ISSN: 2455-1848
International Journal of Innovative Research & Growth
A Peer Reviewed & Referred International Journal
Volume-6 Issue-6 April-2018
Impact Factor-3.0
www.ijirg.com, editor@ijirg.com

Total Hamilton Circuits In A Reflected Code Sequence

[1]Ravindra Singh, [2]Preetam Singh Gour, [3]Preeti Dhatterwal, [4]Sandeep Bairwa [5]Suresh Kr. Burdak, [6]Suresh Kr. Borana
[1,3,5,6]Scholar of M.Sc. (physics), Jaipur National University, Jaipur, Rajasthan, India
[2,4]Assistant Prof., Jaipur National University, Jaipur, Rajasthan, India
E-mail- sisodiyaravindrasingh94@gmail.com, singhpreetamsingh@gmail.com

Abstract
In certain problems, it is easy to verify a solution if an answer key is available but it may be difficult to generate the answer key in the first place. One such problem lies in addressing the total Hamiltonians on a hypercube Qn. This problem is solved using the direct search method. Using MATLAB, we have generated all possible reflected codes for n-tuples by our customized backtracking method, mirror and rolling method and using a sequential circuit. We also developed a pruning method to obtain the total 430008 Hamiltonians for a 4-cube case. Finally we have compared the performance of our implementations with the existing work to set the future problem statement.
Keywords: *Gray Codes, Hamiltonian cycles, algorithm.*

1- INTRODUCTION

Gray or Reflected Code (RC) [1] led us to a problem of finding all possible circuits on the n-cube graph sketched for n-bit reflected code sequence. A circuit is defined as a closed loop that begins at a node and travels on edges of the graph to touch all nodes one at a time and reach back to the initial node. This loop is also known as a Hamiltonian circuit (HC) [4]. Similar concept can be traced back in history with the Icosian game [3].

The solution of our problem involves bulk data handling and thus remains inefficiently answered. Solving such class of problems in polynomial time can bring upon a paradigm shift in the living. For instance, finding cure to cancer, solving problems that our present computers cannot, implementation of highly efficient systems and more.

RCs have found applications in digital systems, spectrometry, genetic algorithms, rotary encoders, Venn diagrams, Karnaugh maps, puzzles and many more.

2- OUR WORK

Drawing a tree to find total HCs is a direct approach. It becomes practically unfeasible due to its large tree size. An example is presented in Fig. 1. Another way is to employ a computer. First create a database of all possible permutations of n-tuple binary sequences. Then remove the parts that violate the RC format. This again becomes impractical for bit size n≥4 due to high memory and computation requirements. For example, in MATLAB, for a practicable result the permutations are capped below a vector length of 10. Our customized backtracking code gives all possible RC code sets for n-tuple. For a 3-tuple, all possible 96 RC sequences were found using dynamic and direct approach in O(mn) time (Fig.2).

A single n-bit RC can be made in O(n) time with.

```
v=[0;1];
f= input('Please enter bits to generate Gray Code(n>1): ')
f=f-1;s=0;
while (s~=f)
c=flipud(v);[m n]=size(v);b=[zeros(m,1) v];a=[ones(m,1) c];
join=[b;a];s=s+1;clc
v=join
```

end

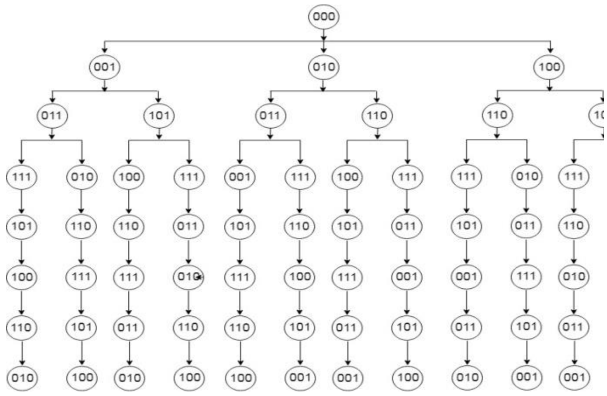


Figure-1 Partial 3-cube tree showing all possible HCs from the node '000'. Eight such graphs will account for the total 96 possible RCs [5].

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

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

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

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

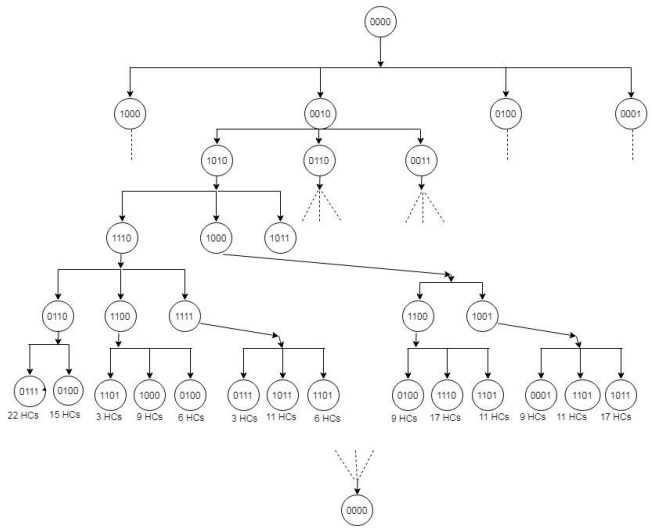
'110' '111' '101' '001' '010' '000' '100' '110'
'011' '010' '110' '100' '111' '101' '100' '000'
'000' '001' '101' '111' '001' '011' '010' '110'
'011' '010' '110' '111' '111' '101' '100' '110'
'101' '100' '000' '001' '010' '000' '001' '011'
'011' '111' '101' '001' '111' '110' '010' '000'
'000' '100' '110'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

Figure-2 All 96 possible Hamiltonians on a 3-cube.

A sequential circuit with shift register and data flip flop is used to generate any n-bit RC sequence (Fig. 3).

In Fig. 4, we propose customized 2-D projections of the 4-cube having minimal overlaps.

Our proposed algorithm to verify total Hamiltonians on a 4-cube analyzes the pruned trees. In a n-tuple code, we begin with a single code and traverse down from an internal node to leaf such that nodes remaining can be permuted in MATLAB (Fig. 5). In our case, the 10 nodes remaining can then be fed to our customized approaches to sift out the valid links. Further by concatenating the results the total possible 43008 HCs are verified for the 4-cube case (Fig. 6).



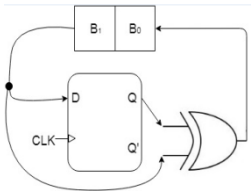


Figure-3 2-Tuple RC generator. Here XOR output =(bi+1)⊕ bi and initially Q = 0.

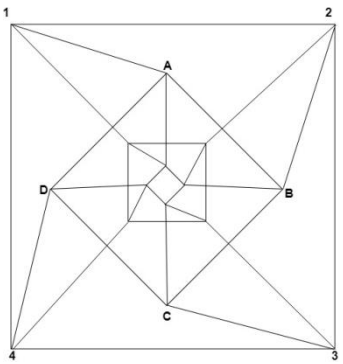


Figure-4 Proposed 4-cube 2D view.

Figure-5 Pruned tree representing HCs on a 4-cube.

	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
TOTAL CODES POSSIBLE:	2688	672	224	75,74 & 75
TOTAL NODES:	16	4	3	3

Figure-6 Total possible branches on a 4-cube tree.

3- CONCLUSION

We have generated total RCs possible for 3-

tuple by our customized backtracking and dynamic method in MATLAB. Further we have implemented a single RC generator using conventional mirror-roll and sequential circuit approach. We have proposed a minimal overlap 2D projection of a 4-cube and a pruning algorithm to reduce computation time for n≥3. All software work was done using computers with/with close specifications to: Intel® Core™ 2 Duo CPU E7500 @ 2.93 GHz and 2GB RAM.

4- FUTURE WORK

Our proposed algorithm can be applied to bit size n≥4. Owing to symmetric shapes [Fig. 4] of a n-cube graph, total RCs can be verified. We aim to generate data to give the total RCs possible by a 7-tuple sequence [6].

5- REFERENCES

[1] Gray, "Pulse Code Communication", U. S. Patent 2 632 058, March 17, 1953.

[2] E. N. Gilbert, "Gray Codes and paths on the n.cube," Bell Systems Technical Journal, 37 (1958), pp. 815-826.

[3] Martin Gardner, "Mathematical Games", Scientific American v.227,n.2 (August, 1972) p.106.

[4] Frank Rubin. A search procedure for hamilton paths and circuits. J. ACM, 21(4):576-580, 1974.

[5] Gardner, M. "The Binary Gray Code." In Knotted Doughnuts and Other Mathematical Entertainments. New York: W. H. Freeman, pp. 23-24, 1986.

[6] A006069, The OEIS Foundation. The Online Encyclopedia of Integer Sequences, 2016. (http://oeis.org)

[7] Shahram Latifi, Eqing Zheng, "Determination of Hamiltonian cycles in cube-based networks using generalized Gray Codes",Computers & ELct. Engg. Vol. 21, No. 3, pp. 189499, 1995, Elsevier Science Ltd.

Appendix A

```
% Customised direct method %
tic
a=[0:3];b=perms(a);[m
n]=size(b);bb=b;bb(:,n+1)=b(:,1);a=zeros(m,
n+1);Gray=0;
for i=1:m;rowtot=0;for
j=1:n;count=biterr(bb(i,j),bb(i,j+1),'overall');
rowtot=rowtot+count;end;if rowtot ==
n;Gray=Gray+1;a(i,:)=bb(i,:);end;
end;c=
a(any(a,2),:);Total_gray_codes=Gray;d =
reshape(cellstr(dec2bin(c)), size(c));
[r c]=size(d);d(:,c)=[];e=d
toc
```

Appendix B

```
% Dynamic method %
clear all;clc;colmn= input('Please enter no.
of codes possible: ');
tic
series=[0:colmn-
1];TotalRows=prod(1:colmn);a=zeros(Total
Rows,colmn+1);
for
Filling=1:TotalRows;series(1,colmn+1)=seri
es(1,1);
rowtot=0;for
jj=1:colmn;count=biterr(series(1,jj),series(1,
jj+1),'overall');
rowtot=rowtot+count;end;if rowtot ==
colmn;a(Filling,:)=series(1,:);end;
series=series(1:colmn);i =
colmn;arr=series;i=colmn;while i > 1 &&
arr(i - 1) >= arr(i)
i = i - 1;end;if i <= 1;result =
[];return;end;result = arr;j = length(result);
while result(j) <= result(i - 1);j = j -
1;end;temp = result(i - 1);
result(i - 1) = result(j);result(j) =
temp;result(i : end) = result(end : -1 : i);
series=result;end;
c= a(any(a,2),:);
d = reshape(cellstr(dec2bin(c)), size(c));
[r c]=size(d);d(:,c)=[]; clc;e=d
[mm nn]= size(e);Total_Codes= mm
toc
```

Appendix C

```
% Proposed Algorithm for level 4 on a 4-
cube %
clear all;clc;a=[0:9];b=perms(a);c=b;[m
n]=size(b);
for i=1:m;for j=1:n;if
b(i,j)==0;b(i,j)=13;elseif b(i,j)==2;
b(i,j)=14;elseif b(i,j)==8;b(i,j)=12;elseif
b(i,j)==9;
b(i,j)=15;else;end;end;end;[m n]=size(b);
```

```
Gray=0;a=zeros(m,n);
for i=1:m;rowtot=0;for j=1:n-1;count=biterr(b(i,j),b(i,j+1),'overall');
rowtot=rowtot+count;end;if rowtot == n-1;Gray=Gray+1;a(i,:)=b(i,:);
end;end;c= a(any(a,2),:);Gray
b=c;e=b;[m n]=size(e);result=zeros(m,n)
for
i=1:m;count=0;counta=biterr(e(i,1),11,'overall');
countb=biterr(e(i,n),0,'overall');count=counta+countb;
if count == 2;result(i,:)=e(i,:);end;end;
ToTal_Paths_on_Route_AA = result(any(result,2),:)
```