International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

An efficient Activity Network Reduction Algorithm based on the Label Correcting Tracing Algorithm

Weng Ming Chu

***Abstract*—**When faced with stochastic networks with an uncertain duration for their activities, the securing of network completion time becomes problematical, not only because of the non-identical *pdf* of duration for each node, but also because of the interdependence of network paths. As evidenced by Adlakha & Kulkarni [1], many methods and algorithms have been put forward in attempt to resolve this issue, but most have encountered this same large-size network problem. Therefore, in this research, we focus on network reduction through a Series/Parallel combined mechanism. Our suggested algorithm, named the Activity Network Reduction Algorithm (ANRA), can efficiently transfer a large-size network into an S/P Irreducible Network (SPIN). SPIN can enhance stochastic network analysis, as well as serve as the judgment of symmetry for the Graph Theory.

***Keywords*—**Series/Parallel network, Stochastic network, Network reduction, Interdictive Graph, Complexity Index.

I. Introduction

ALMOST all of the issues of stochastic networks are more difficult to deal with than those of deterministic networks due to the probability characteristics of activity duration. The approximation of completion time, for example, has been acknowledged as a very difficult problem. Adlakha & Kulkarni [1] has classified the methodology into three types: Exact analysis methods, Approximation & Bounding methods and Monte Carlo Simulation methods (MCS). Let alone their performance, most of these methods have encountered the network size problem which has affected their practical application in the real world and, thus, underlined the need to deal with it before dealing with stochastic network problems. As is well known, MCS has been used to solve the *pdf* of network completion time, first by [18] using the "Crude Simulation." Martin[13], Hartley and Wortham[9], Ringer[14], Burt & Garman[3] and Robillard & Trahan [15] continued this research, with all concentrating on network reduction first and letting the network transfer into an S/P Irreducible Network (SPIN) to lessen activity numbers which can enhance the efficiency of simulation. Fisher, Saisi & Goldstein [8], Dodin [5] and Dodin [6] also focused on network reduction before their approximation of network completion time, as did Kleinderfer [11], Kleinderfer [12] and Splede [17] for bounding the completion time of the network. The structure of SPIN can not be further reduced through the Series or Parallel combined operation on activities in the network, and the remaining activities in the network are in the end composed of various types of Interdictive Graph (IG). Therefore, SPIN can be seen as a network which is pieced together with structures of IG. IG is synonymous to "Wheatstone Bridge" or "Double Wheatstone Bridge" as depicted in Hartley & Wortham [9] and Ringer [14], respectively, who analyzed these specified structures to an approximation of the completion time of a stochastic network. In Bein et al. [2], SPIN was also used to help with the algorithm for finding Dominated Tree (DT), Reverse Dominated Tree (RDT) and, then, the Complexity Graph (CG). The "Complexity Index" must be generated from the CG, and this important concept is required for further applications of the stochastic network [10].

As noted above, network reduction is necessary for and essential to stochastic network research, but most of the literature is lacking in details as to the algorithm itself and instead gives only a simple network example as an explanation. Obviously, this is insufficient information for a large-size network. Thus, the focus of this research was to develop the Activity Network Reduction Algorithm (ANRA), which can do Series/Parallel network reduction very efficiently. and, therefore, can also be used to judge whether or not the network belongs to an S/P network. This judgment is important to and useful for project risk or reliability analysis in stochastic networks.

In this research, the network was in the form of a two-terminal Activity-On-Node(AON) and represented as G=(*V*, *E*), with *V* and *E* as the set of nodes and the edge, respectively, in the network. Let S_m, B_m to be defined as set of the successor and precedent nodes of *node*(*m*), *node*(*i*) and *node*(*j*) are series connected with each other in the network. In case that there exists $S_j = i$ and $B_i = j$, *node*(*i*) and *node*(*j*) can commit the series combination and merge into single node node(*k*), where $S_k = B_j$, $B_k = S_i$. For the other case, if there exists a couple of nodes $\{p^1, p^2, ..., p^n\}$ and $S_{p^i} = X$, $B_{p^i} = Y$, $\forall i = 1, ..., n$, where

Author is with the Department of Electronic Commerce Hsing Kuo University of Management, Tainan, Taiwan of R.O.C. (corresponding author to provide phone: 886-6-2873613; e-mail: chuwengming@gmail.com).

This research is sponsored by National Science Council, the project index number is NSC 97-2221-E-432 -002.

X, Y can be single or set of node, then $\{p^1, p^2, ..., p^n\}$ can commit the parallel combination and merge into single node *node*(r), where $S_r = X$, $B_r = Y$.

For practical consideration, the research issues of stochastic networks should involve instances of large-size networks, which requires that network reduction be applied for reducing the network scale beforehand. In general, network reduction faces two challenges:

1) The reducing algorithm can efficiently operate a large-size network.
2) The reducing algorithm can recognize various types of IG structures from the network in the meantime, since the IGs could be connected with each other.

Colby [4] indicated the above as an NP-hard problem and ANRA was developed to meet these challenges. The algorithm was based on the Label-Correcting Tracing Algorithm (LCTA) introduced by Yao, Chu & Tseng [19] and Yao and Chu [20]. LCTA was originally used for approximating the completion time of a stochastic network, as it could systematically visit each node in the network and commit relative executions. ANRA applies a modified LCTA mechanism and commits Series and Parallel combinations during the node visiting. When LCTA has finished that visiting, the network reduction has been completed. To ensure the validity of ANRA, several requirements had to be met:

1) That any IG structures existing within the network could be identified after all the Series and Parallel combinations had been executed;
2) None of the characteristics of the network could be changed after network reduction (such as the network completion time).
3) The execution time of the algorithm should be reasonable when faced with large-size networks.

This paper has 6 sections: Section II gives a detailed description of LCTA; Section III lists all the conditions for executing Parallel combinations in the networks which have been classified into 2 groups and 5 types; Section IV details the procedures and theorem of ANRA, assisted with pseudo codes; and Section V gives 2 examples to show and prove the performance of ANRA in this research. The last section is the conclusion and comments on future development.

II. Label Correcting Tracing Algorithm

The Label-Correcting Tracing Algorithm (LCTA) is developed by [19] and [20], it was originally developed for the approximation of the completion time of the stochastic networks. It can be easily modified to deal with other applications for stochastic networks. LCTA has first to transfer the network to Expanded-Tress Structure (ETS), and visits each node through the Post-Order Tracing Procedure(POTP) in the ETS. During the node visiting, LCTA proceeds S/P combination The brief descriptions are described as follow:

The first step of LCTA is to transfer the network into a tree structure, we name it as the Expanded-Tree Structure (ETS). Fig. 1 is an example of transformation.

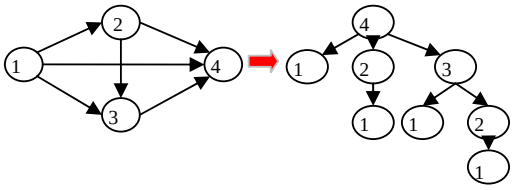

Fig. 1 An example of ETS transformation

LCTA makes some calculations during its tracing visit, and the temporary generated results, status and information must be saved. LCTA has a node data structure for each node to fulfill these requirements. The contents of the data structure are in accordance with the purposed requirements. For this research, the contents of *node*(i) were arranged as shown in Table I:

TABLE I
The Data Structure of Node(I)

i	B_i	S_i
$node(i).flag$	$finish\_flag_i$	$node(i).path$

1) i : index number of *node*(i).
2) B_i : set of precedent nodes of *node*(i).
3) S_i : set of success nodes of *node*(i).
4) $finish\_flag_i$: visited flag value of *node*(i) initially set as 0. When *node*(i) has been visited, it will be set as 1.
5) $node(i).flag$: visited flag set of incident paths of *node*(i) where all are initially set as 0. When the k^{th} incident paths of *node*(i) have been visited, the relative k^{th} position of $node(i).flag$ will be set as 1.
6) $node(i).path$: set of all sub-paths from *node*(*0*) to *node*(i). Each sub-path is recorded as a set of node index numbers by following the sequence of *node*(*0*) to *node*(i).

LCTA starts its visiting from the terminal node of ETS (the root of the tree), and applies POTP to visit each node in ETS. The main rule of POTP is that, before visiting the current node, *node*(j), it must confirm that B_i has visited in sequence from left to right. By following the rule of POTP, LCTA can then visit each node in the proper order. LCTA has a *stack structure* to record its tracing path of where the data get in and out by following the rule of the First In/First Out (FIFO); the data recording and accessing operations are denoted as *push_stack*() and *pull_stack*(), respectively.

Initially, LCTA starts from *node*(*N*) and goes downward until reaching *node*(*1*); the index numbers of the nodes passed are recorded in a *stack structure* in sequence. Then, LCTA begins to bounce up and start the node visiting. By following the rule of POTP, all the nodes are visited systematically. The visiting order is directed and guided by accessing the index numbers from the *stack structure*, which lets all the child nodes find their way back to their father node position. With the status of $node(i).flag$, *node*(i) can not only record the visited state of the child nodes but it can also find the next-to-visit child node. When all of the child nodes have been visited, the $finish\_flag_i$ will be set. With the above mechanism of

tracing, the node-visited track of LCTA is downward and upward repeatedly within ETS, and then finally back to the position where it started, *node*(*N*). When that has happened, LCTA has completed its tracing, as the tracing sketch map of Fig. 1 becomes as shown in Fig. 2:

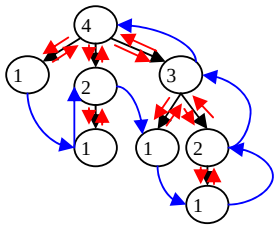

Fig. 2 The tracing sketch map of Fig. 1

LCTA has to check two conditions for current visited node *node*(*cur_node*) during the iterative tracing. The above two checking conditions have been summarized into two procedures: Upward_Trace (*last node*, *current_node*) and Downward_Trace (*current_node*), the details of which are described in the last section.

III. Patterns of Parallel Reduction in the Network

The major consideration of network reduction is to find nodes that can commit Parallel combination and merge them into a single node in the network. This can be easily done artificially for a small-size network, but it is more complex for a large-size network since there are various structure patterns of nodes which can commit the Parallel combination. The main job of ANRA was to find all the eligible patterns in the network and commit the Parallel combination individually. In this section, the eligible structure patterns have been classified into 2 groups with 5 patterns in total, with detailed descriptions below.

A. First group of Parallel combination

There are 2 structure patterns in the first group for Parallel combination, denoted as <1,1> and <1,2> , as shown in Fig. 3.

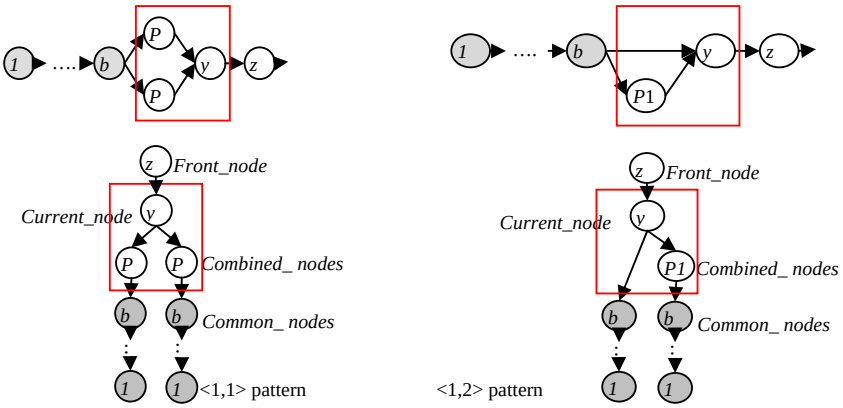

Fig. 3 <1,1> and <1,2> structure patterns of Parallel combination

Parallel combination is done with only two precedent nodes; if *node*(*y*) has more than two precedent nodes, they have to be executed iteratively pair by pair. In Fig. 3, *node*(*y*) has two precedent nodes, *node*(*p1*) and *node*(*p2*). These two nodes have only one path to *node*(*1*) and they share the single path of *Path*(*1*,…,*b*), which is also the main characteristic of the first group of Parallel combination. The <1,2> pattern is actually the special case of <1,2> pattern, of which the duration of *node*(*p2*) is zero.

node(*y*) can be taken as the current visited node in LCTA, if one of the <1,1> and <1,2> structure patterns exist at *node*(*y*). LCTA notices it by the following condition checks:

1) Any two incident paths of *node*(*y*) must share the same nodes, denoted as *common_nodes*. Taking Fig. 3 as an example, they are in *Path*(*1*,…,*b*) and *common_nodes*={ *node*(*1*), …, *node*(*b*)}.
2) If the two incident paths of *node*(*y*) exclude the *common_nodes*, the remaining nodes are denoted as *combined_nodes*, which can be combined as a single node in the network; their precedent nodes would then need to be 1. Taking Fig. 3 as an example, *combined_nodes*={ *node*(*p1*), *node*(*p2*)}, all have only one precedent node, *node*(*y*).

If any one of the above condition checks fails, Parallel combination can not be committed at *node*(*y*). Otherwise, the *combined_nodes* can be merged into a single node and embedded back to the original network.

The *combined_nodes* can include current node *node*(*y*), if *node*(*y*) has no other precedent node except nodes of the *combined_nodes*. Therefore, the combination has two results, as shown in Fig. 4, where *node*(*q*) of Fig. 4a has included *node*(*y*) and *node*(*q*`) has not. Obviously, the precedent and successor relation of *node*(*q*) and *node*(*q*`)within the network is different, which is why the combination must take these two situations into consideration.

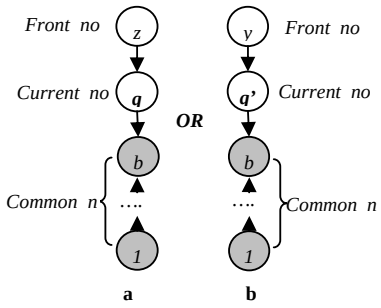

Fig. 4 Two combination results of the first group patterns

A. Second Group of Parallel combination

There are 3 structure patterns in the second group of Parallel combination which are denoted as <2,1> , <2,2> and <2,3> , as shown in Fig. 5.

Parallel combination is done with only two precedent nodes at a time; the current node, *node*(*y*), has two precedent nodes, *node*(*g*) and *node*(*h*). Unlike the first group patterns, the incident paths of *node*(*g*) and *node*(*h*) number more than one, and *common_nodes* is a node structure rather than a single path. This is the main characteristic of second group patterns. The <2,3> pattern is actually the special case of the <2,2> pattern, and the duration of *node*(*g*) is zero.

When LCTA visits *node*(*y*), it also checks if <2,1> , <2,2> or <2,3> structure pattern exists at *node*(*y*). Parallel combination is executed if one of them exists. The condition checks for the second group patterns are much the same as for the first group patterns due to *common_nodes*. The condition checks are as follows:

1) All the incident paths from *node*(*1*) to *node*(*g*) and *node*(*h*) must match, which means that *node*(*g*) and *node*(*h*) share the same *common_nodes*. Taking Fig. 5 as an example, the nodes within the blue-line frame are the *common_nodes* and they are exactly the same for *node*(*g*) and *node*(*h*).

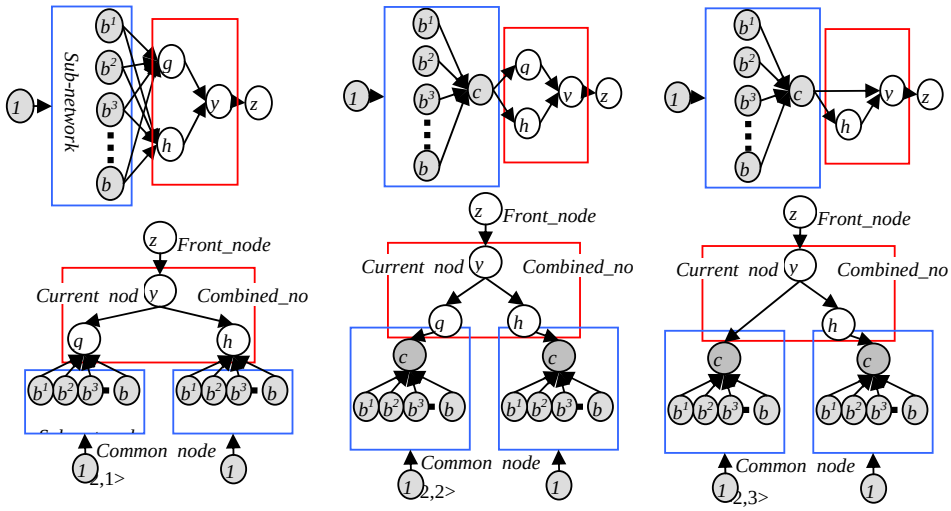

Fig. 5 <2,1> , <2,2> and <2,3> structure patterns of Parallel combination

2) The *combined_nodes* are also defined as the remaining nodes of which the incident paths of *node*(*y*) exclude the *common_nodes*. There can be only one successor node of the *combined_nodes*. As shown in Fig. 5, the *combined_nodes* include *node*(*g*) and *node*(*h*), and there is only one successor node, *node*(*y*).

If any one of the above condition checks fails, Parallel combination cannot be committed at *node*(*y*). Otherwise, the *combined_nodes* are merged into a single node and embedded back to the original network. As for the first group patterns, if *node*(*y*) has no precedent node other than the *combined_nodes*, *node*(*y*) can be included in the *combined_nodes*. As shown in Fig. 6, *node*(*q*) has included *node*(*y*) and *node*(*q*`) has not. The precedent and successor relation of *node*(*q*) and *node*(*q*`) within the network is different.

IV. Activity Network Reduction Algorithm

As discussed in previous sections, ANRA applies LCTA to visit each node of the network and checks the Parallel combination for each visited node. If one of the patterns of Parallel combination exists at the visited node, Parallel combination will be executed and the *combined_nodes* will become a single node and embedded into the network. When LCTA has finished the tracing algorithm, the network reduction has also been completed.

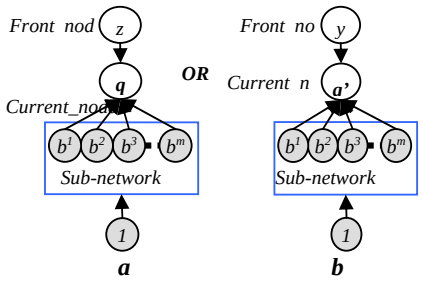

Fig. 6 Two combination results of the second group patterns

To fulfill the condition checks for Parallel combination, ANRA must search all the incident paths for each node in the network. This tedious job can also be solved through LCTA and the results saved in the *node*(*i*).*path* for each *node*(*i*), *i=1*,…,*N*. The searching algorithm of the incident paths will now be introduced, together with the conditions checking algorithm of Parallel combination, and the full algorithm of ANRA presented.

A. Incident Paths Searching Algorithm

Let the k^{th} incident path set of *node*(*i*) be denoted as $path_k(i)$, which goes from *node*(*1*) to *node*(*i*). By applying the mechanism of LCTA node-visiting, all the incident paths of *node*(*i*) can be solved and saved into *node*(*i*).*path*. As shown in Fig. 1, LCTA starts its tracing from *node*(*4*), when LCTA touches down to *node*(*1*), it bounces up and begins to record the node index number of the incident path for the visited node *node*(*i*). Following the rule of POTP, each one of the incident paths was recorded sequentially and saved into *node*(*i*).*path*. When LCTA finished the tracing, the recording of all the incident paths was also completed. We denoted the above incident paths searching algorithm as Record_node (*last_node*, *current_node*) and the step-by-step description for Fig. 1, by referring to Fig. 2:

1) LCTA starts its tracing from *node*(*4*) and goes downward until it touches *node*(*1*).
2) To record index number of *node*(*1*) into *node*(*1*).*path* and then back to *node*(*4*).
3) To record *node*(*1*).*path* and *node*(*4*) to *node*(*4*).*path*(*1*) as $path_1(4)$, the first incident path set of *node*(*4*).
4) LCTA goes down to *node*(*1*) through *node*(*2*) by following the rule of POTP. Since *node*(*1*) has been visited, LCTA upwards and back to *node*(*2*) again.
5) To record *node*(*1*).*path* and *node*(*2*) to *node*(*2*).*path*(*1*) as $path_1(2)$, the first incident path set of *node*(*2*).
6) LCTA goes back to *node*(*4*), and records *node*(*2*).*path* and *node*(*4*) to *node*(*4*).*path*(*2*) as $path_2(4)$, the second path incidence path set of *node*(*4*).
7) LCTA traces its last child nodes, goes downward to *node*(*1*) through *node*(*3*), since *node*(*1*) has been visited, then upward and back to *node*(*3*) again.
8) To record *node*(*1*).*path* and *node*(*3*) to *node*(*3*).*path*(*1*) as $path_1(3)$, the first incident path set of *node*(*3*).
9) LCTA traces the second child of *node*(*3*) and goes down to *node*(*2*). Since *node*(*2*) has been visited, it goes upward and back to *node*(*3*) again and records *node*(*2*).*path* and *node*(*3*) to *node*(*3*).*path*(*2*) as $path_2(3)$, the second incidence path set of *node*(*3*).

B. Conditions Checking Algorithm of Parallel Combination

Let the current visited node be denoted as *node*(*y*) and the numbers of precedent node *m*. For any two precedent nodes, in this case *node*(*a*) and *path*(*b*), the conditions checking algorithm of the Parallel combination is as shown below:

1) Pick one pair of precedent nodes of *node*(*y*), *node*(*a*) and *path*(*b*), and check their incident paths numbers. If they are not equal, there is no need to do the Parallel combination on these two precedent nodes. Pick another pair of precedent nodes of *node*(*y*) and repeat the above checks. Otherwise, go to step B. The above executions continue until there are no additional pairs of precedent nodes of *node*(*y*) to be selected.
2) This step determines to which group patterns the Parallel combination belongs by the numbers of incident paths: if equal to 1, then go to step C and start the checking of the first group patterns; otherwise, go to step D and start the checking of the second group patterns.
3) To check the first group patterns:
   a. Following the description in Section III, check if there are *common_nodes* between the incident paths of *node*(*a*)and *node*(*b*). If it is positive, continue further checking; otherwise, go back to step A.
   b. To solve the *combined_nodes* of *node*(*a*)and *node*(*b*), if any of their successive nodes is greater than 1, go back to step A; otherwise, continue checking.
   c. Following the description in Section III, check if the *combined_nodes* include *node*(*y*). If *m*>2, *node*(*y*) is excluded; otherwise, it is included. Then, merge the *combined_nodes* into a single node and assign it a new node index number; the new generated node should be arranged as to its precedent and successive relation in the network before being embedded into the network.
   d. *m*=*m*-1, then go back to step A.

1) To check second group patterns:
   a. Following Section III, check if all the incident paths of *node*(*a*)and *node*(*b*) are equal with each other. If so, continue checking; otherwise, go back to step A.
   b. To determine the *combined_nodes* of *node*(*a*)and *node*(*b*), if any one of their successive nodes is greater than 1, go back to step A; otherwise, continue checking.
   c. Following Section III, check if the *combined_nodes* include *node*(*y*). If *m*>2, then *node*(*y*) is excluded; otherwise, it is included. It is then necessary to merge the *combined_nodes* into a single node and assign it a new node index number; the new generated node should be arranged as to its precedent and successive relation in the network before being embedded into the network.

The author has arranged the above execution steps into an algorithm and denoted it as the Parallel_Process (*current_node*), where *current_node* is the only input

parameter and the pseudo code is listed as Algorithm 1.

Algorithm 1 (Parallel combination algorithm)

```
current_node =Parallel_Process(current_node)
q=n;
A=node(current_node).precedent;
m=length(node(current_node).precedent);
 while (there exists a pair nodes of A for parallel checking)
do
    Select a pair nodes(a1, a2)from A;
    check_parallel=0;
    if length(node(a1).path)= length(node(a2).path)
      if length(node(a1).path)=1
        com_node=intersect(node(a1).path, node(a2).path);
        combined=setxor({node(a1).path, node(a2).path},
          com_node);
        if ( length(node(ci).succesor)=1)
            ∀c_i ∈ combined, i = 1,...,length(combined)
          check _parallel=1;
          q=q+1;
          Delete_nodes(combined);
          Insert_node(combined , q);
       m=m-1;
       end
       else
        Solve c_node_head1 (the first node with its
            precedent>1 in the node(a1).path);
        Solve c_node_head2 (the first node with its
            precedent>1 in the node(a2).path);
        combined = { node(j) | node(j) ∈
         [path(node(c_node_head1) ~node(a1)) ∪
         path(node(c_node_head1) ~node(a2))] };
      if ( length(node(ci).succesor)=1,
        ∀c_i ∈ combined, i = 1,...,length(combined) )
          if c_node_head1= c_node_head2
            check _parallel=1;
            q=q+1;
            combined = combined-{ c_node_head1};
            Delete_nodes(combined);
            Insert_node(combined , q);
            m=m-1;
          else
           if (node(c_node_head1).precedent=
       node(c_node_head2).precedent)
             check _parallel=1;
             q=q+1;
             Delete_nodes(combined);
             Insert_node(combined , q);
             m=m-1;
           end
          end
        end
       end
  end
  end /(while)
  if check_parallel=1
    if m=1
     combined ={ combined , current_node };
     Delete_nodes({q, current_node });
     q=q+1;
     Insert_node(combined , q);
     current_node=q;
    end
  end
```

The developed ANRA applies the LCTA implanted with algorithms of Record_node (*last_node*, *current_node*) and Parallel_Process (*current_node*) which have been discussed in Sections IV.A and IV.B, respectively. The Record_node (*last_node*, *current_node*) is usually executed before the Parallel_Process (*current_node*) for each node in LCTA and, therefore, they do not interfere with each other which assures the smooth going of Parallel_Process (*current_node*) in LCTA. The algorithm of ANRA is separated into two procedures: Downward_ Trace (*k, j*) and Upward_ Trace (*k, j*), where *j*= *current_node* and *k*=*last_node*. Their pseudo codes are listed as Algorithm 2.

Algorithm 2 (Algorithm of ANRA)

```
j= current_node;
k=last_node;
Downward_ Trace (k, j)
 while ( finish _ flag_n =1) do
    if ( finish _ flag_j =1)
      k= j;
      j = pull_stack();
      Upward_Tracing(k, j);
    else
      if (j=1? )
        finish _ flag_j =1;
      Record_node(0, 1);
      k=j;
      j= pull_stack();
      Upward_Tracing(k,j);
     Else
      k=j;
      push_stack(j);
      j=Select_node(j);
     end
    end
  end

Upward_ Trace (k, j)
while ( finish _ flag_n =1) do
    Record_node(k, j);
    if ( finish _ flag_j =1)
      j=Parallel_Process(j);
      finish _ flag_j =1;
```

```
    k=j;
    j= pull_stack();
  else
    k=j;
    j=Select_node(j);
    push_stack(j);
    Downward_Tracing (k, j);
  end
end
```

The visiting mechanism of LCTA is done through two procedures: the Downward_ Trace(*k, j*) and Upward_ Trace(*k, j*). The initial value of the current node is set as *N* (the terminal node number), and the Downward_ Trace(*N*,*N*) starts first. The tracing of LCTA follows the iterative executions between Downward_ Trace(*k, j*) and Upward_ Trace(*k, j*). Within the two procedures, the visited node executes Parallel_Process(*j*) for the Parallel combination while the Record_node (*k, j*) is usually committed to getting all the incident paths of the current node beforehand. When LCTA finishes visiting all of the network nodes, all the possible Parallel combinations of the network will also be done, and network reduction will have been achieved.

V. Experiment of Examples

ANRA was applied to two examples of networks to help the reader more fully comprehend how it functions. The first network instance was designed to be a Series/Parallel network, which was reduced to a single node after running ANRA. The second network instance was implanted with an IG structure; network reduction shrank the network size to that of an S/P irreducible network.

A. Example 1

The network instance of the first example is shown in Fig. 7, the procedures of ANRA are described as follows, with the middle merged results in Fig. 8.

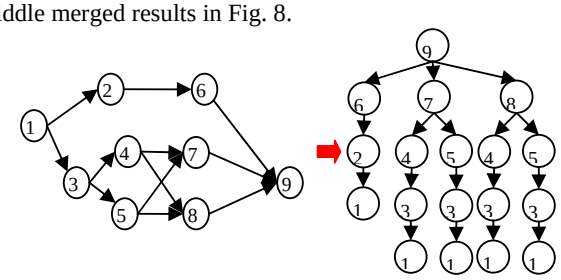

Fig. 7 The network instance of example 1

1) LCTA starts tracing from *node*(*9*), and the Downward_ Trace (*k, j*) and Upward_ Trace (*k, j*) are iteratively executed with each other. The first checking of the Parallel combination stops at *node*(*7*), since it is the first node where the precedent node number is greater than 1.

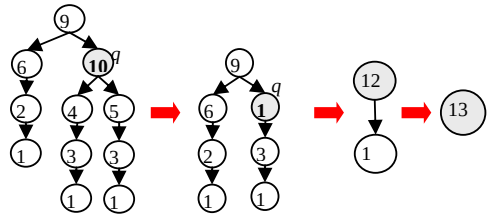

Fig. 8 The results of Parallel combination of Example 1

2) The incident paths of *node*(*7*) is solved by Record_node (*4,7*) and Record_node (*5,7*) and saved in *node*(*7*).*Path*. The executions of Parallel_Process(*7*) are listed below:
   a. Parallel_Process(*7*) indicates that *node*(*7*) belongs to the <1, 1> pattern of Parallel combination, since there is only a single incident path for both precedent nodes of *node*(*7*).
   b. Parallel_Process(*7*) solves *common_nodes* ={1, 3} and *combined_nodes*={4, 5}.
   c. Parallel_Process(*7*) finally denies the Parallel combination on *node*(*7*), since the number of successive nodes of *node*(*4*) and *node*(*5*) is greater than 1.
3) The second checking of Parallel combination stops at *node*(*8*), and the checking result is exactly the same as for *node*(*7*).
4) *node*(*9*) is the third stop for Parallel combination checking. There are in total 3 pairs of precedent nodes for Parallel combination checking, since it has three precedent nodes, {*node*(*6*), *node*(*7*), and *node*(*8*)}. But, Parallel_Process (*i*) found only one pair, {*node*(*7*), *node*(*8*)}, which passed the checking executions listed below:
   a. Parallel_Process(*9*) indicates that it belongs to the <2,1> pattern of Parallel combination, since the incident path number of both *node*(*7*) and *node*(*8*) is greater than 1.
   b. Parallel_Process(*9*) solves *common_nodes* {*node*(*1*), *node*(*3*), *node*(*4*), *node*(*5*)} and *combined_nodes* ={*node*(*7*), *node*(*8*)}.
   c. The *common_nodes* do not include *node*(*9*) since its precedent is greater than 2. Therefore, *node*(*7*) and *node*(*8*) are deleted from the network and merged into a single node, *node*(*10*), with precedent node and successive node as {*node*(*4*), *node*(*5*)} and *node*(*9*), respectively.
5) *node*(*10*)becomes the current node and LCTA proceeds its checking and finds that it has passed:
   a. Parallel_Process(*10*) indicates that it belongs to the <1,1> pattern of Parallel combination as there is only a single incident path for both precedent nodes of *node*(*10*).
   b. Parallel_Process(*10*) solves *common_nodes*={*node*(*1*), *node*(*3*)} and *combined_nodes*={ *node*(*4*), *node*(*5*)}.
   c. The *common_nodes* include *node*(*10*) since its precedent is not greater than 2. Therefore, *node*(*4*) and *node*(*5*) are deleted from the network and merged into a single node, *node*(*11*), with its precedent node and successive node as *node*(*3*) and *node*(*9*), respectively.
6) *node*(*9*) now has two precedent nodes remaining; LCTA

proceeds with its checking and finds that *node*(*9*) has passed:

a. Parallel_Process(*9*) indicates that it belongs to the <1,1> pattern of Parallel combination as there is only a single incident path for both precedent nodes of *node*(*9*).
b. Parallel_Process(*9*) solves *common_nodes*=*node*(*1*) and *combined_nodes*={*node*(*2*), *node*(*3*), *node*(*6*), *node*(*11*)}.
c. The *common_nodes* include *node*(*9*) since its precedent node is not greater than 2. Therefore, *node*(*2*), *node*(*3*), *node*(*6*) and *node*(*11*) are deleted from the network and merged into a single node, *node*(*12*), which has only one precedent node, *node*(*1*).

The last two nodes, *node*(*12*) and *node*(*1*), can be a series merged into *node*(*13*).

B. Example 2

The network instance of example 2, as shown in Fig. 9, has an implanted IG structure which can be clearly displayed after running ANRA. The executed procedures and the middle merged results are listed below and shown in Fig. 10.

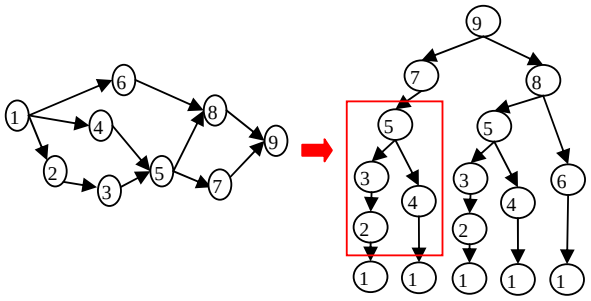

Fig. 9 Network instance of example 2

1) LCTA starts tracing from *node*(*9*), the Downward_ Trace (*k*, *j*) and Upward_ Trace (*k*, *j*) are iteratively executed with each other. The first checking of Parallel combination stops at *node*(*5*), since it is the first node with a precedent node number greater than 1.
2) The incident paths of *node*(*5*) are solved by Record_node (*3*,*5*) and Record_node (*4*,*5*) and saved in *node*(*5*).*Path*. The executions of Parallel_Process(*5*) are as follows:
   a. Parallel_Process(*5*) indicates that it belongs to the <1,1> pattern of Parallel combination as there is only a single incident path for both precedent nodes of *node*(*5*).
   b. Parallel_Process(*5*) solves *common_nodes* = *node*(*1*) and *combined_nodes*={*node*(*2*), *node*(*3*), *node*(*4*)}.
   c. The *common_nodes* include *node*(*5*) since its precedent is not greater than 2. Therefore, *node*(*2*), *node*(*3*) and *node*(*4*) are deleted from the network and merged into a single node, *node*(*10*); its precedent node and successive nodes are *node*(*1*) and {*node*(*7*), *node*(*8*)}, respectively.
3) The second checking of Parallel combination stops at *node*(*8*); Parallel_Process(*8*) proceeds with the checking procedures listed below:
   a. Parallel_Process(*8*) indicates that *node*(*8*) belongs to the <1, 1> pattern of Parallel combination, since there is only a single incident path for both precedent nodes of *node*(*8*).
   b. Parallel_Process(*8*) solves *common_nodes* = *node*(*1*) and *combined_nodes*={ *node*(*6*), *node*(*10*)}.
   c. Parallel_Process(*8*) finally denies the Parallel combination on *node*(*8*) since the numbers of successive node of *node*(*10*) are greater than 1.

LCTA proceeds with its tracing and finally stops at *node*(*9*); there exists a Parallel combination checking over there, and the checking fails since the incident paths numbers of the two precedent nodes of *node*(*9*) are not equal. The final reduced network is an irreducible network achieved with one IG structure (refer to Fig. 10).

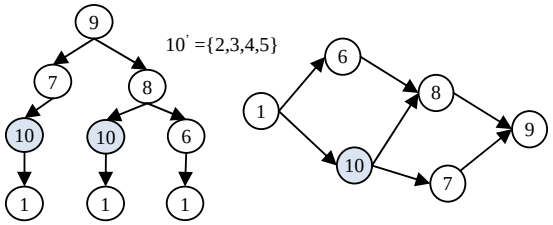

Fig. 10 Results of Parallel combination of example 2

VI. Conclusion

As developed, ANRA can not only efficiently reduce a network into an S/P Irreducible Network (SPIN), but it can also be used as a tool for determining whether or not a network is a Series/Parallel network by displaying all IG structures in the network. The results produced by ANRA are important for stochastic networks since the reduced networks minimize the - burden of computational analysis. As far as the author is aware, most of the relevant literature has been concerned with only simple network cases when discussing network reduction, with none of the algorithms required for its implementation. The development of such an algorithm was the focus of this research by which ANRA has been realized.

ANRA applies LCTA as developed and used for completion time approximation of stochastic networks by [19] and [20]. ANRA modifies the tracing mechanism of LCTA by adding the judgment of various conditions of executing Series/Parallel combination in the network. The author has classified 2 groups and 5 patterns in total, of Parallel combination in this research, which are checked on the visited node during the LCTA tracing. Provided that one of the patterns can be found, ANRA will find combinations of nodes for the respective pattern and the combined node embedded back does not bias the characteristics of the original network. Section III outlined the modified LCTA, which was followed by the description of the checking conditions for the 5 patterns and the details of the combined executions, which have all been assisted with pseudo codes. A clear and comprehensive picture of ANRA was presented in these sections, followed by examples of two network instances executed with ANRA. The results have not only clearly shown the efficiency of ANRA in reducing the

example networks, but they have also verified its validity.

REFERENCES

[1] Adlakha, V.G. & Kulkarni, V.G., "A classified bibliography of research on stochastic PERT networks: 1966-1987", *Information Systems and Operational Research*, 27, n3, 272-296, 1989.
[2] Bein,W.W., Kamburowski, J. and Stallmann, M.F.M., "Optimal reduction of two-terminal directed acyclic graphs," *SIAM J. Computing*, 21, 1112-1129, 1992.
[3] Burt J.M., Garman M.B., "Conditional Monte Carlo: A simulation technique for stochastic network analysis", *Management Science*, 18, 3, 1971.
[4] Colby, A.H., Elmaghraby S.E., "On the complete reduction of directed acyclic graphs", *OR Report No. 197*, N.C. State Univ., Raleigh, NC., 1984.
[5] Dodin, B.M., "Determining the K most critical paths in PERT networks", *Operations Research*, 32, n4, 859-877, 1984.
[6] Dodin, B.M., "Approximating the distribution function in stochastic networks",*Computers and Operations Research*, 12, n3, 251-264,1985.
[7] Duffin, R, "Topology ofseries-parallel networks", *J. Math. Anal. Appl*., 10, pp. 303-318, 1965.
[8] Fisher, D.L., Saisi, D. & Goldstein, W.M., "Stochastic PERT networks: op diagrams, critical paths and the project completion time", *Computers and Operations Research*,12-5, 471–482, 1985.
[9] Harley, H.O., Wortham, A.W., "A statistical theory for PERT critical path analysis", *Management Science*, 12, n 10, 469-481, 1966.
[10] Kamburowski, J., Michael, D.J., Stallman, M.F.M., "Minimizing the complexity index of an activity network", *Networks*, 36, 47–52, 2000.
[11] Kleindorfer G.B., "Bounding distributions for stochastic logic", *Operations Research*, 19, 7, 1586-1601, 1971.
[12] Kleindorfer, G.B., "Bounding distributions for a stochastic acyclic network", *Journal of Operations Research*, 19-7, 1586–1601, 1977.
[13] Martin, J.J., "Distribution of the time through a directed acyclic network", *Operational Research*, 13, 46-66, 1965.
[14] Ringer, L.J., "Numerical operators for statistical PERT critical path analysis", *Management Science*, 16, 136-143, 1969.
[15] Robillard, P., Trahan, M., "The completion time of PERT networks", *Operations Research*. 25, 15-29, 1977.
[16] Sahner R.A., Trivedi K.S., "Performance and reliability analysis using directed acyclic graphs", *IEEE Transactions on Software Engineering*, 13, 1105-1114, 1987.
[17] Splde, J.G., "Bounds for the distribution function of network variables", *First Symposium of Operations Research*, 3, 113-123, 1977.
[18] Van, Slyke, R.M., "Monte Carlo methods and the PERT problem", *Operations Research*, 11, 839-861, 1963.
[19] Yao, M.J., Chu, W.M., Tseng, T.Y., "A Label-Correcting Tracing Algorithm for the Approximation of the Probability Distribution Function of the Project Completion Time", *Journal of Chinese Institute of Industrial Engineers*, 24-2, p.153~165, March 2007.
[20] Yao, M.J., Chu, W.M., "A New Approximation Algorithm for Obtaining the Probability Distribution Function of the Project Completion Time", *Computers and Mathematics with Applications*., 54-2, p. 282~295, July 2007.

Weng Ming Chu was born in Taiwan, Jul. 1962. He received the Ph.D. degree in the Department of Industrial Engineering and Enterprise Information, Tunghai University, Taiwan. Now he is interested in project management, supply chain management, inventory control, and telecommunication networks.

Dr. Chu used to be an officer of Airforce for airplane radar training, Now, he has retired and to be an assistant professor at Hsing Kuo University of Management in Tainan.