Get Network's Disjoint MPs Based on Discrete Events Simulation

Jian Tang[1,a], Tao Han[2,b*], Jianhu Yuan[3,c], Faming Shao[4,d]

[1]College of Field Engineering, PLA University of Science and Technology, Nanjing 210007, China

[a]lgdx_tj@163.com

[b]563524748@qq.com

[c]yjhnj1@163.com

[d]shaofaming@163.com

Keywords: Network's reliability; Disjoint Minimal Path sets; Discrete event simulation; SimEvents

Abstract. From the point of message transmission and processing, this paper puts forward a new disjoint Minimal Path sets (MPs) algorithm to get the disjoint MPs of networks and trys to realize it by means of Discrete Events Simulation (DES). In the new method, the arcs of CoA network are regarded as processing units, nodes as storage units, and network as message transmission network. Taking SimEvents® as the platform, the modeling idea of nodes, arcs and message transmission network are described in detail. An example verifies the correctness of the processing rules and the feasibility of using DES to generate the disjoint MPs automatically.

Introduction

The Minimal Path sets (MPs) methods is an important kind of methods used to compute network's reliability[1-4]. Traditionally, the first step of MPs methods is to obtain the network's MPs and then use total probability formula to calculate the reliability[5]. However, network's MPs are always intersected with each other，if using total probability formula[6] to calculate the reliability based on MPs, the equation will contain a large number of redundant items. So the second step of MPs methods is always to use Inclusion-exclusion principle[7][8], Sum of Disjoint Products (SDP)[9-12], Binary Decision Diagram (BDD)[13-15] and other disjoint algorithm to get the disjoint MPs. Obviously, the two-step methods are cumbersome too and even prone to occur combinational explosion[16]. So some scholars dedicated themselves to find out a way to get disjoint MPs without resorting to MPs[17-22]. In literature [23], Wu put forward an algorithm to directly get all the disjoint MPs by messages' continual transmission according to a set of rules. Wu's algorithm has a characteristic of distributed storage and processing, which inspires us to realize it by means of Discrete Event Simulation (DES). So, this paper will propose a new version of Wu's from the point of DES; and then try to realize it in SimEvents®, a discrete event simulation platform.

Notations

$G(V,A)$ is a acyclic network, $V=\{i,\quad i=1,2,...,n\}$ is the node set; $A=\{a_k,\quad k=1,\dots,m\}$ is the arc set; n and m are respectively the number of nodes and arcs. Among the nodes, 1 is the source node, n is the terminal node, the others are middle nodes. For any arc, there is a mapping $y: A\rightarrow V\times V$, $y(a_k)=(i,j)$, wherein, i is the starting node of a_k, j is the end node of a_k.

m_r is a message composed of the index of arcs, which could be a_k or $\bar{a}_k$ ($k=1,2,\ldots,m$); m_0 is an empty message; M_i is the set of messages stored in node i, $M_i=\{m_r, r=1,\ldots,n_i\}$; IV_i is the set of leading nodes of node i ($i\in V$, $i\neq 1$); IA_i is the set of leading arcs of node i ($i\in V$, $i\neq 1$). IA_i^* is set of node i's valid leading arcs for m_r; OV_i is set of following nodes of node i ($i\in V$, $i\neq n$); OA_i is the set of following arcs of node i ($i\in V$, $i\neq n$), $OA_i=\{a_k | y(a_k)=(i,j), j\in OV_i\}$; OA_i^* is the set of node i's valid following arcs for m_r. In **Error! Reference source not found.**, the standard of how to judge IA_i^* and OA_i^* will be presented in detail.

Disjoint MPs algorithm based on message transmission and processing

In Fig. ,there is a network called Network 1, in which, 1 is the initial node, 3 is the terminal node and 2 is the middle node. The disjoint MPs algorithm is shown in Eq. 1[23].

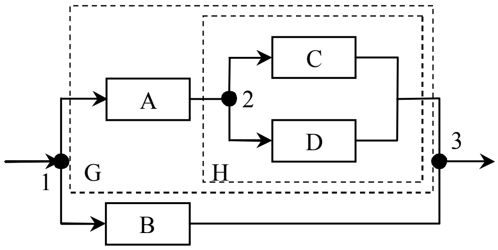

Fig. 1 Network 1

$$\begin{aligned} S &= G+B = G+\bar{G}B = A(C+D)+\overline{A(C+D)}B \\ &= A(C+\bar{C}D)+(\bar{A}+A\overline{(C+D)})B = AC+A\bar{C}D+\bar{A}B+A\bar{C}\bar{D}B \end{aligned} \quad (1)$$

We can let an empty message depart from the network's node 1, and go to node 3 along path 1-2-3. When the message goes through A to node 2, it will be rewritten as A and stored in node 2. Then, it will continue to be transmitted to node 3 and rewritten as AH, i.e. $A(C+D)$, and stored in node 3. In the reverse direction, the message A stored in node 2 will be transmitted along the same path back to its leading node, i.e. 1, and rewritten as $\bar{A}$. Then $\bar{A}$ will be transmitted along another path, i.e. 1-3 to node 3 and rewritten as $\bar{A}B$. As for $C+D$, similarly, an empty message will depart from 2, and then be transmitted to 3 through arc C and rewritten as C. Then C will be transmitted reversely to 2 along the original path and rewritten as . Then $\bar{C}$ will be transmitted to 3 along the other path parallel to arc C, i.e. arc D, and rewritten as $\bar{C}D$.

Obviously, if the transmission and processing rules are formulated correctly, the network's disjoint MPs could be generated automatically during the message transmission in the network.

The message will be transmitted and processed by the following steps:

Step 1. Initialize the message set M_i for each node.

Let $M_1=\{m_0\}$, m_0 is an empty message; $\forall i\in V$ and $i\neq 1$, let $M_i=\varnothing$.

Step 2. $\forall i\in V$ and $i\neq n$, transmit and process each message in M_i forward and backward in accordance with the following rules:

(2.1) Transmit m_r forward.

(2.1.1) Determine OA_i^* for m_r's forward transmission. The rule is: $\forall a_k\in OA_i$, if m_r contains neither a_k nor $\bar{a}_k$, let $a_k\in OA_i^*$; otherwise, $a_k\notin OA_i^*$.

(2.2.2) In a certain order, label the arcs in OA_i^* as a_{k_1}, a_{k_2}, …, temporarily.

(2.2.3) Transmit m_r to a_k's end node j in the aforementioned order and rewrite m_r in obedience to the following rules: Firstly, transmit m_r to a_{k1}, rewrite it as $m_r a_{k1}$, and store it to a_{k1}'s end node j_1; secondly, transmit m_r to a_{k2}, rewrite it as $m_r \bar{a}_{k1} a_{k2}$, and store the new m_r to a_{k2}'s end node j_2; similarly, transmit m_r to the third, fourth, … arcs in OA_i^*, rewrite it as $m_r \bar{a}_{k1} \bar{a}_{k2} a_{k3}$, $m_r \bar{a}_{k1} \bar{a}_{k2} \bar{a}_{k3} a_{k4}$ …, and store the new m_r respectively to node j_3, j_4, ….

(2.2) Transmit m_r backward.

(2.2.1) Determine IA_i^* for m_r's backward transmission. The rule is: $\forall b_k \in IA_i$, if and only if m_r contains b_k, let $b_k \in IA_i^*$; otherwise, $b_k \notin IA_i^*$.

(2.2.2) Temporarily, label the arcs in OA_i as a_{k1}, a_{k2}, ….

(2.2.2) Transmit m_r to each arc in IA_i^* one by one in the reverse direction, and the rule is: Firstly, scan the first arc in OA_i, judge whether m_r contains a_{k1} or $\bar{a}_{k1}$. If the answer is 'No', rewrite m_r as $m_r \bar{a}_{k1}$, if 'Yes', give up rewriting and keep m_r unchanged; then, turn to the second arc a_{k2}, and the rewriting rule is the same as a_{k1}; finally, after all the arcs in OA_i have been scanned, store the new m_r in the leading node of b_k.

(2.3) Delete m_r from M_i.

Step 3: Repeat step 2 until $\forall i \in V$ and $i \neq n$, $M_i = \varnothing$.

Realization by means of Discrete Event Simulation in SimEvetns

Discrete Events Simulation will be tried to realize the new algorithm. The basic idea is to regard:

(1) network as a discrete event system;

(2) arc as processing unit, whose function is to transmit and rewrite messages;

(3) node as storage unit used to temporarily store the messages arriving to it;

(4) entity as message carrier, who can carry message moving in the network;

(5) message as an attribute of entity. It can be rewritten during the simulation.

In SimEvents, entities can pass through all blocks during a simulation, and a block can carry out operations on entities. The data carried by entity are called attribute. In the following depiction, block and port will be denoted by **Bold** and attribute will be denoted by *Italics*.

Take the network in Fig. as an example, we'll respectively set models for three kinds of node, the forward-transmission model of OA_i and the backward-transmission model of IA_i.

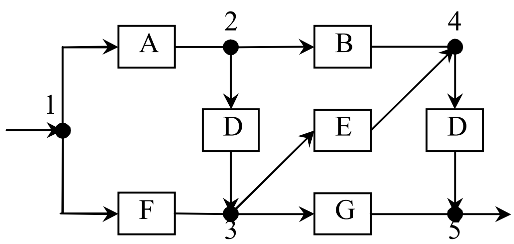

Fig. 2 Network 2

Node model

The main function of nodes is to store the messages transmitted into it temporarily. During the simulation, message will be treated as an attribute of entity, whose attribute name is *Message*.

1. Middle node

Taking node 2 in network 2 as an example, the composition of middle node is shown in Fig. . On the arrival of each entity, **Get Attribute** will get the value of *Message*, and notify **Events-Based Entity Generator** to generate a new entity. Then **Set Attribute 1** will assign the value of *Message* to the new entity and put it into **FIFO Queue** to wait for further processing. Obviously, **FIFO Queue** acts as a temporary storage unit. The old entity will be discarded through **Entity Sink**. Before the new entity goes into the node's following/leading arcs, **Set Attribute 2** will inform it of which arcs are its following/leading arcs by assigning their numbers to entities' *PostArc* and *PreArc* respectively.

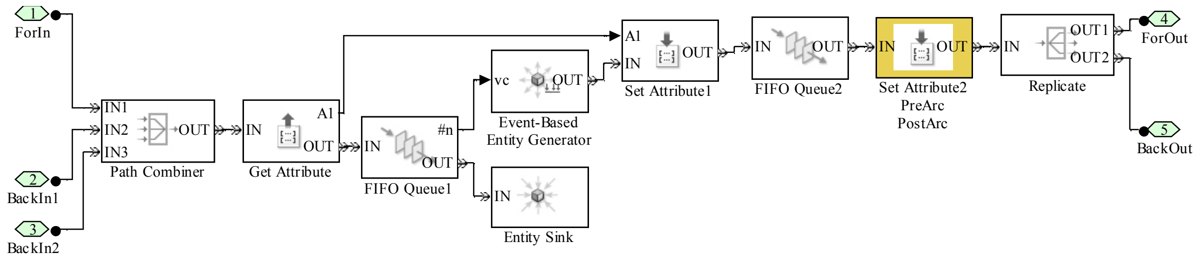

Fig. 3 Middle node

Besides the basic function block, there are still three entity input ports supplied by **Path Combiner** and two output ports supplied by **Replicate**. **ForIn** is the input port set for its leading arc. Because there is only one leading arc A in IA_2, so only one input port is configured. **BackIn1** and **BackIn2** are set for its two following arcs B and D. Among the two output ports, one is **ForOut**, which is used to transmit entity forward to the following arc, the other is **BackOut**, which is used to transmit entities backward to the leading arcs.

2. Source node

The composition of source node is similar to the middle node except that there is only one output port. Because the source node is the end of messages' backward transmission, from which messages will be transmitted forward rather than backward.

3. Terminal node

Terminal node is the destination of all messages, it will only accept but not output entities. Therefore, it does not need to generate new entity.

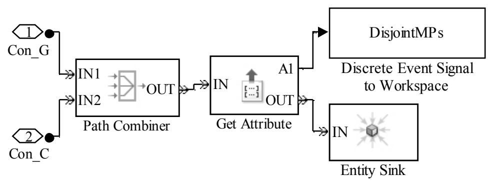

Fig. 4 Terminal node

The terminal node of Network 2 is 5, its composition is shown in Fig. . **Path Combiner** is used to provide input ports for the entities coming from the leading arcs. The messages carried by entities will be obtained by **Get Attribute**, and then output to Matlab's workspace.

Arc model

The main function of arc is to transmit and rewrite messages according to the rules mentioned in **Error! Reference source not found.**. Because messages will be transmitted not only forward but also backward in arcs, and the transmission and the rewriting rules are completely different, so we'll establish a forward-transmission model for OA_i and a back-transmission model for IA_i.

1. Forward-transmission model for OA_i

Take OA_2 as an example, whose model is shown in Fig. and the basic modeling ideas are:

(1) For each message, **Attribute Function1** is used to get the number of arcs included in according to entity's two attributes, *PostArc* and *Message*, and then determine the value of *RouteIndex*. If OA_i^* is $\varnothing$, *RouteIndex*=2, and the entity will be transmitted through **OUT2** of **Output Switch**; Otherwise, *RouteIndex*=1, and the entity will be transmitted through **OUT1** of **Output Switch** for further processing.

(2) **Set Attribute** will initialize entity's *ValidTransArc* and *TransArcNumber* to 0, *RewtrittenIndex* to 2. In the following processing, *ValidTransArc* will be used to record the number of arcs in which the message carried by entity has been rewritten, *TransArcNumber* will be used to record the total number of arcs the entity has gone through, and *RewtrittenIndex* will be used to record whether the message has been rewritten after it leaves its leading node.

(3) In OA_2, we appoint B as the first arc and D the second one in advance. The rewriting function of arc B and D will be respectively implemented by **Attribute Function2** and **Attribute Function3**. The main operations on each message include:

① When the entity goes into the first arc, i.e. B, **Attribute Function2** will judge whether the message contains B or $\bar{B}$. If the answer is 'No', the message will be rewritten in accordance with the rules described in **Error! Reference source not found.**. Then *RewrittenIndex*=1, *ValidTransArc*= *ValidTransArc*+1. If the answer is 'Yes', the message will not be rewritten. After all the operations have been finished, *TransArcNumber*= *TransArcNumber*+1.

② Then, the entity will be further transmitted into two routes provided by. The first route is connected with **Output Switch1** which will decide the output port according to the value of *RewrittenIndex*. If *RewrittenIndex* is 1, the entity will be transmitted to arc B's end node through **Con_B**; otherwise, it will be discarded by **Entity Sink**. The second route is connected with **Attribute Function3**, which will perform the processing function of the next arc.

③ After the entity goes into the second arc, i.e. arc D, the operations in **Attribute Function3** are the same as ①, and the following operations are the same as ②.

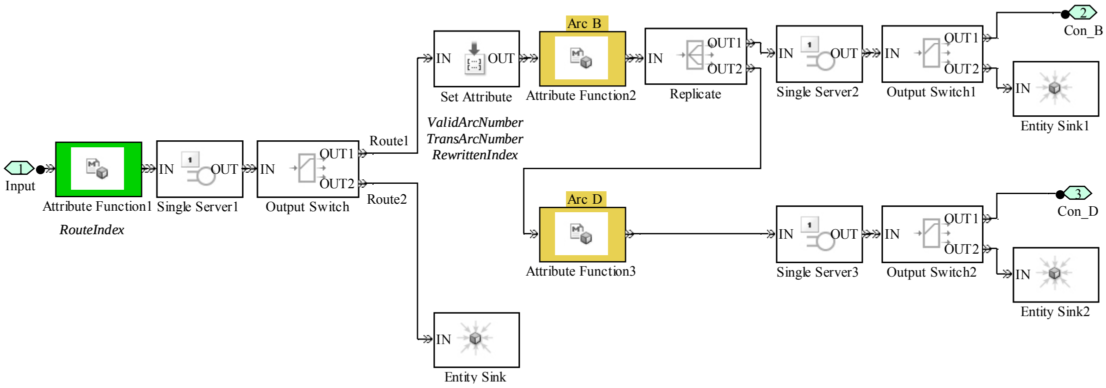

Fig. 5 Forward-transmission model of arcs

2. Backward-transmission model for IA_i

The backward-transmission model is similar to the forward-transmission. The model shown in Fig. is the model of IA_3, which contains arc D and F.

(1) **Attribute Function** is used to select the route of the entity coming from node 3. If IA_3^* is $\varnothing$, *RouteIndex*=1; otherwise, *RouteIndex*=2, the entity will be discarded by **Entity Sink**.

(2) **Set Attribute** will initialize entity's attribute *RewritenIndex* to 2, and **Replicate** will provide an output port for each following arc of node 3.

(3) **Attribute Function_D** and **Attribute Function_F** will execute the message processing function respectively of arc D and F, and record whether *Message* has been rewritten. If the

answer is 'Yes', *RewritenIndex*=1; otherwise, 2. **Output Switch**1 and **Output Switch2** will determine the route to which entity will go according to *RewritenIndex*. If *RewritenIndex*=1, the entity will be transmitted through **Out_D** or **Out_F** to the leading node to wait for further processing. Otherwise, it will be discarded.

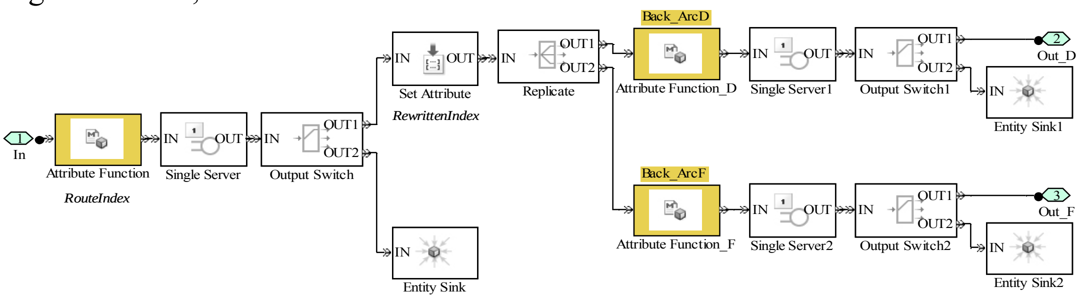

Fig. 6 Backward-transmission model of arcs

Network model

Network 2's transmission model is shown in Fig. , an entity will enter it from port **Conn** at the beginning of simulation. Intuitively, the network is mainly composed of node subsystem, arc's forward-transmission subsystem and arc's backward-transmission subsystem. They are connected by lines according to the relationship between the nodes, IA_i and OA_i. For example, the entities output from node 3 will be transmitted forward to arc G and E, and backward to arc D and F, so in Fig. , Node 3's forward output port **ForOut** is connected with **Arc_G/E**, and its backward output port **BackOut** is connected with **Back_Arc_D/F**.

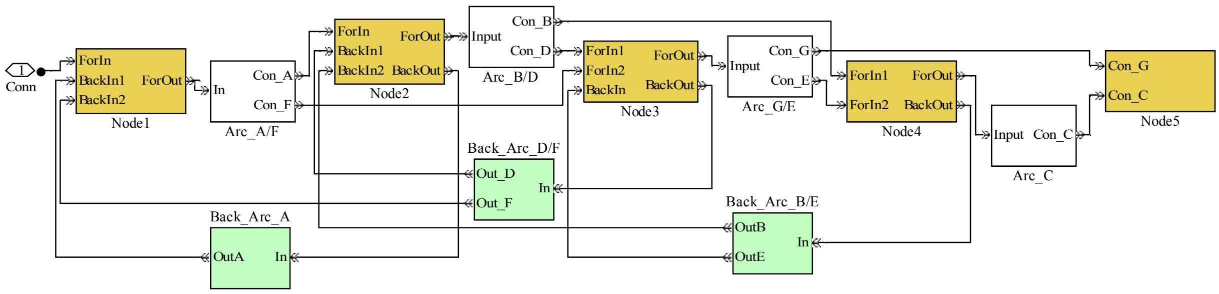

Fig. 7 Simulation model of Network 2

Simulation results

The simulation results are shown as follows, they are exactly the disjoint MPs of Network 2.

① ABC ② $\bar{A}FG$ ③ $A\bar{B}DG$ ④ $\bar{A}F\bar{G}EC$ ⑤ $A\bar{B}CDE\bar{G}$ ⑥ $A\bar{B}\bar{D}FG$ ⑦ $A\bar{B}C\bar{D}EF\bar{G}$ ⑧ $AB\bar{C}DG$ ⑨ $AB\bar{C}\bar{D}FG$

Conclusion

From the point of DES, this paper proposed a new version of disjoint MPs. Then, in SimEvents, the simulation model of network was constructed with the consideration of the encapsulation and reusability of nodes and arcs, which could make the construction of the network model to be greatly simplified. Through the simulation of the model, the new algorithm was realized and the disjoint MPs were automatically generated. So the method proposed by this paper provides an efficient and reliable way for the generation of the disjoint MPs of networks, especially the great-scale one.

References

[1] Lin JS, Jane CC, Yuan J. On reliability evaluation of a capacitated-flow network in terms of

minimal pathsets. Networks 1995; 25(3): 131-138.
[2] Singh B, Ghosh SK. Network reliability evaluation by decomposition. Micro Reliab 1994; 34(5): 925-927.
[3] Shen YL. A new simple algorithm for enumerating all minimal paths and cuts of a graph. Micro Reliab 1995; 35(6): 973-976.
[4] Lin YK. System reliability evaluation for a multistate supply chain network with failure nodes using minimal paths. IEEE Trans Reliab 2009; 58(1): 34-40.
[5] West DB. Introduction to graph theory. New Jersey: Prentice Hall Englewood Cliffs; 2001.
[6] Barlow RE, Proschan F, Hunter LC. Mathematical theory of reliability. Philadelphia: SIAM; 1996.
[7] Dohmen K. An improvement of the inclusion-exclusion principle. Archiv der Mathe 1999; 72(4): 298-303.
[8] Dohmen K. Inclusion-exclusion and network reliability. J Combin 1998; 5: 537-544.
[9] Jane CC, Yuan J. A sum of disjoint products algorithm for reliability evaluation of flow networks. Euro J Oper Res 2001; 131(3): 664-675.
[10]Wilson JM. An improved minimizing algorithm for sum of disjoint products. IEEE Trans Reliab 1990; 39(1): 42-45.
[11]Locks MO. A minimizing algorithm for sum of disjoint products. IEEE Trans Reliab 1987; 36(4): 445-453.
[12]Yeh WC. An improved sum-of-disjoint-products technique for the symbolic network reliability analysis with known minimal paths. Reliab Eng Syst Saf 2007; 92(2): 260-268.
[13]Hardy G, Lucet C, Limnios N. K-terminal network reliability measures with binary decision diagrams. IEEE Trans Reliab 2007; 56(3): 506-515.
[14]Myers A, Rauzy A. Efficient reliability assessment of redundant systems subject to imperfect fault coverage using binary decision diagrams. IEEE Trans Reliab 2008; 57(2): 336-348.
[15]Singhal M. Binary decision diagram based reliability evaluation. Int J Eng Advan Tech 2012; 2(2): 618-626.
[16]Shi Y F, Lu N, Li HM. An algorithm of network systemreliability based on an improved disjoint minimal path set. Comp Eng Sci 2011; 33(1): 31-35.
[17]Ahmad SH, Jamil A. A modified technique for computing network reliability. IEEE Trans Reliab 1987; 36(5): 554-556.
[18]Li DK. New algorithm for computing three-state devices networks reliability by using matrix coulumn transformation. Comp Simu 2010; 27(3): 362-365.
[19]Levitin G. Reliability evaluation for acyclic consecutively connected networks with multistate elements. Reliab Eng Syst Saf 2001; 73(2): 137-143.
[20]Yeh WC. A modified universal generating function algorithm for the acyclic binary-state network reliability. Reliab Eng Syst Saf 2012: 99(3):139-150.
[21]Yeh W C. Multistate-node acyclic network reliability evaluation. Reliab Eng Syst Saf 2002; 78(2): 123-129.
[22]Yeh WC. Multistate-node acyclic networks reliability evaluation based on MC. Reliab Eng Syst Saf 2003; 81(2): 225-231.
[23]Wu XY, Sha JC. A new algorithm for constructing disjoint minimal path set of network. Syst Eng Theo Prac 2000; 23(1): 62-66.
[24]Fishman GS. Discrete-event simulation: Modeling, programming, and analysis.New York: Springer; 2001.