple r and positive integer n equal to the product of the radices in r.
- A list of cases where it is impossible to compute a cyclic mixed-radix dense Gray code for a mixed-radix tuple r and positive integer n strictly less than the product of the radices in r.



(a) (b)

Figure 1: Two possible designs for a rotary encoder for $n = 8$, or 3 bits. Each design shows 8 different settings, with the setting numbers displayed so that the more significant bits are closer to the center of the circular design. Values of the bits are represented in black and white, where black indicates a 1 and white indicates a 0. **(a)** The design shows the binary ordinal sequence 000 to 111 counterclockwise from the arrow. **(b)** The design shows the binary reflected Gray code counterclockwise from the arrow.

Part I
Gray codes in binary

We start in Section 2 with an introduction to the standard binary reflected Gray code, including a set of equations that calculate each integer in the Gray code from just its index in the sequence. Then, in Section 3, we summarize our previous work [3] on Gray codes and describe without proof how to compute non-cyclic dense Gray codes in binary based on the solution for a reflected Gray code. (In Part II, we will prove a more generalized equation for computing non-cyclic dense Gray codes in any radices and apply that equation to the binary case, which will both recreate the equations stated in this part of the thesis and serve as a proof of their correctness.)

In Section 4, the last section of this part, we show how to easily modify our method for a non-cyclic binary dense Gray code to generate a cyclic binary dense Gray code for even-valued sequence lengths.

2 The standard binary reflected Gray code

Originally designed for use in pulse code communication, Gray codes have since proven their versatility in a number of applications including rotary encoders, error-correcting codes, and mechanical puzzles such as Towers of Hanoi. Let's examine how a Gray code can be used to enhance a rotary encoder. We can think of a rotary encoder as a dial-like interface made to map a set of n physical settings to binary numbers that it serializes and sends as a bit stream to a receiving computer. Figure 1 shows two different design possibilities for a rotary encoder. The one on the left contains eight settings that encode for the binary sequence $\langle 000, 001, \ldots, 111 \rangle$, corresponding to the decimal sequence $\langle 0, 1, \ldots, 7 \rangle$, going counterclockwise. Alternatively, the design on the right shows the eight different settings in Gray-code order, which also contains all the binary numbers from 000 to 111 but with the added Gray-code and cyclic properties that only one bit changes between any two adjacent states.

We can assume that each of the $k = \lg n$ bits change independently when we jump from one setting to the next. Notice that with the in-order binary sequence, there is a momentary risk of reading the wrong

setting. Consider the transition from setting 101 to setting 110. In this example, the two least-significant bits change. If the two bits do not change at the same time, the unintended settings 100 or 111 may briefly be in effect. Worse, if we are using the settings to define states in a finite state machine, the state that the computer receives may momentarily be either of these two erroneous readings and, as a result, the computer might start executing the wrong set of instructions pointed to by the incorrectly-read state. The Gray-code property obviates this problem. Therefore, a rotary encoder that defines settings in Gray-code order will never receive an unintended input.

Gray's patent describes a simple recursive process that generates the k-bit binary reflected Gray code for $n = 2^k$. When $n = 2$, the 1-bit binary reflected Gray code is simply the sequence $\langle 0, 1 \rangle$. To create the $(\lg n)$-bit binary reflected Gray code of length n, where $n > 2$, we start with the $(\lg n - 1)$-bit binary reflected Gray code of length $n/2$ and follow a 3-step process: draw a line of reflection after the last integer in the $(\lg n - 1)$-bit binary reflected Gray code, reflect the $(\lg n - 1)$-bit Gray code over the line, and prepend or concatenate 0 as the leftmost bit of the $n/2$ numbers before the line and 1 as the leftmost bit of the $n/2$ numbers after the line. For example, to generate the 3-bit binary reflected Gray code for $n = 8$, imagine a line of reflection below the 2-bit binary reflected Gray code $\langle 00, 01, 11, 10 \rangle$ for $n = 4$; reflect the sequence over that line to generate $\langle 00, 01, 11, 10, 10, 11, 01, 00 \rangle$; and finally, prepend 0 to the four numbers of the original sequence and 1 to the four numbers of the reflected sequence, yielding $\langle 000, 001, 011, 010, 110, 111, 101, 100 \rangle$.

Gray observed a method [5] to generate the xth value of the binary reflected Gray code for 2^k—which we will denote g—in a constant number of word operations. The equation is simply $g = x \oplus \lfloor x/2 \rfloor$, where $\oplus$ is the bitwise exclusive-or operator and $\lfloor \cdot \rfloor$ is the mathematical floor operator. Thus, we simply need to set each bit of g equal to the result of XORing the corresponding bit in x with the next most significant bit of x, leaving the most significant bit alone. If the binary representation of x is $x_{k-1}x_{k-2}\cdots x_0$, then the binary representation $g_{k-1}g_{k-2}\cdots g_0$ of g is calculated as

$$g_{k-1} = x_{k-1} , \tag{1}$$

$$g_i = x_{i+1} \oplus x_i \quad \text{for } i = 0, 1, \ldots, k-2 . \tag{2}$$

In C code, we can denote the set of equations as `g = x ^ (x >> 1)`, where `^` is the bitwise exclusive-or (XOR) operator and `>>` is the bitwise right-shift operator. Assuming that bitwise XOR and right-shift both take a constant number of word operations, then calculating g from x will also take a constant number of word operations.

3 The non-cyclic binary dense Gray code

Our method to generate the binary non-cyclic dense Gray code for n, where n is not a power of 2, is a simple two-step process. We start by taking the first n numbers from the binary reflected Gray code for the next higher power of 2. We then perform a bitwise XOR operation on each of those numbers by the bit mask $m = \lfloor n/2 \rfloor$ to complete the non-cyclic dense Gray code. Let us denote the xth integer of the non-cyclic dense Gray code for n as d. Table 2 then shows the stepwise generation of all sequence elements d for $n = 13$ and $x = 0, 1, \ldots, n-1$. Because $\lfloor 13/2 \rfloor = 6$, the mask m in binary is 0110.

Recall our claim that we can generate d in $\Theta(1)$ word operations given its index x. We start by defining $k = \lfloor \lg n \rfloor$ to be the minimum number of bits needed to represent the value $n - 1$, which is the highest number we will need to compute. Then the binary reflected Gray code for the next higher power of 2 must

ordinal x	binary reflected Gray code g	mask m	non-cyclic dense Gray code d	decimal counterpart
0	0000	0110	0110	6
1	0001	0110	0111	7
2	0011	0110	0101	5
3	0010	0110	0100	4
4	0110	0110	0000	0
5	0111	0110	0001	1
6	0101	0110	0011	3
7	0100	0110	0010	2
8	1100	0110	1010	10
9	1101	0110	1011	11
10	1111	0110	1001	9
11	1110	0110	1000	8
12	1010	0110	1100	12
13	1011			
14	1001			
15	1000			

Table 2: XORing each of the first $n = 13$ numbers in the binary reflected Gray code for 16 numbers with the mask $m = \lfloor n/2 \rfloor = 0110$ produces a permutation of $\langle 0, 1, \ldots, 12 \rangle$ with the Gray-code property.

have sequence length 2^k. Therefore, we compute the xth value of the binary reflected Gray code and XOR the result with $\lfloor n/2 \rfloor$ to generate the xth value of the non-cyclic dense Gray code.

Section 2 showed how to compute g, the xth number in the Gray code, in $\Theta(1)$ word operations, so we now proceed to calculate d using the mask m, which we denote with the binary representation $m_{k-1}m_{k-2}\cdots m_0$. We set m to $\lfloor n/2 \rfloor$, which has the binary representation $0\, n_{k-1}n_{k-2}\cdots n_1$. Thus,

$$m_{k-1} = 0\,, \tag{3}$$

$$m_i = n_{i+1} \quad \text{for } i = 0, 1, \ldots, k-2\,, \tag{4}$$

and therefore, the binary representation $d_{k-1}d_{k-2}\cdots d_0$ of d becomes

$$\begin{aligned} d_{k-1} &= g_{k-1} \oplus m_{k-1} \\ &= x_{k-1} \oplus 0 \qquad \text{(by equations (1) and (3))} \\ &= x_{k-1} \end{aligned} \tag{5}$$

and

$$d_i = g_i \oplus m_i \tag{6}$$

$$= x_{i+1} \oplus x_i \oplus n_{i+1} \quad \text{(by equations (2) and (4))} \tag{7}$$

for $i = 0, 1, \ldots, k-2$. In C code, we have `d = x ^ (x >> 1) ^ (n >> 1)`. Since we already assumed that bitwise XOR takes a constant number of word operations, the process from x to d also requires only constant number of word operations.

With more work, we can also compute the inverse of the non-cyclic dense Gray code function. Let x be the ordinal index where the integer d appears in the non-cyclic dense Gray code. By equation (5), we have

$x_{k-1} = d_{k-1}$. To compute the remaining bits of x, we must first understand how to compute the inverse of the binary reflected Gray code function, which is the ordinal index x where the integer g appears in the binary reflected Gray code. Gray [5] showed that

$$x_i = g_{k-1} \oplus g_{k-2} \oplus \cdots \oplus g_i \quad \text{for } i = 0, 1, \ldots, k-2 \,.$$

XORing x_{i+1} to both sides of equation (2) gives

$$x_i = g_i \oplus x_{i+1} \quad \text{for } i = 0, 1, \ldots, k-2 \,. \tag{8}$$

XORing m_i into both sides of equation (6) and subsequently applying equation (4) gives

$$\begin{aligned} g_i &= d_i \oplus m_i \\ &= d_i \oplus n_{i+1} \quad \text{for } i = 0, 1, \ldots, k-2 \,. \end{aligned} \tag{9}$$

Finally, combining equations (8) and (9) gives, for $i = 0, 1, \ldots, k-2$,

$$x_i = d_i \oplus n_{i+1} \oplus x_{i+1} \,.$$

In C, given d, n, and k, we can compute the inverse x of the integer d in the non-cyclic dense Gray code, assuming that the variables have all been declared as integer types. The following code uses equations (5) and (8) to compute x one bit at a time, from bit $k-1$ down to bit 0. The code places each bit x_i into bit position 0 and then shifts it one position to the left before computing bit x_{i-1}:

```
g = d ^ (n >> 1);
x = (d >> (k-1)) & 1;
for (i = k-2; i >= 0; i--) {
    x <<= 1;
    x |= ((x >> 1) ^ (g >> i)) & 1;
  }
```

This code requires $\Theta(\lg n)$ word operations, assuming that shifting right by at most $k-2$ bits takes $\Theta(1)$ word operations.

4 The cyclic binary dense Gray code

We can easily adapt the method for a length-n non-cyclic dense Gray code to form the cyclic dense Gray code for $2n$. The method we describe here is not the same as the method we published in our 2016 paper, but the intuition behind the new procedure is so obviously correct that it subsumes the work we had done previously. To generate the cyclic dense Gray code for $2n$ using the new method, we take the non-cyclic dense Gray code for n and, as with the binary reflected Gray code, reflect the sequence of numbers over a line of symmetry drawn after the last number of the sequence. Then, append 0 as the rightmost bit of the $n/2$ numbers before the line, and append 1 as the rightmost bit of the $n/2$ numbers below the line to form the set of even numbers $\{0, 2, \ldots, 2n-2\}$ and the set of odd numbers $\{1, 3, \ldots, 2n-1\}$ respectively. The resulting sequence contains the numbers 0 to $2n-1$ and is dense. We can easily see that it maintains the Gray-code and cyclic properties since it mimics the stepwise generation of the binary reflected Gray code for 2^k. Table 3 shows this process in detail for $2n = 26$; we denote the xth number in the cyclic dense Gray code for $2n$ with c.

ordinal x	non-cyclic dense Gray code d	cyclic dense Gray code c	decimal counterpart
0	0110	01100	12
1	0111	01110	14
2	0101	01010	10
3	0100	01000	8
4	0000	00000	0
5	0001	00010	2
6	0011	00110	6
7	0010	00100	4
8	1010	10100	20
9	1011	10110	22
10	1001	10010	18
11	1000	10000	16
12	1100	11000	24
13		11001	25
14		10001	17
15		10011	19
16		10111	23
17		10101	21
18		00101	5
19		00111	7
20		00011	3
21		00001	1
22		01001	9
23		01011	11
24		01111	15
25		01101	13

Table 3: Reflecting the non-cyclic dense Gray code for $n = 13$ and prepending 0s to the original sequence and 1s to the reflected sequence generates the cyclic dense Gray code for $2n = 26$.

In C, given any positive integer n and an ordinal number x in the range 0 to $2n-1$, we can compute the xth integer c in the cyclic dense Gray code as follows:

```
if (x >= n)                    // is x in the second half?
   x = 2*n-1-x;                // if so, use x's reflection
d = x ^ (x >> 1) ^ (n >> 1);   // compute d
c = d << 1;                    // left-shift by 1 to double d
if (x >= n)                    // again, is x in the second half?
  c |= 1;                      // if so, change the least-significant bit to 1
```

Using our previous definition for the inverse function of the non-cyclic dense Gray code, we can easily define an inverse function for the cyclic dense Gray code. Let x be the ordinal index where the integer c appears in the cyclic dense Gray code. In C, we've denoted the inverse function of the integer d in the non-cyclic dense Gray code as `inv(d)` and the modulus operator as `%`. Then, given n and the value c, which lies

in the range 0 to $2n - 1$, we can compute x as follows:

```
d = c >> 1;                    // right-shift c to form d
x = inv(d);
if (c % 2 == 1)                // is c in the reflected half?
  x = 2*n-1-x;                 // if so, reflect x
```

Part II
Non-cyclic Gray codes in fixed and mixed radices

Within this part of the thesis, we will construct a formula that computes each digit in the non-cyclic mixed-radix dense Gray code. Like the binary case, fixed- and mixed-radix dense Gray codes are based on reflected Gray codes generated using the corresponding radices. Therefore, Sections 5 and 6 begin our quest for a mixed-radix dense Gray code by presenting Er's recursive method [4] for a fixed-radix reflected Gray code and generalizing his methods to mixed-radix systems. In each of these sections, we analyze the structure of Er's recursive method and generate formulas that directly compute each element of Er's fixed- and mixed-radix reflected Gray codes given just its index in the sequence, much like the equations that Gray developed to generate each element of his binary reflected Gray code.

With the equations for fixed- and mixed-radix dense Gray codes in hand, Section 7 then builds an intuitive formula for each element of a non-cyclic mixed-radix dense Gray code and subsequently proves its correctness. Finally, Section 8 uses generalized observations about dense Gray codes to circle back and prove the binary case as described in Part I.

5 The fixed-radix reflected Gray code

Inspired by Gray's work, Er [4] expanded the reflected Gray code to any radix $r \geq 2$. His new Gray code, the fixed-radix reflected Gray code, is a sequence of integers represented in radix r: each digit may only take on a value between 0 and $r-1$. Furthermore, Er's fixed-radix reflected Gray code holds the generic Gray-code property, where each integer in a sequence of numbers radix r differs from the preceding integer in exactly one digit. Though Er does not explicitly say so, each pair of consecutive integers in his fixed-radix reflected Gray code not only differs in one digit, but the difference between the two digits that differ is exactly 1. Thus, Er's Gray-code property can be strengthened to make the *strict* Gray-code property: given any two consecutive integers in a fixed-radix Gray code, the absolute difference between the only two differing digits is 1.

Er's algorithm borrows much of the intuition that Gray used generate the binary reflected Gray code. His 3-step recursive process contains some minor alterations to generalize the procedure to radix r. Let us define $n = r^k$ to be the length of the fixed-radix reflected Gray code produced by Er's method. Here, k refers to the number of digits used to represent each value in the sequence. If $k = 1$, the 1-digit fixed-radix reflected Gray code is simply $\langle 0, 1, \ldots, r-1 \rangle$. If $k > 1$, we can define the $(k-1)$-digit sequence as Γ_{k-1}, where $k = \log_r n$. Then, the k-digit sequence Γ_k is calculated by starting with the sequence Γ_{k-1}, appending its reflection (call it Γ_{k-1}^R), then its copy, then its reflection, then its copy, and so on until the resulting sequence is of length r^k. The intermediate sequence now has r parts, $\lceil r/2 \rceil$ of which are copies of the $(k-1)$-digit fixed-radix reflected Gray code and $\lfloor r/2 \rfloor$ of which are reflections. Now, we prepend the numbers $0, 1, \ldots, r-1$ to the r parts of the reflected sequence we computed, assigning 0 for the first n/r numbers, 1 for the second n/r numbers, and so on until we've prepended $r-1$ for the last n/r numbers. If r is odd, then we can write Γ_k as

$$\langle 0\Gamma_{k-1}, 1\Gamma_{k-1}^R, 2\Gamma_{k-1}, 3\Gamma_{k-1}^R, \ldots, (r-1)\Gamma_{k-1} \rangle \,, \tag{10}$$

where xY indicates the digitwise concatenation of x with each number y from the sequence Y. Notice that since we have an odd number of subsequences, the last subsequence that we prepend $r-1$ to must be a copy

of Γ_{k-1}. Alternatively, if r is even, the last subsequence of Γ_k is a reflection of Γ_{k-1}, and thus we can write Γ_k as

$$\langle 0\Gamma_{k-1}, 1\Gamma_{k-1}^R, 2\Gamma_{k-1}, 3\Gamma_{k-1}^R, \ldots, (r-1)\Gamma_{k-1}^R \rangle \ . \tag{11}$$

Let's take an example where r is odd: $r = 3$ and $k = 2$ so that $n = r^k = 9$. We start with the 1-digit ternary ($r = 3$) reflected Gray code for $n = 3$ which is the sequence $\Gamma_1 = \langle 0, 1, 2 \rangle$. First, we append to Γ_1 the sequence Γ_1^R and then Γ_1, giving the sequence $\langle 0, 1, 2, 2, 1, 0, 0, 1, 2 \rangle$. Notice the 3 parts of the intermediate result: $\langle 0, 1, 2 \rangle$, $\langle 2, 1, 0 \rangle$, and $\langle 0, 1, 2 \rangle$. To the $n/r = 3$ numbers in each of these subsequences, we prepend the digits 0, 1, and 2 respectively to generate Er's ternary reflected Gray code for $n = 9$, $\langle 00, 01, 02, 12, 11, 10, 20, 21, 22 \rangle$.

How do we calculate a single element from Γ_k? Given an index x such that $0 \leq x < n$, how do we find the xth number in Er's fixed-radix reflected Gray code? To answer these questions, we need to build a stronger understanding of the reflection function. Let's say a single digit x_i in radix r is reflected. Then, letting $R_r(x_i)$ be the reflection function applied to digit x_i in radix r, we have

$$R_r(x_i) = r - 1 - x_i \ , \tag{12}$$

and we say that the function $R_r(x_i)$ reflects x_i around the radix r. But what if the sequence of single digits $\Gamma_1 = \langle 0, 1, \ldots, r-1 \rangle$ is reflected? In this case, we can describe the reflected sequence $\Gamma_1^R = \langle r-1, r-2, \ldots, 0 \rangle$ as a *descending* sequence in contrast with the original sequence Γ_1, which is *ascending*.

With these definitions in hand, we can derive an algorithm to generate each element of the fixed-radix reflected Gray code from its index x in the sequence. Our first step is to find the connection between the steps to compute a fixed-radix reflected Gray code and the corresponding steps for a fixed-radix ordinal sequence, which is simply a sequence of numbers in increasing order starting from 0. Just as we described Er's method for an fixed-radix reflected Gray code as a recursive 3-step process, we can do the same for the ordinal sequence with one minor step change: instead of building the k-digit fixed-radix ordinal sequence from copies and reflections of the $(k-1)$-digit sequence, we build it solely from copies so that the resulting sequence contains digit positions that follow only the ascending pattern $\langle 0, 1, \ldots, r-1 \rangle$ and never the descending pattern $\langle r-1, r-2, \ldots, 0 \rangle$. Let's walk through this process in detail. We start by defining $n = r^k$ to be the length of the ordinal sequence we are trying to generate, which we will call P_k. As before, if $k = 1$, then P_k is simply the sequence $\langle 0, 1, \ldots, r-1 \rangle$. Otherwise, $k > 1$ and we can construct P_k as

$$\langle 0P_{k-1}, 1P_{k-1}, 2P_{k-1}, 3P_{k-1}, \ldots, (r-1)P_{k-1} \rangle \ , \tag{13}$$

where P_{k-1} refers to the $(k-1)$-bit ordinal sequence. Notice that this equation holds regardless of whether r is even or odd, since we never evaluate its parity to reflect P_{k-1}.

Now, compare how to generate P_k with how to generate Γ_k. What can we say about the pattern of the ith digit in the fixed-radix ordinal sequence compared with the corresponding digit in the fixed-radix reflected Gray code for the digit positions $i = 0, 1, \ldots, k-1$? The ordinal case is easy. From sequence (13), we know that we must construct P_k by prepending each of the digits in $\langle 0, 1, \ldots, r-1 \rangle$ to each element in the (r^{k-1})-length sequence P_{k-1}. Let x^y be a condensed notation representing a y-length sequence of the repeated digit x. Then, if we isolate the most significant digit in each number of P_k—that is, the digits in position $k-1$—we see that they follow exactly the pattern $\langle 0^{r^{k-1}}, 1^{r^{k-1}}, \ldots, (r-1)^{r^{k-1}} \rangle$. Continuing this logic recursively, we find that for $i = 0, 1, \ldots, k-2$, the digit in position i always repeats the (r^{i+1})-length

pattern

$$\begin{array}{c} 0^{r^i} \\ 1^{r^i} \\ \vdots \\ (r-1)^{r^i} \end{array} \tag{14}$$

until the constructed sequence is of length n.

The digits in Er's fixed-radix reflected Gray code follow a similar rule. As with the fixed-radix ordinal sequence, the most significant digit in Γ_k follows the pattern $\langle 0^{r^{k-1}}, 1^{r^{k-1}}, \ldots, (r-1)^{r^{k-1}} \rangle$. Here, however, the recursive rules described in sequences (10) and (11) reflect each alternate subsequence; therefore, the pattern for the ith digit first takes the form of the sequence (14) and then takes its reverse, giving the new pattern

$$\begin{array}{c} 0^{r^i} \\ 1^{r^i} \\ \vdots \\ (r-1)^{r^i} \\ (r-1)^{r^i} \\ \vdots \\ 1^{r^i} \\ 0^{r^i} \end{array} \tag{15}$$

for $i = 0, 1, \ldots, k-2$. Let's describe the above pattern as having two (r^{i+1})-length parts: ascending $\langle 0^{r^i}, 1^{r^i}, \ldots, (r-1)^{r^i} \rangle$ and descending $\langle (r-1)^{r^i}, (r-2)^{r^i}, \ldots, 0^{r^i} \rangle$. If we denote the (r^{i+1})-length pattern that digit position i in Γ_k takes as γ_i, and pattern (14)—the (r^{i+1})-length pattern that digit position i in P_k takes—as ρ_i, then we have simply

$$\gamma_i = \begin{cases} \rho_i & \text{if } \gamma_i \text{ is ascending ,} \\ \rho_i^R & \text{if } \gamma_i \text{ is descending .} \end{cases} \tag{16}$$

For example, Table 4 shows the ternary ordinal sequence juxtaposed with the ternary reflected Gray code for 3 digits. We've denoted the xth element of the sequence Γ_3 by g.

Given a radix r, let us define the fixed-radix representation of ordinal x to be $x_{k-1}x_{k-2}\cdots x_0$. If we take the xth element g from the fixed-radix sequence Γ_k and define its digit representation as $g_{k-1}g_{k-2}\cdots g_0$, then by equation (16), at every digit position $i = 0, 1, \ldots, k-1$, digit g_i is either an element of an ascending sequence identical to the pattern for x_i, or part of a descending sequence reflecting the pattern for x_i. Clearly, g_i is in the ascending sequence if it lies in the first half of the (r^{i+2})-length pattern (15)—that is, if r^{i+1} fits into its index an even number of times—and is in the descending sequence otherwise. Then, using the reflection function R_r for a single digit as given by equation (12), we have

$$g_i = \begin{cases} x_i & \text{if } \lfloor x/r^{i+1} \rfloor \text{ is even ,} \\ r-1-x_i & \text{otherwise .} \end{cases} \tag{17}$$

Notice that the expression $\lfloor x/r^{i+1} \rfloor$ is also equivalent to dropping the rightmost $i+1$ digits of x. This formula completes our algorithm for generating the rth element g in the sequence Γ_k, given only the ordinal x, the radix r, and the number k of digits as our inputs.

ordinal x	fixed-radix reflected Gray code g
000	000
001	001
002	002
010	012
011	011
012	010
020	020
021	021
022	022
100	122
101	121
102	120
110	110
111	111
112	112
120	102
121	101
122	100
200	200
201	201
202	202
210	212
211	211
212	210
220	220
221	221
222	222

Table 4: The ordinal and reflected ternary sequences for $k = 3$ (and $n = 27$). The underlined digits represent the digits that were reflected from their values in the ordinal sequence.

6 The mixed-radix reflected Gray code

Although Er's recursive method for a Gray code [4] was intended for fixed radix, we can easily extend it to compute Gray codes in mixed radices as well. Furthermore, as with the fixed-radix Gray code, we can produce a formula that computes the ith digit of the xth element of the mixed-radix Gray code given just its ordinal index x in the sequence and the set of mixed radices we are using.

We begin with a few guiding principles for working with numbers in mixed radices. Instead of a single radix r, we now consider a k-tuple $(r_{k-1}, r_{k-2}, \ldots, r_0)$ of radices. With such a mixed-radix notation, we can represent the integers 0 to $\left(\prod_{i=0}^{k-1} r_i\right) - 1$. Since this section will discuss products of the radices, we use the notation $p_i = \prod_{j=0}^{i} r_j$ to refer to the product of the rightmost $i + 1$ radices, with the boundary case $p_{-1} = 1$. Thus, if a number has the mixed-radix representation $x_{k-1}x_{k-2}\cdots x_0$, then its integer value is $\sum_{i=0}^{k-1} x_i\, p_{i-1}$. In the special case that all radices are equal and represented with a fixed radix r, then $p_i = r^{i+1}$ and the formula for the value simply becomes $\sum_{i=0}^{k-1} x_i\, r^i$.

Now that we have defined our mixed-radix environment, we analyze the method to construct a mixed-

radix reflected Gray code. Let n be the number of integers in the Gray code, so that the highest integer we will generate is $n-1$. Then, we have $n = p_{k-1}$. Like Er's method, the algorithm for a mixed-radix reflected Gray code follows a 3-step recursive process. If the length k of the radix tuple is 1, then the 1-digit mixed-radix reflected Gray code is simply $\langle 0, 1, \ldots, r_0 \rangle$. Otherwise, $k > 1$ and we denote the $(k-1)$-digit sequence as Γ_{k-1}. The k-digit sequence Γ_k is calculated by appending copies and reflections of the Γ_{k-1} subsequence until the sequence is of length p_{k-1}—a total of r_{k-1} appended subsequences. Finally, to each of the subsequences, we prepend a digit in the range 0 to $r_{k-1}-1$, assigning 0 for the first subsequence, 1 for the second subsequence, and so on until we have assigned r_{k-1} for the last subsequence. If we were to isolate digit $k-1$ from Γ_k, we would see the pattern

$$\langle 0^{p_{k-2}}, 1^{p_{k-2}}, \ldots, (r_{k-1}-1)^{p_{k-2}} \rangle \,, \tag{18}$$

and for digits $i = 0, 1, \ldots, k-2$, we would see the length-$2p_i$ pattern

$$\begin{array}{c} 0^{p_{i-1}} \\ 1^{p_{i-1}} \\ \vdots \\ (r_i-1)^{p_{i-1}} \\ (r_i-1)^{p_{i-1}} \\ \vdots \\ 1^{p_{i-1}} \\ 0^{p_{i-1}} \end{array} \tag{19}$$

until the constructed sequence is of length n.

The digits of the mixed-radix ordinal sequence follow a similar pattern, except they do not recursively reflect subsequences. Therefore, the pattern for digit $k-1$ in the mixed-radix ordinal sequence is $\langle 0^{p_{k-2}}, 1^{p_{k-2}}, \ldots, (r_{k-1}-1)^{p_{k-2}} \rangle$ as in the mixed-radix Gray code, and the pattern for digit i in the mixed-radix ordinal sequence is

$$\begin{array}{c} 0^{p_{i-1}} \\ 1^{p_{i-1}} \\ \vdots \\ (r_i-1)^{p_{i-1}} \end{array} \tag{20}$$

for $i = 0, 1, \ldots, k-2$.

We can view pattern (19) as having two length-p_i halves: an ascending half and a descending half. As in the fixed-radix Gray code, if we are in the descending half of pattern (19) that a digit i takes in Γ_k, then we can simply perform a reflection around radix r_i to get the ordinal pattern (20) for digit i. Table 5 shows the ordinal and reflected Gray-code sequences for the mixed-radix tuple $(2, 3, 4)$. Here, we've denoted the xth integer of Γ_k as g. Notice that each descending sequence in a digit position i is simply a reflection around r_i of the corresponding ascending sequence in the ordinal column.

We end by constructing the formula to compute the xth number of the mixed-radix reflected Gray code from x. Given a k-tuple of mixed radices $(r_{k-1}, r_{k-2}, \ldots, r_0)$, we define the mixed-radix representation of ordinal x to be $x_{k-1}x_{k-2}\cdots x_0$ and similarly define the xth element g from the mixed-radix sequence Γ_k to have the digit representation $g_{k-1}g_{k-2}\cdots g_0$. By our earlier observation, for each digit position $i = 0, 1, \ldots, k-1$, the digit g_i is either an element of an ascending sequence identical to the pattern for x_i, or part of a descending sequence reflective of the pattern for x_i. Clearly, g_i is in the ascending sequence

ordinal x (decimal)	ordinal x (mixed-radix)	mixed-radix reflected Gray code g
0	000	000
1	001	001
2	002	002
3	003	003
4	010	013
5	011	012
6	012	011
7	013	010
8	020	020
9	021	021
10	022	022
11	023	023
12	100	123
13	101	122
14	102	121
15	103	120
16	110	110
17	111	111
18	112	112
19	113	113
20	120	103
21	121	102
22	122	101
23	123	100

Table 5: The numbers 0 to 23 represented using the mixed-radix tuple $(2, 3, 4)$, along with the reflected Gray code.

if it lies in the first half of the length-$2p_i$ pattern (19)—that is, if p_i fits into its index an even number of times—and is in the descending sequence otherwise. Then, using the reflection function R_{r_i} for a single digit as given by equation (12), we have

$$g_i = \begin{cases} x_i & \text{if } \lfloor x/p_i \rfloor \text{ is even ,} \\ r_i - 1 - x_i & \text{otherwise .} \end{cases} \tag{21}$$

As in the case for fixed radix, the expression $\lfloor x/p_i \rfloor$ is equivalent to dropping the rightmost $i+1$ digits of x. The mixed-radix reflected Gray code generated by equation (21) turns out to be equivalent to the sequence generated by an algorithm in Knuth's book [8, p. 300] for a "loopless mixed-radix Gray code," but with the radices in reverse. Thus, running Knuth's algorithm using the ordered radix tuple $(r_0, r_1, \ldots, r_{k-1})$ produces a sequence that, when read from right to left—that is, from least-significant digit to most significant—is exactly the sequence produced by computing equation (21) for the radices $(r_{k-1}, r_{k-2}, \ldots, r_0)$.

7 The non-cyclic mixed-radix dense Gray code

Having discussed the mixed-radix reflected Gray code, we can now build an intuition for what is required for an non-cyclic mixed-radix dense Gray code. As with the non-cyclic binary dense Gray code, we want to expand the possibilities for sequence length n to the set of all whole numbers and produce a permutation

of the fixed-radix sequence $\langle 0, 1, \ldots, n-1 \rangle$ that holds the strict Gray-code property (consecutive pairs of integers differ in only one digit by only 1). Let $(r_{k-1}, r_{k-2}, \ldots, r_0)$ be the k-tuple of mixed radices we will use to generate a Gray code. How could we go about generating the k-digit, mixed-radix dense Gray code for n? We'll first answer this question intuitively by reasoning about ascending and descending patterns in the fixed-radix reflected Gray code. From this intuition, we will build a formula to calculate each xth value of the mixed-radix dense Gray code for n. Finally, we'll use rigorous methods to prove the algorithm's correctness.

We start with an attempt to generate a non-cyclic mixed-radix dense Gray code by taking the first n integers of the mixed-radix reflected Gray code. Suppose our radices are $(3, 3, 4)$ and we wish to produce the dense Gray code for $n = 30$ using these radices. Table 6 shows the first 30 numbers of the mixed-radix reflected Gray code. This approach did not work because we have included the mixed radix numbers 213 and 212 which correspond to the integers 31 and 30, respectively, and are both out of range for our dense Gray code. Meanwhile, we have missed the numbers 210 and 211, or 28 and 29 in decimal.

Notice that the 0th digit (the rightmost digit) within the mixed-radix reflected Gray code is cut off within a descending sequence. Intuitively, if we could make the cut-off point occur outside a descending sequence, then we would favor getting lower numbers in the generated Gray code. To that end, if the ith digit would be cut off within a descending sequence, we reflect all n values for that digit, which either cuts off that digit in an ascending sequence or between ascending and descending sequences. Let us denote the xth value of the mixed-radix dense Gray code, calculated with the mixed-radix tuple $(r_{k-1}, r_{k-2}, \ldots, r_0)$, as d. Then Table 6 shows the correct mixed-radix dense Gray code for $n = 30$.

We now produce a formula to generate each digit of the mixed-radix dense Gray code, using the intuition we gathered about cut-off points. We know that the cut-off point for the ith digit occurs within a descending sequence in the mixed-radix reflected Gray code if $\lfloor n/p_i \rfloor$ is odd. In this case, we reflect all n values for the ith digit of the mixed-radix reflected Gray code, as given in equation (21). Because composing two reflection functions gives the identity function, we can modify equation (21) to get the following formula for the ith digit d_i in the xth integer d of the mixed-radix dense Gray code:

$$d_i = \begin{cases} x_i & \text{if } \lfloor x/p_i \rfloor \bmod 2 = \lfloor n/p_i \rfloor \bmod 2\,, \\ r_i - 1 - x_i & \text{otherwise}\,. \end{cases} \tag{22}$$

Assuming that the k values $p_{k-1}, p_{k-2}, \ldots, p_0$ have all been precomputed (which can be done easily in $\Theta(k)$ time), we can compute each digit in the mixed-radix dense Gray code in constant time.

Proof of our method for the non-cyclic mixed-radix dense Gray code

Here, we show that the digits produced by equation (22) form numbers that give a dense Gray code. We need to prove three properties, which we will prove in the following order:

- Each k-digit number is unique.
- Each k-digit number is in the range 0 to $n - 1$.
- The sequence obeys the strict Gray-code property, so that each number in the sequence differs from the preceding number in exactly one digit, and the values of these digits differ by exactly 1.

ordinal x (decimal)	ordinal x (mixed-radix)	mixed-radix reflected Gray code g	mixed-radix dense Gray code d	decimal counterpart
0	000	000	003	3
1	001	001	002	2
2	002	002	001	1
3	003	003	000	0
4	010	013	010	4
5	011	012	011	5
6	012	011	012	6
7	013	010	013	7
8	020	020	023	11
9	021	021	022	10
10	022	022	021	9
11	023	023	020	8
12	100	123	120	20
13	101	122	121	21
14	102	121	122	22
15	103	120	123	23
16	110	110	113	19
17	111	111	112	18
18	112	112	111	17
19	113	113	110	16
20	120	103	100	12
21	121	102	101	13
22	122	101	102	14
23	123	100	103	15
24	200	200	203	27
25	201	201	202	26
26	202	202	201	25
27	203	203	200	24
28	210	213	210	28
29	211	212	211	29

Table 6: The first n values of the mixed-radix reflected Gray code and the mixed-radix dense Gray code for radices $(3, 3, 4)$ and $n = 30$. Because the 0th digit (the rightmost digit) in the mixed-radix reflected Gray code is cut off within a descending sequence, the mixed-radix dense Gray code is the mixed-radix reflected Gray code with the 0th digit reflected around $r_0 = 4$.

Lemma 1 *Let x and y be whole numbers such that $0 \le x, y < n$ and $x \ne y$. Let x' and y' be the xth and yth values, respectively, of the mixed-radix dense Gray code whose digits are given by the formula in equation (22). Then $x' \ne y'$.*

Proof: Because $x \ne y$, there must be some leftmost digit position j such that $x_j \ne y_j$. As we observed earlier, the value $\lfloor x/p_j \rfloor$ equals the $(k-j-1)$-digit mixed-radix number $x_{k-1}x_{k-2}\cdots x_{j+1}$ for radices $r = (r_{k-1}, r_{k-2}, \ldots, r_{j+1})$. Likewise, the value $\lfloor y/p_j \rfloor$ equals the $(k-j-1)$-digit mixed-radix number $y_{k-1}y_{k-2}\cdots y_{j+1}$ for radices r. By how we defined position j, we have $x_{k-1}x_{k-2}\cdots x_{j+1} = y_{k-1}y_{k-2}\cdots y_{j+1}$. Putting these equalities together, we have

$$\begin{aligned} \lfloor x/p_j \rfloor &= x_{k-1}x_{k-2}\cdots x_{j+1} \\ &= y_{k-1}y_{k-2}\cdots y_{j+1} \\ &= \lfloor y/p_j \rfloor \ . \end{aligned}$$

By equation (22), therefore, we either have $x'_j = x_j$ and $y'_j = y_j$ or we have $x'_j = r_j - 1 - x_j$ and $y'_j = r_j - 1 - y_j$. In either case, since $x_j \ne y_j$, we have $x'_j \ne y'_j$, and so $x' \ne y'$. ∎

Lemma 2 *Let x be a whole number such that $0 \le x < n$, and let x' be the xth value of the mixed-radix dense Gray code whose digits are given by the formula in equation (22). Then $x' < n$.*

Proof: Because $x \ne n$, there must be some leftmost bit position j where x and n differ. Additionally, because $x < n$, we must have $x_j < n_j$. We can use this information along with the following claim to prove our lemma:

For digit positions $i = j, j+1, \ldots, k-1$, we have $x'_{i+1} = n_{i+1}$.

To prove the claim, we note that we have $x_{k-1}x_{k-2}\cdots x_{j+1} = n_{k-1}n_{k-2}\cdots n_{j+1}$ by the definition of j, which implies $x_{k-1}x_{k-2}\cdots x_{i+1} = n_{k-1}n_{k-2}\cdots n_{i+1}$ for $i = j, j+1, \ldots, k-1$. Earlier, we noticed that for any digit position ℓ, the $(k-\ell-1)$-digit mixed-radix number $x_{k-1}x_{k-2}\cdots x_{\ell+1}$ is equal to $\lfloor x/p_\ell \rfloor$ for radices $(r_{k-1}, r_{k-2}, \ldots, r_{\ell+1})$ and, similarly, $n_{k-1}n_{k-2}\cdots n_{\ell+1}$ is equal to $\lfloor n/p_\ell \rfloor$ for the same radices. Therefore, we have

$$\begin{aligned} \lfloor x/p_i \rfloor &= x_{k-1}x_{k-2}\cdots x_{i+1} \\ &= n_{k-1}n_{k-2}\cdots n_{i+1} \\ &= \lfloor n/p_i \rfloor \end{aligned} \tag{23}$$

for $i = j, j+1, \ldots, k-1$. By equation (22) and the definition of j, we have $x'_{i+1} = x_{i+1} = n_{i+1}$ for digit positions $i = j, j+1, \ldots, k-1$, which proves the claim.

Now we return to our proof of the lemma. By the claim, we have

$$x'_{k-1}x'_{k-2}\cdots x'_{j+1} = n_{k-1}n_{k-2}\cdots n_{j+1} \ .$$

Furthermore, by equation (23), we have $\lfloor x/p_j \rfloor = \lfloor n/p_j \rfloor$. Equation (22) and $x_j < n_j$ imply that $x'_j = x_j < n_j$, giving us

$$\begin{aligned} x' &= x'_{k-1}x'_{k-2}\cdots x'_{j+1}x'_j x'_{j-1}\cdots x'_0 \\ &= n_{k-1}n_{k-2}\cdots n_{j+1}x_j x'_{j-1}\cdots x'_0 \\ &< n_{k-1}n_{k-2}\cdots n_{j+1}n_j n_{j-1}\cdots n_0 \\ &= n \ . \end{aligned}$$

Thus, we have shown that if $x < n$, then $x' < n$. ∎

Lemma 3 *Let x and y be whole numbers such that $0 \leq x, y < n$ and $y = x + 1$. Let x' and y' be the xth and yth values, respectively, of the mixed-radix dense Gray code whose digits are given by the formula in equation (22). Then x' and y' differ in only one digit, and the values of those digits differ by* 1.

Proof: Because $y = x + 1$, there must be some leftmost digit position j such that $x_j \neq y_j$, so that $x_{k-1}x_{k-2}\cdots x_{j+1} = y_{k-1}y_{k-2}\cdots y_{j+1}$. Moreover, we must have that $y_j = x_j + 1$ and, for $i = 0, 1, \ldots, j-1$, both $x_i = r_i - 1$ and $y_i = 0$. In other words, we can view x and y as

$$\begin{array}{rcccccccc} x & = & x_{k-1} & \cdots & x_{j+1} & x_j & r_{j-1}-1 & \cdots & r_0 - 1 \quad , \\ y & = & x_{k-1} & \cdots & x_{j+1} & x_j + 1 & 0 & \cdots & 0 \quad . \end{array} \tag{24}$$

We will examine bit positions $k-1, \ldots, j+1$, bit position j, and bit positions $j-1, \ldots, 0$ of x' and y' in three separate cases.

- Bit positions $k-1, \ldots, j+1$:
  Because

  $$\begin{aligned} \lfloor x/p_i \rfloor &= x_{k-1}x_{k-2}\cdots x_{i+1} \\ &= y_{k-1}y_{k-2}\cdots y_{i+1} \\ &= \lfloor y/p_i \rfloor \end{aligned}$$

  for $i = k-1, \ldots, j+1$, equation (22) implies that

  $$x'_{k-1}x'_{k-2}\cdots x'_{j+1} = y'_{k-1}y'_{k-2}\cdots y'_{j+1} \; . \tag{25}$$

- Bit position j:
  Here, we have $\lfloor x/p_j \rfloor = x_{k-1}x_{k-2}\cdots x_{j+1}$ and $\lfloor y/p_j \rfloor = y_{k-1}y_{k-2}\cdots y_{j+1}$. Thus, we have

  $$\begin{aligned} \lfloor x/p_j \rfloor &= x_{k-1}x_{k-2}\cdots x_{j+1} \\ &= y_{k-1}y_{k-2}\cdots y_{j+1} \\ &= \lfloor y/p_j \rfloor \; . \end{aligned}$$

  Therefore, we either have $x'_j = x_j$ and $y'_j = y_j = x_j + 1$ or we have $x'_j = r_j - 1 - x_j$ and

  $$\begin{aligned} y'_j &= r_j - 1 - y_j \\ &= r_j - 1 - (x_j + 1) \\ &= r_j - 2 - x_j \; . \end{aligned}$$

  In either case, we must have that

  $$y'_j = x'_j \pm 1 \; . \tag{26}$$

- Bit positions $j-1,\ldots,0$:
  For bit positions $i = j-1,\ldots,0$, equation (24) gives

$$\begin{array}{rcccccccc} \lfloor x/p_i \rfloor & = & x_{k-1} & \cdots & x_{j+1} & x_j & r_{j-1}-1 & \cdots & r_{i+1}-1 \quad , \\ \lfloor y/p_i \rfloor & = & x_{k-1} & \cdots & x_{j+1} & x_j+1 & 0 & \cdots & 0 \quad . \end{array}$$

  Therefore, we have that $\lfloor y/p_i \rfloor = \lfloor x/p_i \rfloor + 1$, and so $\lfloor y/p_i \rfloor$ and $\lfloor x/p_i \rfloor$ have different parities. Thus, by equation (22), we either have $x'_i = x_i = r_i - 1$ and $y'_i = r_i - 1 - y_i = r_i - 1$ or we have $x'_i = r_i - 1 - x_i = 0$ and $y'_i = y_i = 0$. Either way, we have

$$x'_{j-1}x'_{j-2}\cdots x'_0 = y'_{j-1}y'_{j-2}\cdots y'_0 \,. \tag{27}$$

Combining the three cases in equations (25), (26), and (27) completes the proof. ∎

Thus, we have the following theorem.

Theorem 4 ***Correctness of the formula for generating a mixed-radix dense Gray code***
The method given by equation (22) to generate the ith digit of the mixed-radix dense Gray code of $x = 0, 1, \ldots, n-1$ produces a permutation of $\langle 0, 1, \ldots, n-1\rangle$ such that each pair of consecutive numbers in the permutation differs in just one digit, and the values of these digits differ by exactly 1.

Proof: Immediate from Lemmas 1–3. ∎

We have now proven that our method for each digit of the mixed-radix dense Gray code works for any positive integer n.

8 Special cases for the non-cyclic mixed-radix dense Gray code

Now, we show how to simplify equation (22) for the ith digit of each xth integer of the mixed-radix dense Gray code to cover two special cases: when the radix is fixed for all digit positions (all radices r_i are equal) and for binary Gray codes (when that fixed radix equals 2). We show that in the binary case, we can equate the simplified equation (22) to the previous set of formulas (5) and (7) we discovered and published in our 2016 paper [3].

In the fixed-radix case, the denominator p_i in the conditional expression of equation (22) gives $\prod_{j=0}^{i} r_i$, or r^{i+1} for the fixed radix r. Therefore, the following formula generates the ith digit d_i in the xth integer of the fixed-radix dense Gray code:

$$d_i = \begin{cases} x_i & \text{if } \lfloor x/r^{i+1} \rfloor \bmod 2 = \lfloor n/r^{i+1} \rfloor \bmod 2 \,, \\ r-1-x_i & \text{otherwise} \,. \end{cases} \tag{28}$$

Like the method for the mixed-radix dense Gray code, if we assume that the k values $r, r^2, \ldots, r^k$ have been precomputed, then we can compute each digit of each number of the fixed-radix dense Gray code in constant time.

Observe in Table 7 how we use equation (28) to generate the non-cyclic mixed-radix dense Gray code for $r = 3, n = 16$, and $k = \lfloor n/r \rfloor = 3$. We've denoted the xth element of the non-cyclic mixed-radix dense

ordinal x	mixed-radix reflected Gray code g	mixed-radix dense Gray code d	decimal counterpart
0	000	022	8
1	001	021	9
2	002	020	6
3	012	010	3
4	011	011	4
5	010	012	5
6	020	002	2
7	021	001	1
8	022	000	0
9	122	100	9
10	121	101	10
11	120	102	11
12	110	112	14
13	111	111	13
14	112	110	12
15	102	120	15

Table 7: Applying equation (22) for each digit of the first 16 numbers of the ternary reflected Gray code outputs the non-cyclic ternary reflected Gray code for $n = 16$.

Gray code as d and displayed the first n elements of the fixed-radix reflected Gray code for comparison. A close examination of the two least-significant digits in the reflected Gray code shows that both digits are in descending patterns at the cut-off. Our dense Gray code for $n = 16$ reflects digits 0 and 1 of the fixed-radix reflected Gray code to form only ascending patterns at the point where the sequence is cut. We see from the decimal counterparts that the resulting sequence is indeed dense: it is a permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$.

We now consider when the radix r is fixed at 2—that is, when we want to generate a binary dense Gray code. In this special case, equation (28) simplifies even further to equation (7). To confirm this claim, we first notice that $\lfloor x/2^{i+1} \rfloor$ gives us the binary number $x_{k-1}x_{k-2}\cdots x_{i+1}$. When we take this integer modulo 2, we are simply determining its parity, which is given by just the single bit x_{i+1}. Similarly, we have $\lfloor n/2^{i+1} \rfloor \bmod 2 = n_{i+1}$. Therefore, equation (28) reduces to just

$$x'_i = \begin{cases} x_i & \text{if } n_{i+1} = x_{i+1}\ , \\ 2 - 1 - x_i = \overline{x_i} & \text{otherwise}\ , \end{cases} \tag{29}$$

where $\overline{x_i}$ denotes the logical negation of bit x_i. We can easily see that formulas (7) and (29) are equivalent by examining two cases for each bit $i = 0, 1, \ldots, k-1$:

- Case 1: $n_{i+1} = x_{i+1}$. By equation (7), we have

$$\begin{aligned} x'_i &= x_i \oplus x_{i+1} \oplus n_{i+1} \\ &= x_i \oplus x_{i+1} \oplus x_{i+1} \\ &= x_i\ , \end{aligned}$$

  which matches the first case in equation (29).

- Case 2: $n_{i+1} \neq x_{i+1}$. Here, we have

$$\begin{aligned} x'_i &= x_i \oplus x_{i+1} \oplus n_{i+1} \\ &= x_i \oplus x_{i+1} \oplus \overline{x_{i+1}} \\ &= \overline{x_i} \ , \end{aligned}$$

  which matches the second case in equation (29).

Thus, we see that our earlier method for finding non-cyclic binary dense Gray codes is really just a special case of the method in Section 7 for finding non-cyclic mixed-radix dense Gray codes.

Part III
Cyclic Gray codes in fixed and mixed radices

In this final part, we present methods to generate cyclic mixed-radix Gray codes for any k-tuple of radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and any positive integer $n \leq p_{k-1}$, where as before, $p_i = \prod_{j=0}^{i} r_j$, so that the Gray code produced is a permutation of the sequence $\langle 0, 1, \ldots, n-1 \rangle$. The task of computing a cyclic mixed-radix Gray code is quite hard, so we will weaken our Gray-code property from strict to *modular*, where each integer in the Gray code differs from the preceding integer in only one digit position i, and the values of those digits are either 0 and $r_i - 1$, or they differ by exactly 1. For the remainder of this part, we will use this definition of the Gray-code property to discuss mixed-radix sequences.

Section 9 examines existing literature on cyclic mixed-radix Gray codes. Previously, all known methods for producing a mixed-radix Gray code with the modular Gray-code and cyclic properties were constrained in two ways: first, they produce only *full* Gray codes, where $n = p_{k-1}$; and second, in order to guarantee the full Gray code, they further restrict the radix tuple to only certain values and forms. In Section 10, we will obviate the latter restriction and provide a recursive method to generate a full Gray code for any tuple of mixed radices. Although we are unable to do so for the former restriction, we do provide in Section 11 a graph-theoretic approach to thinking about cyclic mixed-radix non-full Gray codes or, equivalently, cyclic mixed-radix dense Gray codes where n can be any positive integer less than or equal to p_{k-1}. Following this new line of thinking, we then build a list of cases for which we prove it is impossible to generate a cyclic mixed-radix dense Gray code for a particular set of radices and a positive integer n.

9 Previous work for cyclic mixed-radix full Gray codes

Here, we compare previous attempts to generate cyclic mixed-radix full Gray codes. As we mentioned in the introduction to this part, all three methods listed in this section place restrictions upon the mixed-radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ in order to guarantee the cyclic property for the resulting Gray code. We have already seen the first of the three: Er's reflected mixed-radix Gray code [4]. We will use the pattern (18) that digit $k-1$ takes and the pattern (19) that digits $0, 1, \ldots, k-2$ take in Er's mixed-radix sequence to show that Er's method generates a cyclic mixed-radix full Gray code if and only if r_{k-1} is even. The second algorithm we will describe is Cohn's method for a modular fixed-radix Gray code [2], which, when generalized to mixed radices, produces a cyclic full Gray code if and only if each radix r_{i+1} is a multiple of the radix r_i for $i = 0, 1, \ldots, k-2$. Finally, we compare Er's and Cohn's sequences to Anantha and AlBdaiwi's modular Gray code [1], which combines both Er's and Cohn's methods to generate a cyclic mixed-radix full Gray code if and only if each radix has an equal or larger value than the less significant radices and all radices in the tuple share the same parity.

Er's mixed-radix full Gray code

Using the patterns (18) and (19) that we derived for Er's mixed-radix full Gray code in Section 6, we now prove the single case in which Er's method generates a modular cyclic full Gray code.

Theorem 5 ***Modular cyclicity of Er's mixed-radix full Gray code***
Given the radix tuple $(r_{k-1}, r_{k-2}, \ldots, r_0)$*, the pattern (18) that digit* $k-1$ *takes and the pattern (19) that*

digits $0, 1, \ldots, k-2$ *take in Er's mixed-radix full Gray code produce a sequence with the modular cyclic property if and only if* r_{k-1} *is even.*

Proof: To get the modular cyclic property, we must show that the 0th and last integers generated by pattern (18) for digit $k-1$ and pattern (19) for digit i, where $i = 0, 1, \ldots, k-2$, differ in only one digit position j, and the values of those digits are either 0 and $r_j - 1$, or they differ by only 1. To do so, we will examine the digit position $k-1$ and the digit positions $0, 1, \ldots, k-2$ separately.

For digit position $k-1$, which follows the length-p_{k-1} pattern (18), the 0th digit is 0, and the last digit is $r_{k-1} - 1$. Therefore, digit $k-1$ is the digit that differs, and as a result, we can have the modular cyclic property only if, for the remaining digit positions $i = 0, 1, \ldots, k-2$, the 0th and last values of digit i do not differ.

We now examine those digit positions $i = 0, 1, \ldots, k-2$, each of which follows the length-$2p_i$ digit pattern (19) until the constructed sequence has length p_{k-1}. Clearly by pattern (19), the 0th value of each digit i is 0. If the modular cyclic property is to hold, then the last value of each digit must also be 0. This attribute occurs only when p_{k-1}, which is an integer multiple of p_i, is also an integer multiple of $2p_i$ so that the length-p_{k-1} sequence being constructed cuts off exactly when pattern (19) finishes and does not cut off after the first half of pattern (19). Thus, we have that p_{k-1} must be an integer multiple of $2p_{k-2}$, and so $p_{k-1}/p_{k-2} = r_{k-1}$ must be even. Thus, an even radix r_{k-1} is necessary to show that Er's Gray code has the modular cyclic property.

To prove that even r_{k-1} is also sufficient to show that Er's Gray code has the modular cyclic property, consider again the digit positions $0, 1, \ldots, k-2$. Here, we must show that if r_{k-1} is even, then p_{k-1} is an integer multiple of $2p_i$ for $i = 0, 1, \ldots, k-2$, and therefore, the last value of each digit i is 0. We first notice that because p_{k-1} is an integer r_{k-1} multiple of p_{k-2}, the radix r_{k-1} is even, and r_{k-1} is greater than or equal to 2 by nature of being a radix, then p_{k-1} is also an integer multiple of $2p_{k-2}$. Since p_{k-1} is an integer multiple of $2p_{k-2}$, then because

$$\begin{aligned} p_{k-2} &= r_{k-2} \cdot r_{k-3} \cdot \ldots \cdot r_i \cdot r_{i-1} \cdot \ldots \cdot r_0 \\ &= r_{k-2} \cdot r_{k-3} \cdot \ldots \cdot r_i \cdot p_i \\ &= p_i \textstyle\prod_{j=i+1}^{k-2} r_j \ , \end{aligned} \tag{30}$$

we must also have that p_{k-1} is an integer multiple of $2p_i$ for $i = 0, 1, \ldots, k-3$. Thus, having that r_{k-1} is even is both necessary and sufficient to show modular cyclicity in Er's Gray code. ∎

Cohn's mixed-radix full Gray code

Here, we start with Cohn's method for a fixed-radix full Gray code [2], which guarantees a full sequence with the modular Gray-code and cyclic properties for any fixed radix r, and later, we will generalize his method to build a mixed-radix full Gray code that is cyclic for radix tuples of a specific form. Let r be the fixed radix, and let k be the number of digits we will compute for Cohn's fixed-radix Gray code, so that $n = r^k$. Then, like the digit positions in the ordinal sequence for r and n, each digit position $i = 0, 1, \ldots, k-1$ in Cohn's Gray code follows (r_{i+1})-length patterns, except instead of being simple ascending sequences, the

ordinal (decimal)	ordinal (fixed-radix)	Cohn's Gray code
0	000	000
1	001	001
2	002	002
3	010	012
4	011	010
5	012	011
6	020	021
7	021	022
8	022	020
9	100	120
10	101	121
11	102	122
12	110	102
13	111	100
14	112	101
15	120	111
16	121	112
17	122	110
18	200	210
19	201	211
20	202	212
21	210	222
22	211	220
23	212	221
24	220	201
25	221	202
26	222	200

Table 8: Cohn's fixed-radix full Gray code for radix $r = 3$. The sequence has the modular Gray-code and cyclic properties.

digit patterns in Cohn's sequence are of the form

$$\begin{aligned} (m \bmod r)^{r^i} \\ ((1+m) \bmod r)^{r^i} \\ ((2+m) \bmod r)^{r^i} \\ \vdots \\ ((r-1+m) \bmod r)^{r^i} \ , \end{aligned}$$

where m is an integer offset used in conjunction with the modulus operator to cyclically rotate the standard ascending pattern (14) by mr^i positions. We can observe this pattern in Table 8, which shows Cohn's fixed-radix full Gray code for $r = 3$ and $k = 3$, so that $n = r^k = 27$.

Cohn [2] shows how to compute each integer of his Gray code from just its ordinal index in the sequence using matrix-vector multiplication, where the ordinal index is a k-digit vector that is multiplied with a $k \times k$ transformation matrix to compute the corresponding integer in Cohn's code. Alternatively, Sharma and Khanna [9] provide a set of formulas that generate each digit of the xth number x' of Cohn's fixed-radix

sequence. Let x have the digit representation $x_{k-1}x_{k-2}\cdots x_0$, and let x' have the digit representation $x'_{k-1}x'_{k-2}\cdots x'_0$. Then, the formulas are as follows:

$$x'_{k-1} = x_{k-1} , \tag{31}$$

$$x'_i = (x_i - x_{i+1}) \bmod r \quad \text{for } i = 0, 1, \ldots, k-2 , \tag{32}$$

and each digit of x' can be calculated in constant time.

We can easily use equations (31) and (32) of Cohn's fixed-radix full Gray code to produce analogous formulas for Cohn's mixed-radix full Gray code for the k-tuple of radices $(r_{k-1}, r_{k-2}, \ldots, r_0)$. If x' is the xth number of Cohn's mixed-radix full Gray code, then keeping equation (31) as before, so that $x'_{k-1} = x_{k-1}$, then we can modify equation (32) to get

$$x'_i = (x_i - x_{i+1}) \bmod r_i \quad \text{for } i = 0, 1, \ldots, k-2 . \tag{33}$$

Table 9 shows Cohn's mixed-radix full Gray code when we use equations (31) and (33) to generate all digits in the sequence. Notice that not only is the resulting sequence modularly cyclic, but also the radix tuple $(4, 4, 2)$ has the property that each radix is an integer multiple of the radix that is immediately less significant. The following theorem shows that this condition on the radix tuple is both necessary and sufficient for equations (31) and (33) to generate a mixed-radix full sequence with the modular cyclic property.

Theorem 6 ***Modular cyclicity of Cohn's mixed-radix full Gray code***
Given the radix tuple $(r_{k-1}, r_{k-2}, \ldots, r_0)$, Cohn's mixed-radix full Gray code, produced by equations (31) and (33) when applied to each integer in the ordinal sequence $\langle 0, 1, \ldots, p_{k-1} - 1\rangle$, has the modular cyclic property if and only if, for each digit position $i = 0, 1, \ldots, k-2$, the radix r_{i+1} is an integer multiple of r_i.

Proof: Let x' be the solution to equations (31) and (33) when $x = 0$, so that x' is the 0th integer generated in Cohn's mixed-radix sequence. Let y' be the solution to equations (31) and (33) when we use $y = p_{k-1} - 1$ in place of x, so that y' is the last integer generated in Cohn's mixed-radix sequence. Then, to get the modular cyclic property, we must show that x' and y' differ in only one digit position j, and those digits are either 0 and $r_j - 1$, or they differ by only 1.

First, let us examine the digit representation $x'_{k-1}x'_{k-2}\cdots x'_0$ of x' when $x = 0$. From equation (31), we have $x'_{k-1} = x_{k-1} = 0$. Then, for $i = k-2, k-3, \ldots, 0$, we have

$$\begin{aligned} x'_i &= (x_i - x_{i+1}) \bmod r_i && \text{(by equation (33))} \\ &= (0 - 0) \bmod r_i && \text{(by } x = 0\text{)} \\ &= 0 , \end{aligned}$$

giving 0 as the value of all the digits of x'.

Now, we examine the digit representation $y'_{k-1}y'_{k-2}\cdots y'_0$ of y', which is a harder problem. To calculate each digit of y', we evaluate equations (31) and (33), but we replace x with $y = p_{k-1} - 1$, which has the digit representation

$$\begin{aligned} y &= y_{k-1} \quad y_{k-2} \quad \cdots \quad y_0 \\ &= r_{k-1} - 1 \quad r_{k-2} - 1 \quad \cdots \quad r_0 - 1 \quad . \end{aligned} \tag{34}$$

By equations (31) and (34), we have $y'_{k-1} = y_{k-1} = r_{k-1} - 1$. Thus, digit $k-1$ is the digit that differs, and as a result, we can have the modular cyclic property only if, for the remaining digits $i = 0, 1, \ldots, k-2$, the values of x'_i and y'_i do not differ.

ordinal (decimal)	ordinal (mixed-radix)	Cohn's Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	020	020
5	021	021
6	030	031
7	031	030
8	100	130
9	101	131
10	110	101
11	111	100
12	120	110
13	121	111
14	130	121
15	131	120
16	200	220
17	201	221
18	210	231
19	211	230
20	220	200
21	221	201
22	230	211
23	231	210
24	300	310
25	301	311
26	310	321
27	311	320
28	320	330
29	321	331
30	330	301
31	331	300

Table 9: Cohn's mixed-radix full Gray code for the 3-tuple of radices $(4, 4, 2)$. Because every radix r_i, where $i = 1, 2$, is an integer multiple of radix r_{i-1}, this sequence has the modular cyclic property.

	Er	Cohn	Anantha and AlBdaiwi	ours
Gray-code property	strict	modular	modular	strict
modular cyclic property	yes, if r_{k-1} is even	yes, if for $i = 0, 1, \ldots, k-2$, r_i divides r_{i+1}	yes, if for $i = 0, 1, \ldots, k-2$, $r_{i+1} \geq r_i$ and $r_{i+1} \bmod 2 = r_i \bmod 2$	yes
formula for each digit	yes, equation (22)	yes, equations (31) and (33)	yes, not shown	no

Table 10: Comparison of four different methods for a cyclic mixed-radix full Gray code: Er's [4], Cohn's [2], Anantha and AlBdaiwi's [1], and ours (Section 10).

We now examine the values of digit y'_i for $i = 0, 1, \ldots, k-2$, which are given by the following equation:

$$\begin{aligned} y'_i &= (y_i - y_{i+1}) \bmod r_i && \text{(by equation (33))} \\ &= ((r_i - 1) - (r_{i+1} - 1)) \bmod r_i && \text{(by equation (34))} \\ &= (r_i - r_{i+1}) \bmod r_i \\ &= -r_{i+1} \bmod r_i \ . \end{aligned} \tag{35}$$

Recall that in order for Cohn's mixed-radix sequence to be cyclic, we must have that $y'_i = x'_i = 0$ for $i = 0, 1, \ldots, k-2$. Therefore, by equation (35), we must have $-r_{i+1} \bmod r_i = 0$, which occurs if and only if there exists an integer ℓ such that $\ell r_i - r_{i+1} = 0$. Thus, we must have $r_{i+1} = \ell r_i$, and therefore, r_{i+1} must be an integer multiple of r_i for $i = 0, 1, \ldots, k-2$, which we have now proven is necessary and sufficient to show that equations (31) and (33) produce a sequence with the modular cyclic property. ■

Anantha and AlBdaiwi's mixed-radix full Gray code

Now that we've seen both Er's and Cohn's mixed-radix Gray codes, we briefly mention Anantha and AlBdaiwi's method [1] for a mixed-radix full Gray code, which combines both Er's and Cohn's methods to produce a formula that computes each digit of their sequence. Anantha and AlBdaiwi show that their method can guarantee cyclicity if and only if each radix in the radix tuple has an equal or larger value than its less significant radices and all radices have the same parity [1].

Table 10 compares each method for a cyclic mixed-radix full Gray code discussed in this section, summarizing their restrictions on the radix tuple $(r_{k-1}, r_{k-2}, \ldots, r_0)$. In the table, we also include our own recursive method for a cyclic mixed-radix full Gray code, which we describe and prove correct in the following section.

10 A recursive method for a cyclic mixed-radix full Gray code

Now that we have seen previous attempts to generate a cyclic mixed-radix full Gray code, along with the cases in which they do and do not work, we present a novel recursive algorithm that computes this

target sequence for any k-tuple of mixed radices. Our algorithm, which we list below as the procedure RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE, takes three arguments: (1) the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$; (2) the precomputed k-tuple of values $p = (p_{k-1}, p_{k-2}, \ldots, p_0)$, which is easy to calculate in $\Theta(k)$ time; and (3) a most significant digit position j that the procedure will recurse on to generate and return the final sequence of j-digit integers. In order to simplify the discussion of our algorithm's correctness, we now use the integer p_j instead of n to refer to the length of the sequence generated by RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p), since this term lends itself more easily to the logic of our proofs.

Notice that because j represents the most significant bit position that will be generated in the output Gray code, calling the procedure with $j = k - 1$ effectively produces the cyclic mixed-radix full Gray code for length p_{k-1}. In the following algorithm, we define the operator $\|$ to be the prepend or concatenation operator between a digit a and a mixed-radix integer b, so that if b has the digit representation $b_{j-1}b_{j-2}\cdots b_0$, then $a \parallel b$ has the digit representation $a\, b_{j-1}b_{j-2}\cdots b_0$. Finally, although we construct *result* by appending to it, we assume that once we've returned the complete *result* array, we can then index into it in the usual way for arrays (i.e., *result*$[i]$ denotes the ith integer in *result*).

RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p)

```
 1  initialize result to an empty list
 2  if j == 0
 3      // base case: compute the p_0-length Gray code
 4      for i = 0 to r_0 − 1
 5          append i to result
 6  else // recursively compute the p_{j−1}-length mixed-radix Gray code
 7      prev = RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j − 1, r, p)
 8      // prev has length p_{j−1}. Fill in the first p_{j−1} values of result with the values in prev
 9      for i = 0 to p_{j−1} − 1
10          append 0 ‖ prev[i] to result
11      // for each value in the previous Gray code, going back to front, append to it a new
12      // most significant digit, alternating between the digit ascending and descending,
13      // going between 1 and r_j − 1
14      ascending = TRUE
15      for i = p_{j−1} − 1 downto 0
16          if ascending
17              for ℓ = 1 to r_j − 1
18                  append ℓ ‖ prev[i] to result
19          else for ℓ = r_j − 1 downto 1
20                  append ℓ ‖ prev[i] to result
21          ascending = not ascending
22  return result
```

Table 11 shows the output *result* when the procedure is called on the integer $j = 2$, the mixed-radix tuple $r = (3, 2, 5)$, and the product-of-radices tuple $p = (30, 10, 5)$. If concatenating a single digit to a $(j-1)$-digit number takes constant time for any positive integer j, then the algorithm requires only $\Theta(n)$ time to generate and return the cyclic mixed-radix full Gray code for n and r. In another case, if concatenating a single digit to a $(j-1)$-digit integer takes time linear to the total number of digits j, then

ordinal (decimal)	ordinal (mixed-radix)	our Gray code
0	000	000
1	001	001
2	002	002
3	003	003
4	004	004
5	010	014
6	011	013
7	012	012
8	013	011
9	014	010
10	100	110
11	101	210
12	102	211
13	103	111
14	104	112
15	110	212
16	111	213
17	112	113
18	113	114
19	114	214
20	200	204
21	201	104
22	202	103
23	203	203
24	204	202
25	210	102
26	211	101
27	212	201
28	213	200
29	214	100

Table 11: RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE$(2, (3, 2, 5), (30, 10, 5))$ produces our cyclic mixed-radix full Gray code for the 3-tuple of radices $(3, 2, 5)$.

it requires $\Theta(nk)$ time to produce the cyclic mixed-radix full Gray code for n and the k-tuple r.

Proof of our method for a cyclic mixed-radix full Gray code

We will now prove that the above algorithm correctly generates the cyclic mixed-radix full Gray code for the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and the most significant bit position j. To do so, we need to prove several properties of the sequence returned by the procedure, which we will do in the following order:

- The sequence has length p_j.
- Each integer in the sequence has $j + 1$ digits.
- Each integer in the sequence is in the range 0 to $p_j - 1$.
- Each integer in the sequence is unique.
- The sequence obeys the strict Gray-code property, so that each number in the sequence differs from the preceding number in exactly one digit position i, and the values of those digits differ by exactly 1.
- The sequence obeys the modular cyclic property, so that the last and first numbers in the sequence differ in exactly one digit, and the values of those digits are either 0 and $r_i - 1$, or they differ by exactly 1.

We start with the basics. At the very least, RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) should return a sequence of the length that we expect, and each integer in that sequence should be represented using the number of digits that we expect. Lemmas 7 and 8 below prove that our procedure fulfills these two properties necessary for developing a cyclic mixed-radix dense Gray code.

Lemma 7 RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) *produces a sequence of p_j integers.*

Proof: There are only two cases to consider. The first case occurs when $j = 0$, so we execute lines 4–5, which simply store store the values 0 to $r_0 - 1$ as the $r_0 = p_0 = p_j$ integers of *result*. In the other case, we first execute lines 9–10, which fill *result* with the first p_{j-1} integers. Then, we enter a for-loop in line 15, which contributes a factor of p_{j-1}, and within this for-loop, we execute exactly one of the inner for-loops in lines 17–18 or 19–20. Each execution of an inner for-loop appends $r_j - 1$ integers to *result*. Therefore, the length of *result* is given by

$$\begin{aligned} p_{j-1} + p_{j-1}(r_j - 1) &= p_{j-1}(1 + r_j - 1) \\ &= p_{j-1} \cdot r_j \\ &= p_j \ . \end{aligned}$$

In both cases, the call RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) generates a length-p_j sequence. ∎

Lemma 8 *Each integer that* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) *produces is represented with $j + 1$ digits.*

Proof: This proof is by induction on j. For $j = 0, 1, \ldots, k-1$, the inductive hypothesis is that RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) produces a sequence of integers that are all represented with $j+1$ digits. In the base case, where $j = 0$, we execute lines 4–5, which simply create *result* as a list of 1- or $(j+1)$-digit integers. Otherwise, in the inductive step, we execute lines 7–21. By the inductive hypothesis, the recursive call in line 7 returned *prev* with $((j-1)+1)$- or j-digit integers. After the execution of line 7, whenever we build a number to append to *result*, we concatenate one digit to a number in *prev*, forming integers with $j+1$ digits and proving the inductive step. ∎

Now that we've shown that RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) generates a sequence of proper length, with each integer of proper digit length, we can move on to prove the harder requirements of a cyclic mixed-radix full Gray code. In the following lemmas, we use ordinal numbers x and y to index into the sequence produced by RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p), and we let x' and y' be the output values *result*$[x]$ and *result*$[y]$, respectively.

Lemma 9 *Let x be an ordinal index such that $0 \le x < p_j$, and let x' be result$[x]$, where result is generated by* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p). *Then $0 \le x' < p_j$.*

Proof: By Lemma 8, we have that x' is a $(j+1)$-digit integer. Therefore, we let the $(j+1)$-digit representations of x' and $p_j - 1$ be the following:

$$\begin{array}{rcccccc} x' & = & x'_j & x'_{j-1} & \cdots & x'_0 & , \\ p_j - 1 & = & r_j - 1 & r_{j-1} - 1 & \cdots & r_0 - 1 & , \end{array} \tag{36}$$

so that $x' = \sum_{i=0}^{j} x'_i p_{i-1}$ and $p_j - 1 = \sum_{i=0}^{j} (r_i - 1) p_{i-1}$. We will use induction on j, where $0 \le j \le k-1$, to show $0 \le x'_i \le r_i - 1$ for $i = 0, 1, \ldots, j$. This quality will then be sufficient to prove $0 \le x' \le p_j - 1 < p_j$.

For $j = 0, 1, \ldots, k-1$, the inductive hypothesis is that $0 \le x'_i \le r_i - 1$ for $i = 0, 1, \ldots, j$. In the base case, where $j = 0$, we execute lines 4–5, which append the 1-digit integers x' to *result* such that $0 \le x'_0 \le r_0 - 1$. Otherwise, in the inductive step, we execute lines 7–21. By Lemma 8, each integer in *prev* returned by the recursive call in line 7 has j digits. Therefore, whenever we build a number x' to append to *result*, we concatenate the jth digit x'_j to a number in *prev*, where $0 \le x'_j \le r_j - 1$ by lines 10 and 17–20. By the inductive hypothesis, we also have $0 \le x'_i \le r_i - 1$ for $i = 0, 1, \ldots, j-1$. When we put the two equations together, we get $0 \le x'_i \le r_i - 1$ for $i = 0, 1, \ldots, j$, which proves the inductive step. ∎

Lemma 10 *Let x and y be ordinal indices such that $0 \le x, y < p_j$ and $x \ne y$. Let x' and y' be result$[x]$ and result$[y]$, respectively, where* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) *generates result. Then $x' \ne y'$.*

Proof: This proof is by induction on j. For $j = 0, 1, \ldots, k-1$, the inductive hypothesis is that RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) produces integers x' and y' such that if $x \ne y$, then $x' \ne y'$. In the base case, where $j = 0$, we execute lines 4–5, which append the integers 0 to $r_0 - 1$ just once each to our output sequence *result*, and therefore, given any two ordinal positions x and y such that $x \ne y$, we must also have $x' \ne y'$.

In the inductive step, we execute lines 7–21. By the inductive hypothesis, the recursive call returned the array *prev* with all values distinct. We will show that if the same digit is prepended to two values from *prev*, then these two values from *prev* must be unequal. If 0 is prepended, then it must have been prepended in the for-loop of lines 9–10, and therefore, each value of *prev*$[i]$ must be distinct. Otherwise, if two equal values of $\ell \geq 1$ are prepended in the for-loop of lines 15–21, then they must have been prepended in different iterations of this for-loop, so that the value of i differs and, hence, the values of *prev*$[i]$ differ.

In all cases where we are given $x \neq y$, we have $x' \neq y'$. Therefore, we have the inductive step. ■

Lemma 11 *Let x and y be ordinal indices such that $0 \leq x, y < p_j$ and $y = x - 1$. Let x' and y' be result$[x]$ and result$[y]$, respectively, where* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) *generates result. Then x' and y' differ in only one digit, and the values of those digits differ by only* 1.

Proof: This proof is by induction on j. For $j = 0, 1, \ldots, k-1$, the inductive hypothesis is that RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p) produces integers x' and y' such that if $y = x - 1$, then x' and y' differ in only one digit, and the values of those digits differ by only 1. To aid our argument, we define an *append operation* to be the exact instruction within the sequential algorithm at which a new integer is appended to *result*, so that given $y = x-1$, we have that the append operation that appends y' is the one that occurs immediately before the append operation that appends x' to *result*.

In the base case, where $j = 0$, we execute lines 4–5. Here, for every ordinal position x such that $y = x - 1$ is also an ordinal position, the append operation for x' adds the 1-digit integer i to *result*, and the append operation for y', which occurs immediately before, adds the 1-digit integer, $i - 1$. Therefore, x' and y' differ in only one digit, and those digits differ by only 1.

Otherwise, in the inductive step, we execute lines 7–21. Now, there are many different variables we have to consider to prove the inductive step. We must first analyze what section of the procedure we are in when we append x' and y' to *result*. Are we in lines 9–10, are we in lines 14–21, or are we in lines 9–10 when we append y' and lines 14–21 when we append x'? Then, within these cases, we must consider either the value of the local variable i if we are in lines 9–10, or the values of local variables i, ℓ, and *ascending* if we are in lines 14–21. In the following paragraphs, we examine these variables and prove the inductive step through case exhaustion.

We start with the case where both the append operations for x' and y' come from lines 9–10. Line 9 shows that i increments by 1 every time we append to *result*, so that the value of i during the append operation for x' must be exactly 1 more than the value of i during the preceding append operation for y'. Let us define $\widehat{i}$ to be the value of i during the append operation of x'. Then, $\widehat{i} - 1$ is the value of i during the append operation of y' and, by the inductive hypothesis, we have that *prev*$[\widehat{i}]$ and *prev*$[\widehat{i} - 1]$ differ in only one digit, and the values of those digits differ by only 1. Thus, $x' = 0 \parallel$ *prev*$[\widehat{i}]$ and $y' = 0 \parallel$ *prev*$[\widehat{i} - 1]$ also differ in only one digit, and the values of those digits also differ by only 1.

Next, we analyze the case where the append operation for y' comes from lines 9–10 and where the append operation for x' comes from lines 14–21. In this case, we must have that y' is the last integer appended to *result* from lines 9–10, and x' is the first integer appended to *result* from lines 14–21. By lines 9–10, we have $y' = 0 \parallel$ *prev*$[p_{j-1} - 1]$, and by lines 14–21, we have that $i = p_{j-1} - 1$, *ascending* = TRUE, and consequently, $\ell = 1$ during the append operation for x'. Putting these facts together, we get $x' = 1 \parallel$ *prev*$[p_{j-1} - 1]$. Thus, x' and y' differ only in their most significant digit, and the values of those digits differ by 1.

The last case is the most complex: the append operations for x' and y' both come from lines 14–21. Within this section of the procedure, if the append operations for x' and y' come from the same inner for-loop—that is, they both come from either lines 17–18 or lines 19–20—then because the append operation for y' directly precedes the one for x', we must have that i and *ascending* are the same for both operations, while ℓ differs by 1. Therefore, *prev*$[i]$ is the same integer for both append operations, and prepending ℓ to *prev*$[i]$ to compute x' and y' gives two integers that differ in only their most significant digit, with the values of those digits differing by only 1. In the other subcase, the append operations for x' and y' come from different inner for-loops—that is, one comes from lines 17–18 and the other comes from lines 19–20. Here, because one append operation directly precedes the other, we must have that i and *ascending* differ for both operations, and furthermore, the value of i in the two append operations differs by only 1. Let $\widehat{i}$ be the value of i during the append operation for x'. Then $\widehat{i} - 1$ is the value of i during the append operation for y', and by the inductive hypothesis, we have that *prev*$[\widehat{i}]$ and *prev*$[\widehat{i} - 1]$ differ in only one digit, and the values of those digits differ by 1. Now, we must show that when we form the integers x' and y', the most significant digits that we append to the integers *prev*$[\widehat{i}]$ and *prev*$[\widehat{i} - 1]$ do not differ. We have two cases for this most significant digit ℓ. The first is if the append operation for y' is the last operation executed in its instance of the for-loop in lines 17–18, so that *ascending* switches from TRUE to FALSE, and the append operation for x' becomes the first operation executed in the following for-loop at lines 19–20. In this case, ℓ is $r_j - 1$ for y' and remains the same for x'. The second case is if the append operation for y' is the last operation executed in its instance of the for-loop in lines 19–20, so that *ascending* switches from FALSE to TRUE, and the append operation for x' becomes the first operation executed in the following for-loop at lines 17–18. Here, ℓ is 1 for y' and remains the same for x'. Thus, in both cases we append the same value of ℓ to the integers *prev*$[\widehat{i}]$ and *prev*$[\widehat{i} - 1]$ to form x' and y', respectively. As we noticed for *prev*$[\widehat{i}]$ and *prev*$[\widehat{i} - 1]$, the integers x' and y' must also differ in one digit, and the values of the digits that differ must also be 1.

In all cases where we are given $y = x - 1$, we have that x' and y' differ in one digit, and the values of the digits that differ is 1. Therefore, we have the inductive step. ∎

Lemma 12 *Let result be the array returned by* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(j, r, p). *Then the* 0*th integer result*[0] *and the last integer result*$[p_j - 1]$ *differ in only one digit position* i*, and the values of those digits are either* 0 *and* $r_j - 1$*, or they differ by only* 1.

Proof: There are only two cases to consider. The first case occurs when $j = 0$, so we execute lines 4–5, which append the 1-digit values 0 and $r_j - 1$ as *result*[0] and *result*$[r_j - 1]$ = *result*$[p_j - 1]$, respectively. In the other case, we execute lines 7–21. Clearly, we have *result*[0] = 0 ∥ *prev*[0] by lines 9–10, and we have *result*$[p_j - 1] = \ell \parallel$ *prev*[0] by lines 14–21. Let $\widehat{\ell}$ be the value of ℓ that we concatenate to *prev*[0] to form *result*$[p_j - 1]$. By lines 17 and 19, we have either $\widehat{\ell} = 1$ or $\widehat{\ell} = r_j - 1$. If we have the former, then *result*[0] and *result*$[p_j - 1]$ differ in only the most significant digit, and those digits differ by 1. Otherwise, we have the latter, and *result*[0] and *result*$[p_j - 1]$ again differ in the most significant digit, with the most significant digit of *result*[0] equal to 0 and the most significant digit of *result*$[p_j - 1]$ equal to $r_j - 1$. ∎

Theorem 13 ***Correctness of the procedure* RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE**
The procedure RECURSIVE-CYCLIC-MIXED-RADIX-GRAY-CODE$(j, (r_{k-1}, \ldots, r_0), (p_{k-1}, \ldots, p_0))$ *generates a cyclic mixed-radix full Gray code for the radices* $(r_{k-1}, r_{k-2}, \ldots, r_0)$.

Proof: Lemmas 7–10 show that the procedure generates a full sequence, and Lemmas 11–12 show that the output sequence also has the strict Gray-code and modular cyclic properties. ∎

To complement our recursive method of generating the cyclic mixed-radix full Gray code, we provide an iterative algorithm that computes the same output sequence as the recursive one does when, for the former procedure, we use k as the first argument in place of j. This iterative method iterates on j instead of recursing on it. It also fills *result* with the completed p_j-length sequence for each iteration of j instead of returning the previous solution from a function call and then copying over the list elements, just as the recursive procedure does. In this way, the iterative procedure listed below is more space-efficient than its recursive counterpart, requiring only $\Theta(p_{k-1})$ space for the single instance of *result*.

ITERATIVE-CYCLIC-MIXED-RADIX-GRAY-CODE(k, r, p)

```
 1  initialize result to an empty list
 2  // compute the p_0-length Gray code
 3  for i = 0 to p_0 − 1
 4      append i to result
 5  // iteratively compute the p_j-length mixed-radix Gray code for j = 1, ..., k − 1
 6  for j = 1 to k − 1
 7      // for each value in the previous Gray code, going back to front, append to it a new
 8      // most significant digit, alternating between the digit ascending and descending,
 9      // going between 1 and r_j − 1
10      ascending = TRUE
11      for i = p_{j−1} − 1 downto 0
12          if ascending
13              for ℓ = 1 to r_j − 1
14                  append ℓ ‖ result[i] to result
15          else for ℓ = r_j − 1 downto 1
16                  append ℓ ‖ result[i] to result
17          ascending = not ascending
18  return result
```

Like the recursive version of this method, this iterative algorithm requires $\Theta(n)$ time to generate a cyclic mixed-radix full Gray code for n and r if concatenating a single digit to the digit representation of another integer takes constant time regardless of the total number of digits. Otherwise, if concatenating a digit to the digit representation of an integer takes time linear to the total number of digits in the output, then the above algorithm requires $\Theta(nk)$ time to complete a cyclic mixed-radix full Gray code for n and the k-tuple r.

Having provided both a recursive and an iterative procedure to generate cyclic mixed-radix full Gray codes for any tuple of mixed radices, we now move on to consider cyclic dense Gray codes.

11 Cyclic mixed-radix dense Gray codes as Hamiltonian cycles

Unlike what we did in Section 9 for the cyclic mixed-radix full Gray code, we are unable to provide a polynomial-time solution for a cyclic mixed-radix dense Gray code, where, given a radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and a positive integer $n < p_{k-1}$, a cyclic dense Gray code is a permutation of the

sequence $\langle 0, 1, \ldots, n-1 \rangle$ that holds the modular Gray-code and cyclic properties. Thus, instead of providing a solution, this section discusses a graph-theoretical approach we can use to model the cyclic dense Gray code. The following section will use this new approach to prove several cases of r and n for which generating a cyclic mixed-radix dense Gray code is impossible.

We start by defining our model. A *modular Gray graph* for the radix tuple r and the positive integer $n \leq p_{k-1}$ is an undirected graph $G = (V, E)$, where $|V| = n$, such that each vertex $v \in V$ is a unique integer from the set $\{0, 1, \ldots, n-1\}$, and E is the set of edges (u, v) for all $u, v \in V$, such that u and v hold the modular Gray-code property—that is, u and v differ in only one digit position i, and the values of those digits are either 0 and $r_i - 1$, or they differ by only 1. With such a graphical representation of the integers 0 to $n-1$ and the modular Gray-code property, we can equate the problem of generating a cyclic mixed-radix dense Gray code to the task of producing a Hamiltonian cycle—a cycle that traverses each vertex $v \in V$ exactly once—in the modular Gray graph for the corresponding radices r and integer n. Unfortunately, the Hamiltonian cycle problem is NP-complete [7], and even with the special attributes of our modular Gray graph, we are unable to find an algorithm that takes polynomial time in the worst case. Thus, the search for a Hamiltonian cycle, and therefore a cyclic mixed-radix dense Gray code, is infeasible for most cases of n. As we mentioned, however, there are some special attributes that we can observe about modular Gray graphs, especially for cyclic mixed-radix full Gray codes, when $n = p_{k-1}$. This section lists those observations, and Section 12 will relate them to modular Gray graphs for cyclic mixed-radix dense Gray codes, where $n < p_{k-1}$.

Modular Gray graphs for cyclic mixed-radix full Gray codes

We first examine the modular Gray graphs for cyclic mixed-radix full Gray codes, since they are easy to generate and exhibit notable patterns of symmetry. Anantha and AlBdaiwi [1] already introduced modular Gray graphs, which they named "multidimensional mixed-radix tori," when they showed how to generate their cyclic mixed-radix full Gray code, but they did not describe the graphs in detail. Here, we expand upon the graphs where Anantha and AlBdaiwi left off. We draw modular Gray graphs for different sets of r and $n = p_{k-1}$, and then we use our recursive method for a cyclic mixed-radix full Gray code to visualize a Hamiltonian cycle in that graph.

First, we consider how to construct the modular Gray graph for a cyclic mixed-radix full Gray code. Given the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$ and the integer $n = p_{k-1}$, we can easily determine the degree of each vertex $v \in V$ in the corresponding modular Gray graph by examining each radix in the tuple. The following lemma shows how.

Lemma 14 *Given the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, let $G = (V, E)$ be the modular Gray graph for r and $n = p_{k-1}$. Then, each vertex $v \in V$ has the same degree δ, and each digit position $i = 0, 1, \ldots, k-1$ can be mapped to a set of δ_i edges incident on v such that*

$$\delta_i = \begin{cases} 2 & \text{if } r_i \neq 2 \,, \\ 1 & \text{otherwise} \,, \end{cases} \tag{37}$$

and

$$\delta = \sum_{i=0}^{k-1} \delta_i \,. \tag{38}$$

Proof: Let v be a vertex in V, and let the digit representation of v be $v_{k-1}v_{k-2}\cdots v_0$. We will map each digit v_i in v to a set of edges $E(v_i) \subseteq E$ such that $|E(v_i)| = \delta_i$, where δ_i is given by equation (37). Furthermore, we will show that the sets $E(v_0), E(v_1), \ldots, E(v_{k-1})$ are pairwise disjoint, which will suffice to show equation (38).

For each digit v_i in the digit representation of v, let $E(v_i) \subseteq E$ contain all edges $(u, v) \in E$ such that vertices u and v differ in only the ith digit. By how the modular Gray graph is constructed, there are two possible values for this vertex u: the integers u' and u'', and their digit representations are

$$\begin{aligned} u' &= v_{k-1} \quad v_{k-2} \quad \cdots \quad v_{i+1} \quad (v_i + 1) \bmod r_i \quad v_{i-1} \quad \cdots \quad v_0 \quad , \\ u'' &= v_{k-1} \quad v_{k-2} \quad \cdots \quad v_{i+1} \quad (v_i - 1) \bmod r_i \quad v_{i-1} \quad \cdots \quad v_0 \quad . \end{aligned}$$

Now, we examine the radix r_i to determine whether the vertices u' and u'' are equivalent. If r_i is not 2, then we have $(v_i + 1) \bmod r_i \neq (v_i - 1) \bmod r_i$, and the vertices u' and u'' must be distinct. Otherwise, we have $r_i = 2$, so that v_i is either 0 or 1, and we get $(v_i + 1) \bmod 2 = (v_i - 1) \bmod 2$ for both values of v_i, implying that the vertices u' and u'' are the same. Thus, we have shown that $E(v_i)$ has size $\delta_i = 2$ if $r_i \neq 2$ and size $\delta_i = 1$ otherwise—that is, we have shown equation (37). Moreover, since $E(v_i)$ comprises only the edges $(u, v) \in E$ where u and v differ in digit i, it is obvious that the sets $E(v_0), E(v_1), \ldots, E(v_{k-1})$ are pairwise disjoint. This statement proves equation (38). ∎

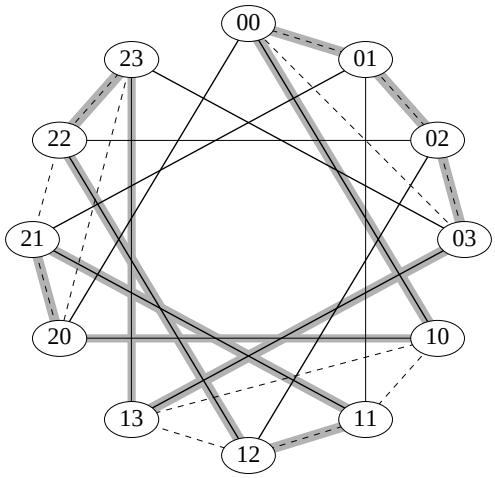
Graph theory defines a graph where all vertices have the same degree δ as a *δ-regular graph*. The following corollary uses Lemma 14 to claim that modular Gray graphs for cyclic full Gray codes—where the number of vertices n is inherently equal to the product p_{k-1} of the radices r—are regular.

Corollary 15 ***Modular Gray graphs for cyclic mixed-radix full Gray codes are regular***
Given the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, let b be the number of radices in r that equal 2. Then, the modular Gray graph for the cyclic mixed-radix full Gray code for r is a $(2k - b)$-regular graph. ∎

Figure 2 shows the modular Gray graph for the radix tuple $r = (3, 4)$ and the integer $n = 12$, along with the Hamiltonian cycle generated by our algorithm for a cyclic mixed-radix full Gray code. Notice that the graph is 4-regular, as stated in Corollary 15. As another example, Figure 3 shows the modular Gray graph for the 3-tuple $r = (2, 2, 3)$ and the integer $n = 12$, which is 4-regular by Corollary 15. Again, we highlight the Hamiltonian cycle that our algorithm generates in this graph.

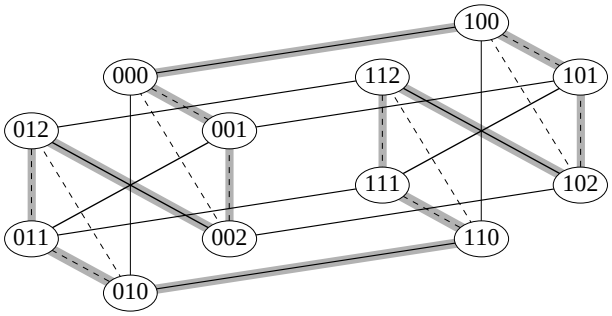
12 When is it impossible to generate a cyclic mixed-radix dense Gray code?

We now consider modular Gray graphs of cyclic mixed-radix dense Gray codes, where, given the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, we have $n < p_{k-1}$. We can think of such a graph as an induced subgraph of the modular Gray graph for the same radices r, where the subgraph is made by discarding the highest-numbered $p_{k-1} - n$ vertices of the larger graph, along with their incident edges. Once we start discarding these vertices, it might become impossible to compute a Hamiltonian cycle with the remaining vertices. Here, we identify two such cases where finding a Hamiltonian cycle, or equivalently, generating a cyclic mixed-radix dense Gray code for a set of values r and n, cannot be done. Theorems 16–17 describe those cases, and in order to simplify the proofs, we assume in all three theorems that k is the minimum number of digits required to represent all n vertices of the graph in the radices $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, so that the $(k - 1)$st digit of $n - 1$ is always positive.



our Gray code
00
01
02
03
13
23
22
12
11
21
20
10

Figure 2: The modular Gray graph for a cyclic mixed-radix full Gray code for $(3, 4)$. The edges are visibly coded: dotted edges represent the modular Gray-code property between two vertices u and v that differ in digit 0; solid edges represent the modular Gray-code property between two vertices u and v that differ in digit 1; and shaded edges represent edges included in the Hamiltonian cycle produced by our algorithm for a cyclic mixed-radix full Gray code.



our Gray code
000
001
002
012
011
010
110
111
112
102
101
100

Figure 3: The modular Gray graph for a cyclic mixed-radix full Gray code for $(2, 2, 3)$. The edges are visibly coded: dotted edges represent the modular Gray-code property between two vertices u and v that differ in digit 0; solid edges represent the modular Gray-code property between two vertices u and v that differ in either digit 1 or 2; and shaded edges represent edges included in the Hamiltonian cycle produced by our algorithm for a cyclic mixed-radix full Gray code.

Theorem 16 ***Graphs with vertices of degree* 1 *are non-Hamiltonian***
Let n be a positive integer such that $n > 2$, and let k be the number of digits required to represent $\widehat{n} = n - 1$ using the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, so that the digit representation of $\widehat{n}$ is $\widehat{n}_{k-1}\widehat{n}_{k-2}\cdots\widehat{n}_0$, where $\widehat{n}_{k-1} > 0$. Let G be the modular Gray graph for r and n. If n has the digit representation

$$n = q\,0\,0\,\cdots\,0\,1\,, \tag{39}$$

where either $q = 1$ or $q < r_{k-1} - 1$, then G has no Hamiltonian cycle.

Proof: There are two cases to consider. The first case occurs when n has the form given by equation (39), and we have $k = 1$. The second case occurs when n is of the same form and $k > 1$. We will show for each case that there exists a vertex $v \in V$ with degree 1, which will suffice to prove that no Hamiltonian cycle exists in G for these forms of n.

- Case 1: $k = 1$.
  Suppose that n has the form given by equation (39). Then, because we are given that $n > 2$, the 1-digit integer n must be in the range $3 \leq n < r_0 - 1$. Let vertex v be the 1-digit integer 0. By the modular Gray graph, v can have edges to only the vertices 1 and $r_0 - 1$, provided that those vertices also exist in V. Because $n \geq 3$, we know that $1 \in V$, and therefore, v has an edge to vertex 1. Since $n < r_0 - 1$, however, we know that $r_0 - 1 \notin V$. Thus, vertex v has degree 1, and G has no Hamiltonian cycle.

- Case 2: $k > 1$.
  Suppose that n has the form given by equation (39). Let vertex v be $n - 1$, which has the digit representation $q\ 0\ \cdots\ 0$. We will show that there is only one edge $(u, v) \in E$ that is incident on v, and the vertex u differs from vertex v in digit $k - 1$.

  First, we must show that for digit positions $i = 0, 1, \ldots, k - 2$, there are no vertices $u \in V$ such that $(u, v) \in E$ and vertices u and v differ in digit position i. By how we construct the modular Gray graph, we have that for each digit position $i = 0, 1, \ldots, k - 2$, vertex v can have edges to the vertices u' and u'', where u' and u'' have the digit representations

$$\begin{array}{rcccccccccc} u' & = & v_{k-1} & v_{k-2} & \cdots & v_{i+1} & (v_i + 1) \bmod r_i & v_{i-1} & \cdots & v_0 & \\ & = & q & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & , \\ & & & & & & & & & & \\ u'' & = & v_{k-1} & v_{k-2} & \cdots & v_{i+1} & (v_i - 1) \bmod r_i & v_{i-1} & \cdots & v_0 & \\ & = & q & 0 & \cdots & 0 & r_i - 1 & 0 & \cdots & 0 & , \end{array}$$

  provided that u' and u'' also exist in V. Since $n - 1$ has the digit representation $q\ 0\ \cdots\ 0$, however, the vertices u' and u'' are larger than $n - 1$, and so u' and u'' cannot exist in V for $i = 0, 1, \ldots, k - 2$. Therefore, there are no vertices $u \in V$ such that we have both $(u, v) \in E$ and u and v differ in a digit position other than $k - 1$.

  Now, we consider the edges (u, v) incident on v such that u and v differ in digit $k - 1$. By how we construct the modular Gray graph, vertex v can have edges to the vertices u' and u'', where u' and u'' have the digit representations

$$\begin{array}{rcccccc} u' & = & (q + 1) \bmod r_{k-1} & 0 & \cdots & 0 & , \\ u'' & = & (q - 1) \bmod r_{k-1} & 0 & \cdots & 0 & , \end{array}$$

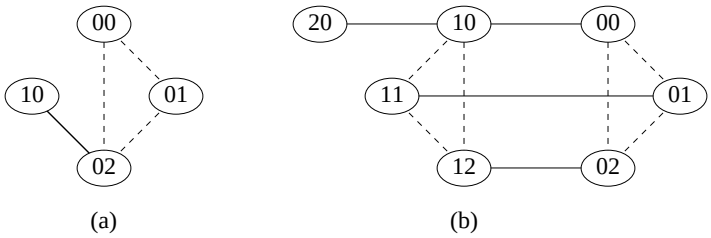


Figure 4: Modular Gray graphs with vertices of degree 1, and therefore, no Hamiltonian cycle. The edges are visibly coded: dotted edges represent the modular Gray-code property between two vertices u and v that differ in digit 0; and solid edges represent the modular Gray-code property between two vertices u and v that differ in digit 1. **(a)** The modular Gray graph for $r = (2, 3)$ and $n = 4$, which has the digit representation 1 1 in r; and **(b)** The modular Gray graph for $r = (4, 3)$ and $n = 7$, which has the digit representation 2 1 in r.

provided that u' and u'' also exist in V. We now prove that $u'' \in V$, but $u' \notin V$ unless $r_{k-1} = 2$, in which case vertices u' and u'' are equivalent. By the definition of k, we have that $q \geq 1$. Thus, we have $q - 1 \geq 0$, which gives u'' the digit representation $(q-1)\,0 \cdots 0$ so that $u'' < n - 1$ and consequently, $u'' \in V$.

Showing the properties of vertex u' is harder. When $r_{k-1} > 2$, then because either $q = 1$ or $q < r_{k-1} - 1$, we have $q + 1 < r_{k-1}$ in both cases. Thus, the vertex u' has the digit representation $(q+1)\,0 \cdots 0$, so that $u' > n - 1$ and consequently, $u' \notin V$. Otherwise, when we have $r_{k-1} = 2$, we must have $q = 1$, and so the value $(q + 1) \bmod r_{k-1}$ is 0. In this case, we have that u' and u'' have the digit representation $((q \pm 1) \bmod 2)\ 0 \cdots 0 = 0\,0 \cdots 0$ and are equivalent.

We have proven that when n has the form given by equation (39), then there is only one edge incident on either vertex 0 or vertex $n - 1$ in the modular Gray graph for r and n; therefore, G has no Hamiltonian cycle. ■

Figure 4 illustrates two graphs, each with a vertex of degree 1, that Theorem 16 describes as non-Hamiltonian. In Figure 4(a), the number n of vertices has the form given by equation (39) with $q = 1$, and in Figure 4(b), n has the same form but with $q < r_{k-1} - 1$, instead.

The following theorem lists another case where a Hamiltonian cycle cannot be found.

Theorem 17 ***Impossibility of both cyclicity and density for some values of r and n***
Let n be a positive integer, and let k be the number of digits required to represent $\widehat{n} = n - 1$ using the radix tuple $r = (r_{k-1}, r_{k-2}, \ldots, r_0)$, so that the digit representation of $\widehat{n}$ is $\widehat{n}_{k-1}\widehat{n}_{k-2} \cdots \widehat{n}_0$, where $\widehat{n}_{k-1} > 0$. Let G be the modular Gray graph for r and n. If n is odd and r_i is even for each digit position $i = 0, 1, \ldots, k-2$, and either r_{k-1} is even or $n_{k-1} < r_{k-1} - 1$, then G has no Hamiltonian cycle.

Proof: There are only two cases to consider, and we will use contradiction to prove that neither of these cases can produce a Hamiltonian cycle. To do this, we will need to use the following claim:

If r_i is even for any digit position $i = 0, 1, \ldots, k-1$, then the number of digit changes made in digit i as we take one trip around the Hamiltonian cycle must be even.

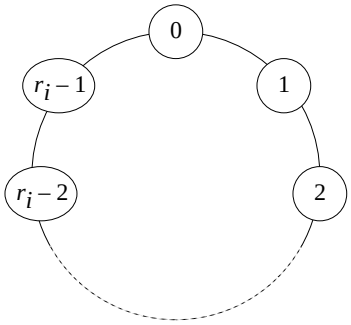


Figure 5: The r_i cycle shows all possible values of digit i, or equivalently, the set of integers $\{0, 1, \ldots, r_i - 1\}$, in a circle.

To prove the above claim, let us first isolate digit i. Figure 5 shows the r_i cycle representing all the possibilities of digit i in a circle.

Assuming that we start from vertex 0, let z_i be the net number of trips we make around the r_i cycle as we take one trip around our Hamiltonian cycle. We define $z_i > 0$ to mean that we made $|z_i|$ net trips clockwise, and $z_i < 0$ to mean that we made $|z_i|$ net trips counterclockwise. These z_i net trips around the r_i cycle account for a total of $|r_i z_i|$ digit changes, and since r_i is even, the net trips overall account for an even number of digit changes. Now that we've accounted for all the net trips around the r_i cycle, the other changes we made in digit i must represent trips where we ventured along one direction from a starting point in the r_i cycle, and then came back along the opposite direction to the same point. Each of these "detours" must have accounted for an even number of digit changes, so that in total, all of the detours accounted for an even number of digit changes. Therefore, all of the digit changes that occur in digit i during one trip around the Hamiltonian cycle account for an even number of digit changes in total. Hence, we have the claim.

Now, we return to the proof for the theorem. In the following two cases, we assume that n is odd and r_i is even for each digit position $i = 0, 1, \ldots, k-2$.

- Case 1: r_{k-1} is even.
  Let us assume to the contrary that a Hamiltonian cycle exists in G. By the Gray-code property, we must make n digit changes in total during one trip around the Hamiltonian cycle. We are given, however, that n is odd, and since r_i is even for digit positions $i = 0, 1, \ldots, k-1$, we know by the claim that each digit contributes an even number of digit changes Therefore, it is impossible to come back to the starting vertex 0 in n digit changes, and we cannot have a Hamiltonian cycle in G.

- Case 2: $n_{k-1} < r_{k-1} - 1$.
  Let us assume to the contrary that a Hamiltonian cycle exists in G. By the claim, we have that each digit $i = 0, 1, \ldots, k-2$ contributes an even number of digit changes. Since $n_{k-1} < r_{k-1} - 1$, we cannot form the full r_{k-1} cycle. The highest digit value that digit $k-1$ may take on is either $n_{k-1} - 1$ when $n = n_{k-1}00\cdots 0$, or n_{k-1} in any other case, and since both these values are less than $r_{k-1} - 1$, neither would hold the modular Gray-code property if it were to change to 0. Thus, z_{k-1} must be 0, and each digit change that occurs in digit $k-1$ must be part of a detour that returns to the starting point in the r_{k-1} cycle, accounting for an even number of digit changes for each detour. By the Gray-code property, and because n is odd, there must be an odd number of digit changes in total during one trip around the Hamiltonian cycle. We can only account, however, for an even number of digit changes in

each digit $i = 0, 1, \ldots, k-1$. Therefore, it is impossible to come back to the starting vertex 0 in n digit changes, and we cannot have a Hamiltonian cycle in G.

We have proven through contradiction in all cases that it is impossible to have a Hamiltonian cycle in G if n is odd and r_i is even for each digit position $i = 0, 1, \ldots, k-2$, and either r_{k-1} is even or $n_{k-1} < r_{k-1} - 1$. ∎

Importantly, although Theorems 16–17 prove several values of r and n for which a Hamiltonian cycle cannot be found, it is not an exhaustive list of these cases. In fact, we know of at least one other such case, which we do not explain here. There is much more still to learn about the modular Gray graphs for cyclic mixed-radix dense Gray codes, and we should think of the theorems in this section as cases that we've found and explained rather than a broader statement about the attributes of modular Gray graphs.

13 Conclusions

We have now shown the five major contributions of this thesis:

- A formula for each digit of the non-cyclic binary dense Gray code for any positive integer n, as given in our earlier paper [3]. With this formula, we can generate each number in the non-cyclic binary dense Gray code in constant time.
- An algorithm that generates a cyclic binary dense Gray code for any even number of integers. The algorithm computes each number in constant time.
- A formula for each digit of the non-cyclic mixed-radix dense Gray code for any mixed-radix tuple r and positive integer n less than or equal to the product of the radices in r.
- A recursive algorithm that generates each integer in the cyclic mixed-radix full Gray code for a mixed-radix tuple r and positive integer n equal to the product of the radices in r.
- A list of cases where it is impossible to compute a cyclic mixed-radix dense Gray code for a mixed-radix tuple r and positive integer n strictly less than the product of the radices in r.

We have yet to determine a digitwise formula for cyclic mixed-radix dense Gray codes, as we were able to do for non-cyclic binary, cyclic binary, and non-cyclic mixed-radix dense Gray codes. In our future work, we will refocus on this goal and potentially find more cases of radix tuples r and integers n for which it is impossible to compute a cyclic mixed-radix dense Gray code. We also hope to study the applications for the dense Gray codes we have developed, as the standard binary reflected Gray code and cyclic mixed-radix dense Gray code have already proven themselves useful in a number of computing [5, 6, 8, 10] and network design [1] problems.

14 Acknowledgments

I would like to thank my thesis advisor and good friend, Professor Thomas H. Cormen of the Dartmouth College Computer Science Department, for his incredible guidance and support throughout my years here as an undergraduate. Tom taught me my very first computer science course— Introduction to Programming and Computation—and has since become my most treasured teacher, colleague, and mentor. When we first started working together in 2015, I could never have imagined all the wonderful work he would help me accomplish: from writing my first technical paper to presenting at my first technical conference at Allerton. Tom has taught me so much in the ways of writing and research, and I am honored to have both started and ended my career as a Dartmouth student with him as my advisor.

I would also like to thank the members of my thesis committee, Professor Prasad Jayanti of the Dartmouth College Computer Science Department and Professor Peter Winkler of the Mathematics and Computer Science Departments, for their encouragement and their expertise. Without their generous support, the work in this thesis would not have been possible.

Thank you,
Jessica C. Fan

References

[1] M. Anantha and B. AlBdaiwi. Mixed radix Gray codes in Lee metric. *IEEE Transactions On Computers*, 56, October 2007.

[2] Martin Cohn. Affine m-ary Gray codes. *Information and Control*, 6:70–78, 1963.

[3] Thomas H. Cormen and Jessica C. Fan. Dense Gray codes, or easy ways to generate cyclic and non-cyclic Gray codes for the first n whole numbers. In *54th Annual Allerton Conference on Communication, Control, and Computing*, October 2016.

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

[5] Frank Gray. Pulse code communication. U. S. Patent 2,632,058, March 1953.

[6] S. Lennart Johnsson and Ching-Tien Ho. On the conversion between binary code and binary-reflected Gray code on boolean cubes. Technical Report TR-20-91, Center for Research in Computing Technology, Harvard University, July 1991.

[7] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, page 94. Plenum Press, 1972.

[8] Donald E. Knuth. *The Art of Computer Programming*, volume 4A, Combinatorial Algorithms, Part 1. Addison-Wesley, 2011.

[9] B. D. Sharma and R. K. Khanna. On m-ary Gray codes. *Information Sciences*, 15:31–43, September 1977.

[10] Vincent Vajnovski and Timothy Walsh. A loop-free two-close Gray-code algorithm for listing k-ary Dyck words. *Journal of Discrete Algorithms*, August 2005.