International Journal of Computer Science and Engineering (IJCSE)
Vol.1, Issue 1 Aug 2012 1-14

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

COMPUTING GLOBAL RELIABILITY FOR WIRELESS SENSOR NETWORKS SUBJECT TO SPANNING TREE ENUMERATION APPROACH

[1]MOHD ASHRAF & [2]RAJESH MISHRA

[1]School of Information & Communication Technology, Gautam Buddha University, Greater Noida, India

[2]School of Information & Communication Technology, Gautam Buddha University, Greater Noida, India

ABSTRACT

In recent advances of Wireless Sensor Networks (WSNs) have given rise to many application areas in daily life. Reliability plays a key role in the performance of any large-scale WSNs application. WSNs reliability must consider several design factors, viz. coverage, connectivity, lifetime, etc. However, connectivity remains the most fundamental factor especially in a large scale harsh environment. It is based on finding minimum paths or spanning trees, need a less memory and computational time. In this paper, we explore the problem of enumeration of these minimum paths that can be suitable method further evaluation of reliability for WSNs.

KEYWORDS: Wireless sensor networks, Spanning trees enumeration, Global Reliability Evaluation

INTRODUCTION

As the cost and size of sensor devices are decreasing fast, the application areas of wireless sensor networks have also expanded rapidly. Wireless sensor devices that can be used to actively monitor human activities have garnered great research interest in recent years. Demand of wearable wireless devices has been on the rise recently. The major application domains [1, 2] are home and office, control and automation, logistics and transportation, environmental monitoring, healthcare, security and surveillance, tourism and leisure, education and training and entertainment. Typical possible application scenarios may include digitally equipped homes, manufacturing process monitoring, vehicle tracking and detection, and monitoring inventory control.

But before the WSN revolution can truly take place, it is critical that communication among these smart sensors be reliable and dependable. Any network outage or loss of transmitted data can decrease the users trust on the system. With the growing dependency of the information and communication technology on wireless networks, network reliability becomes one of the important metric for the successful design, planning and deployment of WSN. The degree to which WSN is able to provide the required services needs to be quantitatively assessed by defining proper measurable quantities. These measurable quantities are called the network reliability measures. The typical network reliability problem is to calculate the probability that an all set of nodes or k set of nodes can send or receive data with each other for a given period of time in certain environmental conditions.

Wilkov [3] first suggested the evaluation of overall network reliability by using the concepts of terminal pair reliability, i.e.,by finding all possible paths between each of the n (n-1)/2 node pairs. Fratta and Montanari [4] also proposed an approximate method based on decomposition technique for all-terminal reliability assessment, but these methods are impracticable for real time communication sensor networks. Considering links failure probabilities and using an adjacency/connection matrix, Jain and Gopal [5] proposed a method for evaluation of all-terminal reliability for complex systems. Hardy et. al. [6] suggested a binary decision diagram based approach for reliability evaluation. Using spanning trees and SDP concept with disjoint grouping approach is one of the good ways of evaluating reliability measures can be found in Chaturvedi and Misra [7]. AboElFotoh et. al. [8] performed a factoring algorithm for WSNs based on algebraic connectivity graph. Like other factoring algorithms, it essentially has exponential complexity. Too many redundant computations from isomorphic sub-networks make this algorithm less efficient.

Reference [9, 10] gives different taxonomy to classify sensor networks according to communication functions, data delivery models, and network dynamics. It proposes that communication within WSN can be conceptually classified into two categories: application and infrastructure. Application communication relates to the transfer of sensed data about the phenomenon whereas infrastructure communication relates to the delivery of configuration and maintenance data which includes the several design factors, viz. coverage, connectivity, lifetime, etc. In this paper, we present a connectivity based reliability measures for only static sensor networks i.e. an initial phase of infrastructure communication is needed to set up the network. Furthermore, if the sensors are energy-constrained, there will be additional communication for reconfiguration.

PROBLEM STATEMENT

In this paper, we consider the problem of modeling, enumeration of spanning trees and evaluating the connectivity oriented reliability of WSNs. We define the WSN reliability as the probability that exist an operating communication path between the sink node, and at least one operational sensor in a target cluster. Due to unique features of WSNs, the reliability evaluation of WSN faces a combination of challenges that are not common to the traditional networks. Several factors contribute to the difficulty of WSN reliability measures and evaluation;

- Wireless sensor nodes are generally densely deployed. The complexity of the reliability evaluation increases sharply as the number of nodes increases to the order of hundreds of nodes [11]. Therefore traditional approaches cannot directly apply to WSN.
- The topology of WSN can change due to the duty cycle adjustment of power-constrained sensor nodes. Sensor nodes are scheduled to sleep provided that the remaining set of active nodes can still maintain the required sensing coverage and network connectivity.
- There are different approaches suggested in the literature to solve reliability evaluation problems involving exact analytical calculation, lower and upper bound construction, and simulation.

To perform exact analytical reliability analysis of such WSNs in order to make the problem of computing probabilistic connectedness there are following major assumptions in our analysis:

- The first assumption is related with statistical independence of edge failures. The assumption, edge failures are statistically independent, implies that the probability of a link being operational is not dependent on the states of the other links in the network. The inherent assumption here is that the link failures are caused by random events which affect all links individually.
- The second assumption is that the perfectly reliable nodes, i.e., their probability of failure is almost zero or insignificant. Much of the development in the area of network reliability measurement is presented under the assumption of perfectly reliable nodes.

PROPOSED ALGORITHM

The algorithm has been implemented in the JAVA. For reader's benefit, each step of the algorithm has been illustrated with suitable example. The proposed algorithm has been implemented in two sections. First section of the algorithm defines the initialization part and rest of the section implies the connectedness of the network. Finally the Step# 8 provides all spanning tree for g-reliability evaluation.

Abbreviations And Acronyms

G	reliability graph
g	sub-graph
V	set of vertices
E	set of edge
S	source node
I [ ] [ ]	incidence matrix of reliability graph
A [ ] [ ]	adjacency matrix of sub-graph
STATUS [ ]	vector for storing visited node

Algorithms

Step # 1: [Initialize] Label all the node of communication network from 1 to N and all the link from 1 to M, where N is the number of node and M is the number of link between the nodes.

Step # 2: Represent the network with incidence matrix I.

Step #3: Generate all possible combination of link MCN-1 with N-1 link out of M link.

Step # 4: Repeat step #5 to step# 7 until the link combination list is empty.

Step # 5: Create the sub graph by taking the entire node (N) and add the N-1 link from reading link combination list.

Step # 6: Generate an adjacency matrix A corresponding to sub-graph.

Step# 7: Call Graph_connectivity (g (V, E), s), apply to the sub graph.

a) **for** all node v **do**

/* Check the status of all node in Sub graph*/

if (STATUS[v]=3) **then** Store the sub graph as Spanning tree and Go to **step# 4**.

b) **Else** reject this combination of link (Sub graph). Go to **step# 4.**

c) End of **step # 4** loop

Step# 8: Display all spanning tree generated in step #7 (a).

Step # 9: Exit.

Procedure graph_connectivity (g (V, E), s)

The **graph_connectivity** is a sub-algorithm which is used to check the connectivity of a sub graph. This algorithm visit all the node of sub graph and store the visited status in data structure array, STATUS[]. If the **graph_connectivity ( )** visited all the node of the graph, Graph is connected otherwise graph is disconnected.

The general idea behind this algorithm beginning at a start node A is as follows. First we examine the starting node A. Then we examine each node v along a path P which begins at A; that is, we process a neighbor of A, then a neighbor of a neighbor of A, and so on. After coming to a "dead end" that is, to the end of path P, we backtrack on P until we can continue along another path P' and so on. A field STATUS is used to tell us the current status of a node.

During the execution of our algorithms, each node v of G will be in one of three states, called the status of v, as follows:

STATUS [v_i] ←1: (**Ready state**) The initial state of the node N.

STATUS [v_i] ←2: (**Waiting state**) The node is in stack, waiting to be processed.

STATUS [v_i] ←3: (**Processed state.)** The Node v has been processed.

Step #1: Initialize all the node to ready state [STATUS[v]←1

Step #2: call Push(STACK, s) ;

/* push() used to insert the vertex on the top of the stack where STACK [1,-- n] be an array implementation of stack, s is the starting vertex/*

Set STATUS[s] ←2*;*

Step # 3: While stack is not empty

Step # 4: **call** Pop(STACK, v)

/* Remove the top node of stack an become visited node N */

set STATUS [v] ←3; /* visited node*/

Step #5: for each neighbor of processed node v

a) **if**(STATUS [next node]= 1)
*/*ready State of the node*/*
then call PUSH(STACK, v) ;
/ insert the adjacent node of N to the top of the stack*/*
Set STATUS[v] ←2 ; */*waiting State*/*

b) **If** (STATUS [next node]= 2)
Call Pop(STACK, top) ;
*/*delete the current node from the stack*/*
Set STATUS [v] ←3;
call PUSH(STACK, v) */*Insert the Adjacent node of v which have STATUS[v] ←1 */*

c) **If** STATUS[next node]= 3 processed state, ignore the vertex.

END for

END while

Step # 6: END graph connectivity

Push( ) and Pop( )

Let STACK[1:n] be an array implementation of stack and top be a variable recording the current top of stack position. top is initialized to 0. v is the node to be pushed in to the stack. n is the maximum capacity of the stack.

Procedure PUSH (STACK, v)

Step #1: **if** (*top*= n) then stack is full;

Step#2: **else** { top←*top*+1;

Step#3: STACK[*top*] ←v; } */* store node a top element of stack*/*

Step #4: **END** PUSH

Procedure POP (STACK , item)

Step#1: **if** (*top*==0) then Stack is empty;

Step#2 : **else** { item=STACK[*top*];

Step#3: *top*← *top*-1;

Step#4: **End** POP

FLOW CHART

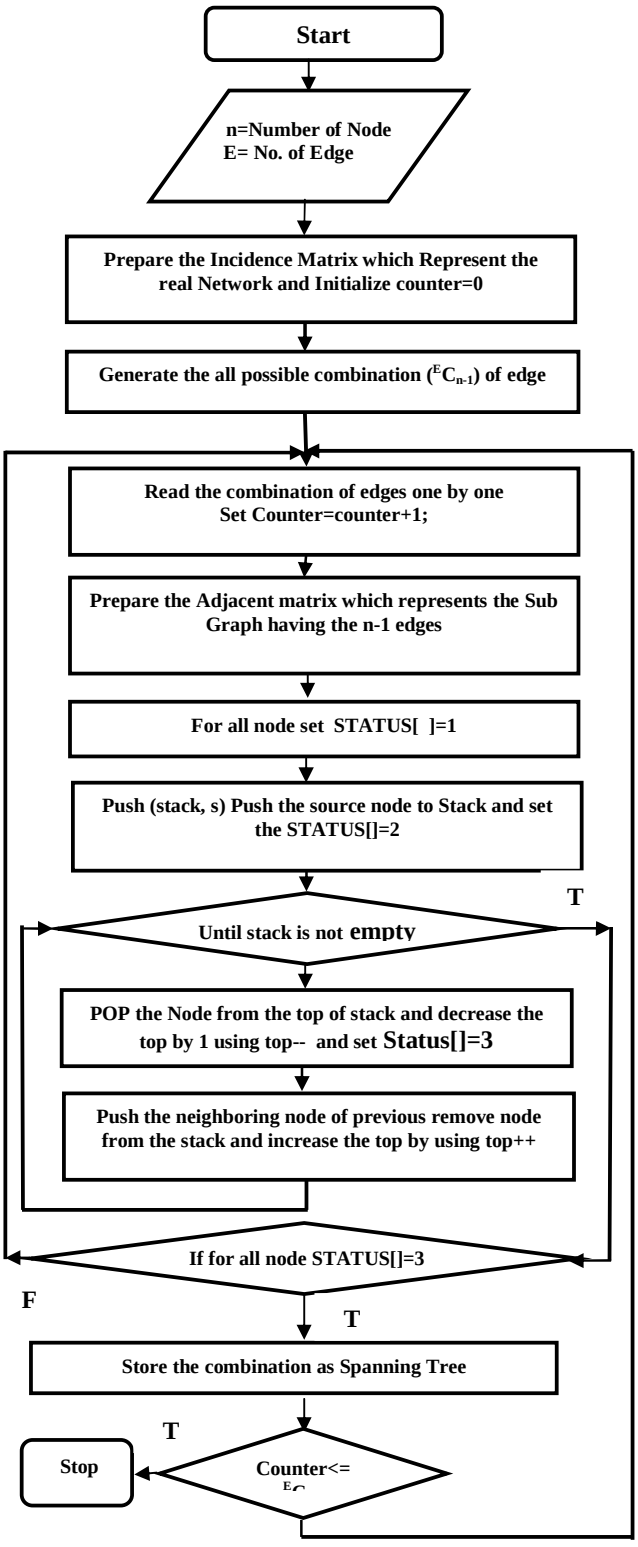

Start
n=Number of Node
E= No. of Edge
Prepare the Incidence Matrix which Represent the real Network and Initialize counter=0
Generate the all possible combination ($^{E}C_{n-1}$) of edge
Read the combination of edges one by one
Set Counter=counter+1;
Prepare the Adjacent matrix which represents the Sub Graph having the n-1 edges
For all node set STATUS[ ]=1
Push (stack, s) Push the source node to Stack and set the STATUS[]=2
T
Until stack is not empty
POP the Node from the top of stack and decrease the top by 1 using top-- and set Status[]=3
Push the neighboring node of previous remove node from the stack and increase the top by using top++
If for all node STATUS[]=3
F
T
Store the combination as Spanning Tree
T
Stop
Counter<=

ILLUSTRATION OF PROPOSED ALGORITHMS

Consider a 6-node, 9-link network with its adjacency matrix shown in Figure 3. The step by step enumeration of spanning trees and their corresponding degree vectors are explain in the following steps:

Step # 1 Label all the node and branches in any arbitrary manner

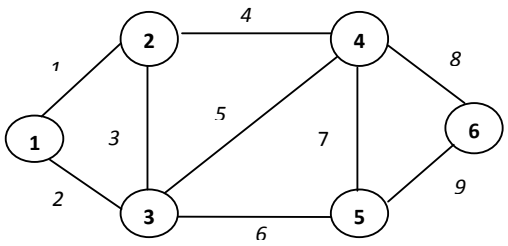


Figure 3: A six node nine link mesh network

Step # 2 Construct the incidence matrix for the graph as shown in Figure 2 and find the minimum degree node.

$$I = \begin{matrix}1\\2\\3\\4\\5\\6\end{matrix}\begin{bmatrix}0&0&0&0&0&0&0&1&1\\1&0&1&1&0&0&0&0&0\\0&1&1&0&1&1&0&0&0\\0&0&0&1&1&0&0&1&1\\0&0&0&0&0&1&1&0&1\\1&1&0&0&0&0&0&0&0\end{bmatrix}$$

Step # 3 Generate the combination of branches (combination having n-1 branches)

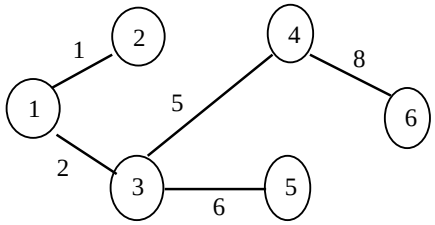
Combination

1. 1 2 3 6 7
2. 1 3 5 6 7
3. 1 2 3 4 7
4. 1 3 4 5 7
5. 3 6 7 8 9

Step# 4 Repeat the Step#5 to Step# 7 until the combination list is empty.

Step# 5 & Step#6 Create the sub graph by reading the combination from the combination list and represent the sub graph by Adjacency Matrix

Example (a) Consider the combination *1, 2, 5, 6, 8* and its corresponding adjacent matrix is given below

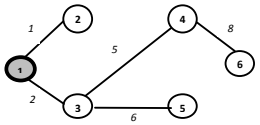


$$A = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

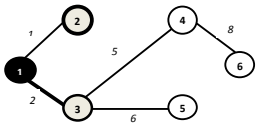
Step#7: Graph_connectivity (g (V, E), s),

a) Initially Stack is empty and i.e initial state of the node is '1' means node are ready to process.
STACK= Φ , top←0, STATUS [ ] = {1, 1, 1, 1, 1, 1}

b) Push the node '1' to the stack and increase the Value of
top by 1. i.e top←top+1 and STATUS [1] ←2
STACK: 1 top←1 STATUS [ ] = {2, 1, 1, 1, 1, 1}

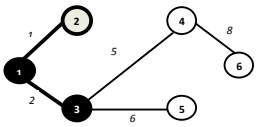


c) POP the node '1' from the stack and decrease the top by 1 i.e. top=0 by top=top-1 and STATUS[1] ←2 and then push on to stack all the neighbors of '1'(those that have Status ←1) and increase the top of the stack
SET the STATUS [2] ←2 and STATUS [3] ←2



STACK: 2, 3 top←2 STATUS [ ] = {3, 2, 2, 1, 1}

d) Pop the top node '3' from the stack and decrease the top by 1 i.e. top←1 by top←top-1 and set STATUS [3]=3 and then push on stack to all the neighbors of '3'(those that have status=1) and change the status of neighbor node of
'3'. The neighboring node of node 3



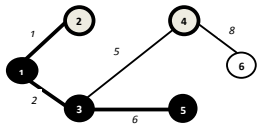
is '4' and '5' and increase the top of the stack by top←top+1

SET STATUS [4] ←2 and STATUS [5] ←2

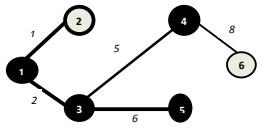
STACK: 2, 4, 5 top←3 STATUS [ ] ← {3, 2, 3, 2, 2, 1}

e) Pop the top node '5' from stack and decrease top by 1 i.e. top←2 by top←top-1 and set STATUS [5] ←3 and then push all neighbor of '5' (those that have status =1) but in this case there is no neighboring node of '5' who have status 1.

Now STACK: 2, 4, top←2 STATUS [ ] = {3,2, 3, 2, 3, 1}



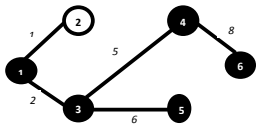
f) Now Pop the top node '4' from the stack and decrease top by 1 i.e. top←1 by top←top-1 and set STATUS [4] ←3.Then push the neighbors node of '4' (those that have status =1). The neighbor node of '4' is '6' and increase the top by 1 i.e. top←top+1 STATUS [6] ← 2



STACK: 2, 6, top←2 STATUS [] = {3, 2, 3, 3, 3, 2}.

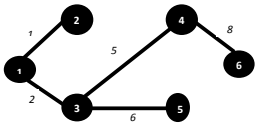
g) Now Pop the top node '6' from the stack and decrease the top by 1 i.e. top=1 by top=top-1 set STATUS[6]=3 of the node from 2 to 3 and push the neighbor node of '6'(those that have status 1) but there is no neighboring node of '6'

Now STACK: 2, top←1 STATUS [ ] = {3, 2, 3, 3, 3, 3}



h) Pop the top node '2' from the stack and decrease the top by 1 i.e. top←0 by top←top-1 Set STATUS [2] ←3 and push the neighbor node of '2' (those that have status 1) but there is no neighboring node of '6' who have status 1.

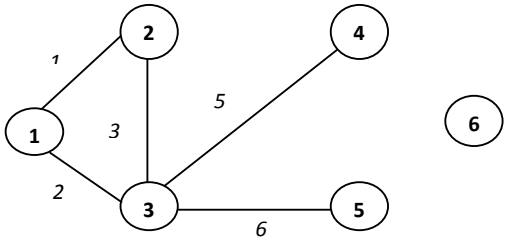
STACK: Φ top←0 STATUS [ ] = {3, 3, 3, 3, 3, 3}



STACK is now empty and every node in STATUS [ ] is 3.

Step 7(a): Store the sub graph as a Spanning tree

Example (b) Consider the combination 1, 2, 3, 5, 6 and its corresponding adjacent matrix is given below:



and its corresponding adjacent matrix

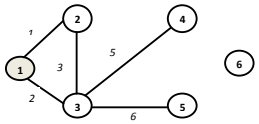
$$\mathbf{A} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step#7: Graph_connectivity (g (V, E), s)

a) Initially Stack is empty and i.e initial state of the node is '1' means node are ready to process.

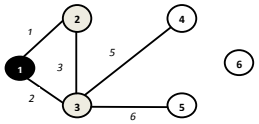
STACK= Φ, top←0, STATUS [ ] = {1, 1, 1, 1, 1, 1}

b) Push the node '1' to the stack and increase the Value of top by 1. i.e top←top+1 and Status[1] ←2



STACK: 1 top←1 STATUS [ ] = {2, 1, 1, 1, 1, 1}

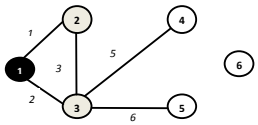
POP the node '1' from the stack and decrease the top by 1 i.e. top=0 by top=top-1 and STATUS[1] ←2 and then push on to stack all



c) the neighbors of '1'(those that have Status ←1) and increase the top of the stack

SET the STATUS [2] ←2 and STATUS [3] ←2

STACK: 2, 3 top←2 STATUS [ ] = {3, 2, 2, 1, 1, 1}

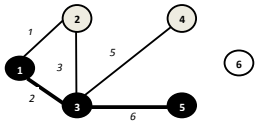


d) Pop the top node '3' from the stack and decrease the top by 1 i.e. top←1 by top←top-1 and set STATUS [3] =3 and then push on stack to all the neighbors of '3' (those that have status=1) and change the status of neighbor node of '3'. The neighboring node of node 3 is '4' and '5' and increase the top of the stack by top←top+1

Set STATUS [4] ←2 and STATUS [5] ←2

STACK: 2, 4, 5 top←3 STATUS [ ] ← {3, 2, 3, 2, 2, 1}

e) Pop the top node '5' from stack and decrease top by 1 i.e. top←2 by top←top-1 and set STATUS [5] ←3 and then push all neighbor of '5' (those that have status =1) but in this case there is no neighboring node of '5' who have status 1.

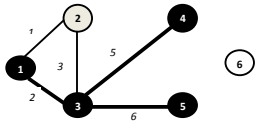


Now STACK : 2, 4 top←2

STATUS [ ] = {3, 2, 3, 2, 3, 1}

f) Now Pop the top node '4' from the stack and decrease top by 1 i.e. top←1 by top←top-1 and set STATUS [4] ←3. Then push the neighbors node of '4'(those that have status =1).

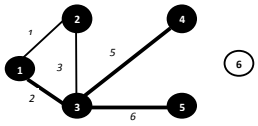
STACK: 2, top←1 STATUS [ ] = {3, 2, 3, 3, 3, 1}



Pop the top node '2' from the stack and decrease the top by 1 i.e. top←0 by top←top-1 Set STATUS [2] ←3 and push the neighbor node of '2' (those that have status 1) but there

g) is no neighboring node of '6' who have status 1.

STACK: Φ top←0 STATUS [ ] = {3, 3, 3, 3, 3, 1}



The STACK is now empty and every node in STATUS [ ]

is not 3

Step#7(b) Reject this sub graph. It is not a spanning tree

EXPERIMENTAL RESULTS

Authors have applied the proposed algorithm to several networks taken from the literature of varied complexities, and verified the reliability obtained by other researchers. Experimental results on several networks taken for this study from the published work. Among them, the results of comparison all spanning trees for few networks (shown in Figure 5.1 – Figure 5.7) [12, 13] are provided in Table 5.1 to show the efficacy of the algorithm and proposed framework to evaluate global reliability using connectivity criterion. Besides, Table 5.1 also provides the enumeration time of each network with reliability.

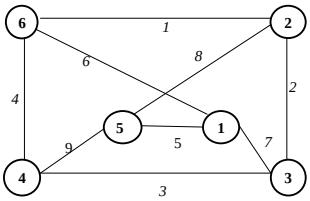


Figure 5.1: 6N9L

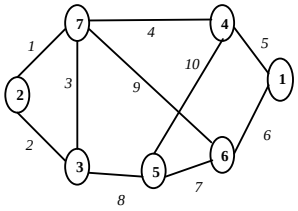


FIGURE 5.2: 7N10L

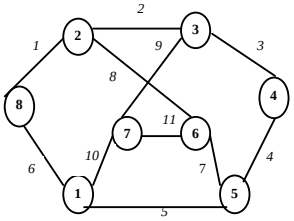


FIGURE 5.3: 8N11L

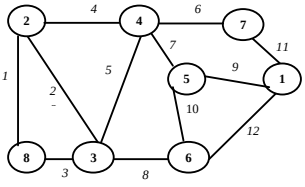


FIGURE 5.4: 8N12L

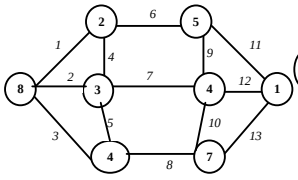


FIGURE 5.5: 8N13L

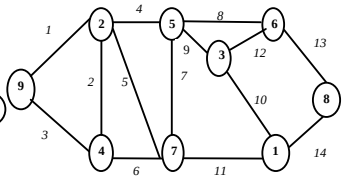


FIGURE 5.6: 9N14L

S.N.	Network	Number of Spanning trees (disjoint terms in reliability expression)	Algorithm[14]	Proposed Algorithm Computation time	all-terminal Reliability for p = 0.9
1	6N9L	81 (81)	$4.8x10^{-3}$ µs	$4.6x10^{-6}$ µs	0.993 263 229
2	7N10L	96 (96)	$6.3\ x10^{-2}$ µs	$6.2\ x10^{-5}$ µs	0.972 218 165
3	8N11L	168(168)	$9.5\ x10^{-2}$ µs	$9.4\ x10^{-5}$ µs	0.969 699 135
4	8N12L	247(247)	$1.1\ x10^{-1}$ µs	$1.07\ x10^{-4}$ µs	0.971 153 157
5	8N13L	576(576)	$1.8x10^{-1}$ µs	$1.72\ x10^{-4}$ µs	0.991 942 810
6	9N14L	647(647)	$5.7x10^{-1}$ µs	$5.0\ x10^{-4}$ µs	0.970 104 348
7	15N25L	111866	0370.93 µs	130.23 µs	0.9965432
8	16N27L	580309	1599.975 µs	789.84 µs	0.9800674
9	17N30L	2215933	11698.5 µs	78902.34 µs	0.9678342

Computation time can be changed a bit depend on the window environment. We have tried ten times for each benchmark and computation time in Table 5.1 is the average of them

CONCLUSIONS

For a full utilization of facilities available in any system, a simple and efficient method for enumeration of all spanning trees has been proposed for evaluating the global reliability of sensor networks. The proposed method is conceptually simple and computationally efficient for all-terminal reliability evaluation of large complex sensor networks and requires less computer memory and computational efforts as compared to other spanning trees based existing methods. Moreover, all methods of reliability computation are known to be computationally intractable or NP-hard, which make it difficult to compare the technique from the aspect of time or memory complexity.

REFERENCES

1. Hamed Y., Mizanian K. and A. M. Jahangir, Modeling and Evaluating the Reliability of Cluster-Based Wireless Sensor Networks, Proceeding of 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 827-834, 2010
2. Yu-Feng X., S. Chen, Xin L. and L. Yu-hong, Elsevier- The Journal of China Universities of post and Telecommunications, Vol. 16(5), pp. 62-70, 2009.
3. R.S. Wilkove, Analysis and design of reliable computer network, IEEE Trans. on Communication. Vol. 20 (3), pp. 660-678, 1972.

4. Fratta and U. G. Montanari, A Vertex Elimination Algorithm for Enumerating All Simple Path Sets in a Graph, Networks, Vol. 20, pp. 151-177, 1975.

5. Jain S. P. and K. Gopal, *An Efficient Algorithm for Computing Global Reliability of a Network*, IEEE Transactions on Reliability, Vol. 37, No. 5, pp. 488-492, 1988.

6. Gary Hardy, Corinne Lucet, and Nikolaos Limnios, K-Terminal Network Reliability Measures with Binary Decision Diagrams, IEEE Trans. Reliability, Vol. 56, no. 3, pp. 506-515, Sept. 2007.

7. Chaturvedi S. K. and K. B. Misra, An Efficient Multi-Variable Inversion Algorithm for Reliability Evaluation of Complex System using Path Sets, International Journal of Reliability, Quality and Safety Engineering, Vol. 9, No. 3, pp. 237-259, 2002.

8. Al-Turjman, Hassanein H. S. and M. A. Ibnkahla, Connectivity optimization for Wireless Sensor Networks Applied to Forest Monitoring, Proceeding of IEEE ICC, 2009.

9. Dohler A., Wireless Sensor Networks: The Biggest Cross Community Design Exercise to Date, Computer Journal of Recent Patents on Computer Science, vol. 1, no. 2, pp. 9-25, 2008.

10. Munir A., Yu B., Ren B. and M. Ma, Fuzzy Logic Based Congestion Estimation for QoS in Wireless Sensor Network, in Proceedings of Wireless Communications and Networking Conference, pp. 4336-4341, 2007.

11. Nitin S. Bhagat, Efficient Spanning Tree Enumeration Using Loop Domain, IEEE Systems Journal, vol. 3, no. 4, pp. 536-545, Dec. 2009

12. .Rath D. and K. P. Soman, A Simple Method for Generating k-Trees of a Network, Microelectronics and Reliability, Vol. 33, No. 9, pp. 1241-1244, 1993.

13. Misra K.B. (Ed.), New Trends in System Reliability Evaluation, Elsevier, Amsterdom, 1993.Mohd Ashraf & Rajesh Mishra " Global Reliability Evaluation of Distributed Communication Networks"17th ISSAT International

14. Conference Reliability & Quality in Design, ISBN 978-o-9763486-7-2, pp. 220-224, August 4-6 2011 , Vancouver, B. C