

Determining All Minimal Paths of a Network

¹H. Salehi Fathabadi, ²M. Soltanifar, ³A. Ebrahimnejad, ⁴S.H. Nasser

¹School of Mathematics, Statistics and Computer Science, Tehran University, Tehran, Iran

²Islamic Azad University, Semnan Branch, Semnan, Iran

³Islamic Azad University, Ghaemshahr Branch, Ghaemshahr, Iran

⁴Department of Mathematics, Mazandaran University, Babolsar, Iran.

Abstract: In studying many network flow problems, such as stochastic capacities and evaluating network flow reliability, the minimal paths and cuts are the essential concepts. In this paper we will develop an algorithm to determine the minimal paths in an acyclic network. Then the algorithm is extended to determine the minimal paths in a general network.

Key words: Minimal paths, Minimal cuts, Network flows, Node-child matrix, Reliability.

INTRODUCTION

Many real systems such as electrical and power systems, telecommunication networks, city traffic and computer networks can be studied using network flow theory. For large systems, designing and implementation of a network flow algorithm needs storing the network's properties. Different methods have been stated to do this. In this paper we use the node-child matrix method to suggest an algorithm for calculating all minimal paths of a network flow with no cycles. We will also give a more general algorithm which produces all the minimal paths of a general network.

The minimal paths problem (MPP) is very old problem. There are some published papers related to the MPP with polynomial bounded complexity such as Gauss-Jordan Algorithm (Douglas, 1991).

The usage history of minimal paths and cuts for evaluation of reliability of a stochastic network flow goes back to the 1970. For more details about applications of MPP, readers are referred to Yeh (2005) and (2004). Arunkumar and Lee (1979) stated an implicit method to produce all the minimal cuts in a digraph. Biegel (1977) used the transpose and sum of matrices for acyclic graphs. Rai and Aggarwal (1980) used Boolean algebra to invert and minimize matrix component. Bicker and Abel (1982) developed a method in which a blocking mechanism was used to produce all the cuts in undirected graphs. Yunlong Shen (1995) and Wie-Chang Yeh (2001) gave out new ideas for producing minimal cuts and paths in a graph. The paper by Read and Tarjan (1975) presented an algorithm with complexity that is linear in the number of s-t paths. Most of these methods require complicated mathematical concepts. In contrast, we suggest a simple and basic structure to represent and store a network properties and also new algorithms that work with this structure.

Definitions and Strategy:

Definitions:

In (out) – degree: In a network, $G = (N, A)$ with the node set of N ($|N|=n$) and arc set of A ($|A|=m$), the number of incoming (outgoing) arcs to (from) a node is called the in (out) – degree of that node.

Origin node: A node with zero in-degree. (Node 1 in Figure 1)

Destination node: A node with zero out-degree, (Node 7 in Figures 1) *Child node:* Node i is called a child of node j if (j, i) exist in a network. *Branch node:* A node with more than one child is called a branch node.

Minimal path: A minimal path is a simple path (a directed path without any repetition of nodes) from an origin node to a destination node.

In order to represent a network, $G = (N, A)$, we use the following method: First, nodes are numbered by natural numbers, (as shown in Figure 1).

Corresponding Author: M. Soltanifar, Islamic Azad University, Semnan Branch, Semnan, Iran
E-mail: soltanifar@khayam.ut.ac.ir (Mehdi Soltanifar)
Tel. :+98-9155332852; Fax: +98-231-3354030

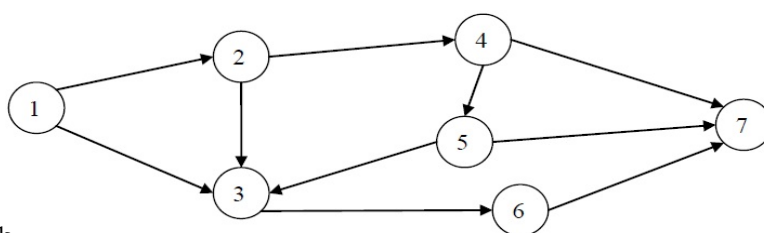


Fig. 1: A Network

Then construct a matrix $B_{n \times p}$ (called node-child matrix) in which p is the maximum out-degree of nodes. The matrix B has one row for each node of the network. Nonzero entries in each row of B are equal to the children's numbers of its corresponding node.

The matrix below is the corresponding node-child matrix of the network shown in Figure 1.

$$B = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \begin{pmatrix} 2 & 3 \\ 3 & 4 \\ 6 & 0 \\ 5 & 7 \\ 3 & 7 \\ 7 & 0 \\ 0 & 0 \end{pmatrix}$$

As it is seen in this structure we only need to introduce all the children of each node. This way of showing the data has many advantages that some of them are mentioned here:

1. Number of columns needed in the node-child matrix is equal to the maximum out-degree of nodes, which is obviously smaller than the number of all nodes (n).
2. The destination node can be easily identified, as it does not have any nodes as a child.
3. This way of representation of the network is very flexible because it's capable of storing bidirectional links and non-planer graphs.
4. There is no limit on the numbers of origin and destination nodes.

The Strategy:

In the upcoming algorithm we will only need the node-child matrix as input. The first algorithm is designed for acyclic network. In the second one this constraint is eliminated using stack data structure. These algorithms work on the node-child matrix B and will use search and swap procedures to produce all the minimal paths. The search method will traverse the available nodes in the current path and will find the next child node using the node-child matrix. If the node, reached at each level, is a destination node the search will stop and the complete path will be stored as a row in the *path matrix*. Otherwise, using the list of branch nodes, another child node of the last branch node (other than the one that already processed) is selected as the current node. The search procedure starts again from the current node.

In the second algorithm we will use a stack data structure. The algorithm is designed in a way that it will recognize the directed cycles and will not store the paths including cycles.

Minimal Paths Algorithm (MPA), Acyclic Network:

Input: S , array of origin nodes; B , the node-child matrix; p , maximum out-degree of nodes, and initialize $j=2, l=1$ (l is the path counter)

Step1: Select an origin node, i , from S , set it as the current node and delete it from S .

Step 2: count the nonzero elements of the i th row of the node-child matrix, say k_i . If $k_i = 0$ then i is a destination node and a path is completed; Store the complete path from origin to destination on the row l of path matrix P ; add one to l , and go to the step 3. Otherwise if $k_i \geq 2$ (if $k_i = 1$ then node ' i ' is not a branch node) then put i on the branch node list, set $B(i, 1)$ as the current node, $i-B(i,1)$ and start from step2.

Step 3: if branch node list is empty you have found all paths started from the ' i '. Proceed to step5, otherwise go to step 4.

Step 4: (*swap process*) take the first element, q , from the branch node list and set $B(q, j)$ as the current node, ($i-B(q,j)$). If $B(q, j+1) = 0$ or $j = p$, then delete q from the branch node list and set $j=2$, otherwise add one to j . go to stage 2. (Note that the branch node list follows FIFO procedure)

Step 5: If there is no more nodes in S, all minimal paths have been found, break out of the algorithm. Otherwise go to step one.

Validity and Time Analysis:

Consider the connected network $G \equiv (N, A)$ which does not have any directed cycles. Assume that this network has only one origin node and at least one destination node. Each time, when a node is chosen as the current node, we might have two different conditions; either the node has a child which the search process will take us one step forward and choose one of the children as the current node (step 2); or the node does not have a child, in which, according to the definition of destination node, we have found a minimal path. Knowing that the number of nodes of $G \equiv (N, A)$ is limited and we do not have any directed cycles; the search process will reach the second condition after a finite number of forward steps. In other words the algorithm is definitely convergent to a minimal path, in each iteration. Now, we show that this algorithm will produce all the minimal paths of $G \equiv (N, A)$. Since all different ways of leaving a branch node is covered by step 4 and all origin nodes in the network are checked in step5, there is no possibility that a minimal path is dropped. Figure 2 shows the algorithm's flowchart.

Theorem 1: The algorithm MPA runs in $O(n^2)$ time.

Proof: First we show that the theorem is true for a network with only one origin node. In the acyclic network, $G \equiv (N, A)$, the search process will reach a destination node after at most n steps. Therefore the order of search process in the worst case is $O(n)$. Also, each node in the worst case will have $n-1$ children; therefore the order of swap process on each node is $O(n)$. This result that the MPA terminates in $O(n^2)$ time. In the case that network has more than one origin node; for each origin node, we operate similar to above process, since the number of origin node is finite, thus the MPA terminates in $O(n^2)$ time.

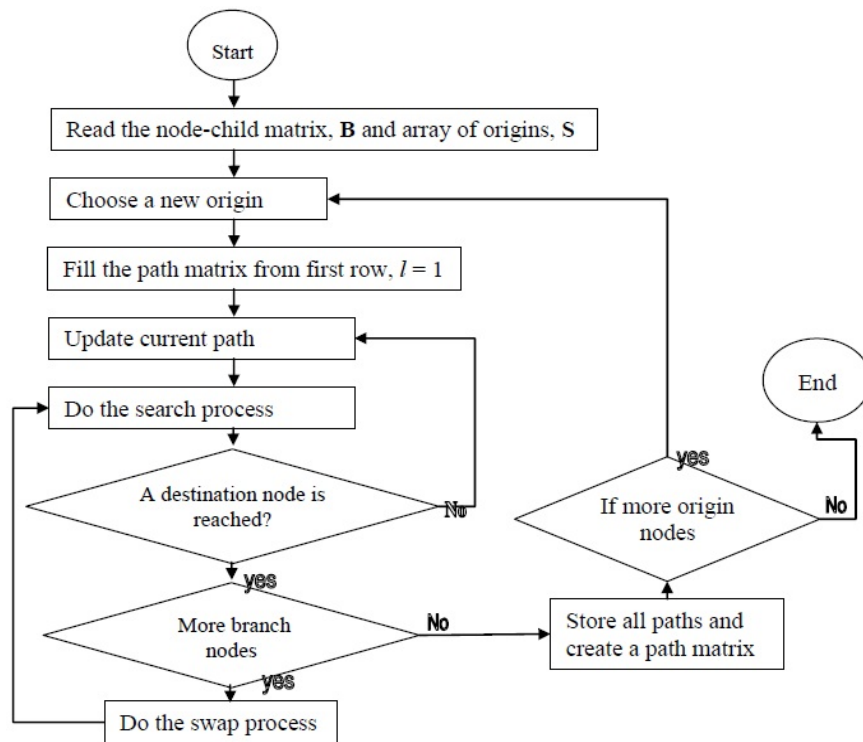


Fig. 2: The Algorithm's Flowchart

Example 1:

In this example we will use the MPA to find all minimal paths of the Figure 1. We also use the node-child matrix defined in Section 2 Initialize ,number of node's children as: $k_1 = 2, k_2 = 2, k_3 = 1, k_4 = 2, k_5 = 2, k_6 = 1, k_7 = 0; l=1, S = \{1\}, j = 2,p=2$. Set the first node as the current node ($i = 1$) since $k_1 = 2$, branch node list will be $Q = \{1\}$. We substitute the current node with the node B ($i, 1$)=2. Continuing of the search

method (repeat of the stage 2) the following path will be found $P_1 = (1, 2, 3, 6, \text{ and } 7)$. Now store the first path on the first row of our path matrix. Now we will set $B(1, 2) = 3$ as the current node. Because our branch list is $Q = \{ 1, 2 \}$ and the first element on the list is node 1, and since the maximum out-degree of nodes of this graph is 2 ($p = 2$), we will delete node 1 from the branch node list and proceed to the stage 2. After completing the algorithm we will have our path matrix as shown bellow. Note that zeros will be placed on the positions left by this algorithm.

$$P = \begin{pmatrix} 1 & 2 & 3 & 6 & 7 & 0 & 0 \\ 1 & 3 & 6 & 7 & 0 & 0 & 0 \\ 1 & 2 & 4 & 5 & 3 & 6 & 7 \\ 1 & 2 & 4 & 7 & 0 & 0 & 0 \\ 1 & 2 & 4 & 5 & 7 & 0 & 0 \end{pmatrix}$$

MPA for General Network:

The algorithm discussed on the section 3 was designed for network flows which do not have directed cycles. Now in this section we will modify the algorithm to determine minimal paths in a network flow with no constraints (i.e. acyclic or not acyclic). This algorithm also will receive an node-child matrix as its input data. A stack is used in the algorithm. We also have a copy of other elements of the stack as a list from which deleting the top element is possible.

This algorithm uses a search procedure that returns the corresponding path matrix. A procedure called *find (stack x, int y)* that takes a stack and an element and returns 1 if the element is found on the stack and 0 otherwise. One of the privileges of this algorithm is the ease of its implementation. Therefore we will use pseudo code to present this algorithm.

Algorithm MPA

```

Begin
  B = the node-child matrix with n rows and p columns;
  S = array of origins;
  k=1;
  Q = the empty stack;
  While (There is an origin node, s) do
  Begin
  Delete all element of Q;
  Set s in Q;
  Search(s);
  End /* end of while */
  End /* end of algorithm */

```

Sub program search(s)

```

Begin
  For (j=1; j~p; j++)
  Begin
  If (B (s,j) =0) then
  Begin
  If (head of Q is a destination node) then Begin
  Pk= node in Q;
  k++;
  (1) Delete from Q;
  (2) s = node of Q;
  (3) Return;
  End /* end of if */ Else
  Begin
  (4) Delete from Q;
  (5) s = top of Q;
  (6) Return;
  End /* end of else */ End /* end of if */
  End

```

```

Else if (find (Q, B(s,j) ) then Continue;
Else
Begin
Set B(s,j) in Q;
s = B(s,j);
Search(s);
End /* end of else */
End /* end of for */
End /* end of sub program */
Sub program find (Q,s)
Begin
If ( s is an element of Q )
Return 1; Else
Return 0;
End /* end of sub program */

```

Algorithm's validity:

Consider the connected network $G \equiv (N, A)$. We assume this network has one origin node and at least one destination node. Each time a node is chosen as the current node, two possible cases may occur: Either this node has a child ($B(s, j) \neq 0$) or it doesn't.

In the first case we will choose a child of the current node and check whether it is in the current path or not. If it is not on the current path ($\text{find}(Q, B(s, j))$ have a false value) we extend the part and choose the child as the current node. Otherwise ($\text{find}(Q, B(s, j))$ have a true value) we will choose another child of the current node (by adding one to j). Now if there are no other children we will use a procedure similar to the swap in the M PA (instruction 4, 5 and 6). In the second case, according to the definition of destination node we have found a minimal path. Since the number of nodes is finite, the search process will reach the second case after a finite number of steps, Therefore the output of the algorithm is certainly all minimal paths. Remember that we will go one step back wards every time we reach a destination node in each path and check every other possible way to reach the destination node (with the instruction of 1, 2, and 3). Therefore the algorithm will find all the minimal paths of the network $G \equiv (N, A)$ starting from the first origin node. Now if we have more than one origin, minimal paths will be found in a similar way for each origin node. Hence the out put of the algorithm is the complete list of minimal paths in the network $G \equiv (N, A)$.

Example 2:

Now we apply this algorithm to find all the minimal paths of the graph shown in Figure 3. The Corresponding node-child matrix is as shown below.

$$\mathbf{B} = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \begin{pmatrix} 2 & 3 & 0 \\ 3 & 4 & 0 \\ 6 & 0 & 0 \\ 5 & 7 & 0 \\ 2 & 3 & 7 \\ 7 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

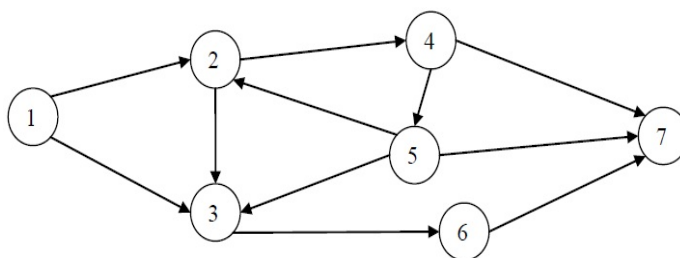


Fig. 3: A network for Example 2

We set $k=1$ and choose the origin node $s=1$ for the start. Now we push s into the stack Q therefore we have $Q = \{ | 1 \}$. (The symbol $\{ | - \}$ is used to show bottom of the stack). Now we will call the search (s) method. Since $j=1$, $B(s, j) = 2$ and we do not have node 2 in the stack we will push 2 into the stack $Q = \{ | 1, 2 \}$ and set $S=2$ and call the search(S) method. Again because $j=1$, $B(s, j) = 3$ and node 3 is not on the stack we will push 3 into stack ($Q = \{ | 1, 2, 3 \}$) and set $s=3$ and call search(S). This time also $j=1$, $B(s, j) = 6$ and node 6 is not in the stack, so we push 6 into the stack ($Q = \{ | 1, 2, 3, 6 \}$), set $S = 6$ and call search(s). Once again $j=1$, $B(s, j) = 7$ and node 7 is not in the stack ($Q = \{ | 1, 2, 3, 6, 7 \}$), now set $s = 7$ and call search(s). This time because $j=1$ and $B(s, j) = 0$ and top element of the stack is out destination node therefore, $P1 = (1, 2, 3, 6, 7)$ and k is set to 2. We also pop an element from stack ($Q = \{ | 1, 2, 3, 6 \}$). Now we will set S to the top element of the stack ($s = 6$) and go one step back. This time around $j=2$, $B(s, j) = 0$ but since the top element of the stack is not a destination node therefore it will be popped from the stack ($Q = \{ | 1, 2, 3 \}$). Now $s=3$ and we go a step back. We have $j=2$ and $B(s, j) = 0$ and in a similar order that we had before $Q = \{ | 1, 2, \}$ and $s=2$. This time $j=2$ and $B(s, j) = 4$ and since 4 is not in the stack, $Q = \{ | 1, 2, 4 \}$ and $s=4$, call search(s). $j=1$, $B(s, j) = 5$, 5 is not in the stack therefore $Q = \{ | 1, 2, 4, 5 \}$ and we'll set $s=5$ and call search(s). This time because $j=1$, $B(s, j) = 2$ and 2 is in the stack we will continue with out any commands. Now we have $s=5$, $j=2$ $B(s, j) = 3$ and 3 is not in the Q so $s=3$, $Q = \{ | 1, 2, 4, 5, 3 \}$ and we call search(S). Now $j=1$, $B(s, j) = 6$ and 6 is not in the stack therefore 6 is pushed into stack and s is set to 6 and search(s) is called. Now, also $j=1$, $B(s, j) = 7$ and 7 is pushed into the stack since it's not already there. ($Q = \{ | 1, 2, 4, 5, 6, 7 \}$) and set $s = 7$. This time search will find path $P2 = (1, 2, 4, 5, 3, 6, 7)$, and will pop an element form stack and will be called again with $s=6$ and stack looking like $Q = \{ | 1, 2, 4, 5, 6 \}$. After finishing the algorithm we will have the path matrix as shown bellow (note that zeroes will be placed on the empty spaces of the matrix).

Conclusion:

For evaluation reliability of a stochastic network flow, we need all minimal paths of that network. In this paper we presented MPA algorithm to generate all minimal path of an acyclic network and then extended it to the general case. Therefore, MPA can be used to evaluation reliability of a stochastic network flow. The advantage of MPA is that there is no limit on the number of origin and destination nodes and it can easily accommodate non-planar graph. However MPA only need basic knowledge about network flow structures.

ACKNOWLEDGMENTS

The fourth author thanks to the Research Center of Algebraic Hyperstructures and Fuzzy Mathematics, Baboslar, Iran and National Elite Foundation, Tehran, Iran for their supports.

REFERENCES

- Ahuja, k., T.L. Magnanti, J. B. Orlin, 1993. Network Flows Theory, Algorithm and Applications. Prentice Hall, Englewood Cliffs, Nj.
- Abel, U., R. Bicker, 1982. Determination of all minimal cut-sets between a vertex pair in a undirected graph. IEEE Trans. Reliab., R-31:167-171.
- Al-Ghanim, A., 1999. A heuristic technique for generating minimal path and cut sets of a general network. Computers and Industrial Engineering., 36: 45-55.
- Arun Kumar, S., S.H. Lee, 1979. Enumeration of all minimal cut sets for a node pair in a graph. IEEE Trans. Reliab., R-28: 51-55.
- Biegel, J.K., 1977. Determination of tie sets and cut sets for a system without feedback. IEEE Trans. Reliab., R - 26: 39-41.
- Douglas, R.S., 1991. Network Reliability and Algebraic Structures. Oxford Science Publications.
- Rai, S., K.K. Aggarwal, 1980. On complementation of path sets and cut sets. IEEE Trans. Reliab., R-29, 139-140.
- Read, R.C., R.E. Tarjan, 1975. Bounds on backtrack algorithms for listing Cycles, Paths, and spanning trees. Networks, 5: 237-255.
- Shen, Y., 1995. A new simple algorithm for enumerating all minimal paths and cuts of a graph. Microelectron.Reliab., 35(6): 973-976.
- Yeh, W.C., 2001. A simple algorithm to search for all d-MP_s with unreliable nodes. Reliability Engineering and System Safety., 73: 49-54.

Yeh, W.C.. 2005. A new approach to evaluate reliability of multistate networks under the cost constraint. *Omega.*, 33: 203-209.

Yeh, W.C., 2004. Multistate network reliability evaluation under the maintenance cost constraint. *Int. J. Production Economics*, 88: 73-83.