A Simple Algorithm for Generating Subsets of k or Fewer Elements of an n-Set

R. E. Sabin
Computer Science Department
Loyola College
Baltimore, MD 21210 USA
E-Mail: RES@loyola.edu

Abstract. Forming all distinct subsets with k or fewer objects from a set with n elements can be accomplished by generating a subset of the binary reflected Gray code. This paper presents a straightforward algorithm that generates the desired Gray codewords by altering the stack which maintains the transition sequence that determines the next codeword position to be altered.

1. Introduction

Problems that require the formation of subsets of k or fewer elements from a set of n elements occur in coding theory [2] and graph theory, where, for an n-cube with each vertex labeled $(x_1, x_2, \ldots, x_n)$, $x_i = 0$ or 1, traversal of vertices with Euclidean distance of $k^{1/2}$ or less from (0, 0, ..., 0) is desired. The most efficient algorithm would generate the subset by the addition or deletion of a single element to the preceding result [3]. If $k < n$, such an algorithm does not exist. Minimally, the task requires that in several cases the next element is generated by a deletion of a single element followed by the addition of an element [4]. The algorithm presented here, while not minimal, is attractively simple to implement. It utilizes the binary reflected Gray code.

2. Generating a Subset of the Binary Reflected Gray Code

A binary reflected Gray code of length n may be represented by a full binary tree of height n with non-root nodes containing 1 or 0. A codeword is the sequence of data values encountered in the acyclic path from the root to the leaf. Figure 1 shows such a representation for the Gray code of length 5. Note that the tree is shown in a reflected and rotated manner with the root at the extreme left of the figure and what would usually be considered the leftmost leaf (00000) at the top of the figure. For clarity, in referring to this rotated, reflected tree, rather than use the usual "left" and "right" designations, we will refer to each node as an "upper" or "lower" node depending on whether it is above or below its sibling, the node with which it shares its parent. Furthermore, we will associate a *weight* with each node in the tree where weight is the number of 1's encountered in the path from the root to that node (including the node).

If Gray codeword g is indexed $g_n \ldots g_1$, each codeword can represent a subset of the set of n elements where the element with subscript i is included in the subset if and

JCMCC 25 (1997), pp. 175-181

only if $g_i = 1$. Then the number of 1's in the codeword, the weight of the codeword, is the cardinality of the subset corresponding to the codeword..

For a given n, construct a tree of height n and consider the codewords corresponding to leaves of the tree. To generate the subset consisting of codewords with weight is less than or equal to k, the algorithm below will "prune" the tree, allowing only the desired leaves to remain. In Figure 1, for $k = 3$, such leaves are marked with an "*."

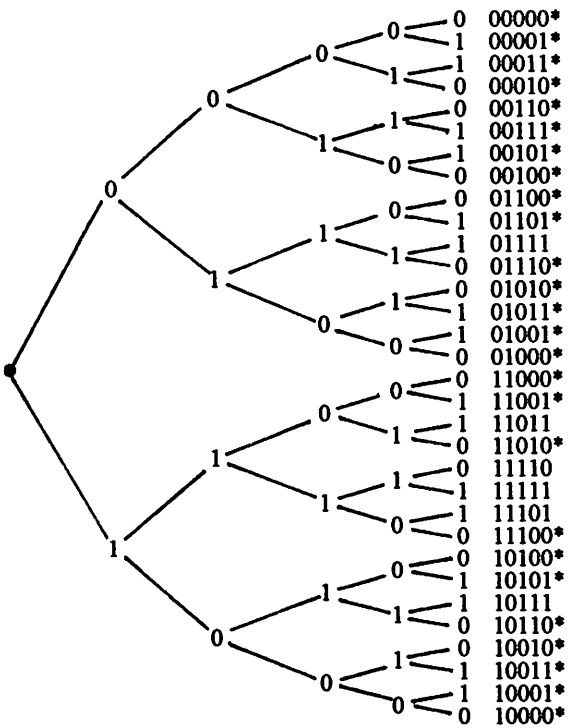


Figure 1 - Tree representation of Gray codewords of length 5

A subsequence of Gray codewords that are not to be generated, *i.e.*,those with weight $k+1$ or more, immediately succeed a codeword of weight k. Since successive codewords differ by weight one, the sequence of "heavy" codewords must terminate with another codeword of weight k. Theorem 1 follows immediately from the structure of the code and is proved in [1].

Theorem 1: Successive binary Gray codewords of length n and weight k differ in exactly 2 positions.

The transition sequence for a binary reflected Gray code is easily generated by use of a stack [1]. The stack is stored in array S where $S(i)$ is initialized to $i+1$, for all i, $0 \le i \le n$ and $S(0)$ is the element at the top of the stack. A value in S is interpreted as a position in the Gray codeword to be altered as well as the location, in S, of the next element in the stack. Gray codewords are generated by Algorithm 1.

Algorithm 1 - Generating binary Gray codewords

$g \leftarrow 0000...0$
$t \leftarrow 0$
while $t < n$ **do**
    $t \leftarrow S(0)$
    complement the t-th position of g
    $S(0) \leftarrow 1$
    $S(t-1) \leftarrow S(t)$
    $S(t) \leftarrow t + 1$

We generate successive Gray codewords of k or lower weight by altering the stack. The following lemmas will prove helpful.

Lemma 1: Of two sibling nodes in the tree representation of a binary reflected Gray code, the upper node has even weight and the lower node has odd weight.

Proof: The nodes on level l represent, in top-to-bottom order, the sequence of Gray codewords of length l. The topmost node represents the all-zero codeword. Since successive Gray codewords differ from each other in exactly 1 digit, the sequence of codewords must have alternating even-odd weights. The sequence of nodes on level l is alternating upper sibling-lower sibling. ■

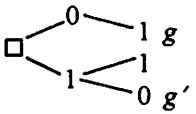
Lemma 2: Let $g_n...g_l$ represent a Gray codeword as well as the path in the binary tree from the root to the leaf corresponding to that codeword. If the rightmost upper node in the path occurs at position t where $t > 1$, then $g_{t-1}=1$ and $g_i= 0$ for all $1 \le i < t$.

Proof: g_t is the root of a subtree representing the binary reflexive Gray code of length $t-1$. The path $g_{t-1}...g_l$ consists of all lower nodes and corresponds to the last Gray codeword of length $t-1$ generated, *viz.*, 100...0. ■

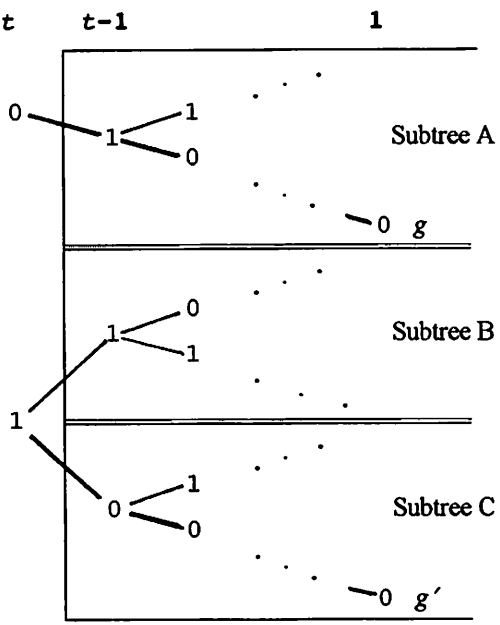
We first consider only the case where k is odd.

Theorem 2: Let $k \le n$ be odd. If Gray codeword g has weight k and the next Gray codeword is to be generated by replacement of a 0 with a 1 in position t of g, the next k-weight Gray codeword, g', differs from g only in positions t and $t-1$ with $g'_t = 1$ and $g'_{t-1} = 0$.

Proof: By Lemma 1, codeword $g = g_n...g_1$ with odd weight k, is a lower node in a leaf pair with a common parent. The next codeword in the sequence is generated by back-tracking from the original codeword (leaf) to the first upper node encountered on the path from that leaf to the root. Here that node is at level $t \geq 2$ with $g_t = 0$. By Lemma 2, $g_{t-1} = 1$ and $g_i = 0 \; \forall \; i, \; 0 \leq i < t$. If $t = 2$, we have the configuration



Clearly, here only 1 leaf need be pruned from the tree; g' is the next codeword of weight k, where g' differs from g only in that $g'_1 = 0$ and $g'_2 = 1$.
If $t > 2$, g has the lowest weight of any leaf in the subtree rooted at $g_{t-1} = 1$, Subtree A below.



Clearly, all leaves in Subtree B have weight greater than k and must be pruned. Leaves in Subtree C form an ordered set that is identical to the leaves of Subtree A except that values at positions t and $t-1$ differ. Hence all leaves in Subtree C except g' are too heavy and are pruned. ■

We test for the conditions of Theorem 2 by storing the weight of the current Gray codeword and the current value of g_t. The structure of the tree reveals that a non-leaf, upper 0 is always the child of another upper node in the tree. The position of an upper node in the tree must be in the stack, since elements in the stack are positions that are to be altered. Thus, when the conditions of Theorem 2 are satisfied, the stack must contain at its top t, the position of the upper 0, and, immediately below it on the stack, $t+1$, the position to the left of the upper zero. The stack's top, t, is stored in $S(0)$. The second element in the stack is $S(t)$ and, when the two top elements of the stack are consecutive, $S(t) = t+1$, the initialized value of $S(t)$. To prune the tree, we generate the next k-weight Gray codeword as described above, then pop the stack to alter the transition sequence. Because of the structure of the stack just described, the stack is popped by the single assignment $S(0) \leftarrow S(t)$.

For the even k case, we initialize g to 100...0, *weight* to 1 and traverse the leaves of Figure 1 in a bottom-to-top order.

Algorithm 2 - Generating binary Gray codewords of weight $\leq k$

```
if k is even
   g ← 1000...0
   weight ← 1
else
   g ← 0000...0
   weight ← 0
t ← 0
while t < n do
     t ← S(0)
     if g_t = 1 or weight < k        { a single adjustment to g is needed }
              if g_t = 1
                   g_t ← 0
                   weight ← weight - 1
              else
                   g_t ← 1
                   weight ← weight + 1
              S(0) ← 1
              S(t-1) ← S(t)
              S(t) ← t + 1
     else                            { two adjustments to g needed - "pruning"}
              g_t ← 1
              g_t-1 ← 0
              S(0) ← S(t)
```

3. Time Complexity of Algorithm 2

Each iteration of the while loop of Algorithm 2 accomplishes the generation of one of the $N = \sum_{i=0}^{k} \binom{n}{i}$ desired Gray codewords. Clearly the algorithm is O(N).

The time may be more precisely estimated by determining the number of times a Gray codeword, representing a subset, is found by pruning the tree, i.e., the number of times a subset is formed by both selecting an element and "unselecting" another element of the last subset formed. This is the number of times the expression $g_t = 1$ or $weight < k$ is false which is the number of times the outer else-block within the while loop of Algorithm 2 is executed. This number can be determined by examining a path on the n-cube.

All distinct n-length strings of 1's and 0's are vertices of an n-cube and may be arranged as a 2-dimensional graph with strings of weight w represented by nodes on level w, as in Figure 2 where $n = 5$. The generation of the binary Gray codewords corresponds to a Hamilton path with all edges drawn between nodes in adjacent levels. (Shown as — in Figure 2.) Algorithm 2 generates a path which consists of portions of this path with one or more edges between elements in level k. Figure 2 displays these edges as dotted --- for $k = 3$.

The number of dotted edges is the number of times the tree is pruned by Algorithm 2, *i.e.*, the number of times the outer else block within the while loop of Algorithm 2 is executed. These edges occur only when a node on level k is the initial point of a downward edge in the Gray graph. Due to the reflective symmetry of the Gray graph, this number is $\binom{n-1}{k}$.

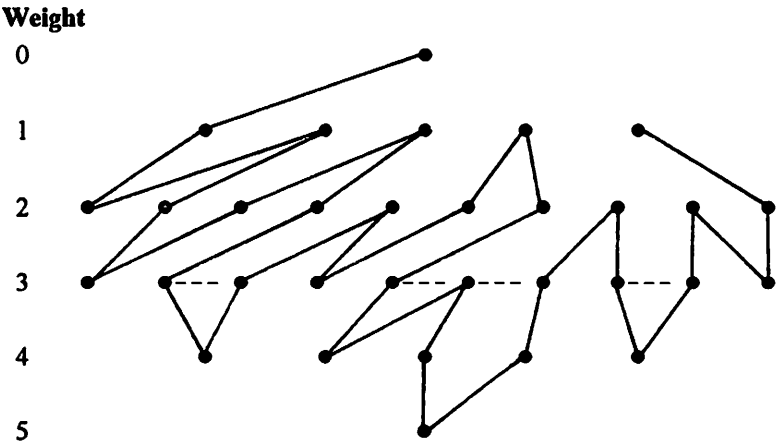


Figure 2 - Paths on the 5-Cube

Let C_1 be the time needed to generate a subset by a single adjustment to the preceding subset and C_2 be the time required to generate a subset that results from pruning. If C_0 is the time required to execute the statements preceding the while loop, total time is estimated by $T(N) = C_0 + C_1 (N - (\binom{n}{k}^{-1})) + C_2 (\binom{n}{k}^{-1})$.

Note that C_1 is greater than C_2 because fewer "stack tending" operations are required to produce a subset by pruning. Pruning involves only popping the stack, a single assignment statement; to produce a subset without pruning involved requires both pushing and popping.

The algorithm does require the actual generation of Gray codewords and not simply transition sequences. In practice, the time involved in updating the Gray codeword (g) can be minimized by considering Gray codewords as arrays of bits and using predefined bit-functions available in most compilers.

The above analysis measures time required by Algorithm 2. If an application uses the Gray codewords generated by Algorithm 2 to operate on an ordered set, a codeword resulting from pruning requires that two adjustments to the selection of elements of the set be made: one previously selected element is to be "unselected" and a previously "unselected" element is to be selected. That these adjustments are always made to elements in adjacent locations (t and t - 1) can lead to optimizations. For example, in the generation of linear codewords that are the sum of k or fewer basis vectors, the real cost of this double alteration may be minimized by pre-computing the sums of "adjacent" basis vectors in the generating set. Then the vector that results from pruning can generated by a single addition to the preceding result.

References

[1] J. Bitner, G. Ehrlich, and E. Reingold, *Efficient Generation of the Binary Reflected Gray Code and Its Applications*, Comm. of the ACM **19** (1976), 517-521.

[2] D. Coppersmith and G. Seroussi, *On the Minimum Distances of Some Quadratic Residue Codes*, IEEE Trans. On Inf. Th. **IT-30** (1984), 407-411.

[3] P. Eades, M. Hickey, and R. Read, *Some Hamilton Paths and a Minimal Change Algorithm*, J. of ACM **31** (1984), 19-29.

[4] V. Job and B. Jamison, *A Minimum Change Algorithm to Generate all Subsets of an N-Set with K or Fewer Elements* (preprint).