Algorithms for Generating Binary Reflected Gray Code Sequence: Time Efficient Approaches

Md. Mohsin Ali, Muhammad Nazrul Islam, and A. B. M. Foysal
Dept. of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna – 9203, Bangladesh
e-mail: mohsin_kuet_cse@yahoo.com, nazrul_bd80@yahoo.com, and foysalking@yahoo.com

Abstract—**In this paper, we propose two algorithms for generating a complete n-bit binary reflected Gray code sequence. The first one is called Backtracking. It generates a complete n-bit binary reflected Gray code sequence by generating only a sub-tree instead of the complete tree. In this approach, both the sequence and its reflection for n-bit is generated by concatenating a "0" and a "1" to the most significant bit position of the n-1 bit result generated at each leaf node of the sub-tree. The second one is called MOptimal. It is the modification of a Space and time optimal approach [8] by considering the minimization of the total number of outer and inner loop execution for this purpose. In this method, both a sequence and its reflection for n-bit is also generated during each inner loop execution instead of generating only one sequence by adding "0" and "2n-1" to the previous n-1 bit sequence. Finally, we evaluate the performance of the proposed and existing algorithms by measuring the execution time with respect to number of bits and compare the results with each other. Simulation results demonstrate that Backtracking takes small execution time up to 10 bits with maximum improvement of 17.02%. The evaluation also demonstrates that MOptimal also takes small execution time for more than 15 bits showing an improvement of 2.10% for 23 bits.**

Keywords - Recursive, Dynamic programming, Space and time optimal, Backtracking.

I. INTRODUCTION

Binary reflected gray code (Gray code) sequence is a binary sequence in which two successive sequences differ only in one bit position. For example, with $N = 3$, the Gray codes are 000, 001, 011, 010, 110, 111, 101, and 100. Gray codes are named for Frank Gray who patented the use of them in shaft encoders in 1953 [1]. It was originally designed to prevent spurious output from electromechanical switches. During the time of designing digital logic circuit from vaccum tubes and electromechanical relays, time counters demanded more power supplygenerating noise spikes since many bits changed at once. In contrast, in Gray code sequence, only one bit is changed when we increment or decrement the value regardless of the number. So, it minimizes the effect of noise spike. Mechanical position sensors use Gray codes to convert the angular position of a disk to a digital form [1]. It has several applications like - solving puzzles such as the Tower of Hanoi and the Brain [2], Hamiltonian circuits in hypercubes [3], Cayley graphs of Coxeter groups [4], capanology (the study of bell-ringing) [5], continuous space-filling curves [6], classification of Venn diagrams [7]. Today, Gray codes are widely used to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

In this paper, we present the derivation and implementation of two algorithms for generating the complete binary reflected Gray code sequence $G(i), 0 \le i \le 2^n - 1$, for a given number of bits n. We evaluate the performance of these algorithms and other related algorithms, and compare their performance with respect to execution time.

The rest of the paper is organized as follows. Related Work is presented in section II. Section III describes the proposed approaches. Performance evaluation is presented in section IV. Finally, section V concludes the paper.

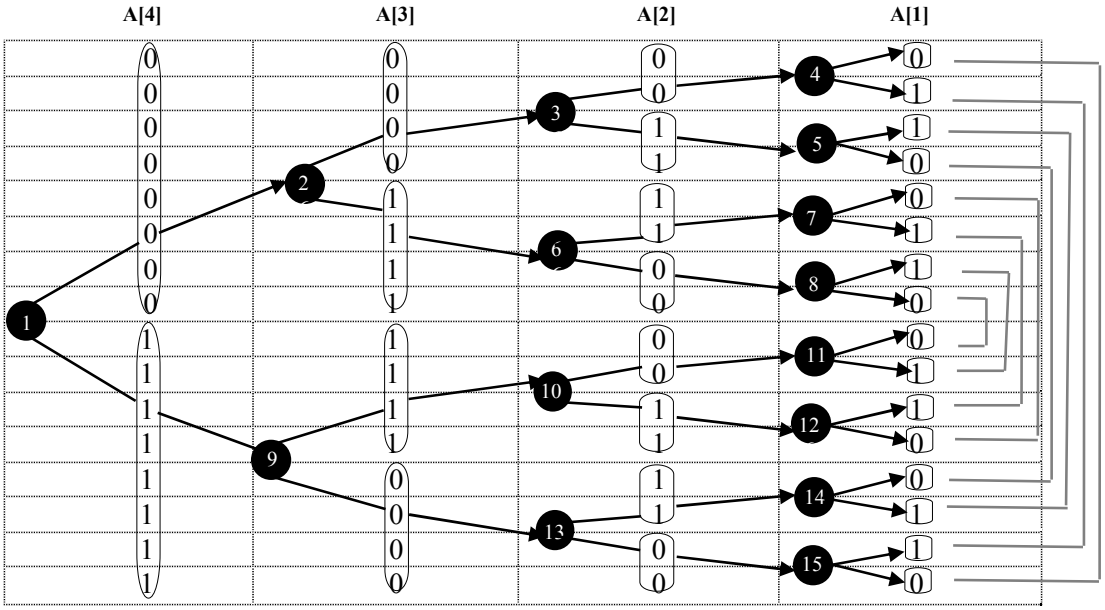
II. RELATED WORK

Phillips and Wick [8] described a *Recursive approach* for generating a binary reflected Gray code sequence. In this approach, the Gray code sequence for n bit system is defined by the recurrence relation. Although, this approach is very efficient for generating only one Gray code value with running time of $O(n)$, but for n bit sequence generation it needs $O(n.2^n)$ time. So, the implementation of this approach is suboptimal, since the desirable and best possible running time for producing and storing 2^n values is $O(2^n)$. The source of inefficiencies of this implementation is that the recursive sub-problems solved during its execution are dependent of each other.

To eliminate the inefficiencies and dependency between the sub-problems of the *Recursive approach*, Phillips and Wick [8] explained a *Dynamic programming approach* for generating a binary reflected Gray code sequence. In this approach, each sub-problem of the original problem is solved only once in such a way that all 1-bit Gray codes are produced before any 2-bit Gray codes, all 2-bit Gray codes are produced before any 3-bit Gray codes, and so on. The implementation of this approach needs to generate and store the entire (n-1)-bit Gray code sequence prior to generating any of the codes in the n-bit Gray code sequence and a two dimensional array is used to store the previous result. Both the space and time complexity of this implementation is

978-0-7695-3591-3/09 $25.00 © 2009 IEEE
DOI 10.1109/ICFCC.2009.41

TABLE I. Illustration of the binary tree based solutions.



$\Theta\left(2^{n+1}\right)$ which is within a constant multiple of the optimal time and space of $\mathrm{O}\left(2^{n}\right)$.

In order to improve the efficiencies of *Dynamic programming approach,* Phillips and Wick [8] proposed a *Space and time optimal approach* called *Optimal* by observing some important principles of the Gray code sequence. The first one of these is that the first half of each *n*-bit Gray code is generated by directly copying the values of the previous (*n*-1)-bit Gray code sequence and pre-pending a leading "0" to it. The second one of these is that the second half of each *n*-bit Gray code sequence is generated by copying the values of the previous (*n*-1)-bit Gray code sequence in reverse order in which they appear in (*n*-1)-bit sequence and pre-pending a leading "1" to it. It produces and stores exactly 2^n values. Both the space and time complexity of this implementation is $O\left(2^{n}\right)$, which is optimal for the generation of a complete *n*-bit Gray code sequence. But in this implementation, the outer loop executes a total of *n-1* times. One of our objectives is to minimize the total number of outer loop execution to achieve the same optimal time complexity.

III. PROPOSED APPROACHES

A. Backtracking approach

Table I shows the 4-bit binary reflected Gray code sequence using binary tree data structure. It shows that visiting right sub-tree or leaf from node 1 through 4 generates "0" for a[4], a[3], a[2], and a[1]. So, the values of a[4], a[3], a[2], and a[1] are initialized to "0". Visiting from right sub-tree or leaf to left sub-tree or leaf from any nodes generate the complemented value of a[4], a[3], a[2], or a[1] for that node. For example, visiting left leaf from node 4 generates the complemented value of a[1] (complement of "0" is "1") and visiting left sub-tree from node 3 generates the complemented value of a[2] (complement of "0" is "1"). After generation of Gray code sequence 0000 at the right leaf of node 4, it backtracks to node 4 and generates Gray code sequence 0001 at the left leaf of node 4. Then, it backtracks to node 3 and generates the left sub-tree. In this process, all the values of the 4-bit binary reflected Gray code sequence is generated after the completion of the complete binary tree according to the indicated order. But, left sub-tree of node 1 (3-bit sequence) is the reflected copy of right sub-tree of node 1 (3-bit sequence). So, we can generate the complete 4-bit binary reflected Gray code sequence by generating only right sub-tree (3-bit sequence) of this node. This is done by generating each Gray code value using 3-bit Backtracking algorithm and pre-pending a "0" to its current array index and a "1" to its reflected array index to generate two values at the same time. The algorithm of this Backtracking approach for generating complete n-bit binary reflected Gray code sequence is shown in Fig. 1.

Notice that each time Backtracking(n) of this algorithm produces 2 values for $n=1$, no values for $n=0$, and $2\times$Backtracking$(n-1)$ values for remaining values of n. So, the space and time complexity of this algorithm is as follows

$$Backtracki\ ng(n)=\begin{cases}2 & if\ n=1\\ 2\times Backtracki\ ng(n-1) & if\ n>1\end{cases} \quad (1)$$

```
Algorithm Backtracking (n)
// a[1:N] is an array of size N which is initialized to 0. N is the
// number of bits in each Gray code and N>1. p is initialized
// to "0", result, result1, and result2 are initialized to NULL,
// and q is initialized to "1". The algorithm is called with
// Backtracking (N) and result1 ∪ result2 is the solution.
1.  result3 ⟵ result
2.  If n>1 and n!=N Then
3.    Do result ⟵ result ∪ a[n]
4.  Else If n=1 Then
5.    Do result1 ⟵ result1 ∪ p ∪ result ∪ a[n] ∪ '\n'
6.       result2 ⟵ q ∪ result ∪ a[n] ∪ '\n' ∪ result2
7.    return
8.  Backtracking (n-1)
9.  a[n-1] ⟵ !a[n-1]     // complement a[n-1]
10. Backtracking (n-1)
11. result ⟵ result3
```

Figure 1. Backtracking approach of generating all *n*-bit gray codes.

So, the algorithm generates

$$Backtracking(n) = 2 \times Backtracking(n-1)$$
$$= 2^{n-1} \times Backtracking(1) = 2^n$$

values. Thus, the time and space complexity of this algorithm is $O(2^n)$.

B. Modification of space and time optimal approach: MOptimal

A close observation of *Space and time optimal approach* [8] reveals that initially it generates 2 Gray code sequences, the outer loop executes for j = 1 to *n*-1, inner loop executes for i = 1 to twoHatJ where Min(twoHatJ) = 2^1 and Max(twoHatJ) = 2^{n-1}, and generates 1 Gray code sequence for each inner loop execution. But, it is observed from the property of Gray code sequence that half of the sequences are the reflection of the remaining half. So, we modify *Space and time optimal approach* [8] by minimizing the total number of outer and inner loop execution by considering this property. This modified algorithm called *MOptimal* is shown in Fig. 2. In this approach, outer loop executes for i = 1 to $\left\lceil \frac{n}{2} \right\rceil - 1$, inner loop executes for j = 1 to twoHatJ where Min(twoHatJ)= 4 and Max(twoHatJ) = $4^{\left\lceil \frac{n}{2} \right\rceil - 1}$, and generates 1 or 3 Gray code sequence and its reflection for each execution of inner loop. Therefore, the algorithm produces and stores

$$4 + \sum_{i=1}^{\left\lceil \frac{n}{2} \right\rceil - 1} 4^i(k) = 2^n \qquad (2)$$

values.
Where,

$$k = \begin{cases} 1 & if\ \ 2i+1 = n \\ 3 & if\ \ 2i+1 \neq n \end{cases} \qquad (3)$$

IV. PERFORMANCE EVALUATION

To evaluate the performance of the proposed and all other algorithms, we implement each algorithms using Java language and execute them using jdk1.3 on a Windows XP Operating System installed computer having configuration of Pentium 4 processor with 3.00 GHz speed and 512 MB of

```
Algorithm MOptimal (n)
// g[0:2^n-1] is an array of size 2^n and g[0], g[1], g[2] and g[3] are initialized with 0, 1, 3 and 2 respectively. n is
// the number of bits in each Gray code. twoHatJ is initialized to 1, doubleTwoHatJ and tripleTwoHatJ are
// initialized to 0 and count is initialized to 4. Array g holds the solution.
1.  For i ⟵ 1 To ⌈n/2⌉ - 1 Do
2.      twoHatJ ⟵ 4×twoHatJ
3.      For j ⟵ 1 To twoHatJ Do
4.          g[count] ⟵ twoHatJ+g[twoHatJ - j]
5.          If (2×i+1)!=n Then
6.              Do doubleTwoHatJ ⟵ 2×twoHatJ
7.                  tripleTwoHatJ ⟵ doubleTwoHatJ + twoHatJ
8.                  g[tripleTwoHatJ-j] ⟵ doubleTwoHatJ + g[count]
9.                  g[doubleTwoHatJ+count] ⟵ doubleTwoHatJ + g[twoHatJ-j]
10.         count ⟵ count+1
11.     count ⟵ 2×count
12. return g
```

Figure 2. Modification of space and time optimal approach [8] for the generation of all *n*-bit gray codes.

TABLE II. Execution time (Milliseconds) of different algorithms for $n = 6$ to $n = 23$.

No. of bits (n)	Existing			Proposed	
	Recursive	*Dynamic Programming*	*Optimal*	*Backtracking*	*MOptimal*
6	0	0	0	0	0
7	15	0	0	0	0
8	31	15	15	15	15
9	62	47	47	31	47
10	109	94	94	78	94
11	203	172	156	156	156
12	469	406	297	407	297
13	1016	875	594	984	594
14	2765	2453	1156	4485	1156
15	5188	4578	2282	20750	2282
16	10938	9625	4532	127656	4500
17	26219	22719	9563	466547	9453
18	56751	49984	19625	936412	19391
19	147761	109375	39859	----	39593
20	453635	305951	76844	----	76421
21	1287722	727854	170781	----	168172
22	----	----	358859	----	354844
23	----	----	721223	----	706140

RAM. The executions of these algorithms are conducted five times for each values of n (number of input bits in the algorithms) and obtain an average of these measured execution times. This process is continued for $n = 2$ to 23. Table II summarizes the measured execution times of these algorithms for $n = 6$ to 23 and Fig. 3 demonstrate the corresponding graph. The observations are as follows.

It is observed from Table II that *Backtracking* takes small time for $n = 6$ to 10 and the improvement is 17.02% compared to *Optimal* for $n = 10$.

From Table II and Fig. 3 it is also observed that *MOptimal* always takes smaller time than all other existing algorithms for more than 15 bits and the improvement is 2.10% compared to *Optimal* for $n = 23$. The reason behind this that the outer loop of *MOptimal* executes for

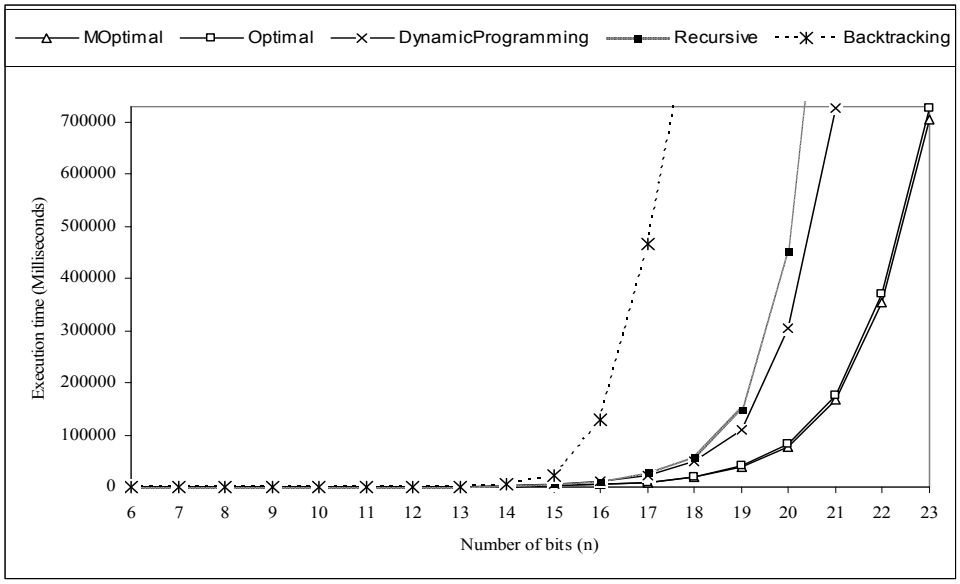


Figure 3. Number of bits Vs. Execution time of different algorithms.

$i = \left\lceil \frac{n}{2} \right\rceil - 1$ instead of j = 1 to *n*-1 that *Optimal* executes and all other remaining algorithms execute loops for more times.

V. CONCLUSION

In this paper, we presented the derivation and implementation of two algorithms named *Backtracking* and *MOptimal* for the generation of complete *n*-bit binary reflected Gray code sequence. Then, we have evaluated each algorithm by executing each program for different input bit values *n* and have obtained the average of five execution times for each input *n*. The evaluation results demonstrate that *Backtracking* takes small time to execute up to a limit of *n* with a significant improvement over *Optimal*. These results further demonstrate that *MOptimal* takes small time to execute for more than a specific *n* values with a significant improvement over *Optimal*.

We are now currently working with genetic algorithms for generating a complete *n*-bit binary reflected Gray code sequence.

REFERENCES

[1] F. Gray, "Pulse code communication," *U.S. Patent 2 632 058,* March 17, 1953.

[1]. M. Gardner, "The binary Gray code," *in Knotted donuts and other mathematical entertainments,* F. H. Freeman and Co., New York, 1986.

[2]. E. Gilbert, "Gray codes and paths on the n-cube," *Bell System Technical Journal 37*, pp. 815-826, 1958.

[3]. J. Conway, N. Sloane, and A. Wilks, "Gray codes and reflection groups," *Graphs and combinatorics 5,* pp. 315-325, 1989.

[4]. A. White, "Ringing the cosets," *Amer. Math. Monthly 94*, pp. 721-746, 1987.

[5]. W. Gilbert, "A cube-filling Hilbert curve," *Math Intell 6,* pp. 78, 1984.

[6]. F. Ruskey, "A Survey of Venn Diagrams," *Elec. J. of Comb.,* 1997.

[7]. A. T. Phillips, M. R. Wick, "A Dynamic Programming Approach to Generating a Binary Reflected Gray Code Sequence," *in Proc. 38th Annual Midwest Instruction and Computing Symposium*, Eau Claire, WI, 9 April 2005.