Grey Codes in Black and White

Wes Cleveland, Erin Connerley, Becca Maddrell, Kasey Aderhold, Abel Yehdego

December 18, 2007

1 Introduction

A Gray code represents each number in the sequence of integers $0, 1, \ldots, 2^n - 1$ as a binary string length N in an order such that adjacent integers have Gray code representations that differ in only one bit position. In order to increase a number, one changes the Gray code representation by changing one bit. This property defines a Gray code and makes it different from other codes, like the binary, where more than one bits may change per increase of integer. This property is called the **adjacency property**. We define a valid Gray code as one that has this adjacency property.

The utility of Gray codes as opposed to binary codes is that each sequential number only changes 1 bit. Binary, however, has an average of 2 bits changing for each sequential number. In addition, for large n-bit binary numbers the number of bit changes is significantly large. This large number of bit changes creates a significantly large chance for errors to occur. Through this exploration of the binary code we determine strong connections with the properties the Gray code. The Gray code also has unique properties and ways it is formed.

2 Creating Gray Codes with Trees

Each Gray code representation in the sequence must only change by one bit. Formally, a representation of numbers by a Gray code is said to satisfy the **adjacency property** if the code assigned to n and $n + 1$ differ in only one bit location, for $0 \leq n \leq 2^n - 2$. The sequence may repeat itself in a

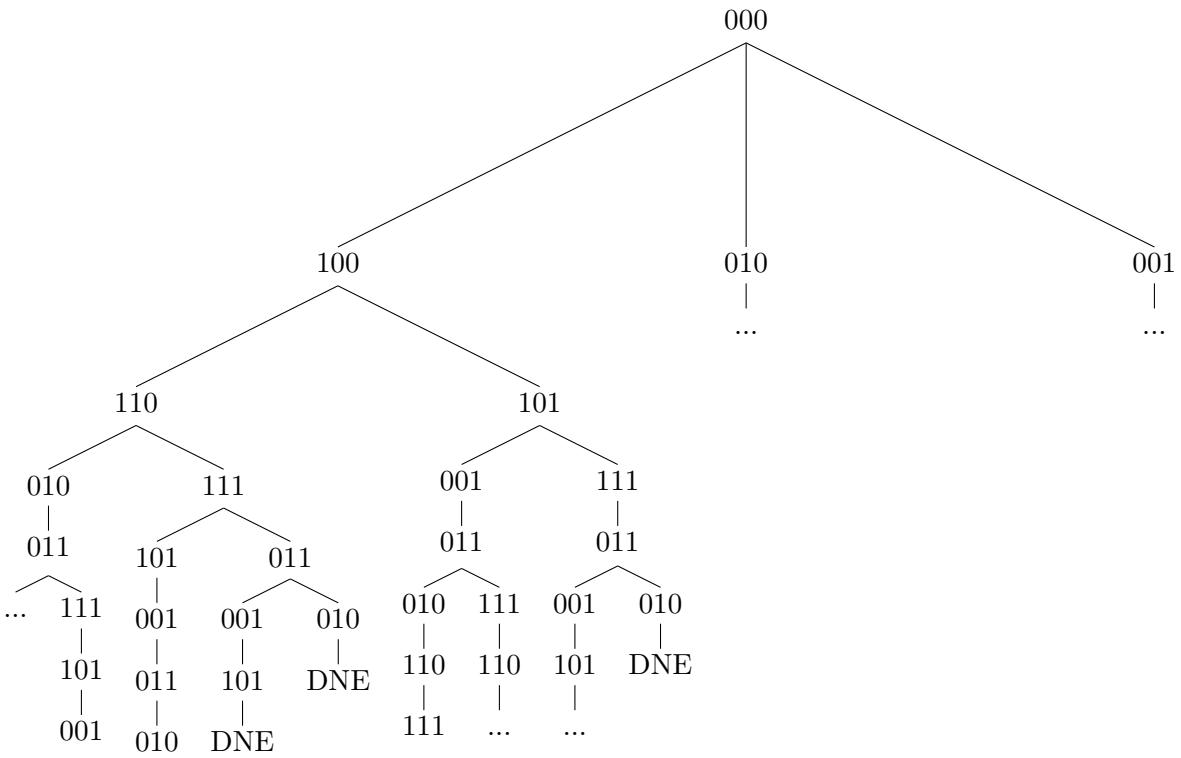

Figure 1: Gray code creation tree

cyclic way because of the adjacency property. That is, the Gray code representation of $2^n - 1$ in an n-bit Gray code may be able to be followed by the first representation in the same n-bit sequence in such a way that the representation of 0 and $2^n - 1$ differ in only one bit position. Such a Gray code is called a "**good**" Gray code.

A Gray code can be created in several ways. To form an n-bit Gray code one may start with n-zeros e.g. a three bit Gray code you may start with three zeros. Starting with 0 is a convention that makes it convenient to compare Gray codes. Each location in the string, each zero, has the ability to change from a zero to a one. Using this method a tree can be used to create a Gray code. One must eliminate a sequence of Gray codes if a term is repeated. If a term is repeated, it means that the number representation being repeated, which is not admissible if a Gray code is the representation of increasing numbers. Once a sequence is created, it may be ruled out if it does not have the adjacency property.

A Gray code can be created using one of the n-bit sequences of Gray code by cutting rows vertically and horizontally and reflecting, by cyclic permutation, and by switching the zeros and ones.

3 Creating Gray Codes through Reflection

After experimenting with using reflection to create new Gray codes, we encountered this question: Is it true that any n-bit Gray code can be turned into a $(n+1)$-bit Gray code by reflection? To answer this question we must first define a **reflection**.

Theorem 1 (Reflecting Theorem)**.** *If given an n-bit Gray code, one can create an $n+1$-bit Gray code by mirroring the n-bit Gray code, placing a 0 before the first $n/2$ terms in the sequence and a 1 before the remaining terms.*

Here we are given a 2-bit Gray code:

00	Now it is 'mirrored' downward:	00	Now add 0's and 1's:	000
01		01		001
11		11		011
10		10		010
		10		110
		11		111
		01		101
		00		100

Since the $n+1$ bit Gray code is based on a valid n-bit Gray code, and we never introduce more than 1 change while keeping the original Gray code, we know that reflection will result in a valid $n+1$ bit Gray code. The n-bit Gray code $g_0, g_1, g_2, \ldots, g_{2n-1}$ is changed to a $n+1$ bit Gray code by reflecting n bit strings and adding 0s and 1s to the front of the terms. The resulting code created by reflection satisfies the adjacency property. The last term in the sequence changes by one bit to create the first term in the sequence. Therefore the Gray code created by mirroring is by definition a good code.

4 Unique 3-bit Gray Codes

When first writing down all unique 3-bit Gray codes, we came up with eighteen different codes:

000	000	000	000	000	000	000	000	000
100	100	100	100	100	100	010	010	010
110	110	110	101	101	101	110	110	110
010	010	111	001	001	111	100	100	111
011	011	101	011	011	110	101	101	011
001	111	001	010	111	010	001	111	001
101	101	011	110	110	011	011	011	101
111	001	010	111	010	001	111	001	100

000	000	000	000	000	000	000	000	000
010	010	010	001	001	001	001	001	001
011	011	011	101	101	101	011	011	011
111	001	001	111	100	100	111	010	010
110	101	101	011	110	110	101	110	110
100	111	100	010	010	111	100	100	111
101	110	110	110	011	011	110	101	101
001	100	111	100	111	010	010	111	100

We decided that in order to be a Gray code, the code must be cyclical. That is, the bit changes from the last term to the first term must also be one. With this, the code can cycle through without ever changing by more than one. By imposing this requirement, we cut our unique 3-bit Gray codes down to twelve:

000	000	000	000	000	000	000	000	000	000	000	000
100	100	100	100	010	010	010	010	001	001	001	001
110	110	101	101	110	110	011	011	101	101	011	011
010	111	001	111	100	111	111	001	111	100	111	010
011	101	011	110	101	011	110	101	011	110	101	110
111	001	111	010	111	001	100	111	010	111	100	111
101	011	110	011	011	101	101	110	110	011	110	101
001	010	010	001	001	100	001	100	100	010	010	100

We also decided that every Gray code should start with 000 because any code that didn't would simply be a rotation of the same order. For

example if we started our first code at the term 100 instead and cycled through accordingly ending with 000, it would not entail a new unique Gray code. We exclude all of these rotations from our list.

Looking at the columns of these Gray codes reveals something very interesting. There are only six different columns in our entire list of codes:

0	0	0	0	0	0
1	0	0	1	0	0
1	1	0	1	1	0
0	1	0	1	1	1
0	1	1	1	0	1
1	1	1	0	0	1
1	0	1	0	1	1
0	0	1	0	1	0

Since each Gray code in our list is made up of some combination of these six columns, we decided that each is created through manipulating the columns of one of the Gray codes. To create the previous twelve codes in our table, we only need these six columns in the form of two Gray codes. Every other one is simply a manipulation of one of these two. We have then divided our previous twelve codes into two unique groups. Any code within each of the two groups can be chosen to represent the entire group, since each code within a group has the exact same columns as any other in that group. For example, these two codes:

000	000
100	100
110	110
010	111
011	101
111	001
101	011
001	010

Represent the two different groups. Permuting the three columns of each will create the rest of our twelve codes.

In conclusion, there are only two unique groups of 3-bit Gray codes.

5 3- bit Gray Code Bit Change Pattern

5.1 Not "Good" Gray code (not cyclic)

The symbol c_i denotes changing i^{th} bit of a binary code. In a and b below the first and third change in the sequence and the fifth and seventh change in the sequence changed the same bit. These bit changes, the first and the third changes and the fifth and the seventh changes were opposite in manner. If the first three changes were c_1, c_2, c_1, the last three changes were the opposite pattern with c_2, c_1, c_2. The middle change was always the bit change that was not a part of the pattern because the third bit had not been changed before.

a.

$$000 \overset{c_1}{\rightarrow} 100 \overset{c_2}{\rightarrow} 110 \overset{c_1}{\rightarrow} 010 \overset{c_3}{\rightarrow} 011 \overset{c_2}{\rightarrow} 001 \overset{c_1}{\rightarrow} 101 \overset{c_2}{\rightarrow} 111$$

b.

$$000 \overset{c_1}{\rightarrow} 100 \overset{c_3}{\rightarrow} 101 \overset{c_1}{\rightarrow} 001 \overset{c_2}{\rightarrow} 011 \overset{c_3}{\rightarrow} 010 \overset{c_1}{\rightarrow} 110 \overset{c_3}{\rightarrow} 111$$

5.2 Type A "Good" Gray code

Gray codes formed from this type of 3 bit row change, let us call it type A, have the same bit change every other time starting with the first bit.

$$000 \overset{c_1}{\rightarrow} 100 \overset{c_2}{\rightarrow} 110 \overset{c_1}{\rightarrow} 010 \overset{c_3}{\rightarrow} 011 \overset{c_1}{\rightarrow} 111 \overset{c_2}{\rightarrow} 101 \overset{c_1}{\rightarrow} 001$$

5.3 Type B "Good" Gray Code

All Gray codes formed from this type of 3 bit row change, let us call type B, have the same bit change every other time starting with the 2nd bit change.

$$000 \overset{c_1}{\rightarrow} 100 \overset{c_2}{\rightarrow} 110 \overset{c_3}{\rightarrow} 111 \overset{c_2}{\rightarrow} 101 \overset{c_1}{\rightarrow} 001 \overset{c_2}{\rightarrow} 011 \overset{c_3}{\rightarrow} 010...$$

Theorem 2. *In a "good" n-bit Gray code beginning with a string of n 0s, the string of n 1s can possibly represent one of only* $2^{n-1} - n + 1$ *of the integers from 0 to* $2^n - 1$.

Example 1: 2-bit Gray code

$$g_0, g_1, g_2, g_3$$

$$00, 10, 11, 01$$

There is only one position g_2 with all ones in a 2-bit gray code.

Example 2: 3-bit Gray Code

$$000, 001, 011, 111, 101, 100, 110, 010$$
$$000, 001, 011, 010, 110, 111, 101, 100$$
$$000, 001, 011, 010, 110, 100, 101, 111$$

Each of the first two rows has an interval entry consisting of a string of 3 1's; these represent good gray codes, the last row represents a bad gray code. The good codes have the bit consisting of only of ones in the only places possible to make a good Gray code, g_3 and g_5, assuming that g_0 is 000. This is because only one bit is able to change at a time so each consecutive number is changed by one bit from the first. Since we started with 000 the g_2 has to include a 1 . g_3 has to either contain no 1's or two 1's. This creates a pattern of every odd numbered g having an even number of ones in it and every even numbered g having an odd number of ones in it. As such only an even numbered row can contain all ones in the 3-bit code. If the arbitrary n-bit Gray code is $g_0, g_1, , g_{2^n-1}$ then the ones that have odd number of 1's are the ones of the form $g_{2^k-1} 1 \leq k \leq 2^{n-1}$.

In addition all ones cannot be located until g_4 because you can only change one bit at a time. This means that only g_4, g_6, g_8 are capable of filling up with ones in a 3-bit code. In an n-bit Gray code the first position capable of filling up with one's is the $(n+1)$th position. After that, every other position is capable of having all ones.

However, for good Gray codes, we can exclude the last one because it does not allow the code to be cyclical. As such, the last code that has g_n filled with ones will have that position exactly n rows away from the end of the code. Since there are 2^n rows we can say that position $2^n - 1$ of the code that is not capable of being filled with ones and making a good Gray code gives us 2 times the number of unique Gray codes minus one. This is how the theorem is proved.

6 Average Number of Bit Changes in Binary Code

The binary code is a numerical system that represents numeric values using 0 and 1. One counts in binary the same as any other numerical system. The binary code is a base 2 system in positional notation. A bit is a digit in a binary number. For example, the number 001 has three bits and when it becomes 010 two of the bits changed. When counting in binary at least one bit changes every time you increase the number by one, but sometimes counting up one more will cause more than one bit change. The following shows counting from zero to 7 in binary and showing each respective bit change.

000

001 (1 bit change from 0 to 1).

010 (1 bit change from 1 to 2).

011 (1 bit change from 2 to 3).

100 (1 bit change from 3 to 4).

101 (1 bit change from 4 to 5).

110 (1 bit change from 5 to 6).

111 (1 bit change from 6 to 7)

The pattern of the number of bit changes when counting in binary depends on the number of bits used. The maximum number of bits that can change when increasing the number by one is the number of bits in the code. For example, in a three bit code, the maximum number of bits that can change is three.

Theorem 3. *In an n-bit binary number, n bit changes occur when counting from* $2^{n-1} - 1$ *to* 2^{n-1}.

For example: In a 3 bit code, the greatest change is from 3 to 4 which is basically half-way to the greatest number that can be represented. These bit changes are equal to adding one to the binary carry sequence.

Definition 1. *For n-bit binary numbers, a(n) represents the sequence of numbers of bit changes that occur when counting from* 0 *to* $2^n - 1$. *In particular,* $a(n)_k$ *is the number of binary bits changed when counting from* $k - 1$ *to* k.

As you increase to 4-bits the bit change sequence is the 3-bit sequence a(3) followed by a 4 and then another 3-bit sequence. Using proof by induction, the pattern continues up to a n-bit code which has a sequence of the $n-1$ bit sequence preceding it followed by n and then another $n-1$ bit sequence. We can then see that the maximum number of bit changes, n in this case, is always found in the middle of the sequence.

So for the pattern of the number of bit changes here are the sets respectively of each n-bit binary code with the subscript number representing the n-bit binary codes that we are working with for that particular sequence.

$$\begin{aligned} a(1) &= 1 \\ a(2) &= 1,2,1 \\ a(3) &= 1,2,1,3,1,2,1 \\ a(4) &= 1,2,1,3,1,2,1,4,1,2,1,3,1,2,1 \end{aligned}$$

Now assume that the n^{th} sequence is correctly represented by this model. We claim that $a(n)$ is simply $\{a(n-1), n, a(n-1)\}$.

For a base case we will use a 2 bit sequence.

00 01 10 11

First we note that a 3 bit sequence uses this sequence with a zero in front until the first two bits have filled up with ones and then changes to a 1.

000 001 010 011

The first two bits are filled with ones and now the next bit changes to a one and all the others change to a zero. This is key to understanding the proof by induction.

In 100 101 110 111, notice how the pattern of the first 2 bits repeats.

Assume that the k^{th} bit sequence will take the form $a(k-1), k, a(k-1)$.

We can say that the next sequence which corresponds to $n = k+1$ will have the same pattern up until the first k bits have filled up with ones and

then the $(k+1)^{\text{th}}$ bit will change to a 1 and stay a one and the first k-bits will change back to the beginning of the sequence and repeat. Since we can do this we can say this pattern is true for all n-bit sequences and their pattern of the number of bit changes. Now that we know how we arrive at each pattern of bit changes for each n-bit code we can go on to find the average for every n-bit code.

Writing out the number of bit changes as you count from 0 up to 31 you obtain:

$$1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,1,3,1,2,1,4,1,2,1,3,1,2,1$$

There are 31 times that there are changes and the total number of bit changes is 57. This means the average number of bit changes is 57/31 for a five bit system.

But we do not need to go through this long process to make the average for each n-bit set. Notice once again for the 5 bit binary code pattern of bit changes we simply reflect that of the 4 bit code over the number 5 or the number of bits in the code. So if we know the previous one we can always solve the next.

So for a six bit pattern we would simply have:

1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,1,3,1,2,1,4,1,2,1,3,1,2,1, - $a(5)$

6,

1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,1,3,1,2,1,4,1,2,1,3,1,2,1 - $a(5)$

Now we are going to notice something else. In order to take the average we must first take the total number of bit changes and divide it by the number of times there were changes in the bits. An easy way to do this is to find the sum of the entries in $a(n)$ code and then multiply it by two and add n. This is because of the **Reflecting Theorem** in which we obtained the next n-bit code by reflecting the previous over the number of bits. So for a simple example we can reflect a one bit code to a two bit code.

1

$1,2,1$

10

And then to a three bit code

$$1, 2, 1, 3, 1, 2, 1$$

For these codes we obtain the sum by adding the numbers up. So respectively

$$1 = 1$$

$$1 + 2 + 1 = 4 \tag{1}$$

$$1 + 2 + 1 + 3 + 1 + 2 + 1 = 11$$

The easy method we described would be to remember the previous code's sum and double it and then add the number n to that.

So, for the 2-bit, you take $1 * 2 + n$ where n=2 and get 4. This is the correct sum for the 2-bit pattern.

Once again we will do this for the three bit pattern to find its sum.

$$4 * 2 + 3 = 11$$

So all we have to do to find the sum of $a(n)$ is this in the sequence notation we were using:

$$2\left(\sum_{k=1}^{2^{n-1}-1} a(n-1)_k\right) + n = 2^{(n+1)} - (n+2). \tag{2}$$

Indeed, we could obtain this equality by induction, setting the base case equal to n=2:

$$2\left(\sum_{k} a(2-1)_k\right) + 2 = 2^{(2+1)} - (2+2) = 4,$$

which accords with the value obtained in (1).

Assuming that the equality holds for n, we see that

$$\begin{aligned}
\sum_k a(n+1)_k &= \sum_k a(n)_k + n + \sum_k a(n)_k \\
&= 2(2^{n+1} - (n+2)) + (n+1) \\
&= 2^{n+2} - 2n - 4 + n + 1 \\
&= 2^{n+2} - n - 3 \\
&= 2^{n+2} - ((n+1) + 2),
\end{aligned}$$

thus establishing (2) for all n.

Now, the denominator for the average number of bit changes is simply the number of times there are changes or the number of times we counting up by one. So for any binary code you can only count $2^n - 1$ times. For example, you can only count up seven times in a three bit code or $2^3 - 1 = 7$.

So, the average number of bit changes is the total sum of bit changes divided by the number of times there are bit changes, so we have:

Theorem 4. *The average number of bit changes for an n-bit binary code is* $\dfrac{2^{n+1} - (n+2)}{2^n - 1}$.

We can use this formula to prove that as n approaches infinity the average number of bit changes approaches 2:

$$\begin{aligned}
\lim_{n\to\infty} \frac{2^{n+1} - n - 2}{2^n - 1} &= \lim_{n\to\infty} \frac{2^{n+1} \times \left[1 - \dfrac{n}{2^{n+1}} - \dfrac{2}{2^{n+1}}\right]}{2^{n+1} \times \left[\dfrac{2^n}{2^{n+1}} - \dfrac{1}{2^{n+1}}\right]} \\
&= \lim_{n\to\infty} \frac{1 - \dfrac{n}{2^{n+1}} - 2^{1-(n+1)}}{2^{n-(n+1)} - 2^{0-(n+1)}} \\
&= \lim_{n\to\infty} \frac{1 - \dfrac{n}{2^{n+1}} - 2^{-n}}{2^{-1} - 2^{-n-1}} \\
&= \lim_{n\to\infty} \frac{1 - \dfrac{n}{2^{n+1}} - \dfrac{1}{2^n}}{\dfrac{1}{2} - \dfrac{1}{2^{n+1}}}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\lim_{n\to\infty} 1 - \lim_{n\to\infty} \frac{n}{2^{n+1}} - \lim_{n\to\infty} \frac{1}{2^n}}{\lim_{n\to\infty} \frac{1}{2} - \lim_{n\to\infty} \frac{1}{2^{n+1}}} \\
&= \frac{1 - \lim_{n\to\infty} \frac{n}{2^{n+1}} - 0}{\frac{1}{2} - 0} \\
&= \frac{1 - \lim_{n\to\infty} \frac{n}{2^{n+1}}}{\frac{1}{2}}
\end{aligned}
$$

Flipping the one half and using L'Hospital's rule we are able to simplify the problem further.

$$
2 \times \left(1 - \lim_{n\to\infty} \frac{1}{\ln 2 \times 2^{n+1}}\right) = 2 \times (1 - 0) = 2
$$

Thus we have proven that the limit as n approaches infinity of the average number of bit changes in the usual binary representations of integers is 2.

7 Conclusion

Gray codes and binary codes are useful sequences in problem solving. There are good and bad types of Gray code depending on whether the code follows the adjacency property and is cyclic. Binary code creates a larger chance for error to occur when changing one number than Gray code because binary codes have an average of 2 bits changes for each sequential number while Gray code has only one.