Volume 1, No. 4, November 2010
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

GENERATION OF ALL SPANNING TREES A COMBINATORIAL APPROACH

Saptarshi Naskar[1], Krishnendu Basuli[2] and Samar Sen Sarma[3]

[1]Dept. of Computer Science, Sarsuna College, Kolkata, India
sapgrin@gmail.com
[2]Dept. of Computer Science, WBSU, Kolkata, India
krishnendu.basuli@gmail.com
[3]Dept. of Computer Science and Engg., University of Calcutta, Kolkata, India
sssarma2001@yahoo.com

Abstract: This paper deals with all spanning tree generation of a simple, symmetric and connected graph. Since, number of spanning trees of a graph is asymptotically exponential it is our endeavor to generate, all trees in reasonable amount of time and space[1]. The method here is qualitatively and quantitatively better than existing methods. The reason behind the claim is minimum number of duplicate tree comparison and no circuit testing at all for its realization[1,6-11]. We are hopeful that betterment of the algorithm lies in the target of no duplicate tree generation.

Key words: Spanning Tree, Fundamental Circuit Matrix, Degree of Freedom, Transition Vector, Gray Code.

INTRODUCTION

Generation of all spanning trees of a simple, symmetric, connected graph G(V,E); where |V| = number of vertices and |E| = number of edges(In this paper G is used for graphs with the mentioned qualification), is a very old and well known problem of graph theory[2,3]. There are three basic class of tree generation algorithm:

(a) Trees by test and select method.
(b) By decomposition method.
and (c) By Elementary Tree Transformation method.

In this paper we are only concentrating on the 'Elementary tree Transformation method'. The major problem of all tree generation algorithms are[1]:

(a) Huge storage space
(b) Duplicate tree generation
and(c) Exponential time complexity.

That is why this problem is treated as an intractable problem[1,4,5]. According to elementary tree transformation method, first one need to generate an initial tree and then a chord is added to the tree and that forms a circuit with some tree branches[3,5]. Then one tree branch, participating in the circuit, is to be removed at a time, keeping the chord branch intact to form unique trees. Thus all trees can be generated by taking one chord at a time, two chords at a time, so on and replacing them in the tree with same number of tree branches[3,5].

The major advantage of the tree generation using elementary tree transformation algorithm over the other class of tree generation algorithms is it definitely do not produce any non-tree[1]. A close look to the method clears that while generating all spanning trees by the elementary tree transformation method, we need to generate all circuits which is again an intractable problem[1]. The above problem is intractable in the sense that the solution space is exponential[1].

In reference 1 there is an interesting approach by which computational overhead of the generation of all circuits is reduced. We have adopted this idea in our problem and extended this concept by which we reduced the computational overhead, space complexity i.e. storage space and duplicate tree generation.

In the section 2 there is a detail description of the algorithm and the outline of the algorithm. There are some definitions related to the algorithm. We have proposed two theorems and two lemmas and also there are the proofs for the corresponding lemmas and the theorems. In the section 3 there is the description of the algorithm. In the section 4 there are the salient features of the algorithm. In the section 5 there is the main algorithm. In the section 6 there is an example worked out. Also we can find the corresponding technique how the algorithm works. In the section 7 there are two tables one is showing the trees generated and another comparing the time required by a computer having specification (P-VI, Dual Core 2.8 GHZ processor, 512(400) MB RAM) of the present method and the previous other algorithms. In the section 8 there is an explanation of the algorithm for its open problem nature. In the section 9 there is an explanation of the theoretical complexity of the present algorithm. We conclude in the section 10.

ALGORITHM OUTLINE

In this section first we must clearly define some terms related to our algorithm.

Fundamental Circuit Matrix(FCM): This is a matrix with $|e|$ columns and $|e\text{-}r|$ rows, where r is the rank of the graph. First $|n\text{-}1|$ columns are branch edges of the initial tree, next $|e\text{-}r|$ columns are chord branches.

ATM: This is the abbreviation of All Tree Matrix. This matrix contains valid trees.

TATM: This is the abbreviation of Temporary All Tree Matrix.

Set-G: This is the set of cardinality $2^{|e\text{-}r|}$-1. Each element is the gray codes of $|e\text{-}r|$ bit length.

The unit distance chords are gray codes. There are 2^n numbers of bit strings for n bit string, such that successive codes differ by one bit. A gray code can be represented by its transition sequence that is ordered list of the bit positions, numbered from right to left, that change as we go form one code string to the next. The transition sequence can be generated at constant time. If we consider the variables of the gray codes are the chords of a graph and the nullity is $|e\text{-}r|$, then each bit string of Set-G represent transition vectors. The transition vectors help us in the generation of all circuit vectors by modulo-2 sum of two circuit vectors only instead of choosing one circuit, two circuits etc. This reduces the time from $en2^{(n-1)}$ to $e2^n$, by $2/n$ times approximately[1].

In our algorithm in the initial tree a transition vector is added and it results into a circuit. This circuit is broken by deleting m number of tree branches, where m is the existing degree of freedom of the transition vector. Each tree is stored into TATM as dth distance tree. ith transition vector is applied for each of the dth distance tree for formation of next circuit vector i.e. for formation of (d+1)th distance tree. If (i-1)th transition vector contains less number of existing degree of freedom than ith transition vector, otherwise it is applied to initial tree and TATM is made NULL.

THEOREM 1: Algorithm Elementary-Gray-Tree(EGT) generates all trees of G.

PROOF:

According to elementary tree transformation method all possible combinations of circuits are generated using gray code generation tool. Since all possible combination of n bits can be generated using gray code. Here in the EGT generated combinations are actually all possible combinations. Therefore in the above algorithm all circuits are generated, and hence all trees are generated.

THEOREM 2: Duplicate trees are generated at the same distance.

PROOF:

We know that gray codes are unit distance unique code.

According to EGT algorithm a circuit is generated from a gray code and all the spanning trees are stored for the same distance code. Next distance gray code differs from the previously used gray code, in the previous distance by one bit. This bit is nothing but the chord bit.

Hence present tree set must contain a new chord which was present in the previously generated tree set.

Hence it is not possible to generated duplicate trees in the different distances. Hence the theorem is proved.

LEMMA 1: If a circuit vector contains no tree branches that can not generate any new tree.

PROOF:

A circuit vector contains some cord branches and some tree branches. That indicates the chords are forming circuits with the tree branches.

If a circuit vector contains no tree branches that means a circuit is formed using only chords. According to EGT there is no tree branches remains that can be replaced by the chords. Hence no new trees are possible in this case.

Hence, the lemma is proved.

LEMMA 2: Collection of all rows in ATM is the set of all trees of G.

PROOF:

Since, Set-G is the all possible combination of chords. As all the transition vectors in Set-G are unique, each combination is unique. Hence generated trees from each distinct combination are unique. As all possible combination of such trees are generated. Hence the lemma is proved.

ALGORITHM DESCRIPTION

An initial tree T_0 is generated from G by **DFS** method. Say rank of the graph is r and nullity is $|e\text{-}r|$ .Then store row-0 by {1,1,…….,rth },{0,0,….(e-r)th } at **ATM**.

Transition sequence is generated by using a stack operation as follows:

The stack initially contains $n(=|e\text{-}r|)$ elements as n,(n-1),(n-2)………..,2,1. (with 1 on the top). Algorithm POPs off the top element, i and puts it into the sequence;i-1, i-2,……1 are then pushed onto the stack. If there is nothing to push, stop, from the transition sequence.

Set-G is constructed. **FCM** is also constructed.

Add g_i from Set-G in jth row of **TATM** ($\forall$j=1,2,……..). If the circuit row contains (m+1) 1's; ring sum all the chord rows of **FCM** to have the branch set is to be deleted one at a time.

If ring sum results into ϕ reject the g_i i.e. chords are forming circuit to themselves.

If circuit contains (n+1) 1's identifies new chord in g_i which is not in g_{i-1} makes extra circuit to branches. Hence need to remove.

This new row is inserted in **TATM** at the end if ring sum to existing rows to the new one is not ϕ else the new row is rejected. Thus duplication is removed.

Same process is repeated for all jth rows in **TATM** when j is maximum then copy **TATM** and **ATM** and empty **TATM**.

If g_{i+1} contains more 1's than g_i ,g_i is assigned by g_{i+1} process is repeated. Otherwise g_i is assigned by g_{i+1} and **ATM** and **TATM** is made empty except row ϕ of **ATM** and same process is repeated. Thus all distance spanning trees are generated, since we have exhausted all the distinct circuits.

SALIENT FEATURES OF THE ALGORITHM

(i) All circuits generated using gray code reduces the time complexity of all circuit generation.

(ii) Since only a single chord is added if necessary, for generation of a new circuit. The tree generation is practically reduced to addition of only a new tree branch. This reduces space complexity of the algorithm drastically.

(iii) Duplicate tree testing is there, but the storage requirement is under practical limit.

PROPOSED ALGORITHM

Input: Incidence matrix of a simple, symmetric and connected graph *G*.

Let $N = 2^{(e-r)}-1$

Output: ATM matrix of T×*e* order. Where, T is number of trees and e number of edges of the graph.

ALGORITHM: ALL_TREES

```
Step 1: input the incidence matrix M of graph G.
Step 2: $T_0$=Call DFS.
Step 3: B={ $b_i$; $\forall$ i=1,2,…,r; r is the rank of the graph}
        C={$c_i$; $\forall$ i=1,2,…, (e-r); e is the number of edges of G}
        GM={$g_i$; $\forall$ i=1,2,…, N}
        To construct GM, Call GRAY_CODE_MAT.
Step 4: F=Fundamental Circuit Matrix, To construct F Call FCM
Step 5: Initialize ATM and TATM by $T_0$. i$\leftarrow$1.
Step 6: Identify the Chord edges from $g_i$, and ring-sum the corresponding rows of the
        F to identify the circuit branches.
Step 7: Eliminate one branch at a time, except the chord branches and store them in
        $TATM_j$ i.e. at jth row.
Step 8: If jth row contains n elements goto step 17
Step 9: j$\leftarrow$j+1.
Step 10: Repeat through Step 7 while all branches are not been exhausted.
        Step 11:Remove the twin rows from TATM, and
        copy all the rows of TATM to ATM.
Step 12: If $g_i > g_{i+1}$ goto Step 15, else Continue.
Step 13: i$\leftarrow$i+1;
Step 14: If $I \le N$ goto Step 6, else goto Step 18.
Step 15: Delete All rows from TATM except the 1st row, copy all the rows of TATM
          back to ATM.
Step 16: i$\leftarrow$i+1, goto Step 6.
Step 17: Repeat through Step 11.
Step 18: Stop.
```

ALGORITHM: DFS[5]

```
Step1: i$\leftarrow$ i+1
Step2: num(v) $\leftarrow$ i
Step3: For $w \in$ Adj (v) Do
Step4:      If num(w) = 0 Then
Step5:          (v,w) is a tree edge
Step6:          T $\leftarrow$ T $\cup$ {(v,w)}
Step7:          Call Procedure DFS(w,v)
Step8:      Else If num(w) < num(v) and w $\neq$ u Then
Step9:          (v,w) is a back edge
Step10:     B$\leftarrow$ B $\cup$ {(v,w)}
Step11:     End if
Step12: End For
Step13: Return
```

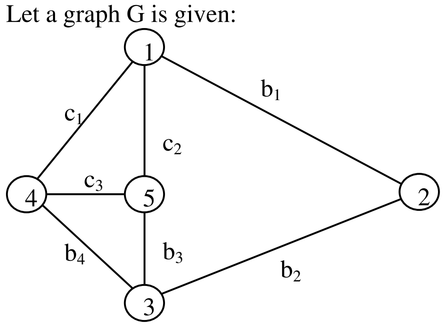
ALGORITHM: GRAY_CODE_MAT[7,10]

```
Step1: GM(i)$\leftarrow$i  $\forall$i=1,2,3,…
Step2: Copy Entire GM to TGM
Step3: i$\leftarrow$0, r$\leftarrow$0
Step4: If TGM(i)$\neq$0 Then Continue Else Goto Step 14
Step5:          n=PAT(r)=TGM(i)
Step6:          r$\leftarrow$r+1
Step7:          Shift Left one bit of TGM
Step8:          j=Position of n in GM
Step9:          IF j$\neq$0 Then Continue Else Goto Step 13
Step10:         Shift Right TGM by one bit and Assign the value in GM
Step11:  End IF
Step12: End If
Step13: Repeat Through Step 4
Step14: Return
```

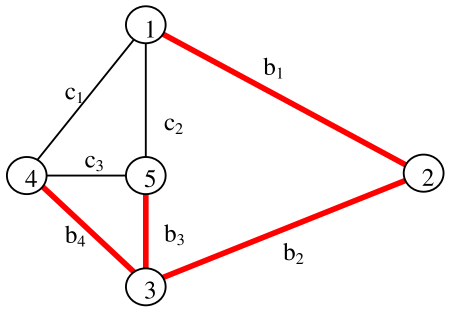
ALGORITHM: FCM[6]

EXAMPLE WORKED OUT

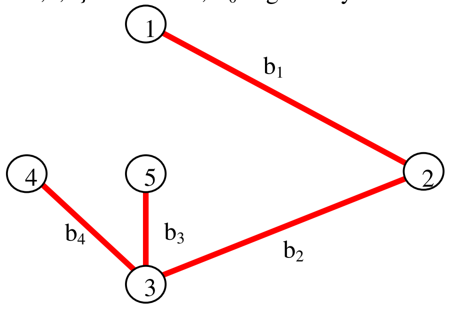
Let a graph G is given:



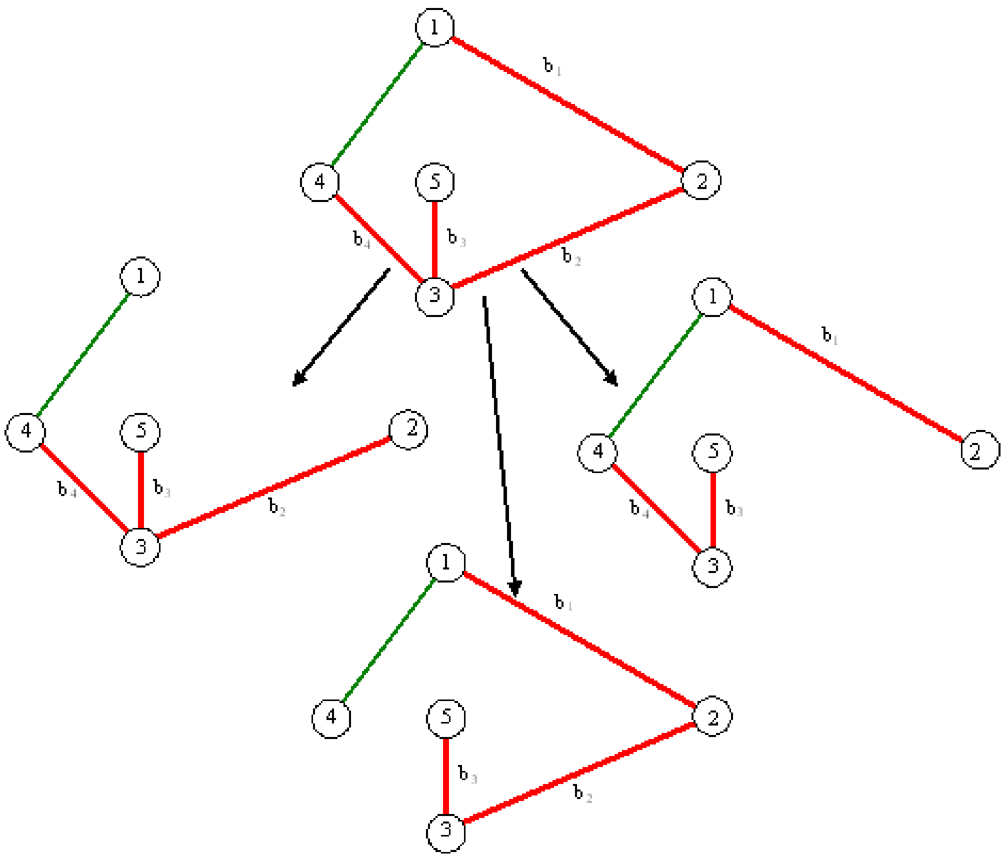
Step 1: A tree T_0 is generated from G using DFS.



Here, set of branches of the initial spanning tree, B={b_i, $\forall$ i= 1,2,3,4} and set of chords of the initial tree, C={c_i, $\forall$ i=1,2,3}. Therefore, T_0 is given by:



Step 2:Now chord c_1 is placed in the tree and produced a circuit with b_1, b_2 and b_4. Then keeping c_1 intact and deleting b_1, b_2 and b_4 one by one we produce all the distance 1 trees.



Thus all distance trees can be generated.

Table 1: This table can be used to show all the trees ATM(*All Tree Matrix*)

Tree Distance	**Tree No.**	b_1	b_2	b_3	b_4	c_3	c_2	c_1	**Gray Code**
0	1	1	1	1	1	0	0	0	000
1	2	0	1	1	1	0	0	1	001
	3	1	0	1	1	0	0	1	
	4	1	1	1	0	0	0	1	
2	5	1	0	1	0	0	1	1	011
	6	0	1	1	0	0	1	1	
	7	1	1	0	0	0	1	1	
	*	0	1	1	0	0	1	1	
	8	0	1	0	1	0	1	1	
	*	1	0	1	0	0	1	1	
	9	1	0	0	1	0	1	1	
1	10	1	0	1	1	0	1	0	010
	11	1	1	0	1	0	1	0	
	12	0	1	1	1	0	1	0	
2	13	1	0	1	0	1	1	0	110
	14	1	0	0	1	1	1	0	
	15	0	1	0	1	1	1	0	
	*	1	0	0	1	1	1	0	
	16	1	1	0	0	1	1	0	
	17	0	1	1	0	1	1	0	
	*	0	1	0	1	1	1	0	
3	#	0	0	0	0	1	1	1	111
	#	0	1	1	1	1	0	1	

2	#	1	0	1	1	1	0	1	101
	#	1	1	0	1	1	0	1	
2	18	0	1	1	0	1	0	1	101
	19	0	1	0	1	1	0	1	
	20	1	0	1	0	1	0	1	
	21	1	0	0	1	1	0	1	
	22	1	1	0	0	1	0	1	
	*	0	1	0	1	1	0	1	
	*	1	0	0	1	1	0	1	
1	23	1	1	1	0	1	0	0	100
	24	1	1	0	1	1	0	0	

Here # means Non-trees and * means Duplicate Trees.

EXPERIMENTAL RESULTS

Table 2: This table contains a comparison of the algorithm with the test and select method in CPU time.

		Test and Select	Present	%
Vertices	Edges	Time (ms)	Time (ms)	
4	6	0	0	0
5	7	0.01	0.01	0
5	9	0.021	0.01	1.1
6	10	0.049	0.04	0.9
6	12	0.049	0.045	0.4
7	13	0.068	0.057	1.1
7	15	0.07	0.069	0.1
8	16	0.087	0.081	0.6
8	18	0.09	0.0815	0.85
9	19	0.09	0.09	0
9	21	0.12	0.1	2
10	22	0.133	0.129	0.4
10	24	0.144	0.141	0.3
11	25	0.155	0.153	0.2
11	27	0.167	0.165	0.2
12	28	0.299	0.27	2.9
12	30	0.351	0.35	0.1
13	31	0.47	0.43	4
13	33	0.566	0.51	5.6
14	34	1.1	1.09	1
14	36	1.29	1.23	6
15	37	1.5	1.5	0
15	39	1.8	1.5	30
16	40	2.01	1.883	12.7
16	42	2.01	1.9	11
17	43	2.4	2.3	10
17	45	2.55	2.5	5
18	46	2.9	2.839	6.1
18	48	3.1	3.078	2.2
19	49	3.61	3.5	11
19	51	3.667	3.6	6.7
20	52	3.8	3.795	0.5
20	54	4.12	4.034	8.6

Figure 1: This is the corresponding graph representation of the Table 2 data.

OPEN PROBLEM

We are in search of an algorithm which can generate all spanning trees of a simple, symmetric and connected graph. The existing algorithms all takes too much time for direct and/or indirect testing of circuits. Our algorithm which takes cyclic interchange procedure needs indirect way for generation of all trees some circuit testing and also some time is taken for duplicate tree testing. The most efficient decomposition algorithm takes O(eN) time where e is the number of edges and N is number of spanning trees. It is open to reduce the efficiency of all trees generation in O(N) times. All though we have no way to reduce exponential times N as it is the number of spanning trees of the graph.

COMPUTATIONAL COMPLEXITY

Let, number of vertices for the given graph $\boldsymbol{G}$ is $\boldsymbol{V}$, number of edges is $\boldsymbol{E}$ and the number of tree possible for the graph is **N**. Then the algorithm must output **N** number of trees. Now, to generate a single tree form the module **ALL_TREES**:
(i) To generate T_0 the required time complexity is of $\mathbf{O}(\boldsymbol{E}\mathbf{log}(\boldsymbol{V}) + \boldsymbol{V})$.
(ii) For the module **Gray_Code_Mat** $\mathbf{O}(\boldsymbol{V}^2)$
(iii) For the module FCM time required is $\mathbf{O}(\boldsymbol{V}^2)$
Since, in the limiting case **N** actually dominating factor over V^2 then over all time complexity for the proposed algorithm is $\mathbf{O}(\boldsymbol{E}\mathbf{log}(\boldsymbol{V}) + \boldsymbol{V} + \mathbf{N})$.
Now the space complexity for the algorithm is only the space required to represent the single tree. And since, only single level trees are required to store then at most V rows are required. Hence the space required for this algorithm is $\mathbf{O}(\boldsymbol{V}^2)$.
Since, **N** represents number of trees it may be exponential in the worst case. Obviously, the conclusion is trivial as we are generating all trees of any graph.

CONCLUSION

We have highlighted elementary tree transformation method in this paper. The method takes the advantage of all circuit generation using the advantage of transition sequence generation from binary to gray code. For a large number of trees, the advantage in enormous. And we consider that the method supersedes the existing methods, mainly at this point. The secondary reason for its supremacy is that, since duplicate testing are limited to a small fraction of circuits the space complexity is within a practical limit. We conclude here and wait for the betterment in near future.

REFERENCES

[1] B. Rao and V. G. K. Murti. Enumeration of All Trees a Graph Computer Program, Electronics Letters, Vol. 6, No. 4, 1970.
[2] B. Rao and V. G. K. Murti. Enumeration of All Trees a Graph Computer Program, Electronics Letters, Vol. 6, No. 4, 1970.
[3] H. M. Trent, A Note on the Enumeration and Listing of All Possible Trees in a Connected Linier Graph, Proc. Nat. Acad. Sci. U.S.A., Vol. 40, p. 1004.
[4] I. Pak and A. Postnikov, Enumeration of Spanning Trees of Graphs, Harvard University, Massachusetts Institute of Technology, 1994.
[5] M. Peikarski, Listing of All Possible Trees of a Linear Graph, lbid., CT-12, Correspondence, pp. 124-125, 1965.
[6] N. Deo, Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall of India Private Limited, New Delhi, 2003.
[7] Reingold, Nievergelt and Deo, Combinatorial Algorithms, Prentice-Hall, Inc., 1977.
[8] S.L. Hakimi, On the Trees of a Graph and Their Generation. J. Franklin Inst. 270, pp. 347-359, 1961.

[9] S. Seshu and M. B. Reed, Linear Graphs and Electrical Networks, Rading, Mass, Addision-Wesley, 1961.
[10] S. Sen Sarma, A. Rakshit, R. K. Sen, A. K. Choudhury. An Efficient Tree Generation Algorithm, Journal of the Institution of Electronics and Telecommunications Engineers (IETE), Vol. 27, No. 3, pp. 105-109, 1981.
[11] W. Mayeda, Graph Theory, Wiley Inter-science, 1972.
[12] W. Mayeda & S. Seshu, Generation of Trees without Duplications, IEEE Trans, CT-12, pp. 181-185, 1965.