IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 9, No. 2, June 2020, pp. 175∼182
ISSN: 2252-8938, DOI: 10.11591/ijai.v9i2.pp175-182

Enumeration of the minimal node cutsets based on necessary minimal paths

Yasser Lamalem, Khalid Housni, Samir Mbarki
MISC Laboratory, Department of computer sciences, Faculty of sciences, Ibn Tofail University, Morocco

Article Info

Article history:

Received Dec 21, 2019
Revised Feb 20, 2020
Accepted Apr 4, 2020

Keywords:

Graph
Minimal paths
Network
Node cutsets
Reliability

ABSTRACT

Reliability evaluation is an important research field for a complex network. The most popular methods for such evaluation often use Minimal Cuts (MC) or Minimal paths (MP). Nonetheless, few algorithms address the issue of the enumeration of all minimal cut sets from the source node s to the terminal node t when only the nodes of the network are subject to random failures. This paper presents an effective algorithm which enumerates all minimal node cut-sets of a network. The proposed algorithm runs in two steps : the first one is used to generate a subset of paths, called necessary minimal paths, instead of all minimal paths. Whereas the second step stands to build all minimal cut-sets from the necessary minimal paths.

Corresponding Author:

Yasser Lamalem,
MISC Laboratory, Department of computer sciences,
Faculty of Sciences, Ibn Tofail University,
University Campus, BP 133, Kenitra, Morocco.
Email: yasserlamalem@gmail.com

1. INTRODUCTION

Network reliability can be defined as a probabilistic measure that determines whether a network remains functional, depending on the problem of interest, when its elements fail at random. Since the World War II, the reliability study has become a very active line of research [1–13], especially for critical systems such as telecommunication systems, transportation systems, mechanical systems and oil/gas production systems [5–7, 9, 10, 14, 15]. The objective of reliability studies is the global performance of the system. In literature, many studies have been realized to evaluate the network reliability [1, 3–6, 14, 16–23] and most of them use network-based algorithms founded in terms of minimal cut-sets (MCs) or minimal path-sets (MPs). And to enumerate MCs and MPs, several algorithms have been proposed [1, 4, 5, 14, 16–19, 21, 22, 24–26]. Some of these algorithms can be used only for directed acyclic network like the algorithm proposed by [5]. As introduced in [4] and in [5], it is less expensive to enumerate MCs than to enumerate MPs. The example given in [4] is that of the 2x100 lattice network which has 10000 paths and 299 cuts.

As already introduced, the network reliability can be evaluated using many methods. One of these methods is minimal cut-sets enumeration [1, 4, 16, 19, 21, 22]. In [4], the authors presented a method which can enumerate edge cut set one by one using recursive algorithm. In [21], the enumeration of the MCs is carried out on the basis of a necessary and sufficient condition that a subset of the nodes must verify to be a MC. In [27], the author determines, in the first place, the set of minimal paths from a structure called basic minimal path tree. In a second place, to obtain all minimal vertex cut-sets from MPs, the author uses the logical expressions OR and AND. In [22], the authors present an algorithm based on the reduction of the incidence matrix (result of the failure of a subset of nodes) to determine the subsets of the nodes that cause the loss of connectivity between a pair of nodes.

The methods which consider that the nodes are not reliable and the links are reliable to find all minimal node-cuts can be classified into two classes:

(a) Direct methods, like Patvardhan method [21], Singh B. method [22] and W.C. Yeh method [12]. These methods determine directly from graph the set of minimal node-cuts.

(b) Indirect methods: these methods enumerate, at first, the minimal paths and then extract minimal node-cuts from them in the second time. Some of these methods count only basic minimal paths, like in [27], which contain less paths than all minimal paths.

In this paper we propose a method that finds minimal node cut-sets in two steps : a) The first step is to generate the set of paths called the necessary minimal paths. b) The second step is to extract all minimal cut-sets from the necessary minimal paths using one of the algorithms [20, 27, 28]. The rest of the paper is organized as follows : The notations, functions, and definitions that will be used all along this paper are given in section 2. In section 3, we present the proposed algorithm which makes it possible to enumerate the set of necessary minimal paths. Section 4 contains an illustrative example that demonstrates how the algorithm works. To generate the node cut-sets from the necessary minimal path, an algorithm is given in section 5. Finally, Section 6 gives a comparative study in which the efficiency of our algorithm is compared to three other algorithms [21, 22, 27].

2. NOTIONS AND DEFINITIONS

2.1. Notions and nomenclature

SPath: Shortest minimal path. **Paths**: Set of all minimal paths. **Vin**: Table containing the number of adjacent nodes belonging to *N* for each node. **N**: Set containing the nodes through which the shortest path passes.

2.2. Definitions

Residual graph: A residual graph is the graph obtained after the removal of some nodes.
Path: A path can be defined as a sequence of vertices or a finite sequence of edges connecting a sequence of vertices that are all distinct from each other. In our case, a path will be represented by its sequence of vertices such that for each two successive vertices, there is an arc between them. A path always has a first node, called source node, and a last node, called sink node.
Minimal path: Minimal Path (MP) is a path which contains no other sub-paths between between the source and sink.
Cut: A cut is a set of nodes such that removing these nodes divides the graph into two connected graphs.
Minimal cut: A cut c is a Minimal Cut (MC) if it does not contain other cuts. Necessary minimal paths: Necessary minimal paths is a minimum subset of minimal paths required to generate minimal cuts from them.

2.3. Functions

shortesPath(G: Graph, source: node, terminal: node): Returns the shortest minimal path between the source node and the terminal node. If there is no path between the source and the sink, it returns null. In similar cases, we use dijkstra algorithm, but in this paper we will use the breath first search algorithm because all the edges have the same weight, in other words, because processed graphs are not weighted.
createPath(Paths : Set of paths, Node : node, N : Set of nodes): This function takes as an argument all paths generated by the algorithm (path that are generated in the previous iteration in addition to the current shortest path) and construct other paths using the node *Node* given in the argument. In the first step, the function determines all the adjacent nodes of *Node* and which are in *N*. The second step consists of looking for all the paths containing these nodes. After these two steps, the function replaces each combination of a pair of nodes in the list of adjacent nodes of *Node* by the nodes of the path found in the second step. Example : $Paths = \{\{s,2,t\},\{s,1,3,t\}\}$, $N = \{s,1,2,3,t\}$ and $node = 4$. $Adjacence(4) = \{1,3\} \subset N$, the path $\{s,1,3,t\}$ contain $\{1,3\}$. Therefore, we replace all nodes between 1 and 3 by 4. The new path found is $\{s,1,4,3,t\}$.
removeNodesFromGraph(G: Graph, N: Set of nodes): This function remove all nodes containing in the set *N* from the graph G to obtain the residual graph.
Adjacence(G: Graph, Node : node): This function return all adjacent nodes of the node passed in argument.
push: push is a function that adds an item to a given collection.
pop: This function returns and removes an element from a given collection.

3. ALGORITHM

The algorithm presented in this section enumerates all minimal nodes cut-sets from the list of necessary minimal paths. The proposed approach starts with the enumeration of all necessary minimal paths (algorithm 1). This list of the necessary minimal paths, generated by algorithm 1, is used in the second step to generate all minimal nodes cut-sets.

The key idea of our algorithm is focused on the iterative construction of all the necessary paths. The first instructions of the loop consists of generating the residual graph by eliminating all the nodes in the set *setOfNode* from the current graph (for the first iteration, the *setOfNode* list is empty). In the resulting residual graph, all nodes have the value 0 in the *Vin* table. After this, and at each iteration, the algorithm determines the shortest path in the resulting residual graph and adds it to the set of all paths. later, the algorithm updates the set *N* as follows: for each node *Node1* in *N*, and for each node *x* in *adjacence(Node1)*, *Vin[x]* is incremented by 1. After each increment, if the value of *Vin[x]* become 2 then another path will be generated from the node *x*. This node that has the value 2 in *Vin* will then be added to *N*.

For a given node *x*, the value 1 in the table *Vin* means that there are links between *x* and the nodes in the found paths. Consequently, each pair of nodes whose value in Vin is equal to 1 and which have a path between them will be used to generate other minimal paths. For the first iteration, the set *N* is empty, which implies that in this case the residual graph is the initial graph.

At the end of the algorithm, each pair of *Vin* nodes whose value equal to 1 are used, together with the residual graph and the set *N*, as new parameters to next iteration as shown in algorithm 1.

Algorithm 1 GeneratePaths (Graph, s, t) *// where s is the source node and t is the sink node*

```
local variables:
Voisin, Node, Source, Sink: node;   Vin, N, setOfNodes : table of nodes;   $To\_Treat$: list of node pair
SPath: Path;   Paths : Set of Paths;   residualGraph: Graph
Begin
$Vin[x] \leftarrow 0$ for all x in V
To_Treat ← {[s,t,{∅}]}:
while To_Treat is not empty do
  {[Source, Sink,setOfNodes]} ← pop(To_Trait)
  residualGraph ← RemoveNodesFromGraph(Graph , setOfNodes)
  SPath ← shortesPath(residualGraph, Source, Sink);
  if SPath != null then
    N← SPath;
    Add(Paths,SPath);
    for all Node in N do
      for all Voisin in Adjacence(Node, residualGraph ) do
        if Voisin not in N then
          Vin[Voisin] ← Vin[Voisin] + 1 ;
        end if
        if $Vin[Voisin] \geq 2$ then
          Add(N,Voisin);  CreatePath(Paths,Voisin,N);
        end if
      end for
    end for
    for all Node1 in residualGraph such that Node1 $\notin$ N do
      for all Node2 in residualGraph such that Node2 $\notin$ N do
        if $Vin[Node1] == 1$ and $Vin[Node2] = 1$ and $Prev(Node1) \neq Prev(Node2)$ then
          Push(To_Trait, {[Node1, Node2,setOfNodes∪ N] })
        end if
      end for
    end for
  end if
end while
```

4. ILLUSTRATION OF THE ALGORITHM

For illustrating the algorithm, we consider the graph in Figure 1 (a).

Initialization: Vin={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, To_Trait=[s,t,Ø], Paths ={Ø}

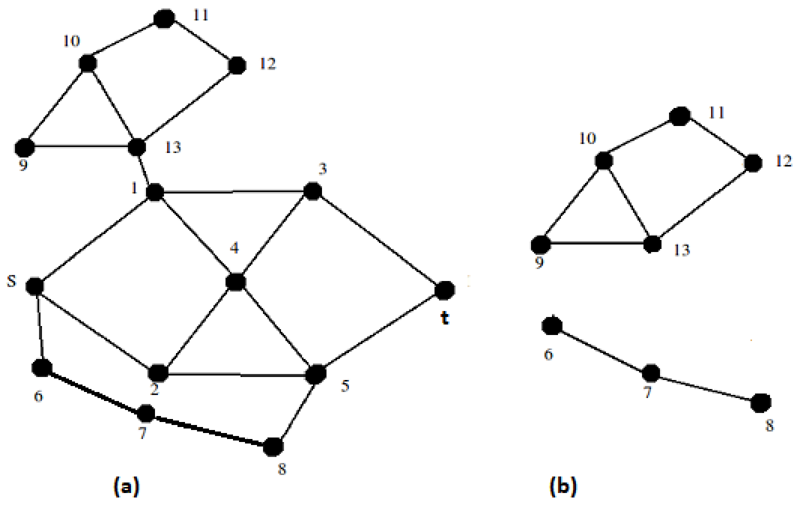

Figure 1. (a) An undirected network example indexed by node. (b) The residual graph obtained after the execution of the first iteration of our algorithm.

First iteration :
Source = s; Sink = t; setOfNodes = $\{\emptyset\}$;
Delete all nodes in setOfNodes from the Graph to get ResidualGraph,
ResidualGraph ← RemoveNodesFromGraph(Graph,N).
SPath = {s,1,3,t}; N = {s,1,3,t};
Paths = {{s,1,3,t}};
For x **in** {s,1,3,t} // increment of vin[j] for all $j \in Adjacent(x)$, in our case :
Case of x=s:
Vin[2] = Vin[2] + 1 = 1;
Vin[6] = Vin[6] + 1 = 1;
Case of x= 1
Vin[4] = Vin[4] + 1 = 1;
Vin[13] = Vin[13] + 1 = 1;
Case of x=3
Vin[4] = Vin[4] + 1 = 2;
Since the node 4 has $Vin[4] >= 2$, we will add 4 to N. N={s,1,3,t,4} and we will call the function *CreatePath*
with the parameters (Paths,4,N). Paths becomes $Paths = \{\{s,1,3,t\},\{s,1,4,3,t\}\}$
Case of x=t
Vin[5] = Vin[5] + 1 = 1;
Case of x=4
Vin[2] = Vin[2] + 1 = 2;
Vin[5] = Vin[5] + 1 = 2;
Since the node 2 has $Vin[2] >= 2$, we will add 2 to N. N={s,1,3,t,4,2} and we will call the function *CreatePath*
with the parameters (Paths,2,N). Paths becomes Paths = {{s,1,3,t},{s,1,4,3,t},{s,2,4,3,t}}
Since the node 5 has $Vin[5] >= 2$, we will add 5 to N. N={s,1,3,t,4,2,5}and we will call the function *CreatePath*
with the parameters (Paths,5,N). Paths becomes Paths={{s,1,3,t},{s,1,4,3,t},{s,2,4,3,t},{s,1,4,5,t},{s,2,4,5,t}}
Case of x=2
Nothing change in Vin table.

Case of x=5
    Vin[8] = Vin[8] + 1 = 1;
Now we have iterated the Set N : To_Trait = {[6, 8, setOfNodes∪ N], [6, 9, setOfNodes∪ N], [8, 9, setOfNodes∪ N]};
The second iteration:
Source = 6; Sink = 8; setOfNodes = {s,1,3,t,4,2,5};
Delete all nodes in setOfNodes from the Graph to get ResidualGraph,
ResidualGraph ← RemoveNodesFromGraph(Graph,N), the residual graph is shown in Figure 2.
Path = {6,7,8}, N = {6,7,8}.
Paths = {{s,1,3,t},{s,1,4,3,t},{s,2,4,3,t},{s,1,4,5,t},{s,2,4,5,t},{s,6,7,8,5,t},{s,6,7,8,5,t}}
For x **in** {6,7,8}
case of x=6
    Nothing change in *Vin* table.
Case of x=7
    Nothing change in *Vin* table.
Case of x=8
    Nothing change in *Vin* table.
The third iteration:
Source = 6; Sink = 9; setOfNodes = {s,1,3,t,4,2,5};
Delete all nodes in setOfNodes from the Graph, ResidualGraph ← RemoveNodesFromGraph(Graph,N),
$Path = \{\}$. //shortest path
There is not path between node 6 and node 9.
The fourth iteration:
Source = 8; Sink = 9; setOfNodes = {s,1,3,t,4,2,5};
Delete all nodes in setOfNodes from the Graph, ResidualGraph ← RemoveNodesFromGraph(Graph,N),
$Path = \{\}$. //shortest path
There is not path between node 6 and node 9.
End of algorithm.

Result : The algorithm has generated six paths {{s,1,3,t},{s,1,4,3,t},{s,2,4,3,t},{s,1,4,5,t},{s,2,4,5,t}, {s,6,7,8,5,t}}. Using those paths we can extract the minimal cuts from them using one of the algorithms [20, 27, 28].

5. EXTRACTING MINIMAL NODE CUT-SETS

Subsequent to the generation all the necessary minimal paths from the graph, the next step is the extraction of the minimal node-cuts from paths. To this end, there are already some algorithms used to solve this problem, such as [20, 27, 28], in this paper we are going to use the Prasad algorithm [27].

5.1. Algorithm

Step 1 : Generation of necessary minimal paths using previous algorithm,
Let C be the set that will include (towards the end) all minimal vertex cutsets,
Put (s) and (t) in C.
Ignore s and t in each necessary minimal path.
Step 2 : Obtain for each necessary minimal path, an expression using the OR logic function on its terms of vertices.
Step 3 : Multiply all the expressions obtained in step 2 using AND logic function.
Step 4 : Store the terms obtained by the previous step in C (minimal vertex cut-sets).

5.2. A numerical illustration of the minimal cut generation algorithm

Step 1 : The necessary minimal paths obtained using previous algorithm :
{{s,1,3,t}, {s,1,4,3,t}, {s,2,4,3,t}, {s,1,4,5,t}, {s,2,4,5,t}, {s,6,7,8,5,t}}
Step 2 : Eliminating *s* and *t*, the logic expressions for the six necessary minimal paths are
(1+3), (1+4+3), (2+4+3), (1+4+5), (2+4+5), (6+7+8,5) respectively .
Step 3 : The resultant product using AND logic expression is (1+3)(1+4+3)(2+4+3)(1+4+5)(2+4+5)

(6+7+8,5).

Step 4 : The simplified expression is ($\underline{126}$ + $\underline{127}$ + $\underline{1\ 2\ 8}$+$\underline{1\ 2\ 5}$+$\underline{1\ 4\ 5}$+$\underline{3\ 5}$+$\underline{3\ 4\ 2\ 6}$+$\underline{3\ 4\ 2\ 7}$+$\underline{3\ 4\ 2\ 8}$) .

Result :

- Minimal cuts of order 1 are : (s) and ((t)).
- Minimal cuts of order 2 are : (3, 5)
- Minimal cuts of order 3 are : (1, 2, 6), (1, 2, 7), (1, 2, 8), (1, 2, 5), (1, 4, 5).
- Minimal cuts of order 4 are : (3, 4, 2, 6), (3, 4, 2, 7), (3, 4, 2, 8).

6. COMPARATIVE STUDY

In this section we present a comparative study between our approach and those published in [21, 22, 27] while using three networks shown in Figure 2, each one of them has it own difference from the others. As shown in Table 1, the algorithm proposed in this paper is faster than [21, 22, 27]. This is due, On the one hand, to the fact that the proposed approach uses only a subset of the minimal paths. On the other hand, to the technique used for the generation of the minimum paths which allows, at each iteration, to exploit the previous information (the list of the paths which are already generated).

The algorithm in the paper [22] generate many combination so the complexity of the algorithm depend on the number of nodes , and in each one of these combinations the algorithm will look if the two conditions are verified to check if that combination is a minimal cut or not. The time that this is going to take for the execution of the two conditions is too long. The second method [27] count only the basic minimal paths that are needed for generating the minimal nodes cutsets. The problem of this method is that sometimes it does not generate all basic minimal paths that are needed for generating minimal nodes cut-sets, instead of that our method generate all necessary minimal paths for generating minimal cuts from them. The third method [21] is an algorithm that generates all the minimal cuts directly instead of being limited to all basic paths. This is based on the recognition of the fundamental structure of the cut-sets. the time complexity to Generate cut-sets is O($(m.n)^2$). The problem of this method is it does not find only minimal cut-sets but all cut-sets.

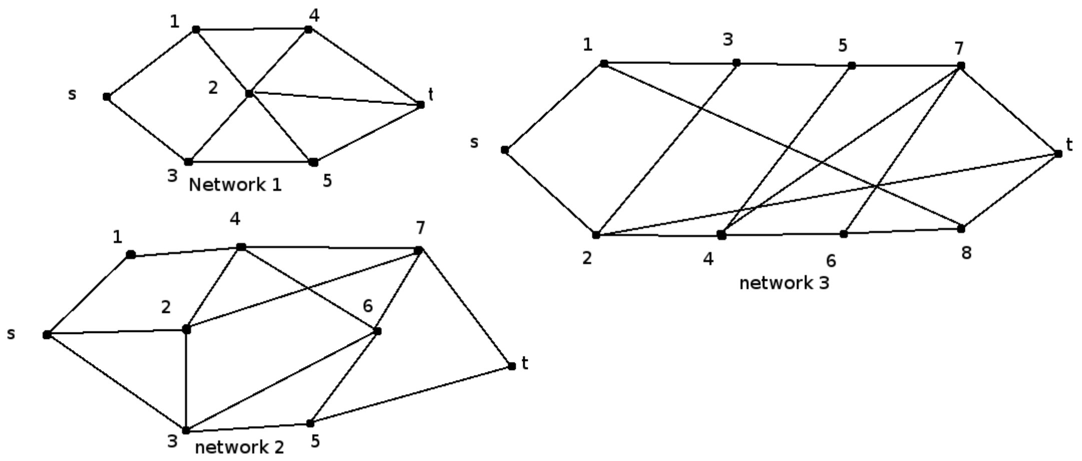

Figure 2. Networks used in comparison tests

Table 1. The result of the comparison

	Network 1	Network 2	Network 3
Method [22]	50 ms	78 ms	87 ms
Method [27]	30 ms	35 ms	44 ms
Method [21]	21 ms	24 ms	32 ms
Our method	17 ms	18 ms	22 ms

7. CONCLUSION

Unlike the existing approaches, this paper presents a new approach to generate all minimal node cut-sets, using a set of paths, called necessary minimal paths, that have generated from the algorithm proposed in the section 3. The algorithm consists of two main steps; the first step is to enumerate the necessary minimal paths from the graph, and the second step is to extract all minimal node cut-sets from the necessary minimal paths. The algorithm is very fast compared to the other existing algorithms in the literature, because it solely enumerates the minimal paths that are necessary for the extraction of minimal cuts. As a perspective to our work, we will develop an approach which will allow us to extract the necessary paths in a single graph traversal.

REFERENCES

[1] S. Soh and S. Rai, "CAREL: Computer Aided reliability evaluator for distribution computing networks", *IEEE Transactions of Parallel Distribution and Systems*, vol. 2(2), pp. 199-213, Apr 1991.
[2] L. He and X. Zhang, "Fuzzy reliability analysis using cellular automata for network systems", *Information Sciences*, Vol. 348(20), pp. 322-336, Jun 2016.
[3] A.C. Nelson, *et al.*, "A computer program for approximating system reliability", *IEEE Trans. Reliability*, vol. R-19(2), pp. 61 – 65, May 1970.
[4] M. Benaddy and M. Wakrim, "Cutset Enumerating and Network Reliability Computing by a new Recursive Algorithm and Inclusion Exclusion Principle". *I. J. of Computer Applications*, vol. 45(16), pp. 22-25. May 2012.
[5] W.C. Yeh, "New Method in Searching for All Minimal Paths for the Directed Acyclic Network Reliability Problem", *IEEE Transactions on Reliability*, vol. 65(3), pp. 1263–1270, Jul 2016.
[6] R. Moghaddass, *et al.*, "Reliability and availability analysis of a repairable k-out-of-n:G system with R repairmen subject to shut-offrules", *IEEE Trans. Reliability*, vol. 60(3), pp. 658–666, Sep 2011.
[7] W.C. Yeh and S.C. Wei, "Economic-based resource allocation for reliable grid-computing service based on grid bank", *Future Gener. Comput. Syst.*, vol. 28(7), pp. 989–1002, Jul 2012.
[8] G. Levitin, "The Universal Generating Function in Reliability Analysis and Optimization". *London, U.K.: Springer-Verlag*, 2005.
[9] D.K. Panda and R.K. Dash, "Reliability Evaluation and Analysis of Mobile Ad Hoc Networks", *International Journal of Electrical and Computer Engineering (IJECE)*, Vol. 7(1), pp. 479-485, Feb 2017.
[10] A. Lekbich, *et al.*, "An analytical multicriteria model based on graph theory for reliability enhancement in distribution electrical networks", *International Journal of Electrical and Computer Engineering (IJECE)*, Vol. 9(6), pp. 4625-4636, Dec 2019.
[11] Y.K. Lin and P. C. Chang, "Maintenance reliability estimation for a cloud computing network with nodes failure", *Expert Systems with Applications*, vol. 38, pp. 14–185, Oct 2011.
[12] W. C. Yeh, "An improved sum-of-disjoint-products technique for the symbolic network reliability analysis with known minimal paths", *Rel Eng and System Safety*, vol. 92(2), pp. 260–268, Feb 2007.
[13] Y.K. Lin, "Two-commodity reliability evaluation of a stochastic-flow network with varying capacity weight in terms of minimal paths", *Comp & Oper Research*, vol. 36(4), pp. 1050-1063, Apr 2009.
[14] W. C. Yeh, "A novel node-based sequential implicit enumeration method for finding all d-MPs in a multistate flow network", *Information Sciences*, vol. 297(10), pp.283–292, Mar 2015.
[15] K. Lin, "A novel algorithm to evaluate the performance of stochastic transportation systems", *Expert Systems with Applications*, vol. 37(2), pp. 968–973, Mar 2010.
[16] W.C Yeh, "A Simple Heuristic Algorithm for Generating All Minimal Paths", *IEEE Transactions on Reliability*, vol. 56(3), pp. 488–494, Sep 2007.
[17] S.G. Chen and Y.K. Lin, "Search for all minimal paths in a general large flow network", *IEEE Transactions on Reliability*, vol. 61(4), pp. 949–956, Dec 2012.
[18] B. Roberts and DP. Kroese, "Estimating the number of s-t paths in a graph", *Journal of Graph Algorithms and Applications*, vol. 11(1), pp. 195–214, 2007.
[19] A.M. Al-Ghanim, "A heuristic technique for generating minimal path and cutsets of a general network", *Computers Industrial Engineering*, vol. 36(1), pp. 45-55, Jan 1999.
[20] K.D. Heidtmann, "Inverting paths and cuts of 2-state systems", *IEEE Transactions on Reliability*, vol. R-32(5), pp. 469-471, Dec 1983.
[21] C. Patvardhan, *et al.*, "Vertex cutsets of undirected graphs", *IEEE Transactions on Reliability*, vol. 44(2),

pp. 347–353, Jun 1995.

[22] B. Singh, "Enumeration of node cutsets for an st network", *Microelectronics and Reliability*, vol. 34(3), pp. 559-56, Mar 1994.

[23] W.P. Dotson and J.0. Gobien, "A new analysis technique for probabilistic graphs", *IEEE Transactions on Circuits and Systems*, vol. 26(10), pp. 855-865, Oct 1979.

[24] G.H. Bai, *et al.*, "An improved algorithm for finding all minimal paths in a network". *Reliability Engineering and System Safety*, vol. 150, pp. 1–10, Jun 2016.

[25] K. Housni, "An Efficient Algorithm for Enumerating all Minimal Paths of a Graph", *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 10(1), pp. 450-460, 2019.

[26] S. Chen, "Search for all minimal paths in a general directed flow network with unreliable nodes", *International Journal of Reliability and Quality Performance*, vol. 2(2), pp. 63–70, 2011.

[27] V.C. Prasad, *et al.*, "Generation of vertex and edge cutsets". *Microelectronics and Reliability*, vol. 32(9), pp.1291-1310, Sep 1992.

[28] M.O. Locks, "Inverting and minimalizing pathsets and cutsets", *IEEE Transactions on Circuits and Systems*, vol. R-27(2), pp.107-109, 1978.