ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse95932020.pdf
https://doi.org/10.30534/ijatcse/2020/95932020

An Improved Algorithm for Network Reliability based on Finding all Node Cutsets

Yasser Lamalem[1], Khalid Housni[2]
[1]Ibn Tofail University, Morocco, yasserlamalem@gmail.com
[2]Ibn Tofail University, Morocco, khalidhousni@uit.ac.ma

ABSTRACT

We can define the network reliability as the ability of a network to carry out a desired operation such as communication. In the literature, several methods have been proposed to evaluate the reliability of networks. We can cite as an example state enumeration methods and cut/path set enumeration methods. The algorithm proposed by BRIJENDRA SINGH in his paper titled "ENUMERATION OF NODE CUTSETS FOR AN s-t NETWORK" belongs to the category of methods which are based on the enumeration of the minimal node cuts set. This paper is made for the purpose to improve the Singh's algorithm. The improvement includes the efficiency of the algorithm and the execution time.

Key words : Reliability, Node cutsets, minimal paths, network, graph.

1. INTRODUCTION

Network reliability is defined as the ability of a network to carry out a desired operation. As defined in [1], the network is said to be operational if the sink is accessible from the source (existence of at least one path connect the source to the sink). Since the World War II, the reliability studies has become very important especially for critical systems such as telecommunication systems, transportation systems, mechanical systems and oil/gas production systems [2]-[6],[8],[9] etc.
Several methods can be used to evaluate the reliability of networks [10]-[28]. Among these methods, there are those based on the enumeration of list of Minimal Cuts (MCS) but many of them assume that the nodes are fully reliable as Benaddy [10], Amjed Al-Ghanim [20] and S. Hasanuddin Ahmad [21]. Other researchers assume that nodes and edges are not reliable as Nasser S. Fard [17] and Yi-Kuei Lin [16]. Few of the methods consider that the nodes are unreliable and the edges are perfectly reliable as Singh B. [13] , Prasad VC, Sankar V, Prasaka Rao KS [11] and Prasad VC[12].

As we have already introduced, network reliability is characterized by success of at least one path between two specified nodes. And for that, it is necessary to take into account all the possible sources of the failures of the systems. Unfortunately, most of the state of the art methods consider that the nodes of the network are perfectly reliable. However, in a practical communication network or computer network, nodes are also subject to failures with certain probabilities. Thus under such circumstance, reliability evaluation that assumes perfect nodes is not realistic.
The main classes of methods to find the minimal node cuts:

- Direct methods [12], [13].
- Methods based on enumeration of all minimal paths [20], [21], [28].
- Methods that count only basic minimal paths which contain less paths than enumerating all minimal paths [11], [27].

The main purpose of this paper is to present and improve the Singh's algorithm to search for all of the minimal node cuts in a graph.
The rest of the paper is organized as follows: in section 2 we present an introduction to the Sing B.'s method with the notations and the definitions that will be used all along this paper. In section 3, the implementation detail of Sing B.'s algorithm to find all minimal cuts and the problems of it will be presented. An improvement of the algorithm and the solutions for its problems are given in section 4. Section 5 contain all the proofs for the improvements we have added and so their effectiveness. Finally, Section 6 contains an illustrative example that demonstrates how the algorithm will works with the new solutions.

2. NOTIONS, DEFINITIONS AND FUNCTIONS

2.1 Definitions

Cut: a set of nodes whose removal divides the graph into two connected graphs.
Minimal cut (MC): a cut c is a minimal if it does not contain other cuts.
Reduced matrix (Ir): it is the matrix obtained after the removal of some lines and columns.

Residual graph: is the graph obtained after the deletion of the nodes.
Order of combination: is the number of nodes in a combination.
Shortest minimal path: is a path from source node s to terminal note t which contains the minimum number of nodes.
Isolated node: is a node which has a row with all 0 values in the incidence matrix.

2.2 Notions

s: the source node.
t : the sink node.
G(V,E) : the graph, where V is the set of all nodes, and E is the set of all links.

2.3 Functions

getAdjacence(Node1,Set1) : return all the adjacency nodes of Node1, where those nodes do not contain the nodes in Set1.

3. SING B. ALGORITHM

This section will present the Singh B. [13] algorithm that finds all the minimal nodes cutsets in a graph, and all the steps of the algorithm.
The algorithm at the first step accept as inputs the incidence matrix of the graph (I), and a set of nodes (Pm) that contain all the nodes in the shortest minimal path, except two, the source and terminal nodes. The algorithm also takes as parameter a set (N) that contain all nodes except the source and terminal nodes.
In the second step, the algorithm initialize a set (S) by an empty set, and then in step 3 store the source node *s* and the terminal node *t* in the set (S) = {(s),(t)}.
In the next step (step 4), the algorithm choose element by element from the set (Pm), and delete the column and the row that correspond to that element from the incidence matrix (I), to get the reduced matrix (Ir). If either the reduced matrix (Ir) is in a block diagonal form or (Ir) has a row with all 0 values, then store that element of (Pm) in the set (S), because it's a minimal cut.
In the step 5, a variable K which contains the order of nodes combinations is initialize by 2, thus a node combinations of that order is generated.
In the step 6, the algorithm take all nodes from the set N, and forms the set (P) of all node combinations of order (K-1). Thereafter the algorithm combines each element in the set (P) with an element of (Pm) to obtain a node combination (Ck) of order K.
The next step (step 7), is to take those combinations stored in (Ck) and delete all elements that have a subset in (S) to form the set (Cr).
Now (Cr) contain all the candidates for the minimal node cuts of order K. The step 8 check if (Cr) is an empty set or not. If it is, go to the step 10, otherwise continue to step 9.
In the step 9, the algorithm take each element of (Cr) and delete the column and the row that correspond to that element from the incidence matrix (I) to get the reduced matrix (Ir), and check if either the reduced matrix (Ir) is in a block diagonal form or (Ir) has a row with all 0 values. If it is, then the algorithm store that element of (Cr) in the set (S).
In the two next steps 10 and 11, the order of the combinations of nodes will be incremented K = K + 1, and check if K ≤(T_n+1), where T_n is the total number of nodes in (N). If it is,go to step 6. Otherwise, the algorithm is over.
It seems that the algorithm works perfectly, but if we look at step 4, it seems that it will not work in all cases. Considering the following example given in Figure 1 and Figure 2, the shortest minimal path in the graph is {s,2,t}, as a consequence (Pm) = {2}, if we take the step 4 and we delete the column and the row that correspond to it from the incidence matrix (I) to get the reduced matrix (Ir), we find that the values in row number 4 are 0. But it's not a minimal node cut, because there is another path that connect the source node *s* and the terminal node *t*. This implies that the algorithm can only work for reduced networks (networks that contain only the nodes which can appear in the set of minimal cuts).

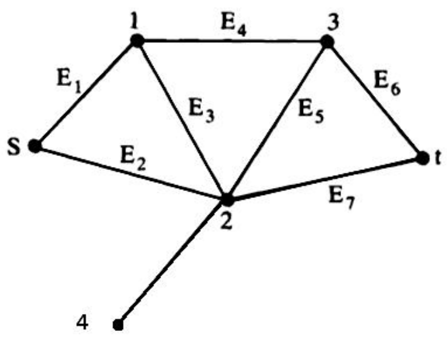

Figure 1: Graph1

$$Ir = \begin{bmatrix} & E_1 & E_4 & E_6 \\ \mathbf{s} & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 0 \\ \mathbf{3} & 0 & 1 & 1 \\ \mathbf{4} & 0 & 0 & 0 \\ \mathbf{t} & 0 & 0 & 1 \end{bmatrix}$$

Figure 2: The reduced matrix (Ir) of the graph 1

This is not the only problem of the algorithm. The step 9 has the same problem. Considering the following example given in Figure 3 and Figure 4. The set (Cr) generated by the algorithm contains (2,4,6) ⊂ (Cr) , now if we delete the column and the row that correspond to those nodes from the

incidence matrix (I) to get the reduced matrix (Ir), we will find that line number 5 has only values 0, but it's not a minimal cut, because there are other paths connecting the source node s to the terminal node t.
After giving some examples where the algorithm cannot correctly generate the set of minimal cuts, we propose in the next section the improvements that can make the algorithm more efficient and faster.

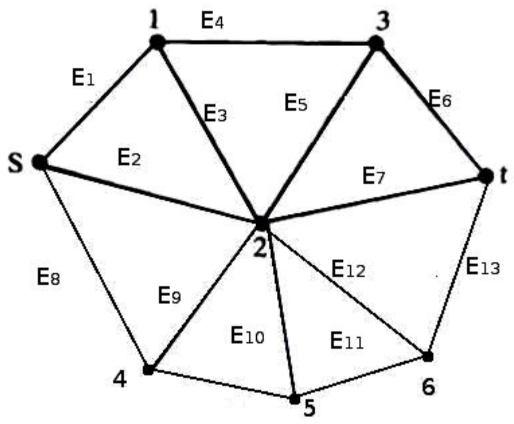

Figure 3: Graph2

$$Ir = \begin{bmatrix} & E_1 & E_4 & E_6 \\ \mathbf{s} & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 0 \\ \mathbf{3} & 0 & 1 & 1 \\ \mathbf{5} & 0 & 0 & 0 \\ \mathbf{t} & 0 & 0 & 1 \end{bmatrix}$$

Figure 4: The reduced matrix (Ir) of the graph 2

4. IMPROVING SING B. ALGORITHM

In this section we present all the proposed improvements of the Singh B. algorithm [13], which includes the complexity, the computational time and also proposing solutions for the failures of the algorithm.
In the rest of this document, we represent a set of nodes by {rep, $node_1$,..., $node_n$}. In this new notation we add a unique number, noted rep, that's will be calculated using a simple algorithm (Left_Shifts) for each set of nodes.

The Left_Shifts algorithm will use the Binary *Left Shift Operator* << (the bits of the left operand are shifted to the left by the number of bits specified by the right operand exp :
1 << 3 → 8). So each time we want to check if a given node, denoted x, belongs to a given set, instead of browsing the whole set we will only calculate the result of the bit-by-bit AND operation (denoted &) between the node x and the number representing the set *rep*. So if the result equal to *x* then *x* belongs to the set. Otherwise, *x* does not belong to the set.

```
Left_Shifts (NodeSet)
Begin
Result = 0;
  For Node IN NodeSet DO
    Result = = Result + (1 << Node);
  EndFor

  RETURN Result;
End
```

For illustrating the algorithm, we consider the node set {2,3,4} as an argument of the algorithm.
Result = 0;
First iteration
For 2 in {2,3,4}
Result = 0 + (1 << 2) = 4.
Second iteration
For 3 in {2,3,4}
Result = 4 + (1<<3) = 12;
Third iteration
For 4 in {2,3,4}
Result = 12 + (1<<4) =28;
The representing number of the node set {2,3,4} is 28. the set {2,3,4} will then be represented by {28, 2, 3, 4}

4.1 First Improvement

For the first improvement of the algorithm, we will propose a solution for the two failures of the the two steps 4 and 9. The case where Ir has a row with all 0 values and the elements of (Pm) does not represent a cut as already illustrated in the previous examples.

The condition that say "if either Ir has a row with all 0 values " means that any node from the reduced matrix that have a row with all 0 value is an isolated node, but that doesn't mean that there is no path between s and t, unless the isolated node is the source node or the terminal node.

4.1.1 The proposed solution

The improvement for the step 4 is:
Step 4 : For each element of (Pm):
- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir
- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values or, Ir is in block diagonal form then store the elements of (Pm) in (S). Otherwise choose the next element from (Pm).

The improvement for the step 9 is:
Step 9 : For each element of (Pm):
- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir.
- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values or, Ir is in block diagonal

form then store the elements of (Cr) in (S). Otherwise choose the next element from (Cr).

With this change of conditions in step 4 and step 9, two improvement have been made, the first is to fix the Singh B. algorithm to work for all cases, the second is to reduce the execution time of the algorithm.

4.1.2 Complexity

The complexity of the previous condition (used in Singh's algorithm) is of the order of *O*(n*n) (n is the number of nodes in the reduced matrix). Because we have to check all the lines. For the new condition the complexity is *O*(n*2), because it is enough to check if at least one of the lines corresponds to the source node *s* and the terminal node *t* has only values 0.

4.2 Second Improvement

Now the algorithm has been improved in the term of checking the first condition, for the step 4 and step 9. But there is still a condition that take too much time to test if the reduced matrix is in a form of block diagonal or not. For obtaining the block diagonal form of a matrix, a permutation of any two rows, or columns is made until the matrix could be written as: *I(G) = [I(g1)/0|0/I(g2)]*, where g1 and g2 are two disconnected sub graphs such as $s \in g1$ and $t \in g2$. As it can be noticed for a small network it can be accepted as a solution, because the first operation is to permute the columns or rows, and after each permutation, a test is made to check if the resulting matrix is written in a block diagonal form or not. in the case where there is a connection between the two sub-graphs, it is necessary to make all the permutation between the rows and the columns in order to conclude that there is a connection between the source and the sink. For a large network this solution is not acceptable because the complexity and the computational time for this operation is too height.

For this issue, this paper propose a new technique to replace the block diagonal form, by a simple algorithm.

4.2.2 The proposed algorithm for testing connection

The main idea of the proposed algorithm is to test if there is a connection, between the source node *s* and the terminal node *t* by starting from the source node *s* to the terminal node *t*.

```
testConnection (Graph,Source,Terminal)
Begin
  Set SourceSet = {s};
  For Node IN SourceSet DO
    NodeAdj = getAdjacence(Node,SourceSet);
    If t ∈NodeAdj Then
      RETURN TRUE;
    ELSE
      Add the NodeAdj set to the set SourceSet.
    EndIF
  EndFor

  RETURN FALSE;
End
```

4.2.2 Illustration for the proposed algorithm

For illustrating the proposed algorithm, consider the network in the Figure 1 :
Initialize the parameters:
N = 6;
SourceSet = {s};
First iteration
For *s* in SourceSet
NodeAdj = {1,2}
t not in NodeAdj ⇒ SourceSet ={s,1,2}
Second iteration
For 1 in SourceSet
NodeAdj = {3}
t not in NodeAdj ⇒ SourceSet = {s,1,2,3}
Third iteration
For 2 in SourceSet
NodeAdj = {4,t}
t not in NodeAdj ⇒ there is a connection between *s* and *t*.

The algorithm found that there is a connection between the source node s and the terminal node *t* in the third iteration, instead of doing the block diagonal technique, that require to do all the permutation between the rows and the columns to know that there is a connection between s and t.

4.2.3 Complexity

The complexity of the algorithm which makes it possible to test whether a matrix is on a diagonal block form or not is of order (n! * n!), Because it is necessary to permute all the columns and rows of the matrix. For the proposed solution, the complexity is *O*(| V | + | E |).

4.2.4 The result of improvement

The result of the improvement for step 4 is:
Step 4 : For each element of (Pm):

- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir
- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values, or there is a no connection between the source and the terminal node using the *testConnection* algorithm, then store the elements of (Pm) in (S). Otherwise choose the next element from (Pm).

The improvement for the step 9 is:
Step 9 : For each element of (Pm):

- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir

- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values, or there is a no connection between the source and the terminal node using the *testConnection* algorithm, then store the elements of (Cr) in (S). Otherwise choose the next element from (Cr).

4.3 Third Improvement

The other thing that slows down Singh's algorithm is the operation of generation of combinations of order k (step 6). This is due to the fact that each time we repeat step 6, it regenerates the order combinations (k -1), which increases the computation time.

The proposed solution consists in using the set (Cr), which contains the minimal cuts obtained from the previous time. For the first iteration (k=2), and since (Cr) is not yet defined, the algorithm uses the classical method to generate the combinations of order (k). But from iteration 2, the proposed solution will use (Cr) to accelerate the algorithm while reducing the time necessary for the construction of the combinations of order (k) (see the proposed solution below).
The improvement for the step 6 is:
Step 6 : Generation of node combinations of order k:
1) If K = 2 then

- Write the set of nodes combination of order (k-1) from the set N to obtain (P).
- Combine each element from (P) with an element of (Pm) to obtain (Ck), and calculate the corresponding unique number for each new generated element.

2) If K != 2 then

- If Cr = ∅ then Stop.
- Combine each element el ∈ (P) with a node of the set (N) where v ∈ (N) and *v* not in el to form (Ck), recalculate the unique number for each generated candidate.

While generating the new candidates for the minimal node cutsets, the algorithm will recalculate the unique number for each set of nodes. Four example considering the set P = {{28, 2,3,4}} and the set N = {5}, the new set of nodes will be {{60, 2,3,4,5}} (28 + (1 << 5) ⇒ 60)

4.3.1 Complexity

For the previous condition (step 6 in Singh's algorithm), the algorithm generate all the combinations of order K-1, the complexity for this operation in each time is *O*(M! / ((M-K+1)!(K-1)!)) (K is the order of combination and M is the number of nodes in the set N). The complexity of the proposed solution, for the first time is M, because it will generate combination of order 2 : *O*(M! / ((M-1)!1!)) = M. In the next, it will use the set (Cr) that contains the MCS generated in the last time step. Therefore the complexity is O(|Cr|*M) in the worst cases.

4.4 Fourth Improvement

The fourth improvement is for the step 7, that delete those elements of (CK) which have a subset in (S) to obtain the set (Cr).
For example : CK={{2,3}}, and S = { {2,4,5}, {2,1,5}, {3,4,5}}. The first step is to compare the set {2,3} with the set {2,4,5}, for this operation we have to check if each node x ∈ {2,3}, x ∈ {2,4,5} (check if 2 ∈ {2,4,5} and if 3 ∈ {2,4,5}). The execution time of this operation is very large. The purpose of this improvement is to present a technique for reducing the execution time of the step 7.

The main idea of the proposed solution is to use the unique number that's been calculated for each set to check if (R) ⊂ (Q) or not (R and Q two sets).

4.2.1 The proposed algorithm

For deleting the MCS candidate from (Ck), which have a subset in (S) to form the set (Cr). The proposed algorithm compare all the elements *EC* ∈ CK with the all elements *ES* ∈ S using the bitwise operation & to tell if *ES* ⊂ *EC*.

```
DeletingSubsets (Ck,S)
Begin
  Cr = Ck;
  For Candidate IN Ck DO
    For Cut IN S DO
      IF{(Candidate & Cut) == Cut}
        Remove the Cut from the set Cr;
        BREAK;
      ENDIF
    EndFor
  EndFor

  RETURN Cr;
End
```

4.2.1 Illustration of the proposed algorithm

For illustrating the proposed algorithm, we consider the set Ck = {{2,3,4,5}[60]} and the set S = {{1,2}[5],{2,3}[12], {4,6}[80]} as an arguments of the algorithm.
Cr = {{2,3,4,5}[60]};
Iteration 1
For {2,3,4,5}[60] in Ck
For} {1,2}[5] in S;
We have (60 & 5) == 4, therefor {1,2} not in {2,3,4,5}.
Iteration 1.1
For {2,3,4,5}[60] in Ck
For {2,3}[12] in S;

We have (60 & 12) == 12, therefor {2,3} ⊂ {2,3,4,5}, we remove {2,3,4,5} from (Cr).

5. THE PROPOSED ALGORITHM

In this section the improved algorithm to search for all minimal node cutsets is presented, with all the steps.

Step 1 : Input the following.

- Incidence matrix (I) of the graph.
- (Pm) → set of nodes in the shortest minimal path except source node (s) and terminal node (t).
- (N) → set of nodes in the graph except (s), (t), and Pm.

Step 2 : Initialize: set of cutsets S = ∅, and the set B = ∅.

Step 3 : Store the source node (s) and terminal node (t) as elements of (S) : (S) = ((s),(t)).

Step 4 : For each element of (Pm):

- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir
- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values, or there is a no connection between the source and the terminal node using the *testConnection* algorithm, then store the elements of (Pm) in (S). Otherwise choose the next element from (Pm).

Step 5 : Initialize order of nodes combination K = 2.

Step 6 : Generation of node combinations of order k:

1) If K = 2 then

- Write the set of nodes combination of order (k-1) from the set N to obtain (P).
- Combine each element from (P) with an element of (Pm) to obtain (Ck), and calculate the corresponding unique number for each new generated element.

2) If K != 2 then

- If Cr = ∅ then Stop.
- Combine each element el ∈ (P) with a node of the set (N) where v ∈ (N) and *v* not in el to form (Ck), recalculate the unique number for each generated candidate.

Step 7 : Delete those elements of (Ck) which have a subset in (S) to obtain the set (Cr) using the *DeletingSubsets* algorithm.

Step 8 : If (Cr) = ∅ then STOP, otherwise to step 9.

Step 9 : For each element of (Pm):

- Delete the column and row that correspond to the element from the incidence matrix to get the reduced matrix Ir : I ⇐ Ir
- If either the source node *s* or the terminal node *t* of Ir has a row with all 0 values, or there is a no connection between the source and the terminal node using the *testConnection* algorithm, then store the elements of (Cr) in (S). Otherwise choose the next element from (Cr).

Step 10 : K ⇐ K + 1

Step 11 : If K ≤ ((Tn+1) where Tn is the total number of nodes in (N) then go to step 6. Otherwise, go to step 12.

Step 12 : STOP.

6. ILLUSTRATION OF THE ALGORITHM

Consider the network shown in Figure 1.

Step 1 : Input the following.

- Incidence matrix (I) of the graph.

$$I = \left[\begin{array}{cccccccc} & E_1 & E_2 & E_3 & E_4 & E_4 & E_6 & E_7 \\ \mathbf{s} & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ \mathbf{2} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \mathbf{3} & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \mathbf{t} & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

- (Pm) = (2) in the shortest minimal path (s-2-t).
- (N) = (1, 3, 4).

Step 2: Initialize set of cutsets : (S) = ∅.

Step 3: Store the source node (s) and terminal node (t) as elements of (S).

Step 4: Repeat for all element (Pm) = 2.

- Choose 2: the result is:

$$Ir = \left[\begin{array}{cccc} & E_1 & E_4 & E_6 \\ \mathbf{s} & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 0 \\ \mathbf{3} & 0 & 1 & 1 \\ \mathbf{4} & 0 & 0 & 0 \\ \mathbf{t} & 0 & 0 & 1 \end{array}\right]$$

- There is no link between source node s and terminal node t, neither the source node s nor the terminal node t of Ir has a row with all 0 values.

Step 5: Initialize order of nodes combination k=2.

Step 6: Generate node combinations of order 2 in the following steps.

- P = ((1), (3), (4)).
- Ck = ((1,2) , (2,3) , (2,4)).

Step 7: Cr = ((1,2) , (2,3) , (2,4)).

Step 8: Since Cr != ∅ go to step 9.

Step 9 : Repeat for all element of Cr = ((1,2) , (2,3) , (2,4))

- Choose (1,2) the result is :

$$Ir = \left[\begin{array}{cc} & E_6 \\ \mathbf{s} & 0 \\ \mathbf{3} & 1 \\ \mathbf{4} & 0 \\ \mathbf{t} & 1 \end{array}\right]$$

- The row corresponding to s has all 0 values. Store the node combination (1, 2) as an element of (S).

- Choose (2,3) the result is :

$$Ir = \begin{bmatrix} & E_6 \\ \mathbf{s} & 1 \\ \mathbf{1} & 1 \\ \mathbf{4} & 0 \\ \mathbf{t} & 0 \end{bmatrix}$$

- The row corresponding to t has all 0 values. Store the node combination (2, 3) as an element of (S).
- Choose (2,4) the result is :

$$Ir = \begin{bmatrix} & E_1 & E_4 & E_6 \\ \mathbf{s} & 1 & 0 & 0 \\ \mathbf{1} & 1 & 1 & 0 \\ \mathbf{4} & 0 & 1 & 1 \\ \mathbf{t} & 0 & 0 & 1 \end{bmatrix}$$

:

- There is no connection between source node s and the sink node t, neither the source node s nor the terminal node t of Ir has a row with all 0 values.

Step 10: K = 3.
Step 11: If 3 ≤ ((T_n+1) T_n = 4, then go to step 6.
Step 6: Generate node combinations of order 2 in the following steps.

- Since Cr != ∅ and k != 2 then P = ((1,2) , (2,3) ,(2,4)).
- Ck = ((1,2,3) , (1,2,4) , (2,3,4)).

Step 7: Cr = ∅.
Step 8: Since Cr = ∅, go to step 10.
Step 10: K = 4.
Step 11: If 4 ≤ ((T_n+1); T_n = 4, then go to step 6.
Step 6: Generate node combinations of order 3 in the following steps.

- P = ((1, 3, 4)).
- Ck = ((1, 2, 3, 4)).

Step 7: Since (Cr) = 0, go to step 10.
Step 10: K = 5.
Step 11: If 5 ≤ ((T_n+1); T_n = 4, then go to step 12.
Step 12: Stop.
Result : (s) , (t) , (1 , 2) , (2 , 3) .

7. BENCHMARK AND TEST

To evaluate the efficiency of the proposed improvements of the algorithm, we carried out an experiments. In these experiments, we have tested on typical grid networks, shown in Figure 5, in each experiment we incremented the number of edges and nodes. The result of this test, as expected, show that our algorithm is more efficient than [13].
As for the implementation language we used JAVA, and for equipment HP core i5 second generation 6 GB Ram.

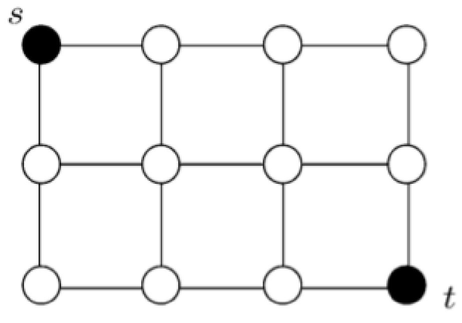

Figure 5: A typical grid network with 12 nodes 17 edges.

Table 1: Comparison result of the second test

Network (N*E)	Nb of MCS	Our Algo	[13]
12*17	22	0.136 s	1.163 s
16*24	65	0.399 s	3.271 s
20*31	141	2.393 s	20.186 s
24*38	191	12.744 s	103.098 s
28*45	265	153.426 s	1352.339 s

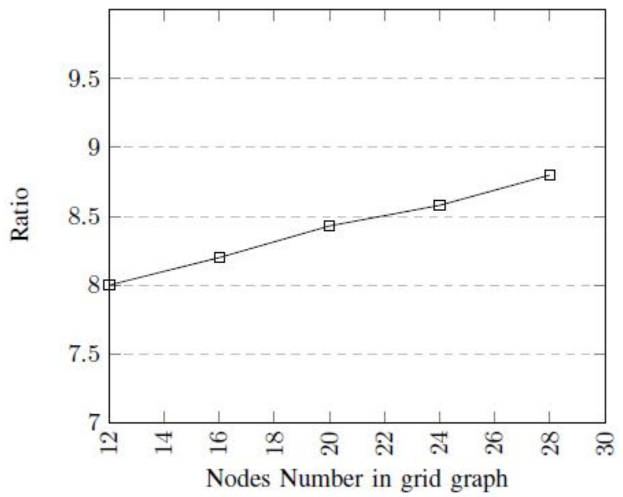

Figure 6: Ratio = CPU time of [13] / CPU time of the proposed improvements.

As a results of experiments (Figure 5, Table 1, Figure 6), it show's that the execution time of the proposed improvements is less than [13]. In addition, the execution time of our improvements gets shorter than [14] as more we add nodes and edges to the grid graph.

8. CONCLUSION

Based on Singh B. method, this paper make many improvements to the Singh B. algorithm, and present them by an illustrated examples. Multiple solutions are proposed by adding and editing conditions in the algorithm, and proposing an method for replacing the block diagonal form that take too much time, which lead us to reduce the execution time of the algorithm as the result of the experiments shows.

REFERENCES

1. Ball, MO, Colbourn, CJ, Provan, JS. Network reliability.Handbooks in operations research and management science, 7; 1995. p.673762. https://doi.org/10.1016/S0927-0507(05)80128-8
2. W.C. Yeh, **New Method in Searching for All Minimal Paths for theDirected Acyclic Network Reliability Problem,** IEEE Transactions onReliability, vol. 65(3), pp. 12631270, Jul 2016.
3. R. Moghaddass, et al., **Reliability and availability analysis of a repairablek-out-of-n:G system with R repairmen subject to shut-offrules**, IEEETrans. Reliability, vol. 60(3), pp. 658666, Sep 2011. https://doi.org/10.1109/TR.2011.2161703
4. 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.
5. 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.
6. 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. https://doi.org/10.11591/ijece.v9i6.pp4625-4636
7. 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.
8. 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.
9. 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. https://doi.org/10.1016/j.eswa.2009.05.080
10. 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.
11. Prasad VC, Sankar V, Prasaka Rao KS. **Generation o vertex and edge cutsets**. Microelectronics and Reliability 1992;32:1291 310
12. Patvardhan C, Prasad VC, Prem Pyara V. **Vertex cutsets of undirected graphs**.IEEE Trans. Reliab. 1995; 44(2).
13. Singh B. **Enumeration of node cutsets for an s–t network, microelectron**. Reliability 1994; 34(3): 559–561.
14. Yoo YB, Deo N. **A comparison of algorithms for terminal pair reliability**. IEEE Trans. Reliab. 1988; 37 (June):393±8. https://doi.org/10.1109/24.3743
15. Netes VA, Filin BP. **Consideration of node failures in network-reliability calculation**. IEEE Transactions on Reliability 1996; 45(1):127–8.
16. Lin Y. **Using minimal cuts to evaluate the system reliability of a stochastic-flow network with failures at nodes and arcs**. Reliab Eng Syst Saf 2002; 75:41–6.
17. Fard NS, Lee TH. **Cutset enumeration of network systems with link and node failures**. Reliab Engng Syst Safety 1999; 65(2):141±6.
18. W.-C. Yeh, **A simple algorithm to search for all MCs in networks,** European Journal of Operational Research, vol. 174, no. 3, pp.1694–1705, 2006/11
19. Jasmon, G. B. O. S. Kai., 1985, **A New Technique in Minimal Path and Cutset Evaluation**, IEEE Transactions on Reliability, Vol. R-34, no. 2, pp. 136-143.
20. Al-Ghanim AM., 1999, **a heuristic technique for generating minimal path and cutsets of a general network,** Computers Industrial Engineering, 36: 45–55.
21. Ahmad SH. **Simple enumeration of minimal cutsets of acyclic directed graph**. IEEE Transactions on Reliability 1988. https://doi.org/10.1109/24.9868
22. J.A. Abraham. **An improved algorithm for network reliability.** IEEE Transaction on Reliability, 28:58–61, 1979.
23. Lin P. M et al., (1976), **A new algorithm for symbolic system reliability analysis**, IEEE Transactions of Reliability, R-25, 1976.
24. A. Dr.A.Rajesh and R.Shankari, **A Novel Opinion Computational model of multi path POR Routing Protocol based on subjective logic in Mobile Ad hoc Networks**, International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8 No. 5, 2019. https://doi.org/10.30534/ijatcse/2019/48852019
25. Ahmad Tayyar, **A shortest path method on a surface in space**, International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8 No. 2, 2019. https://doi.org/10.30534/ijatcse/2019/16822019
26. Marielle Anne Roque and Lawrence Materum, **Interface for the Factor-inclusion Weighting Approach in Determining the Number of Multipath Propagation Clusters**, International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8 No. 4, 2019.
27. Y. Lamalem, K. Housni, S. Mbarki, **Enumeration of the minimal node cutsets based on necessary minimal paths,** IAES International Journal of Artificial Intelligence, Vol 9, No 2, 2020.
28. Y. Lamalem, K. Housni, M. Bennady, **New and Fast Algorithm to Enumerate all Minimal Paths Starting from s and t**, Journal of Advanced Research in Dynamical and Control, 12(04):1101-1113, 2020.