

# Generation of All Spanning Trees in the Limelight

Saptarshi Naskar<sup>1</sup>, Krishnendu Basuli<sup>2</sup>, and Samar Sen Sarma<sup>3</sup>

<sup>1</sup> Department of Computer Science, Sarsuna College, India  
sapgrin@gmail.com

<sup>2</sup> Department of Computer Science, WBSU, India  
krishnendu.basuli@gmail.com

<sup>3</sup> Department of Computer Science and Engineering, University of Calcutta  
92, A. P. C. Road, Kolkata – 700 009, India  
ssarma2001@yahoo.com

**Abstract.** Many problems in science and engineering [1, 3, 8, 10] can be formulated in terms of graphs. There are problems where spanning trees are necessary to be computed from the given graphs. Connected subgraph with all the  $n$  vertices of the graph  $G(V,E)$ , where  $|V|=n$ , having exactly of  $n-1$  edges called the spanning tree of the given graph. The major bottleneck of any tree generation algorithm is the prohibitively large cost of testing whether a newly born tree is twin of a previously generated one and also there is a problem that without checking for circuit generated subgraph is tree or non-tree. This problem increases the time complexity of the existing algorithms. The present approach avoids this problem with a simple but efficient procedure and at the same time ensures that a large number of non-tree subgraphs are not generated at all.

**Keywords:** Spanning Tree, Vertex Connectivity, KMTT, PRIES, SPRIES.

There are three distinct classes of existing tree generation algorithms, viz., (a) Trees by examination of all  ${}^eC_{n-1}$  sets of edges, where  $e$  and  $n$  are number of edges and number of vertices of a simple, connected graph, respectively [2, 11] (b) Trees by cyclic interchange method [3, 6, 10] and (c) Trees by decomposition method [3]. Generation of trees involves three basic questions: (a) What percent of  ${}^eC_{n-1}$  edge combinations turns out to be tree? (b) How efficient is the tree-testing algorithm? and (c) How much storage is required?

In this context, the algorithm proposed in this paper generates a very small number of non-trees; its storage requirement is independent of the number of trees and the associated testing procedure is simpler and efficient. The works of Peikarski [7] and Sen Sarma [1] contain the idea of the present method. The major achievements include rejection of non-trees prior to testing and the removal of storage limitations and a part of generated subgraph is tree without checking for circuit.

An undirected graph  $G=(V,E,F)$  consists of a set of vertices  $V$ , a set of edges  $E$  and a function  $F$  that maps each edge  $e \in E$  onto an unordered pair of vertices. Here  $G$  is a simple, symmetric and connected graph, i.e.,  $G$  has no self-loops, parallel edges and edge orientations. A tree  $T$  of a graph  $G$  of  $V$  vertices is a loop free subgraph encompassing all the vertices of  $G$  [3, 8].

*Kirchhoff's Matrix Tree Theorem (KMTT)*: For a given a graph  $G$  with no self-loops, parallel edges and edge orientations, let  $B_0$  be its incidence matrix with one row removed, and  $B_0^t$  be the transpose of  $B_0$ , then determinant  $|B_0.B_0^t|$  gives the number of spanning trees of  $G$  [5].

Since a tree  $T$  of  $G$  with  $n$  vertices has  $n-1$  edges, one can generate all  $(n-1)$  edge combinations and filter a subset of them by a testing sieve that allows only trees to pass. A preferable algorithm will be that generates only trees. The present algorithm though generates non-tree edge combinations, but the number of such combinations relative to the number of trees is drastically reduced. For this we generate a set of convenient data structures, namely *PRIES* and *SPRIES* of a graph, at the outset from the incidence matrix of the given graph. Now the proposed algorithms are given below:

**Algorithm 1.** *Generation of SPRIES Matrix.*

**Input:** The incidence matrix  $A$  of the given graph  $G$ .

**Output:** The *SPRIES* matrix.

Step 1: From the incidence matrix  $A$  find the maximum degree vertex, delete the row and hence obtain *PRIES* [1].

Step 2:  $i \leftarrow 2$ ;  $\text{maxlim} \leftarrow \lfloor 2e/n \rfloor$

Step 3: Choose next highest degree vertex.

Step 4: Keep all the edges in the  $i^{\text{th}}$  column, leaving the edges already chosen and shift other edges to the right.

Step 5:  $i \leftarrow i+1$

Step 6: Repeat through Step 3 while  $i \leq \text{maxlim}$ .

Step 7: For  $i=1$  to  $n-1$

Step 8: For  $j=1$  to  $\text{maxlim}$

Step 9: If  $A[i][j] \neq \text{null}$

Step 10:            $\text{arr}[i]=A[i][j]$

Step 11:            $\text{break};$

Step 12: End If

Step 13:  $j \leftarrow j+1$

Step 13: End For

Step 14:  $i \leftarrow i+1$

Step 15: End For

Step 16: For  $i=1$  to  $n-1$

Step 17: If  $\text{arr}[i] = \text{null}$

Step 18:           Insert all edges that are incident to the priority vertices of  $A[i]$  into the priority columns.

Step 19: End If

Step 20:  $i \leftarrow i+1$

Step 21: End For

Step 22: Stop.

**Algorithm 2.** *Combination Generation for Trees only from Privileged Columns.*

**Input:** The *SPRIES* matrix  $A$  (derived from Algorithm 1).

**Output:** The trees one at a time.

Step 1: Choose elements from *SPRIES* in such a way that (i) at least one element from 1<sup>st</sup> column, (ii) exactly one element from each row.

Step 2: If all elements are selected from privileged columns, then go to Step 3 otherwise go to Step 1.

Step 3: If all the selected elements are unique then output the combination as tree.

Step 4: Repeat through Step 1, while all privileged columns are exhausted.

Step 5: Stop.

**Algorithm 3.** *Combination Generation for Trees from Non-Privileged Columns.*

**Input:** The *SPRIES* matrix.

**Output:** The trees one at a time.

Step 1: Choose elements from *SPRIES* in such a way that (i) at least one element from 1<sup>st</sup> column, (ii) exactly one element from each row.

Step 2: If chosen elements are taken from non-privileged columns, go to Step 3 else go to Step 1.

Step 3: If all the chosen elements or just subset of the elements are present in the privileged columns and also they are present in the same row or the combination are not distinct do not choose the combination as a tree.

Step 4: Repeat through Step 1, while all privileged columns are exhausted.

Step 5: Stop.

**Algorithm 4.** *Tree Testing Algorithm* [3]

**Input:** The adjacency matrix  $M$  of the given graph  $G$ .

**Output:** Checking whether an  $(n-1)$ -edge combination of  $G$  is a tree.

Step 1: Read  $G$  initialize subgraph  $g$  by  $G$ .

Step 2: Select vertex  $i$  in  $g$ .

Step 3: Fuse all vertices adjacent to  $i$  with  $i$ , and call the new vertex  $i$ .

Step 4: Is the number of vertices nonadjacent to  $i$  same as before fusion? If no, repeat through Step 3.

Step 5: Delete from  $g$ , vertex  $i$  (along with all vertices fused with  $i$ ) call the remaining subgraph as  $g$ .

Step 6: Is any vertex left in  $g$ ? If yes, do not take the combination as tree, else call the combination as tree of the graph  $G$ .

Step 7: Stop.

In the present method the algorithms presented, graphs are represented by the adjacency list, hence the storage requirement i.e. space complexity is proportional to

$en$ , where  $e$  is number of edges and  $n$  is the number of vertices of the graph  $G$ . Hence, the space complexity is  $O(en)$ . The time complexity is given below:

For Algorithm 1: In worst case  $\lfloor 2en \rfloor$  is  $(n-1)$ , and choosing the highest degree vertex then need to search total  $n^2(n-1)/2$  elements . So the time complexity is  $O(n^3)$ .

For Algorithm 2: In this algorithm the time complexity is  $O(n)$ . Since  $n-1$  combination is required for individual tree generation.

For Algorithm 3: In this algorithm the time complexity is  $O(n)$ . Since  $n-1$  combination is required for individual tree generation.

For Algorithm 4: In worst case all  $n-1$  columns are fused with  $i^{\text{th}}$  column and in each fusion one perform at most  $n$  logical addition. Hence the time complexity is  $O(n^2)$ .

Since exponential trees are the output of the algorithm, physical time measurement is of no concern here. However we have noted that, for a large graph the computation time is of the order of several minutes using a Pentium IV based Personal Computer.

V	E	C			T	% of Tree (T/C)*100		
		No of Combinations				Brute-Force	PRIES	SPRIES
No. of vertices	No. of edges	Brute-Force	PRIES	SPRIES	Trees			
4	6	20	16	16	16	80	100	100
5	8	70	42	40	40	56	95	100
5	9	126	79	75	75	60	95	100
6	12	792	348	336	300	38	86	89
7	17	12376	3934	4025	3024	25	76	75
7	19	27132	10320	8575	8232	30	80	96

In this paper an algorithm is proposed for computing all possible spanning trees of a simple, connected, non-oriented graph. This algorithm, in general, outperforms the algorithm for computing the same in [1], where a large number of non-tree combinations are generated. This algorithm is also generating such undesired non-tree combinations, but the number of generating such combinations is much less here. In every case of the experimental results, it is been verified that results computed by present algorithm and by the method developed in [1] with the results computed using *KMTT* [5]. The immediate objective is to enrich the algorithm so that no non-tree combinations are obtained.

As the number  $n$  of vertices of a simple, connected, non-oriented graph increases, the procedure predominantly surpasses the *PRIES* technique [1]. Observed that, for the graphs with much less number of vertices of degree much higher than the rest, the percentage of non-tree generation becomes negligibly small. Since the time required in computing the trees is proportional to the number of combinations of edges to be tested for trees, a little extrapolation shows how efficient the present algorithm is. The efficiency criterion is still far behind the ideal situation, where no testing for trees is necessary.

## References

- [1] Sen Sarma, S., Rakshit, A., Sen, R.K., Choudhury, A.K.: An Efficient Tree Generation Algorithm. *Journal of the Institution of Electronics and Telecommunications Engineers (IETE)* 27(3), 105–109 (1981)
- [2] Rao, B., Murti, V.G.K.: Enumeration of All Trees a Graph Computer Program. *Electronics Letters* 6(4) (1970)
- [3] Deo, N.: *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall of India Private Limited, New Delhi (2003)
- [4] Sen Sarma, S., Rakshit, A., Sen, R.K., Choudhury, A.K.: An Efficient Tree Generation Algorithm. *Journal of the Institution of Electronics and Telecommunications Engineers (IETE)* 27(3), 109 (1981)
- [5] Bollobas, B.: *Modern Graph Theory – Kirchhoff’s Matrix Tree Theorem*, p. 54. Springer International Edition, New York (2002)
- [6] Hakimi, S.L.: On the Trees of a Graph and Their Generation. *J. Franklin Inst.* 270, 347–359 (1961)
- [7] Peikarski, M.: Listing of All Possible Trees of a Linear Graph, *Ibid*, CT-12, Correspondence, pp. 124–125 (1965)
- [8] Sen Sarma, S., Rakshit, A., Sen, R.K., Choudhury, A.K.: An Efficient Tree Generation Algorithm. *Journal of the Institution of Electronics and Telecommunications Engineers (IETE)* 27(3), 109 (1981)
- [9] Pak, I., Postnikov, A.: *Enumeration of Spanning Trees of Graphs*. Harvard University, Massachusetts Institute of Technology (1994)
- [10] Mayeda, W.: *Graph Theory*. Wiley Inter-science (1972)
- [11] Mayeda, W., Seshu, S.: Generation of Trees without Duplications. *IEEE Trans.* CT-12, 181–185 (1965)