Get Network’s Disjoint MPs Based on Discrete Events Simulation

Jian Tang\textsuperscript{1,a}, Tao Han\textsuperscript{2,b}\textsuperscript{*}, Jianhu Yuan\textsuperscript{3,c}, Faming Shao\textsuperscript{4,d}

\textsuperscript{1}College of Field Engineering, PLA University of Science and Technology, Nanjing 210007, China
\textsuperscript{a}lgdx_tj@163.com
\textsuperscript{b}563524748@qq.com
\textsuperscript{c}yjhnj1@163.com
\textsuperscript{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 tries 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\textsuperscript{®} 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\cite{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\cite{5}. However, network’s MPs are always intersected with each other, if using total probability formula\cite{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\cite{7,8}. Sum of Disjoint Products (SDP)\cite{9-12}, Binary Decision Diagram (BDD)\cite{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\cite{16}. So some scholars dedicated themselves to find out a way to get disjoint MPs without resorting to MPs\cite{17-22}. In literature \cite{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\textsuperscript{®}, a discrete event simulation platform.

Notations

$G(V,A)$ is a acyclic network, $V = \{i, \ i = 1,2,...,n\}$ is the node set; $A = \{a_k, \ k = 1,...,m\}$ 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 $\psi : A \rightarrow V \times V$, $\psi(a_k) = (i,j)$, wherein, $i$ is the starting node of $a_k$, $j$ is the end node of $a_k$.
is a message composed of the index of arcs, which could be \(a_k\) or \(\overline{a_k}\) \((k = 1, 2, \ldots, m)\); \(m_0\) is an empty message; \(M_i\) is the set of messages stored in node \(i\), \(M_i = \{m_r | r = 1, \ldots, n\}\); \(IV_i\) 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 \(i\) \((i \in V, \ i \neq n)\), \(OA_i = \{a_i | \Psi(a_i) = (i, j), j \in OV_i\}\); \(OA_i^*\) 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. 1, 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].

\[
S = G + B = G + \overline{G}B = A(C + D) + \overline{A}(C + D)B
\]

\[
= A(C + \overline{C}D) + (\overline{A} + A(C + D))B = AC + A\overline{C}D + \overline{A}B + A\overline{C}\overline{D}B
\]

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 \(\overline{A}\). Then \(\overline{A}\) will be transmitted along another path, i.e. 1-3 to node 3 and rewritten as \(\overline{AB}\). 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 \(\overline{C}\). Then \(\overline{C}\) will be transmitted to 3 along the other path parallel to arc C, i.e. arc D, and rewritten as \(\overline{CD}\).

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 \(M_i = \{m_0\}\), \(m_0\) is an empty message; \(\forall i \in V\) and \(i \neq 1\), let \(M_i = \emptyset\).

**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:

1. (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 \(\overline{a_k}\), 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 \(a_1, a_2, \ldots\), temporarily.
(2.2.3) Transmit $m_r$ to $a_i$’s end node $j$ in the aforementioned order and rewrite $m_r$ in obedience to the following rules: Firstly, transmit $m_r$ to $a_{i1}$, rewrite it as $m_r a_{i1}$, and store the new $m_r$ to $a_{i1}$’s end node $j_1$; secondly, transmit $m_r$ to $a_{i2}$, rewrite it as $m_r a_{i1} a_{i2}$, and store the new $m_r$ to $a_{i2}$’s end node $j_2$; similarly, transmit $m_r$ to the third, fourth, … arcs in $OAl$, rewrite it as $m_r a_{i1} a_{i2} a_{i3}, m_r a_{i1} a_{i2} a_{i3} a_{i4}, ...$, 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 $OAl$ as $a_{i1}$, $a_{i2}$, ….

(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 $OAI$, judge whether $m_r$ contains $a_{i1}$ or $\bar{a}_{i1}$. If the answer is ‘No’, rewrite $m_r$ as $m_r a_{i1}$, if ‘Yes’, give up rewriting and keep $m_r$ unchanged; then, turn to the second arc $a_{i2}$, and the rewriting rule is the same as $a_{i1}$; finally, after all the arcs in $OAl$ have been scanned, store the new $m_r$ in the leading node of $b_l$.

(2.3) Delete $m_r$ from $M_i$.

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

Realization by means of Discrete Event Simulation in SimEvents

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 $OAI$ and the backward-transmission model of $IAi$.

![Fig. 2 Network 2](image_url)

**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, 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 and a back-transmission model for IA.

1. Forward-transmission model for OA

Take OA 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 $O\bar{A}'$ is $\emptyset$, 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, RewrittenIndex 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 RewrittenIndex will be used to record whether the message has been rewritten after it leaves its leading node.

(3) In $OA_3$, 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 $IA_3$

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 $\emptyset$, RouteIndex=1; otherwise, RouteIndex=2, the entity will be discarded by Entity Sink.

(2) Set Attribute will initialize entity’s attribute RewrittenIndex 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’, 
RewrittenIndex=1; otherwise, 2. Output Switch1 and Output Switch2 will determine the route to which entity will go according to RewrittenIndex. If RewrittenIndex=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.

**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, \( I_A \) and \( O_A \). 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.

**Simulation results**

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

1. \( ABC \)
2. \( AFG \)
3. \( ABDG \)
4. \( AFGEC \)
5. \( AB\overline{C} DE\overline{G} \)
6. \( \overline{A}BDFG \)
7. \( ABC\overline{D}EF\overline{G} \)
8. \( ABCDG \)
9. \( ABCDFG \)

**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


