Available online at www.ijpe-online.com

International Journal of Performability Engineering

Vol. 13, No. 6, October 2017, pp. 796-806
DOI: 10.23940/ijpe.17.06.p2.796806

Analysis of Large Fault Trees based on Simplified BDD Algorithm

Wei Liu[a], Yong Zhou[a,*], Hongmei Xie[a], Zhengxian Wei[b]

[a]*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China*
[b]*System Engineering Research Institute, China State Shipbuilding Corporation, Beijing 100036, China*

Abstract

A simplified BDD analysis method (SimBDD) is proposed to solve the problem of temporal and spatial explosion and low computational efficiency of getting the fault tree's minimum cut sets in BDD method. On the basis of using SimBDD to get the minimum cutting sets, the failure paths of the fault tree are obtained by the hierarchical processing method. Tests are taken on large fault trees with this method. The results demonstrate that the method is efficient.

Keywords: fault tree; SimBDD; minimum cut set; failure path; hierarchical processing method

(Submitted on July 25, 2017; Revised on August 30, 2017; Accepted on September 15, 2017)

(This paper was presented at the Third International Symposium on System and Software Reliability.)

1. Introduction

The fault propagation paths that start from basic events and end at the top event is called failure path of the fault tree. In the process of analyzing the reliability of a system, if we want to know the impact of the subsystem on the whole system and the failure of the subsystem, the failure paths of the system should be obtained. While using the traditional descending method to get failure paths, the time exponentially increases with the number of basic events in general and the redundant paths cannot be effectively eliminated. In this paper, a method is proposed to find failure paths with minimum cut sets of a tree.

Binary decision diagram is one of the most effective methods to analyze fault tree, which is widely used in reliability analysis and system diagnosis. Lee [6] and Akers [1] introduced a binary decision diagram (BDD) by representing Boolean functions as decision graphs. Rauzy [9,10] initiated the BDD application to the reliability analysis. However, in the analysis of large-scale fault trees, the use of BDD method will have a problem of space explosion. The size of the BDD structure depends largely on the ordering of the variables. Andrews [2,11] gave a way to find the appropriate variable sort. Deng and Wang [5] introduced a binary decision diagram (BDD) based on modularization for fault tree analysis to deal with large-scale fault trees.

This paper presents a simplified form of the BDD (SimBDD) for solving the above problems. The SimBDD method is based on the BDD connection method [3,4,7]. In the SimBDD structure, the number of SimBDD nodes is linearly related to the number of leaf nodes in the fault tree, so the storage of the SimBDD structure requires only a small amount of space. Through the connection of the nodes, this method can quickly get the corresponding SimBDD of a fault tree.

After obtaining the minimum cut sets of a fault tree, you can get the smallest basic events combination that causes the top event to occur, but you cannot know how the failure propagates from the basic events to the top event. The paths from the basic events to the top event are called the fault path of the fault tree. On the basis of using SimBDD to find the minimum cut sets, this paper presents a hierarchical processing method to find the failure paths of a fault tree.

* Corresponding author.
E-mail address: zhouyong@nuaa.edu.cn.

The structure of this paper is as follows. The next section describes the normal BDD algorithm. Section 3 provides the details of SimBDD. Section 4 explains the hierarchical processing method, which is efficient to get the failure paths. Benchmark tests were performed to show the efficiency of the SimBDD algorithm and the hierarchical processing method for large fault trees and the results are described in Section 5. The conclusions of the study are provided in Section 6.

2. Binary decision diagram

BDD is a directed acyclic graph describing Shannon decomposition, and a data structure for representing Boolean functions [9]. Each variable node has two branches. The "1" branch represents the corresponding event occurs and the "0" branch indicates that the event does not occur. NodeOne indicates that the root event failed and nodeZero means that the root event is in a safe state. All nodes on each path start at root node and terminal on nodeOne form a cut set of a system.

Rauzy [10] proposed an algebraic framework based on Shannon decomposition to calculate the minimum cut set of the fault tree. The Shannon decomposition formula can be expressed as

$$f(X) = X_i f_1 + \overline{X_i} f_0 \tag{1}$$

where $f_1 = f_1(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$, $f_0 = f_0(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$

The corresponding If-Then-Else (ite) structure of the formula (1) is $ite\left(x_i, f_1, f_0\right)$. It can be intuitively expressed as: a binary tree structure whose parent node is x_i, with f_0 and f_1 as its left and right branches. It describes that the event x_i is passed to the child nodes f_0 and f_1 in two different states, respectively.

An important feature of ite is that the operators can be nested to definite. With this feature, the state of the ite child node can continue to be passed to the next child node until terminal nodes 1 and 0 are encountered. It is because of this nesting feature of ite that the ite operator can fully describe the fault tree failure mode and propagation path.

If $index(x_1) > index(x_2)$, then

For "and" gate operation:

$$ite(x_1, G_1, G_2) \bullet ite(x_1, H_1, H_2) = ite(x_1, G_1 \bullet H_1, G_2 \bullet H_2) \tag{2}$$

$$ite(x_1, G_1, G_2) \bullet ite(x_2, H_1, H_2) = ite(x_1, G_1 \bullet h, G_2 \bullet h) \tag{3}$$

For "or" gate operation:

$$ite(x_1, G_1, G_2) + ite(x_1, H_1, H_2) = ite(x_1, G_1 + H_1, G_2 + H_2) \tag{4}$$

$$ite(x_1, G_1, G_2) + ite(x_2, H_1, H_2) = ite(x_1, G_1 + h, G_2 + h) \tag{5}$$

where $h = ite(x_2, H_1, H_2)$

The fault tree in Figure 1 can be transformed into a BDD structure (Figure 2) using Equations (2-5). In Figure 2 $\{a,b\}$, $\{a,\overline{b},c\}$, $\{a,\overline{b},\overline{c},d\}$, $\{\overline{a},\overline{b},\overline{c},d,e\}$, $\{\overline{a},\overline{b},c,d\}$, $\{\overline{a},b,\overline{c},d,e\}$ and $\{\overline{a},b,c,d\}$ are seven paths which start at the root node and terminate at node 1. It can be simplified as $\{a,b\}$, $\{a,\overline{b},c\}$, $\{a,\overline{b},c,d\}$, $\{\overline{a},\overline{c},d,e\}$ and $\{\overline{a},c,d\}$. So the minimum cut sets of the FT are $\{a,b\}$, $\{a,c\}$, $\{a,d\}$, $\{c,d\}$ and $\{d,e\}$. We can get the probability of the top event using formula (6):

$$P_T = P_a P_b + P_a(1-P_b)P_c + P_a(1-P_b)(1-P_c)P_d + (1-P_a)(1-P_c)P_d P_e + (1-P_a)P_c P_d \tag{6}$$

3. Simplified BDD

3.1. Structure of SimBDD

Figure 3 presents the general SimBDD structure. Its corresponding FT is T. We call the corresponding SimBDD of T as BDD_T. The nodes in rectangle are omitted in Figure 3. We name the head of BDD_T as T_head.

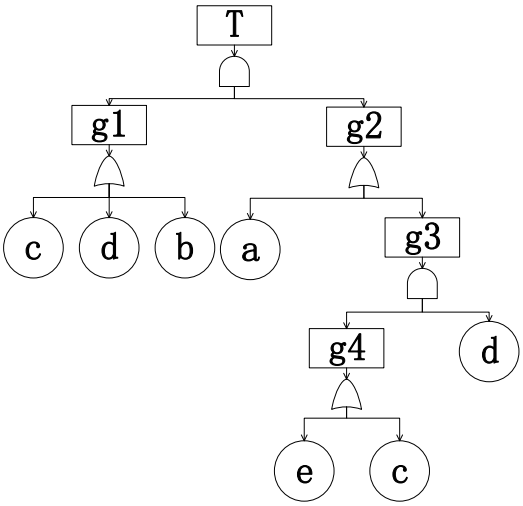


Figure 1. Fault tree T

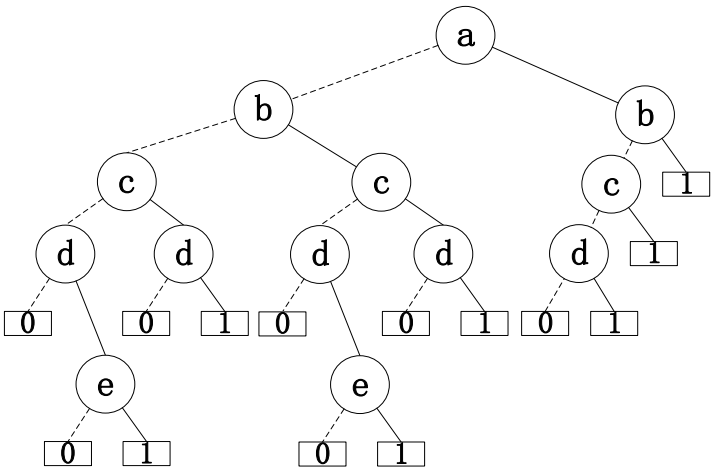


Figure 2. The BDD of fault tree T under the order a< b< c< d< e

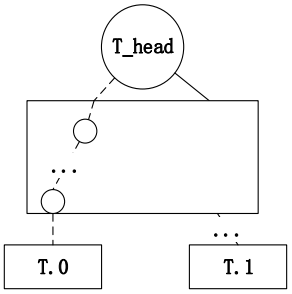


Figure 3. Structure of SimBDD

Definition: leftmost node

Each leaf node in the SimBDD is connected to a different nodeZero. We call the edge that represents that the node occurs as edgeOne and the edge which represents that the node does not occur as edgeZero. Assume that the edgeZero of each node in a SimBDD is on the left side of its edgeOne, the leftmost node of a SimBDD represents the zero node that the leftmost path connects to.

The structure of SimBDD can be described as:

```
Struct SimBDD{
   String node                 //name of the node
   SimBDD zeroLink             //linked node when the node not occur
   SimBDD oneLink              //linked node when the node occur
   List<SimBDD> parents //parent nodes of the node
   SimBDD zeroNode             //leftmost node of the SimBDD
   SimBDD oneNode              //nodeOne of the SimBDD
}
```

In Figure 3, T.1 is the terminal nodeOne shared by all the nodes in BDD_T. That is, all paths that lead T to be true will eventually connect to T.1. T.0 in Figure 3 represents the leftmost nodeZero of BDD_T. We can find that the leftmost path of SimBDD consists of edgeZeros of nodes.

3.2. Operations of SimBDD

Suppose there are two SimBDD, BDD_T1 and BDD_T2, and the head nodes are T1_head and T2_head, respectively.

As shown in Figure 4(a), when $T = T1 \bullet T2$, we first connect the parent nodes of T1_head.oneNode to T2_head with their edgeOne and add these parent nodes to T2_head.parents. Then, we assign T2_head.oneNode to T1_head.oneNode. Finally, the SimBDD whose head is T1_head is the one that T corresponds.

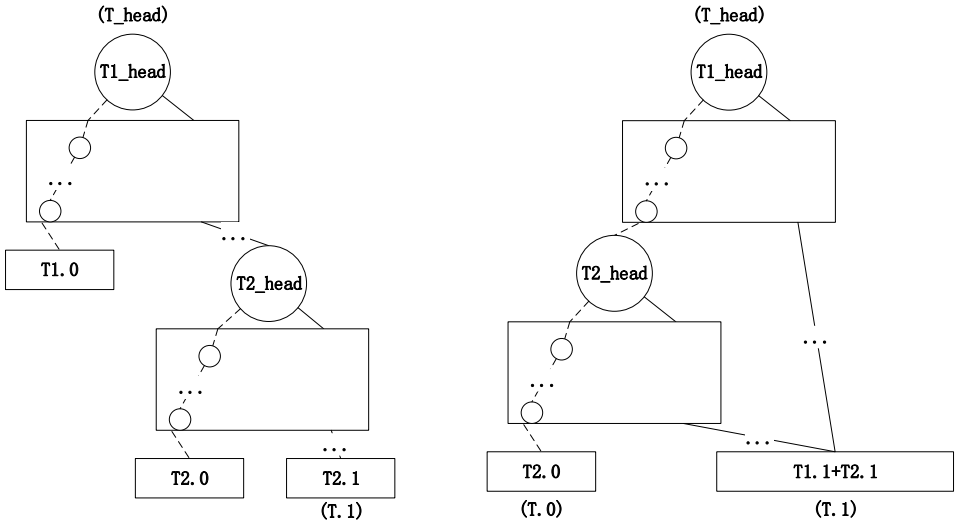
The procedure of getting $T = T1 \bullet T2$ the SimBDD is as follows:

```
/* "and" operation of SimBDD*/
SimBDD_AND(T1_head,T2_head)
   parents ← T1_head.oneNode.parents
   for each parent in parents
      parent.oneLink ← T2_head
      T2_head.parents ← add(parent)
   T1_head.oneNode ← T2_head.oneNode
return T1_head
```

As Figure 4(b) shows, when $T = T1 + T2$, we first connect the parent node of T1_head.zeroNode to T2_head with their edgeZero and add the parent node to T2_head.parents. Then, we merge T1_head.oneNode and T2_head.oneNode. Finally, the SimBDD whose head is T1_head is the one that T corresponds to.

The procedure of getting the SimBDD of $T = T1 + T2$ is as follows:

```
/* "or" operation of SimBDD*/
SimBDD_OR(T1_head,T2_head)
   parents ← T1 _head.zeroNode.parents
   for each parent in parents
      parent.zeroLink ← T2_head
      T2_head.parents ← add(parent)
   parents ← T2 _head.oneNode.parents
   for each parent in parents
      parent.oneLink ← T1_head.oneNode
      T1_head.oneNode.parents ← add(parent)
   T1_head.zeroNode ← T2_head.zeroNode
return T1_head
```



(a) (b)

Figure 4. (a) "and" operation of SimBDD (b) "or" operation of SimBDD

3.3. Convert from FT to SimBDD

In this paper, FT is converted to SimBDD in depth-first order.

The following pseudocode shows the procedure of converting from FT to SimBDD, where T.root represents the root node of the fault tree T, ftnode represents a fault tree node, ftnode.name represents the name of fnode, ftnode.child represents the set of ftnode's child nodes, ftnode.child [i] represents the (i-1)th subnode of ftnode, ftnode.calc represents the corresponding gate of intermediate event node ftnode. The function TransToSimBDD uses the root node of the FT as its input and returns a SimBDD that corresponds to the FT.

```
/* The procedure of transforming from FT to SimBDD */
TransToSimBDD(T.root)
TransToSimBDD(ftnode)
   theBDD ← create a new SimBDD node
   if ftnode.childs is null
      theBDD.node ← ftnode.name
      theBDD.zeroLink ← theBDD.zeroNode
      theBDD.oneLink ← theBDD.zeroNode
      theBDD.zeroNode.parents ← add(theBDD)
      theBDD.oneNode ← add(theBDD)
   else
      for i ← 0 to the length of ftnode.childs
         child ← ftnode.childs[i]
         childBDD ← TransToSimBDD(child)
         if i=0
            theBDD ← childBDD
         else
            if ftnode.calc="AND"
               theBDD ← SimBDD_AND(theBDD,childBDD)
            else if ftnode.calc="OR"
               theBDD ← SimBDD_OR(theBDD,childBDD)
return theBDD
```

3.4. Method of getting MCSs

Definition: positive node
If edgeOne of a node is in a path, then the node is called the positive node of the path.

Definition: A ⇒ B, A ⇒ B ⇒ C

A ⇒ B represents a set of all paths from SimBDD node A to SimBDD node B. A ⇒ B ⇒ C represents a set of all the paths from SimBDD node A to SimBDD node B and then to SimBDD node C.

Definition: {A ⇒ B}, {A ⇒ B ⇒ C}

A different positive node on each path in A⇒B can combine a set, {A⇒B} represents a collection of all these sets. Distinct positive nodes on each path in A ⇒ B ⇒ C can make up a set, and {A ⇒ B ⇒ C} represents a collection consist of all these sets.

T1 and T2 are any two fault trees, and their corresponding SimBDDs are BDD_T1 and BDD_T2. Head nodes of BDD_T1 and BDD_T2 are T1_head and T2_head. Let T = T1 <op> T2 (<op> denote "+" or "•"). The corresponding SimBDD of T is BDD_T, and its head node is T_head.

For a path p, lets n(p) denote the number of positive nodes on p. let p [1] p [2] ... p [n (p)] denote all positive nodes on p.

1) When $T = T1 \bullet T2$, change the connection of T1_head.oneNode's parent nodes to T2_head with its edgeOne and point T1_head.oneNode to T2_head.oneNode. Thus, all paths to nodeOne in BDD_T are from T1_head to T2_head.oneNode.

2) When $T = T1 + T2$, connect the parent node of T1_head.zeroNode to T2_head with its edgeZero and merge T1_head.oneNode and T2_head.oneNode into T1_head.oneNode. Thus, all paths to nodeOne in BDD_T are either from T1_head to T1_head.oneNode or from T1_head to T2_head.oneNode.

Lemma 1: If A is a cut set of T1 for any $A \in \{T1\_head \Rightarrow T1\_head.oneNode\}$ and B is a cut set of T2 for any $B \in \{T2\_head \Rightarrow T2\_head.oneNode\}$, then C is a cut set of T for any $C \in \{T\_head \Rightarrow T\_head.oneNode\}$.

Proof **of Lemma 1:**

1) When $T = T1 \bullet T2$:

$$C \in \{T\_head \Rightarrow T\_head.oneNode\}$$
$$= \{T1\_head \Rightarrow T2\_head \Rightarrow T2\_head.oneNode\}$$
$$= \{Z \mid Z = A \cup B, A \in \{T1\_head \Rightarrow T1\_head.oneNode\}, B \in \{T2\_head \Rightarrow T2\_head.oneNode\}\}$$

Because A is a cut set of T1, $A \in \{M \mid T1 = 1 \ when \ a = 1 \ for \ every \ a \in M\}$.

Similarly, $B \in \{N \mid T2 = 1 \ when \ b = 1 \ for \ every \ b \in N\}$. Then:

$$C \in \{Z \mid Z = A \cup B, A \in \{M \mid T1 = 1 \ when \ a = 1 \ for \ every \ a \in M\}, B \in \{N \mid T2 = 1 \ when \ b = 1 \ for \ every \ b \in N\}\}$$
$$= \{Z = M \cup N \mid T1 = 1 \ when \ a = 1 \ for \ every \ a \in M, T2 = 1 \ when \ b = 1 \ for \ every \ b \in N\}$$
$$= \{Z \mid T1 = 1 \ and \ T2 = 1 \ when \ c = 1 \ for \ every \ c \in Z\}$$
$$= \{Z \mid T = 1 \ when \ c = 1 \ for \ every \ c \in Z\}$$

As a result, C is a cut set of T for any $C \in \{T\_head \Rightarrow T\_head.oneNode\}$.

2) When $T = T1 + T2$. Since there is no positive node on the path from T1_head to T2_head,

$$\{T2\_head \Rightarrow T2\_head.oneNode\}$$
$$= \{T1\_head \Rightarrow T2\_head \Rightarrow T2\_head.oneNode\}$$
$$= \{T1\_head \Rightarrow T2\_head.oneNode\}$$

Then,

$C \in \{T\_head \Rightarrow T\_head.oneNode\}$
$= \{T1\_head \Rightarrow T1\_head.oneNode\} \bigcup \{T2\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T1\_head.oneNode\} \bigcup \{T1\_head \Rightarrow T2\_head.oneNode\}$
$= \{M \mid T1 = 1 \text{ when } a = 1 \text{ for every } a \in M\} \bigcup \{N \mid T2 = 1 \text{ when } b = 1 \text{ for every } b \in N\}$
$= \{Z \mid T1 = 1 \text{ or } T2 = 1 \text{ when } c = 1 \text{ for every } c \in Z\}$
$= \{Z \mid T = 1 \text{ when } c = 1 \text{ for every } c \in Z\}$

As a result, C is a cut set of T for any $C \in \{T\_head \Rightarrow T\_head.oneNode\}$.

Above all, Lemma 1 is correct.

.

Lemma 2: If $\{T1\_head \Rightarrow T1\_head.oneNode\}$ contains all the MCSs of T1 and $\{T2\_head \Rightarrow T2\_head.oneNode\}$ contains all the MCSs of T2, then $\{T\_head \Rightarrow T\_head.oneNode\}$ contains all the MCSs of T.

Proof of Lemma 2:

If A is a MCS of T1, $A \in \{M \mid T1 = 1 \text{ when } a = 1 \text{ for every } a \in M\}$. If B is a MCS of T1, $B \in \{N \mid T2 = 1 \text{ when } b = 1 \text{ for every } b \in N\}$.

1) To make $T = T1 \bullet T2 = 1$, we should set $T1 = 1$ and $T2 = 1$. That is, each cut set of T contains all elements of a MCS of T1 and all elements of a MCS of T2. As a result, each MCS of T is a union of a MCS of T1 and a MCS of T2. Let A be a MCS of T1, B be a MCS of T2 and C be a MCS of T.

$C \in \{Z \mid Z = A \bigcup B, A \in \{T1\_head \Rightarrow T1\_head.oneNode\}, B \in \{T2\_head \Rightarrow T2\_head.oneNode\}\}$
$= \{T1\_head \Rightarrow T2\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T2\_head.oneNode\}$
$= \{T\_head \Rightarrow T\_head.oneNode\}$

As a result, $\{T\_head \Rightarrow T\_head.oneNode\}$ contains all the MCSs of T.

2) To make $T = T1 + T2 = 1$, we should set $T1 = 1$ or $T2 = 1$. That is, each cut set of T should contains a MCS of T1 or a MCS of T2. As a result, each MCS of T is a MCS of T1 or a MCS of T2. Let C be a MCS of T.

Since there is no positive node on the path from T1_head to T2_head,

$\{T2\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T2\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T2\_head.oneNode\}$

Then,

$C \in \{T\_head \Rightarrow T\_head.oneNode\}$
$= \{T1\_head \Rightarrow T1\_head.oneNode\} \bigcup \{T2\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T1\_head.oneNode\} \bigcup \{T1\_head \Rightarrow T2\_head.oneNode\}$
$= \{T1\_head \Rightarrow T1\_head.oneNode\}$
$= \{T\_head \Rightarrow T\_head.oneNode\}$

As a result, $\{T\_head \Rightarrow T\_head.oneNode\}$ contains all the MCSs of T.

Above all, Lemma 2 is correct.

Proposition 1: For a fault tree, if we convert it to a SimBDD, then:

1) Each set of positive nodes of every path in SimBDD start from the head node and terminate at nodeOne is a cut set of the corresponding FT.

2) All cut sets made up by the above method contain all the MCSs of the corresponding FT.

Proof of Proposition 1:

If T is a fault tree, then T is in one of the three cases: 1) T is a basic event, 2) $T = T1 \bullet T2$, 3) $T = T1 + T2$. (T1 and T2 are two FTs)

When T is a basic event, then T satisfies Proposition 1 obviously.

When $T = T1 \bullet T2$ or $T = T1 + T2$, T1 and T2 satisfy Proposition 1 by the inductive hypothesis. We can get a conclusion from Lemma 1 and Lemma 2 that T satisfies Proposition 1.

Thus, the fault tree T satisfies the Proposition 1.

Using the conclusion of proposition 1, we get the method of finding cut sets of a fault tree using SimBDD:

For the fault tree T, the corresponding SimBDD is BDD_T, and the head node of the SimBDD is T_head. As long as all paths from T_head to T_head.oneNode are found, the positive nodes on each path form a cut set of T.

The pseudocode of getting the cut sets of a fault tree is as follows. The input of the function GetCSS is the head node of a FT's corresponding SimBDD and the output of the function is the cut sets of the fault tree.

```
/* Get cut sets*/
GetCSS(T_head)
   list ← create a empty list
   CSS ← create a empty list
   FindCS(T_head,list,CSS)
   return CSS
/*Traverse to find all cut sets*/
FindCS(bddNode,list1,CSS)
   if bddNode=T_head.oneNode
      CSS ← add(list1)
   else
      if bddNode.zeroLink is not null or a nodeZero
         list2 ← copy(list1)
         FindCS(bddNode.zeroLink,list2,CSS)
      if bddNode.oneLink is not null
         if list1 do not contains bddNode.node
            list1 ← add(bddNode.node)
          FindCS(bddNode.oneLink,list1,CSS)
```

From the Proposition 1, the cut sets obtained by the above method contain all the minimum cut sets of the fault tree. After all the cut sets are obtained, delete all cut sets that contain other cut sets. After the above procedure is completed, the remaining sets are the minimum cut sets of the fault tree.

3.5. Use Case

The process of converting the fault tree in Figure 1 into a SimBDD is shown in Figure 5 and the other nodeZeros except the leftmost node are hidden in the figure. In the order of depth-first, we get the corresponding SimBDDs of c+d (Figure 5(a)), g2 (Figure 5(b)), g6 (Figure 5(c)), g5 (Figure 5(d)), g3 (Figure 5(e)) and T (Figure 5(f)).

In Figure 5(f), the paths from the head node to nodeOne are $\{c,a\}$, $\{c,\bar{a},e,d\}$, $\{c,\bar{a},\bar{e},c,d\}$, $\{\bar{c},d,a\}$, $\{\bar{c},d,\bar{a},e,d\}$, $\{\bar{c},d,\bar{a},\bar{e},c,d\}$, $\{\bar{c},\bar{d},b,a\}$, $\{\bar{c},\bar{d},b,\bar{a},e,d\}$, and $\{\bar{c},\bar{d},b,\bar{a},\bar{e},c,d\}$. Positive nodes on these paths are $\{c,a\}$, $\{c,e,d\}$, $\{c,d\}$, $\{d,a\}$, $\{d,e,d\}$, $\{d,c,d\}$, $\{b,a\}$, $\{b,e,d\}$ and $\{b,c,d\}$ respectively after remove the duplicate nodes. After delete all cut sets which contain other cut sets, we get the MCSs of the FT which are $\{c,a\}$, $\{c,d\}$, $\{d,a\}$, $\{d,c\}$ and $\{b,a\}$.

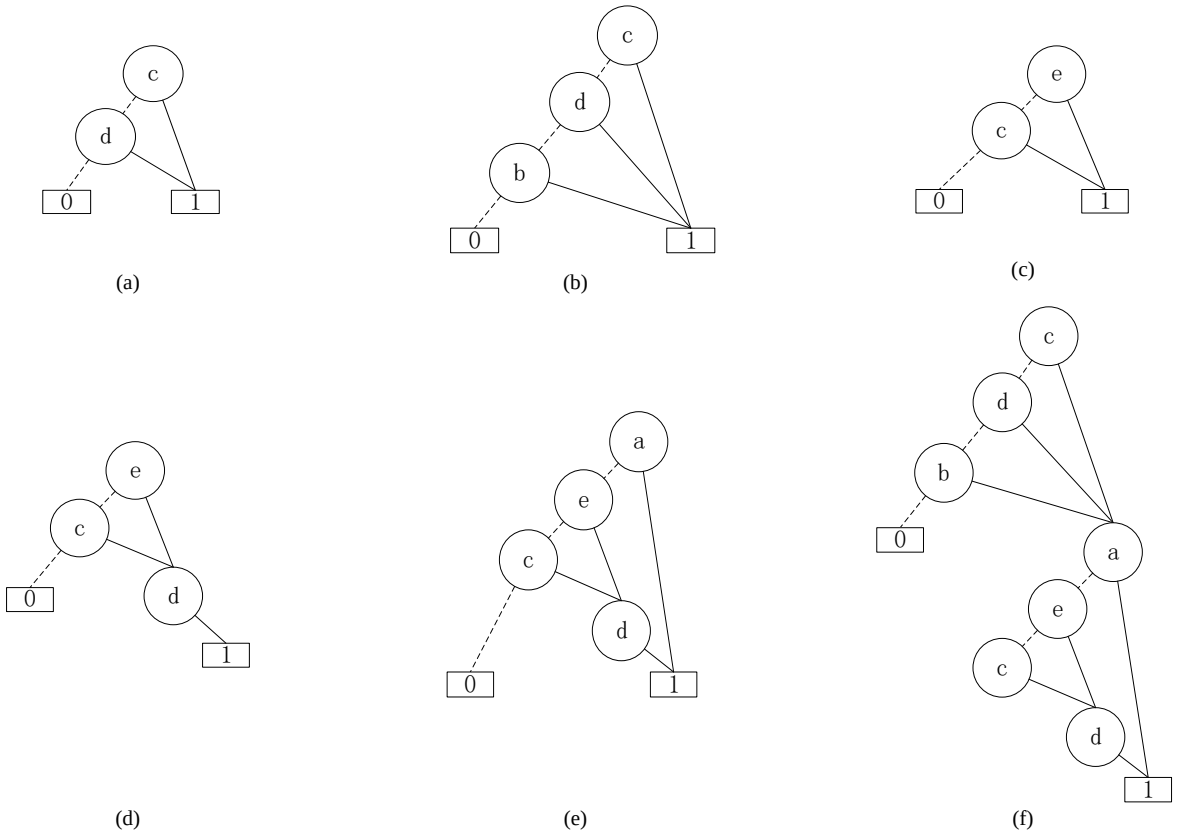


Figure 5. (a) The SimBDD of c+d (b) The SimBDD of g1 (c) T

4. Hierarchical processing method

After using the SimBDD method to obtain the minimum cut sets of the fault tree, we get the smallest basic events combination that causes the top event to occur. For each minimum cut set, the failure of all events in the cut set will cause the top event to fail, but it is not possible to know how these failure propagate from the bottom to the top, resulting in failure of the top event.

4.1. Procedure of getting failure paths

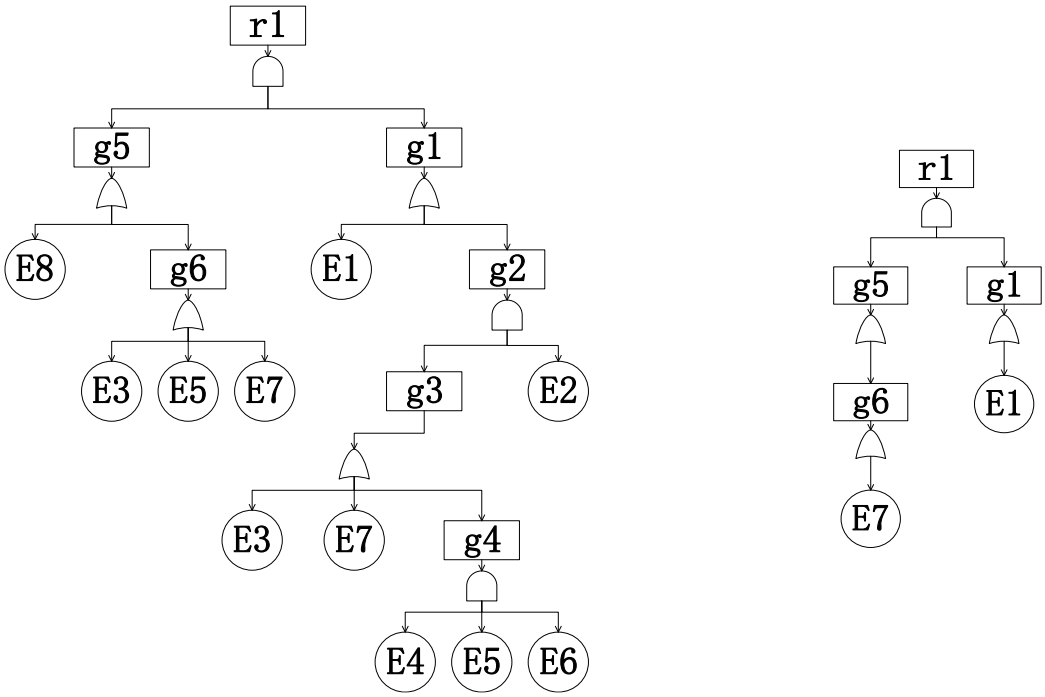
In order to get the failure path of a fault tree, we should:

- Step.1. Traverse the fault tree and get the level of each fault tree node. The level of the top event is 0, and the level of the child node is the level of the parent node plus one. After traversing all nodes, we get the maximum level of the fault tree nodes, assuming that it is MaxLevel.
- Step.2. Utilizing SimBDD to obtain the minimum cut sets for the fault tree, we can find the corresponding failure path for each minimum cut set. For a minimum cut set, a failure path list named FailureLists is created, whose size is MaxLevel + 1 and each layer of whom retains an empty list. After the initialization of the failure path list is complete, the basic events corresponding to each element of the minimum cut set is added to the corresponding hierarchy.
- Step.3. Process from the layer MaxLevel to the layer 0 of the failure path list. When dealing with the list of the i-th layer, the nodes in the list are traversed.
  - Situation 1. If the parent gate of a node is an OR gate and the parent node of the node is not in the i-1 layer, the parent node of the node is added to the i-1 layer in the list.
  - Situation 2. If the parent gate of the node is an AND gate and all child nodes of the parent node are in the i-th list, the parent node of the node is added to the i-1 list and skips the traversing for all childs of the parent node. If the parent gate of the node is an AND gate but the parent node has a child that is not in the i-th list, all descendants of the parent child node are removed from the failure path list.
- Step.4. Construct a failure path tree based on the parent-child relationship between adjacent nodes in the failure path list.

4.2. Use case

In order to obtain the failure path list corresponding to the cut set {E1, E7} in Figure 6(a), Table 1 shows the complete process from initializing the failure path list to dealing with the list of layer 1. Starting from the second column, each column in the table indicates the status of the failure path list after finishing an action. Step 1 is an initialization process for the failure path list. In step 2, E7 is found in the fourth list and its parent g3 is added to the third list. Because E7 in the third list has a parent g6 and its parent gate is OR gate, g6 is added to the second list in step 3. The parent gate of g3 is AND gate and g3's brother E2 is not in the third list, so delete g3 and its descendants from the path list in step 4. Step 5 gets g6's parent g5 and E1's parent g1 and adds them to the first list. The parent of g5 and g1 is r1, add it to the zeroth list.

After the above steps, a failure path tree can be constructed based on the parent-child relationship between adjacent nodes in the failure path list. According to the above steps, the failure path list corresponding to the minimum cut set {E1, E7} is [[r1], [g5, g1], [g6, E1], [E7], [], []]. According to the parent-child relationship between adjacent nodes, a failure path tree is constructed. The failure path tree in Figure 6(b) clearly shows how the fault propagates from E1 and E7 to the top event r1.



(a) (b)

Figure 6. (a) Fault tree r1 (b) The failure path tree of r1's cut set {E1, E7}

Table 1. Procedure of getting a failure path

	Process					
Level	Init	Deal with the fourth level	Deal with E7 in the third level	Deal with g3 in the third level	Deal with the second level	Deal with the first level
0						r1
1					g5,g1	g5,g1
2	E1	E1	E1,g6	E1,g6	E1,g6	E1,g6
3	E7	E7,g3	E7,g3	E7	E7	E7
4	E7	E7	E7			
5						

5. Hierarchical processing method

We have tested open source tool OpenFTA and the SimBDD algorithm on six fault trees from a benchmark set [8] and get the time of them to obtain the MCSs. The cost of time to obtain the failure paths of the six fault trees with the Hierarchical Processing Method (HPM) is also gotten and the results are given in table 2.

Table 2 lists the time to find the minimum cut set through OpenFTA and SimBDD. The size in Table 2 is the total number of gates and basic events in FT. In general, we can get the correct MCSs in both ways. Comparing the time of the two methods, it is found that obtaining the MCS through the SimBDD algorithm is much faster than the OpenFTA.

For most large-scale fault trees in the benchmark set, the time of getting failure paths with hierarchical processing method is less than one second. The time is a lot less compared to the time of getting MCSs. As a result, it is efficient to be applied to get the failure paths.

Table 2. Test results

FT	Size	Basic Events	Solutions	Time(s)		Time(ms)
				OpenFTA	SimBDD	HPM
Das9201	204	122	14217	197.2	34.5	844.9
Isp9606	130	89	1776	3.6	0.1	39.3
Das9204	83	53	16704	320.7	76.4	795.5
Das9205	71	51	17280	120.4	14.6	526.4
Das9206	233	121	19518	735.4	54.0	1243.6
ftr10	269	175	305	<0.1	<0.1	9.0
isp9603	186	91	3434	43.2	7.2	284.8

6. Conclusions

This article has described a simplified BDD algorithm to get a fault tree's MCSs and hierarchical processing method to obtain failure paths of a fault tree. The SimBDD algorithm is much faster to get the MCSs with less memory consumption for large fault trees. With the MCSs obtained by SimBDD, the hierarchical processing method can be used to get the failure paths. The results in above table show that this method is efficient.

Acknowledgements

This paper is funded by National Key R&D Program of China (2016YFB1000802). And the paper is supported by the Fundamental Research Funds for the Central Universities (NS2016088) and China's 13th plan of key basic research project (JCKY2016206B001 & JCKY2014206C002).

References

1. S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509-516, 1994
2. J. D. Andrews and L. M. Bartlett, "Efficient Basic Event Orderings for Binary Decision Diagrams," in *Reliability and Maintainability Symposium*, pp. 61-68, 1998
3. K. S. Brace, R. L. Rudell, R. E. Bryant, "Efficient Implementation of a BDD Package," in *IEEE Design Automation Conference*, pp. 40-50, 1991
4. R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Computer Society*, vol. 35, no. 8, pp. 677-691, 1986
5. Y. Deng, H. Wang and B. Guo, "BDD Algorithms Based on Modularization for Fault Tree Analysis," *Progress in Nuclear Energy*, vol. 85, pp. 192-199, 2015
6. C. Y. Lee, "Representation of Switching Circuits by Binary-Decision Programs," *Bell System Technical Journal*, vol. 38, no. 4, pp. 985-999, 1959
7. R. Prescott, "System Failure Modelling Using Binary Decision Diagrams," *Loughborough University*, 2007
8. "A Benchmark of Boolean Formulae," A. Rauzy, Available at http://iml.univ-mrs.fr/~arauzy/aralia/benchmark.html, Last accessed on July 20, 2017
9. A. Rauzy, "A Brief Introduction to Binary Decision Diagrams," *Journal Européen Des Systèmes Automatisés Hermes*, vol. 4, no. 4, pp. 206-207, 1996
10. A. Rauzy, "New Algorithms for Fault Trees Analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203-211, 1993
11. R. M. Sinnamon and J. D. Andrews, "Improved Efficiency in Qualitative Fault Tree Analysis," *Quality & Reliability Engineering International*, vol. 13, no. 5, pp. 293-298, 1997