Proc. Natl. Sci. Counc. ROC(A)
Vol. 22, No. 6, 1998. pp. 841-848

(Scientific Note)

Generalized Gray Codes with Applications

DAH-JYH GUAN

Institute of Applied Mathematics
National Sun Yet Sen University
Kaoshiung, Taiwan, R.O.C.

(Received February 19, 1998; Accepted April 23, 1998)

ABSTRACT

An efficient algorithm for enumerating a sequence of all elements in $(Z_n)^k$ in a special order is presented. The sequence enumerated is called the (n,k)-Gray code. It has the property that each pair of adjacent elements differs in only one digit and the difference is either +1 or −1. This sequence can be efficiently applied to calculation of the weight polynomial of a group code. We also show that the Tower of Hanoi problem, in which disks can only be moved to the adjacent pegs, can be solved by enumerating the $(3,k)$-Gray code, where k is the number of disks to be moved. Finally, we show that the enumeration of the (n,k)-Gray code can also be regarded as a Hamiltonian path in a generalized hypercube network.

Key Words: (n,k)-Gray code, weight polynomial, tower of Hanoi, Hamiltonian path, hypercube network

I. Introduction

Consider the problem of enumerating of all elements in $(Z_n)^k$. For example, for n=3 and k=2, Table 1 shows two possible sequences which contain all the elements of $(Z_n)^k$.

We may regard each element of $(Z_n)^k$ as a k-digit number of base n. The first sequence, $a_i=(a^i_{k-1}, a^i_{k-2}, ..., a^i_0)$, i=0, 1, ..., n^k−1, is listed in the order of the numerical values of the elements, $\sum_{j=0}^{k-1} a^i_j n^j$. The second sequence, b_i, i=0, 1, ..., n^k−1, contains the same set of elements as does the first one but in different order. Note that each pair of adjacent elements in the second sequence differ in only one digit, and that the difference is either +1 or −1. We shall call the second sequence, b_i, i=0, 1, ..., n^k−1, the (3,2)-*Gray code*. In this paper, we shall present a simple and efficient algorithm to enumerate the (n,k)-Gray code. We shall also describe its applications.

Enumeration of the (n,k)-Gray code is useful in the calculation of the weight polynomial of a group code generated by a generator matrix. Let n be the basis, l be the length of the code word, and k be the length of the message. Thus, each code word is an l-digit base n number, and each message is a k-digit base n number. The total number of messages is n^k. Direct enumeration of the set of all code words requires kln^k multiplications and $(k\text{-}1)ln^k$ additions. Using the (n,k)-Gray code, the computation time needed to enumerate the set of all code words can be reduced to ln^k additions and no multiplications. The reduction in computation time makes computation of longer codes possible. Furthermore, our algorithm is not recursive. The algorithm can incorporate checkpoint and restart operations very efficiently. Therefore, the calculation of the weight polynomial can be started from any element of $(Z_n)^k$ in the (n,k)-Gray code. In particular, the program can continue from the last point at which the program terminated abnormally. We can also divide the whole computation into parts and execute each part on different computers. Since computation of the weight polynomial usually takes a long time when l and k are large, checkpoint and restart operations are necessary. We have applied this technique in finding a ternary

Table 1. Two Sequences for All the Elements of $(Z_n)k$

i	a_i	b_i
0	(0 0)	(0 0)
1	(0 1)	(0 1)
2	(0 2)	(0 2)
3	(1 0)	(1 2)
4	(1 1)	(1 1)
5	(1 2)	(1 0)
6	(2 0)	(2 0)
7	(2 1)	(2 1)
8	(2 2)	(2 2)

constabelian [100,16,48]-code whose minimum distance exceeds that of the best known ternary [100,16,45]-code (Cheng and Guan, 1996).

In the next section, we shall describe an efficient algorithm to enumerate the (n,k)-Gray code. We shall also show how to apply the algorithm to calculate the weight polynomial of a group code. This technique can also be applied to any other calculation of a linear function whose value depends on all the elements of $(Z_n)^k$. The (n,k)-Gray code can also be regarded as a solution to a variation of the Tower of Hanoi problem, in which all the moves of the disks are restricted to moves between adjacent pegs. Furthermore, the (n,k)-Gray code can also be regarded as a Hamiltonian path of a generalized hypercube network. We shall describe these two applications in Section III and draw conclusions in Section IV.

II. Efficient Algorithm to Enumerate (n,k)-Gray Code

Our algorithm for generating (n,k)-Gray codes is based on the following theorems. Let $(d_{k-1}, d_{k-2}, ..., d_0)$ and $(g_{k-1}, g_{k-2}, ..., g_0)$ be two elements in $(Z_n)^k$. Let

$$s_j=\left(\sum_{i=j+1}^{k-1} g_i\right) \bmod 2,\ j=0,\ 1,\ ...,\ k\text{-}2.$$

Define a function σ_0: $Z_n \to Z_n$ as the identity function, that is, $\sigma_0(i)=i$ for every $i \in Z_n$. Define a function σ_1: $Z_n \to Z_n$ such that $\sigma_1(i)=(n-1)-i$ for every $i \in Z_n$. Define a function f: $(Z_n)^k \to (Z_n)^k$ such that

$$f((d_{k-1}, d_{k-2}, ..., d_0))=(g_{k-1}, g_{k-2}, ..., g_0),$$

where

$$g_{k-1}=d_{k-1}$$

$$g_j=\sigma_{s_j}(d_j),\ j=k-2,\ k-3,\ ...,\ 0.$$

Theorem 1. *The function f is bijective.*

Proof. Since f is a map from $(Z_n)^k$ into itself, it is suffice to show that f is injective. Assume that $f((d_{k-1}, d_{k-2}, ..., d_0))=(g_{k-1}, g_{k-2}, ..., g_0)$, and $f((d'_{k-1}, d'_{k-2}, ..., d'_0))=(g_{k-1}, g_{k-2}, ..., g_0)$. By definition, $d_{k-1}=g_{k-1}=d'_{k-1}$. For $j=k-2, k-3, ..., 0$, $\sigma_{s_j}(d_j)=g_j=\sigma_{s_j}(d'_j)$, which implies that $d_j=d'_j$. □

We regard $(d_{k-1}, d_{k-2}, ..., d_0)$ as a number of base n and define its numerical value, $(d_{k-1}, d_{k-2}, ..., d_0)_n$, as $\sum_{j=0}^{k-1} d_j n^j$.

Theorem 2. *If* $(d_{k-1}, d_{k-2}, ..., d_0)_n-(d'_{k-1}, d'_{k-2}, ..., d'_0)_n=1$, *then* $f((d_{k-1}, d_{k-2}, ..., d_0))$ *and* $f((d'_{k-1}, d'_{k-2}, ..., d'_0))$ *differ in only one digit, and the difference is* 1.

Proof. Assume that

$$f((d_{k-1}, d_{k-2}, ..., d_0))=(g_{k-1}, g_{k-2}, ..., g_0),$$

and that

$$f((d'_{k-1}, d'_{k-2}, ..., d'_0))=(g'_{k-1}, g'_{k-2}, ..., g'_0).$$

Since $(d_{k-1}, d_{k-2}, ..., d_0)_n-(d'_{k-1}, d'_{k-2}, ..., d'_0)_n=1$, there exists some j, $0 \le j \le n-1$, such that

(1) if $j<n-1$, then $d_i=d'_i$, for $i=k-1, k-2, ..., j+1$;
(2) $d_j=d'_j+1$;
(3) if $j>0$, then $d_i=0$ and $d'_i=n-1$, for $i=j-1, j-2, ..., 0$.

Since $d_i=d'_i$ for $i=k-1, k-2, ..., j+1$, it is clear that $g_i=g'_i$ for $i=k-1, k-2, ..., j+1$. For $m=1, 2, ..., k-1$, let $s_m=\left(\sum_{i=m+1}^{k-1} g_i\right) \bmod 2$ and $s'_m=\left(\sum_{i=m+1}^{k-1} g'_i\right) \bmod 2$. Note that $g_i=g'_i$ for $i=k-1, k-2, ..., j+1$ implies that $s_i=s'_i$ for $i=k-2, k-3, ..., j$. Since $d_j=d'_j+1$, g_j and g'_j must be different. The absolute value of their difference is $|\sigma_{s_j}(d_j)-\sigma_{s'_j}(d'_j)| = |\sigma_{s_j}(d_j)-\sigma_{s_j}(d_j+1)|=1$. Since $|g_j-g'_j|=1$, $s_{j-1} \ne s'_{j-1}$, which implies that either $g_{j-1}=n-(d_{j-1}+1)$ or $g'_{j-1}=n-(d'_{j-1}+1)$, depending on whether s_{j-1} is 1 or 0. In both cases, $g_{j-1}=g'_{j-1}$. It is now easy to see that, for $i=j, j-1, ..., 0$ the two sequences of numbers $g_{k-1}g_{k-2}...g_i$ and $g'_{k-1}g'_{k-2}...g'_i$ differ only in the j-th digit, and $|g_j-g'_j|=1$. Therefore, $s_i \ne s'_i$ for $i=j-1, j-2, ..., 0$. Since $d_i=n-1$ and $d'_i=0$ for $i=j-1, j-2, ..., 0$, $g_i=g'_i$ for $i=j-1, j-2, ..., 0$. □

By the above two theorems, an (n,k)-Gray code can be generated by converting elements in $(Z_n)^k$ in ascending order of their values.

Note that a $(2,k)$-Gray code is an ordinary k-bit binary Gray code. Define $x \oplus y=1$ if, and only if, $x \ne y$. It is well known that a k-bit binary Gray code $(a_{k-1}, a_{k-2}, ..., a_0)$ can be generated from a k-bit binary number $(b_{k-1}, b_{k-2}, ..., b_0)$ as follows:

$$a_{k-1}=b_{k-1}$$

$$a_j=a_{k-1} \oplus a_{k-2} \oplus ... \oplus b_j,\ j=k-2,\ k-3,\ ...,\ 0.$$

It is easy to see that a $(2,k)$-Gray code is a special case of an (n,k)-Gray code, and that the above formula can be derived from our generalized formula.

It is useful to convert a k-digit (n,k)-Gray code, $(g_{k-1}, g_{k-2}, ..., g_0)$, back into the corresponding k-digit number of base n, $(d_{k-1}, d_{k-2}, ..., d_0)$. Define $t_j=\left(\sum_{i=j+1}^{k-1} g_i\right)$ mod 2. Let σ_0 and σ_1 be defined in the same way as above. Define a function h: $(Z_n)^k \rightarrow (Z_n)^k$ such that

$$h((g_{k-1}, g_{k-2}, ..., g_0))=(d_{k-1}, d_{k-2}, ..., d_0),$$

where

$$d_{k-1}=g_{k-1}$$

$$d_j=\sigma_{t_j}(g_j),\ j=k-2, k-3, ..., 0.$$

Theorem 3. $h(f((d_{k-1}, d_{k-2}, ..., d_0)))=(d_{k-1}, d_{k-2}, ..., d_0)$.

Proof. Assume that

$$f((d_{k-1}, d_{k-2}, ..., d_0))=(g_{k-1}, g_{k-2}, ..., g_0),$$

and let

$$h((g_{k-1}, g_{k-2}, ..., g_0))=(d'_{k-1}, d'_{k-2}, ..., d'_0).$$

It is clear that $d'_{k-1}=g_{k-1}=d_{k-1}$. For $j=k-2, k-3, ..., 0$,

$$\begin{aligned} d'_j &= \sigma_{t_j}(g_j) \\ &= \sigma_{t_j}(\sigma_{s_j}(d_j)) \\ &= \sigma_{s_j}(\sigma_{s_j}(d_j)) \\ &= d_j. \end{aligned}$$

Note that, by the above definitions, $s_j=t_j=(\sum_{i=j+1}^{k-1} g_i)$ mod 2. □

We have presented a method for generating an (n,k)-Gray code. Since there are n^k codes to be generated and each code needs $O(k)$ time to generate the k digits of the code, a direct implementation takes $O(kn^k)$ time to generate all the elements of the code. We shall present a more efficient algorithm to generate all of the codes. The idea is to generate the next code from the current one. We need to store the current code in an array and to determine which digit should be increased or decreased for the next code. We can decide which digit should be changed by using Theorem 2.

The algorithm is written in C and is shown in Fig. 1. The first code is (0, 0, ..., 0). Each iteration from line 12 to line 25 generates the next Gray code, which is stored in the vector $g[0]$, $g[1]$, ..., $g[k]$. A vector

```
 1 #include <stdio.h>
 2 #define N 20
 3 #define K 16
 4 main()
 5 {
 6       int n[K+1];          /* the maximum for each digit */
 7       int g[K+1];          /* the Gray code */
 8       int u[K+1];          /* +1 or −1 */
 9       int i, j, k;
10
11       for (i=0; i<=K; i++) {g[i]=0; u[i]=1; n[i]=N;};
12       while (g[K]==0) {
13            printf("(");
14            for (j=K-1; j>=0; j--) printf (" %d", g[j]);
15            printf (")\n");
16
17            i=0;           /* enumerate next Gray code */
18            k=g[0]+u[0];
19            while ((k>=n[i]) || (k<0)) {
20                  u[i]=-u[i];
21                  i++;
22                  k=g[i]+u[i];
23            };
24            g[i]=k;
25       };
26 }
```

Fig. 1. Algorithm to generate the (n,k)-Gray code.

$u[0]$, $u[1]$, ..., $u[k]$ is used to indicate whether the algorithm should increase or decrease the value of the $g[i]$, for $i=0, 1, ..., k$. Note that only one digit is changed in each iteration. Initially, all the values of $u[0]$, $u[1]$, ..., $u[k]$ are 1. Each digit of the generated code can only be a number in $[0, n-1]$. The algorithm first tries to increase or decrease the least significant digit $g[0]$, depending on the value of $u[0]$. If the value of $g[0]+u[0]$ is in $[0, n-1]$, then the algorithm has found the next (n,k)-Gray code. The new code is obtained from the old code by replacing the value of $g[0]$ with $g[0]+u[0]$. Otherwise, the value $g[0]+u[0]$ is outside of $[0, n-1]$, the value of $u[0]$ is multiplied by -1, and the value of $g[0]$ remains unchanged. The next digit, $g[1]$, is considered to be the candidate to be increased or decreased. This process is repeated until the algorithm finds a digit $g[i]$ such that $g[i]+u[i]$ is in $[0, n-1]$. Note that the algorithm uses vectors of dimension $k+1$, indexed by 0, 1, ..., k, but that only the first k digits are printed. When the most significant digit $u[k]$ is changed from 0 to 1, the algorithm stops. It is easy to check that, when the algorithm stops, all the elements in $(Z_n)^k$ have been enumerated.

It is sometimes useful to generalize the algorithm described above in such a way that each digit in the (n,k)-Gray code has its own maximum value, instead of a common value n. We store the maximum values in an array, $n[0]$, $n[1]$, ..., $n[k]$. Instead of checking whether $0 \le g[i]+u[i] \le n$, the algorithm checks if

$0 \le g[i]+u[i] \le n[i]$.

III. Applications of the Generalized Gray Code

In this section, we shall describe applications of the (n,k)-Gray code. The first application is calculation of the weight polynomial of a group code. The second application is finding the solution to a variation of the tower of Hanoi problem. The third application is finding a Hamiltonian path in a generalized hypercube network.

1. Calculation of the Weight Polynomial of a Group Code

As a result of noise in a transmission channel, information received by a receiver may contain errors in digital communications. This required the design of good error correction codes, so that transmission errors can be detected and corrected. Algebraic coding theory plays an important role in the design of good error correction codes (Berlekamp, 1984; Blahut, 1983; Blake and Mullin, 1975; MacWilliams and Sloane, 1981).

Let the information being transmitted be the strings of the alphabet $\{0, 1, ..., n-1\}$ for some positive integer n. The value of n is usually a small number, for example, 2, 3, or 4. Assume that the information to be transmitted is divided into blocks of length k. Each block of information is first transformed into a *code word* of length l. The value of l must be at least k and is usually greater than k. The code words are then transmitted to the receiver.

Each block of information can be represented as an element in $(Z_n)^k$. Each code word can be regarded as an element in $(Z_n)^l$. Since $l>k$, not every element in $(Z_n)^l$ is a code word. The receiver can detect errors when a messge which is not a code word is received. Let x and y be two code words. The *Hamming distance* between x and y is defined as the number of digits by which the code words x and y differ. In addition to detecting errors, the receiver can correct m digits of errors if the Hamming distance between any pair of code words is at least $2m+1$, and if there are at most m error digits in each message block received.

Most of the error correction codes are *group codes*, which are codes in which the set of code words is a group. One advantage of group codes is that the minimum distance of the code is equal to the minimum number of nonzero digits over all nonzero code words. Therefore, we need not calculate the Hamming distance between every pair of code words. We need only count the nonzero elements in each nonzero code word.

```
for i=0 to k do w_i=0;
for each u=(u_1, u_2, ..., u_k) in (Z_n)^k do
    d=0;
    for i=1, 2, ..., l do
        s=0;
        for j=1, 2, ..., k do
            s=s+u_j×a_{j,i};
        end for
        if (s mod n)≠0 then d=d+1
    end for
    w_d=w_d+1;
end for
```

Fig. 2. Simple algorithm to compute the weight polynomial of a group code.

The weight polynomial of a group code is the polynomial in x in which the coefficient of x^i is equal to the number of code words which has exactly i nonzero digits. The minimum distance of a group code is the smallest i, such that the coefficient of x^i is nonzero.

A group code is usually generated using a generator matrix. Let A be the generator matrix of a group code. The algorithm shown in Fig. 2 computes the weight polynomial of the group code generated by the generator matrix A, which has dimension $l \times k$.

For each element $u \in (Z_n)^k$, the algorithm shown in Fig. 2 needs kl multiplications and $(k-1)l$ additions to generate the code word. Since there are n^k elements in $(Z_n)^k$, the algorithm needs kln^k multiplications and $(k\text{-}1)ln^k$ additions. When k and l are large, the computation takes a long time to complete. For example, when $n=3$, $k=20$ and $l=100$, the computation takes about a day on a low end workstation. When searching for good error correction codes, we need to try a large number of different generator matrices. Therefore, we need to design a more efficient algorithm to compute the weight polynomial of a group code.

Since there are n^k elements in $(Z_n)^k$ and the weight polynomial depends on each element in $(Z_n)^k$, it is impossible to design a polynomial time algorithm, in terms of n and k, for computation of the weight polynomial. However, we may rearrange the order of the elements in $(Z_n)^k$ properly so as to reduce the computational time. For example, if the elements are ordered in the same way as the generalized Gray code, then matrix multiplication is not needed to generate the code word. The first code word, which is the image of the zero element (0, 0, ..., 0) in $(Z_n)^k$, is (0, 0, ..., 0) in $(Z_n)^l$. If the previous code word has been generated, the next code word can be obtained by adding or substracting the i-th column of the matrix A to the previous code word, where i is the index of the digits in which the two elements disagree. For each code

Table 2. Generator Matrix for Ternary Constabelian [100,16,48]-Code

36758484753675848475367584847536758484753675848475
21638463212163846321216384632121638463212163846321
48757548214875754821487575482148757548214875754821
12751263631275126363127512636312751263631275126363
36363636368484848484757575757575757575758484848484
12121212126363636363757575757563636363631212121212
57575757578484848484848484848457575757571212121212
21212121218484848484212121212163636363636363636363
33887777888877778833777788338877883388778833887777
11667766116677661111776611116666111166771111667766
55888855118888551155885511558855115588881155888855
22882266668822666622226666228866662288226622882266
33778888778877337788778888773377337778888888773377
11771166666666117711771166661166117711661166661177
55881188558855555881188118855555555881188118855558
22226688668866222266226688662266222266886688662222

Table 3. Coefficients of the Weight Polynomial for Ternary Constabelian [100,16,48]-Code

i	coefficient
0	1
48	11600
51	47200
54	331600
57	1354800
60	4098040
63	7683200
66	10915000
69	9737200
72	5952400
75	2247200
78	592800
81	67400
84	8200
90	80

word, we need only l additions and no multiplications. The computation time is reduced to ln^k. The new algorithm will be more than $2k-1$ times faster because most computers can do addition much faster than multiplication. The new algorithm is shown in Appendix.

We have applied the algorithm in finding a good ternary code. It is a special kind of group code, called the *constabelian code*. It is the ideal of the algebra

$$A_F=A(F;a,t)$$

$$=F[X_1,X_2,...,X_r]/(X_1^{t_1}-a_1,\ X_2^{t_2}-a_2,\ ...,\ X_r^{t_r}-a_r).$$

They are *constacyclic codes* as defined by Berlekamp (1984) when r=1. They are *abelian codes* as defined by Berman (1967a, 1967b) and MacWilliams (1970) when a=(1, 1, ..., 1), where 1 is the identity element of F. We have applied the above algorithm in finding many good constabelian codes. The most important one is the ternary constabelian [100, 16, 48]-code. The minimum distance of this code exceeds that of the best known ternary [100, 16, 45]-code listed in the table compiled by Brouwer and Sloane[1].

Let F=GF(3), r=4, a=(2,1,1,1), and t=(2,2,5,5). Thus,

$$A_F=A(F;a,t)$$

$$=F[X_1,X_2,X_3,X_4]/(X_1^2-2,\ X_2^2-1,\ X_3^5-1,\ X_4^5-1).$$

Table 2 shows the generator matrix of the ternary [100,16,48]-code, constructed using the algorithm described by Cheng and Guan (1996). Note that the numbers in Table 2 need to be expanded in base 3. For example, 7 in Table 2 represents two elements 2 and 1 of GF(3) in sequence, and 367 in the first row of Table 2 represents 102021.

Finally, Table 3 lists the non-zero coefficients of the weight polynomial of the ternary [100, 16, 48]-code.

2. Other Applications

In this section, we will describe two additional applications of the (n,k)-Gray code. The first one is finding the solution to a variation of the Tower of Hanoi problem. The second one is finding a Hamiltonian path in a generalized hypercube network.

The tower of Hanoi problem involves transfer of a tower of k disks from the left peg to the right peg using an intermediate peg in between. Only one disk can be moved at a time, and a smaller disk must be placed on top of a larger disk each time. We will consider a variation of the tower of Hanoi problem, in which disks can only be moved to adjacent pegs (Graham *et al*., 1994). The goal is to find the shortest sequence of moves that transfers a tower of k disks from the left peg to the right peg.

First, we number the pegs from left to right as 0, 1, and 2, and number the disks from the largest diameter to the smallest diameter as 1, 2, ..., k, respectively. Assume that disk i is on peg x_i, for i=1, 2, ...,

[1]Brouwer, A. E. and N. J. A. Sloane, "Table of lower bounds on $d_{max}(n,k)$ for linear codes over field of order 3." Private Communication.

k. We shall use $(x_1, x_2, ..., x_k)$ to record the state of the disks. Initially, all the disks are on peg 0, and the state is (0, 0, ...,0). The first move must be to move disk n from peg 0 to peg 1. The state then becomes (0, 0, ..., 0, 1). An example of the tower of Hanoi problem with 3 disks is shown in Fig. 3.

Theorem 4. *The tower of Hanoi problem in which disks can only be moved to adjacent pegs can be solved by the enumeration of the* (3,*k*)*-Gray code, where k is the number of disks to be moved.*

Proof. Let the initial state be (0, 0, ..., 0), and let the j-th state be the j-th (3,k)-Gray code. We will prove, by induction on j, that all moves are legal moves. It is clear that the first move from state (0, 0, ..., 0, 0) to state (0, 0, ..., 0, 1) is a legal move. Let

$$f((d_{k-1}, d_{k-2}, ..., d_0))=(g_{k-1}, g_{k-2}, ..., g_0)$$

and

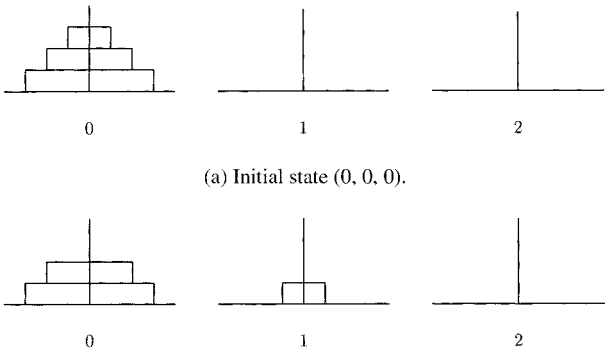
$$f((d'_{k-1}, d'_{k-2}, ..., d'_0))=(g'_{k-1}, g'_{k-2}, ..., g'_0)$$

be two states of consecutive moves. Then, $(d_{k-1}, d_{k-2}, ..., d_0)_n-(d'_{k-1}, d'_{k-2}, ..., d'_0)_n=1$. By Theorem 2, there exists some j, $0\le j\le k-1$, such that

(1) if $j<n-1$, then $g_i=g'_i$, for $i=k-1, k-2, ..., j+1$;
(2) $|g_j-g'_j|=1$;
(3) if $j>0$, then $g_i=g'_i=0$ or $g_i=g'_i=n-1$, for $i=j-1, j-2, ..., 0$.

That is, when the j-th disk is to be moved from peg g_j to peg g'_j, all the smaller disks are either on peg $l=0$ or $l=2$. Since disk j is to be moved from peg g_j to peg g'_j, we need to show that g_j, g'_j, and l are different. Consider the first time that the j-th disk is to be moved. When, for the first time, the j-th digit is changed in the enumeration of the (3,k)-Gray code, the $(j-1)$-th digit must be increased from 0 to $n-1$. Therefore, it must be the case that the j-th disk is to be moved from peg 0 to peg 1, and that all the smaller disks are on peg 2. Consider the second time the j-th disk is to be moved. The $(j-1)$-th digit must be decreased from $n-1$ to 0. It is easy to see that the j-th disk is to be moved from peg 1 to peg 2, and that all the smaller disks are on peg 0. Note that our algorithm generates reflective Gray codes. Following the same argument, we can show that when the j-th disk is to be moved from peg 2 toward peg 0, all the smaller disks will be on one peg, and the peg will not be the same one that the j-th disk was originally on, or the peg that the j-th disk is to be moved to. □

For example, the solution to the tower of Hanoi problem with 3 disks is listed as follows:



(a) Initial state (0, 0, 0).

(b) Move the smallest disk from peg 0 to peg 1, (0, 0, 1).

Fig. 3. The tower of Hanoi problem, n=3.

(0,0,0) (0,0,1) (0,0,2) (0,1,2) (0,1,1) (0,1,0) (0,2,0) (0,2,1) (0,2,2) (1,2,2) (1,2,1) (1,2,0) (1,1,0) (1,1,1) (1,1,2)(1,0,2) (1,0,1) (1,0,0) (2,0,0) (2,0,1) (2,0,2) (2,1,2)(2,1,1) (2,1,0) (2,2,0) (2,2,1) (2,2,2).

Another application of the algorithm is finding a Hamiltonian path in a generalized hypercube network. In a k dimensional binary hypercube, the vertices are k-digit binary numbers. Two vertices are adjacent if, and only if, their binary numbers differ in only one binary digit. Hypercube networks have become popular because they have good connectivity and the degree of each vertex is small.

We consider a generalization of binary hypercube. The vertices of a generalized hypercube are the k-digit base n numbers. Two vertices in a generalized hypercube are adjacent if, and only if, their numbers differ in only one digit, and the difference is either +1 or −1.

Finding a Hamiltonian path is an important problem in graph theory. It is also important in practical network design. Although it is generally difficult to find a Hamiltonian path in a graph, it is known that a binary hypercube has a Hamiltonian path. We can show that a generalized hypercube also has a Hamiltonian path. The path can be found by the algorithm shown in Fig. 1. That is, the Hamiltonian path can be represented by a sequence of vertices, which is the sequence of the (n,k)-Gray code.

Note that when n is even, the sequence of the (n,k)-Gray code has the property that the last element and the first element in the sequence also differ by one digit; therefore, the graph has a Hamiltonian cycle.

IV. Conclusions

We have presented an efficient algorithm to

enumerate the (n,k)-Gray code. It can be used to reduce the computation time required to calculate the weight polynominal of a group code. The algorithm is not recursive. Therefore, checkpoint and restart operations can be implemented efficiently. Computations can also be divided into parts and executed on different computers. It is a useful tool for searching for good group codes. We have used this technique to find the [100,16,48]-code and many other good codes.

We have also shown that the (n,k)-Gray code can be used to solve a variation of the tower of Hanoi problem, in which the disks can only be moved to adjacent pegs. The (n,k)-Gray code can also be used to find a Hamiltonian path in a generalized hypercube network.

Appendix

Algorithm to Compute the Weight Polynomial of a Group Code

```
#include <stdio.h>

#define N 100
#define K 16
#define Q 3

main()
{
    int A[K] [N];       /* generator matrix */
    int u[K+1];         /* the message */
    int g[K+1];         /* to generate Gray code */
    int v[N];           /* the code */
    int w[N+1];         /* the weight polynomial */
    int d;              /* the distance */
    int i, j, k, s;

    for (i=0; i<K; i++) /*  generate a random matrix */
        for (j=0; j<N; j++)
            A[i] [j]=random() % Q;

    for (i=0; i<=K; i++) {u[i]=0; g[i]=1;};
    for (i=0; i<N; i++) v[i]=0;
    for (i=0; i<=N; i++) w[i]=0;
    w[0]=1; u[0]=1;
    i=0; d=N;
    while (u[K]==0) {
        k=0;
        if (g[i]==1) {
            for (j=0; j<N; j++) {
                v[j]=(v[j]+A[i][j]);
                if (v[j]>=Q) v[j]-=Q;
                if (v[j] !=0) k++;
            }
        } else {
            for (j=0; j<N; j++) {
                v[j]=(v[j]-A[i][j]);
                if (v[j]<0) v[j]+=Q;
                if (v[j] !=0) k++;
            }
        };
        w[k]++;
        if (d>k) d=k;

        i=0;            /* generate next Gray code */
        s=u[0]+g[0];
        while ((s==Q) || (s==-1)) {
            g[i]=-g[i];
            i++;
            s=u[i]+g[i];
        };
        u[i]=s;
    };
    printf("\nNN=%d, KK=%d, DD=%d\n", N, K, d);
    for (i=0; i<=N; i++) {
        if ((i%10)==0) printf ("\n");
        printf ("%d", w[i]);
    };
    printf("\n");
}
```

References

Berlekamp, E. R. (1984) *Algebraic Coding Theory*. Agean Park Press, Laguna Hills, CA, U.S.A.

Berman, S. D. (1967a) On the theory of group codes. *Kibernetika*, **3**(1), 25-31.

Berman, S. D. (1967b) On the theory of group codes. *Kibernetika*, **3**(2), 21-30.

Blahut, R. E. (1983) *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, MA, U.S.A.

Blake, I. F. and R. C. Mullin (1975) *The Mathematical Theory of Coding*. Academic Press, New York, NY, U.S.A.

Cheng, Y. and D. J. Guan (1996) *Constabelian Codes*. Technical report, Department of Applied Mathematics, National Sun Yat-sen University, Kaohsiung, Taiwan.

Graham, R. L., D. E. Knuth, and O. Patashnik (1994) *Concrete Mathematics*, 2nd Ed. Addison-Wesley Publishing Company, Reading, MA, U.S.A.

MacWilliams, F. J. (1970) Binary codes which are ideals in the group algebra of an abelian group. *BSTJ*, **49**, 987-1011.

MacWilliams, F. J, and N. J. A. Sloane (1981) *The Theory of Error Correction Codes*. North-Holland, New York, NY, U.S.A.

Gray Code的推廣與其應用

官大智

國立中山大學應用數學系

摘 要

我們提出一個依特定順序產生$(Z_n)^k$的所有元素的演算法。此序列稱為(n,k)-Gray碼。在此序列中，每一對相鄰的元素僅差一個位元，且其差值為+1或-1。此序列可以應用於群碼（group code）權重多項式（weight polynomial）的計算，以減少計算時間。此演算法為非遞回（nonrecursive），且產生下一個元素時只和上一個元素有關。因此可以將計算量大的工作分段來做。此演算法所產生之序列亦可用來解一種Hanoi Tower問題，在此問題中，圓盤僅能移到相鄰的柱子上。此問題的解恰好是$(3,k)$-Gray碼，其中k為圓盤的個數。最後，我們證明(n,k)-Gray碼也是推廣高次立方體（generalized hypercube）中一個Hamiltonian path。