A Cut-Based Algorithm for Reliability Analysis of Terminal-Pair Network Using OBDD

Yung-Ruei Chang[†] , Hung-Yau Lin, Ing-Yi Chen[‡] , and Sy-Yen Kuo
Department of Electrical Engineering, National Taiwan University
sykuo@cc.ee.ntu.edu.tw

Abstract

In this paper, we propose an algorithm to construct the Ordered Binary Decision Diagram (OBDD) representing the cut function of a terminal-pair network. The algorithm recognizes isomorphic sub-problems and thus avoids redundant computations. The system reliability could be efficiently computed by the OBDD. Finally, we propose an approach to compute the importance measures for multiple components by traversing the OBDD only once. The correctness and the effectiveness of our approach are demonstrated by experiments on 30 benchmark networks. The experimental results on a 2-by-100 lattice network, which has 2^{99} paths or 10,000 cuts, show an impressive improvement compared to the previous works using the sum of disjoint products method that have exponential complexity. The CPU time of our method, including the calculation of not only the reliability but also the importance measures, for a 100-stage lattice network is only about 0.24 seconds. Thus, this approach is very helpful for the reliability and sensitivity analysis of large networks.

1. Introduction

In recent literature [1-7], the existing algorithms for computing the terminal-pair reliability of a network can be grouped into two categories according to their approaches. The algorithms in the first category require enumeration of all the simple paths. These methods deal with a complete set of non-disjoint events and bring a lot of computation complexity. The algorithms in the second category are based on decomposing a network into a disjoint event tree. Since all the paths of the tree are disjoint, the network reliability is the sum of the probabilities of these disjoint paths. However, identifying all the disjoint paths in a network is difficult and is a well-known NP-hard problem [4]. Hence, determining the terminal-pair reliability of a network is thus very time-consuming.

Most of previous works [1-7] focused on speeding up calculations by reducing the computation as much as possible. In general, these algorithms lack effective methods to enumerate all the simple paths and do not support efficient manipulation of Boolean algebra. Although these algorithms have been demonstrated with a reasonable efficiency on medium-scale networks, they have two inherent drawbacks. First, the sum of disjoint product forms is inefficient in dealing with larger Boolean functions. Second, the tree-based partition algorithm does not consider the merging of isomorphic sub-problems, so that redundant computations cannot be avoided.

Since 1986, when Bryant [8] first proposed the Ordered Binary Decision Diagram (OBDD) representations of Boolean functions and proved some fundamental results on OBDDs, lots of researches have been developed based on this structure and its variations. OBDD is based on the Shannon expansion and can be recognized as a graph-based set of disjoint products. Based on this property, Kuo [9] first proposed a feasible OBDD-based algorithm for computing the terminal-pair reliability of a large network. The main idea, which makes the approach in [9] much more efficient than previous works, is that the OBDD can be automatically constructed by converging isomorphic sub-problems during traversing the network from source to sink. Therefore, the reliability can be quickly derived from the OBDD. The method in [9] focused on the path set of a terminal-pair network. However, in this paper we try to use the cut method to construct the OBDD.

Moreover, identifying the critical components is also an important issue for the reliability analysis and the optimization design of network topology. In this paper, we will propose an OBDD-based algorithm to compute the importance measures of multiple components of a network during a single-pass traversal of the OBDD. The experimental results on a 2-by-100 lattice network show that our method is much better than previous algorithms, which have expo-

[†] Y.R. Chang is also with Institute of Nuclear Energy Research, Atomic Energy Council, Taiwan.
[‡] I.Y. Chen is with Department of Computer Science and Information Engineering of National Taipei University of Technology, Taipei, Taiwan.
Acknowledgment: This research was supported by the National Science Council, Taiwan, R.O.C. under Grant NSC 92-2623-7-002-004-NU.

nential complexity by using the sum of disjoint products. The proposed algorithm will be useful for the terminal-pair reliability and sensitivity analysis of large networks.

Section 2 illustrates the preliminaries of OBDD. A cut-based method for constructing the OBDD of a terminal-pair network is proposed in Section 3. This method avoids the redundant calculations on isomorphic sub-problems. Section 4 presents several OBDD-based algorithms to compute the reliability measures including the reliability (availability) and the Birnbaum importance measures of a terminal-pair network. Section 5 shows the experimental results on 30 benchmark networks. Section 6 gives the conclusions.

2. Preliminaries

OBDD [8] is based on a disjoint decomposition of a Boolean function called the Shannon expansion. Given a Boolean function $f(x_1,\cdots,x_n)$, then for any $i \in \{1,\cdots,n\}$; $\bar{x}_i \equiv \neg x_i = 1 - x_i$:

$$f = x_i \cdot f_{x_i=1} + \bar{x}_i \cdot f_{x_i=0} \tag{1}$$

In order to express the Shannon decomposition concisely, the if-then-else (*ite*) format [10][11] is defined as:

$$f = ite(x_i, f_{x_i=1}, f_{x_i=0})$$

The way that OBDDs are used to represent logical operations is simple. Let Boolean expressions *f* and *g* be:

$$f = ite(x_i, f_{x_i=1}, f_{x_i=0}) = ite(x_i, F_1, F_0)$$
$$g = ite(x_j, g_{x_j=1}, g_{x_j=0}) = ite(x_j, G_1, G_0)$$

A logic operation between *f* and *g* can be represented by OBDD manipulations as:

$$ite(x_i, F_1, F_0) \lozenge ite(x_j, G_1, G_0)$$
$$= \begin{cases} ite(x_i, F_1 \lozenge G_1, F_0 \lozenge G_0) & \text{ordering}(x_i) = \text{ordering}(x_j) \\ ite(x_i, F_1 \lozenge g, F_0 \lozenge g) & \text{ordering}(x_i) < \text{ordering}(x_j) \\ ite(x_j, f \lozenge G_1, f \lozenge G_0) & \text{ordering}(x_i) > \text{ordering}(x_j) \end{cases}$$

where $\lozenge$ represents a logic operation such as AND or OR. For more details on using the operations of OBDD, please refer to [8]. In practice, using logical operations on variables generates the OBDD.

A useful property of OBDD is that all the paths from the root to the leaves are mutually disjoint. If *f* represents the system reliability expression, based on the property of the disjoint decomposition of OBDD, the reliability (or availability) of the system can be recursively evaluated by (1).

$$\Pr\{f\} = \Pr\{x_i\} \cdot \Pr\{f_{x_i=1}\} + \Pr\{\bar{x}_i\} \cdot \Pr\{f_{x_i=0}\} \tag{2}$$

where $\Pr\{\cdot\}$ means $\Pr\{\cdot = 1\}$ for simplification. For example, if $\Pr\{x_i\}$ is the reliability R_i of component *i* and U_i is the unreliability of component *i*, then the system reliability *R* is

$$R = \Pr\{f\} = R_i R_{x_i=1} + U_i R_{x_i=0} = R_i R_{x_i=1} + (1 - R_i) R_{x_i=0} \tag{3}$$

where $R_{x_i=1}$ and $R_{x_i=0}$ represent $\Pr\{f_{x_i=1}\}$ and $\Pr\{f_{x_i=0}\}$ respectively. Similarly, the unreliability of a system can be calculated as:

$$U = \Pr\{g\} = U_i U_{\bar{x}_i=1} + R_i U_{\bar{x}_i=0} \tag{4}$$

where *g* is the system unreliability expression and the dual of f; i.e. $f(x_1, x_2, \cdots, x_n) \equiv 1 - g(1 - x_1, 1 - x_2, \cdots, 1 - x_n) \equiv 1 - g(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n)$, $U_{\bar{x}_i=1}$ and $U_{\bar{x}_i=0}$ represent $\Pr\{g_{\bar{x}_i=1}\}$ and $\Pr\{g_{\bar{x}_i=0}\}$ respectively. In this paper, we will focus on the unreliability expression *g* based on the cut-set method.

3. Constructing OBDD Based on Cut Method

This section presents a cut-based method to construct the OBDD that represents the cut function of a terminal-pair network. The OBDD is automatically constructed with the convergence of isomorphic sub-problems during traversing the network from the source to the target in edge reduction diagrams. Therefore, our approach avoids redundant calculations and reduces the execution time significantly.

Notation

G_k	a graph representing a terminal-pair network in the edge reduction diagram.
C_k	the cut function corresponding to G_k.
s_k, t_k	the [source, target] of G_k.
E_k	the index set of edges connected to s_k in G_k.
e_i	an edge connected to s_k in G_k, $i \in E_k$.
G_{k*i}	the sub-graph of G_k obtained by deleting all edges connected to s_k and moving s_k to the node where e_i is connected.
RN(G_{k*i})	eliminating the redundant nodes from G_{k*i}; the redundant node is the node which has only one edge connected with.
$x_i, \bar{x}_i$	event variables, $[x_i = 1, \bar{x}_i = 1]$ represents the edge e_i to be [functional, failed].
$\wedge, \vee$	Boolean [AND, OR]: [conjunction, disjunction].

A terminal-pair network means a network with a given source vertex and a given target vertex. For a terminal-pair network, the graph G_k in the edge reduction diagram is composed of the union of the sub-graph G_{k*i} and the edge e_i connected to the source. Therefore,

$$G_k = \bigcup_{i \in E_k} \left(e_i \, G_{k*i} \right) \tag{5}$$

To avoid redundant calculation, except s_k and t_k in G_{k*i}, some redundant nodes which are connected with only one edge can be eliminated. For example, G_1 is equivalent to RN(G_{0*1}) and G_8 is equivalent to RN(G_{5*6}) in Fig. 1. Since only redundant nodes are eliminated, the cuts of RN(G_{k*i}) is equivalent to the cuts of the original graph G_{k*i}. Fig. 1 shows the complete edge reduction diagram of an example terminal-pair network G_0. Each rectangle G_k represents a network that is constructed from the parent graph by deleting all edges connected to the source node and eliminating the redundant nodes. Each G_k has a corresponding cut function C_k that is composed of a set of cuts. Therefore, C_0 represents the cut function of the example terminal-pair network G_0. We need to construct an OBDD to represent C_0.

A dash line in Fig. 1 represents a cut of the network since it separates the terminal graph t from the root graph G_0. That means if we cannot reach the target, then the system has failed. To find all of the cuts (dash lines) of G_0, we found that there exist some relationships between the cut function of a graph and that of its sub-graphs. For G_0, there are two edges, x_1 and x_2, connected to the source of G_0. That means there are two branches of G_0 in the edge reduction diagram. A branch will fail if its corresponding edge or sub-graph fails. Therefore, the rule to find a cut of G_0 is to let both branches fail. Hence, the cuts of G_0 are $\{ \bar{x}_1 \wedge \bar{x}_2$, $\bar{x}_1 \wedge$ the cuts of G_2, $\bar{x}_2 \wedge$ the cuts of G_1, the cuts of $G_1 \wedge$ the cuts of $G_2\}$ as the dash lines in Fig. 1. Note that, for simplification, not all of the cuts (dash lines) of G_0 are shown in Fig. 1. This forms the recursive relationship of the cut function, i.e.

$$C_k = \bigwedge_{i \in E_k} \left(\bar{x}_i \vee \text{the cuts of } \mathrm{RN}(G_{k*i}) \right) \qquad (6)$$

For example, C_0 is:

$$C_0 = (\bar{x}_1 \vee C_1) \wedge (\bar{x}_2 \vee C_2) \qquad (7)$$

Based on the disjoint property of OBDD, the cut function of a given terminal-pair network could be easily constructed by using the AND, OR manipulations of OBDD. Additionally, the ordering of the variables is determined by the breadth-first searching method [9] for a compact size of the OBDD.

Moreover, instead of the conventional tree-based partitions, our diagram-based reduction can avoid the redundant computations of isomorphic sub-graphs. Fig. 1 contains nine non-terminal nodes and one shared isomorphic graph G_8. In our algorithm, we use a hash table to record the network topology and its corresponding cut function (i.e. cut-based OBDD) for each shared isomorphic graph. This hash table can avoid the redundant computations on isomorphic graphs. If we get a hit in the hash table, we do not need to recalculate the information of this graph. We can retrieve it from the hash table. A proper hash table can reduce the time to compare network topologies. When the network becomes larger and more complex, it brings a significant growth on the number of isomorphic graphs. The benefit of using the hash table becomes significant. Fig. 2 shows the cut-based OBDD of G_0 derived by our method. The cut-based algorithm for constructing the OBDD of a terminal-pair network using edge reduction diagram is shown in Fig. 3.

4. Reliability Analysis Using OBDD

4.1. Reliability (Availability)

In the previous section, the cut-based OBDD of a given terminal-pair network, g, is obtained. Based on the disjoint property of OBDD, we can efficiently derive the reliability

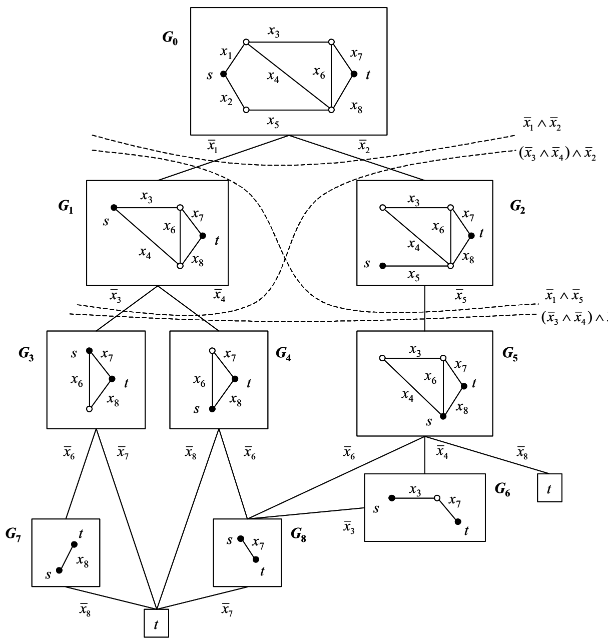
G_0
G_1
G_2
G_3
G_4
G_5
G_6
G_7
G_8
$\bar{x}_1 \wedge \bar{x}_2$
$(\bar{x}_3 \wedge \bar{x}_4) \wedge \bar{x}_2$
$\bar{x}_1 \wedge \bar{x}_5$
$(\bar{x}_3 \wedge \bar{x}_4) \wedge \bar{x}_5$
s
t

Figure 1. Edge reduction diagram for a terminal-pair network.

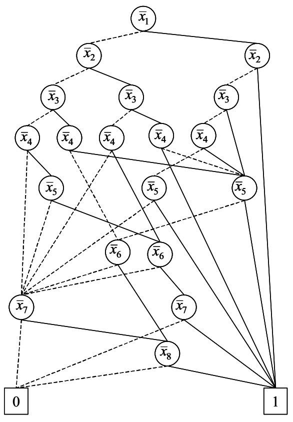

Figure 2. The cut-based OBDD of G_0 in Figure 1.

```
Procedure bdd Cut_BDD_Construct($G_k$)
    bdd bdd_op, bdd_result;
    $s_k$ = the source vertex of $G_k$;
    if ($G_k$ is the target) then return (bdd_one);
    if ( ( bdd_result = find_hash_table($G_k$) ) is a hit) then return (bdd_result);

    bdd_result = bdd_one;
    For each $e_i$ connected to the $s_k$ in $G_k$ {
        compute $G_{k*i}$;
        eliminate redundant nodes in $G_{k*i}$;     // i.e. RN($G_{k*i}$)
        bdd_op = Cut_BDD_Construct($G_{k*i}$);
        bdd_op = BDD_or($\bar{e}_i$, bdd_op);
        bdd_result = BDD_and(bdd_result, bdd_op);
    }
    insert_hash_table($G_k$, bdd_result);

    free temp bdd nodes during manipulations;
    return(bdd_result);
}
```

Figure 3. The cut-based algorithm for constructing the OBDD of a network.

(R) or availability (A) of a network system from the use of the reliability or availability of each component in probability calculation, respectively.

$$R = 1 - \Pr\{g\} \text{ or } A = 1 - \Pr\{g\} \quad (8)$$

4.2. Importance Measure

The Birnbaum importance measure of a component (say component *k*) represents the probability that a system is in a critical state with respect to that component, i.e. the probability that the system is initially in a good state and the failure of component *k* causes the system to fail. In [12] two algorithms are proposed to compute the Birnbaum importance measure using OBDD. However, the algorithms presented in [12] take more computational time and calculate only one component's importance measure at a time. They need to be run again to obtain the importance measure of another component. In this section, we propose an algorithm to compute multiple components' importance measures with only a single-pass OBDD traversal.

The Birnbaum importance measure is defined as the partial derivative of the system unreliability with respect to the failure probability of component *k*:

$$I_k^B(t) \equiv \frac{\partial F(t)}{\partial F_k(t)} = \Pr\{g_{\bar{x}_k=1}\} - \Pr\{g_{\bar{x}_k=0}\} \quad (9)$$

where $F(t)$ is the system failure probability at time *t*, $F_k(t)$ is the failure probability of component *k* at time *t*; $F_k(t) = \Pr\{g_{\bar{x}_k=1}\}$ where $\bar{x}_k = 1(0)$ means component *k* is faulty (good). *g* is the system structure function and $\Pr\{g_{\bar{x}_k=1}\}$ ($\Pr\{g_{\bar{x}_k=0}\}$) is the unreliability of the system given that component *k* has failed (not failed).

A) Two-pass traversal

This method traverses the OBDD twice to obtain the importance measure for component *k* [12].

- Find $\Pr\{g_{\bar{x}_k=1}\}$ using OBDD; i.e., find the system unreliability by assuming component *k* has failed.
- Find $\Pr\{g_{\bar{x}_k=0}\}$ using OBDD; i.e., find the system unreliability by assuming component *k* is not faulty.
- $\Pr\{g_{\bar{x}_k=1}\} - \Pr\{g_{\bar{x}_k=0}\}$ gives the Birnbaum importance measure for component *k*.

B) Modified single-pass traversal

There are two steps in a single traversal. In the first step, from definition (9), the importance measure depends on the probability of state transition of component *k*. Therefore, a disjoint path, which goes to terminal one and does not include component *k* in it, will not contribute to the importance measure of component *k*. We should delete this type of paths or let the probabilities of the paths be 0 when traversing the OBDD.

The second step is similar to the procedure in Method A for nodes in finding $\Pr\{g_{\bar{x}_k=1}\}$ and $\Pr\{g_{\bar{x}_k=0}\}$ except for the node corresponding to component *k*. Therefore, we combine the two calculations at the node corresponding to component *k* and compute the probability of each node (say *i*) in OBDD using the following rules:

- If node *i* is corresponding to component *k*, then
$$\Pr\{g_k\} = \Pr\{g_{\bar{x}_k=1}\} - \Pr\{g_{\bar{x}_k=0}\} \quad (10)$$
- If node *i* is not corresponding to component *k* and ordering(*i*) > ordering(*k*), then
$$\Pr\{g_i\} = \Pr\{\bar{x}_i\} \cdot \Pr\{g_{\bar{x}_i=1}\} + (1 - \Pr\{\bar{x}_i\}) \cdot \Pr\{g_{\bar{x}_i=0}\} \quad (11)$$
- If node *i* is not corresponding to component *k* and ordering(*i*) < ordering(*k*), then also use (11) to calculate $\Pr\{g_i\}$ except that let the probability of sub-tree $\Pr\{g_{\bar{x}_i=1}\}$($\Pr\{g_{\bar{x}_i=0}\}$) be 0 in (11) if the right (left) sub-tree is independent of component *k*. To check if the sub-tree of node *i* is independent of component *k* is simple. Let node *j* be the sub-node of node *i*. If ordering(*j*) > ordering(*k*) then the sub-tree is independent of component *k*.

Finally, when we have finished traversing the OBDD, we get the probability of the root, $\Pr\{g\}$. $\Pr\{g\}$ gives the Birnbaum importance measure of component *k*.

C) Single-pass traversal for multiple components

This method traverses the OBDD only once to get the importance measures of multiple components. This method is extended from Method B. If Ω is the set of components whose Birnbaum importance measures are to be calculated, $\Pr\{g(0)\}$ is the system unreliability, and $\Pr\{g(k)\}$ is the Birnbaum importance measure of component *k*, then we can compute the following at each node (say *i*):

- For each node *i*
$$\Pr\{g_i(0)\} = \Pr\{\bar{x}_i\}\Pr\{g_{\bar{x}_i=1}(0)\} + (1 - \Pr\{\bar{x}_i\})\Pr\{g_{\bar{x}_i=0}(0)\} \quad (12)$$
- For each $k \in \Omega$
  - If node *i* is corresponding to component *k*,
$$\Pr\{g_i(k)\} = \Pr\{g_{\bar{x}_i=1}(0)\} - \Pr\{g_{\bar{x}_i=0}(0)\} \quad (13)$$
  - If node *i* is not corresponding to component *k* and ordering(*k*) > ordering(*i*), then let the probability of sub-tree $\Pr\{g_{\bar{x}_i=1}(k)\}$ ($\Pr\{g_{\bar{x}_i=0}(k)\}$) be 0 in (14) if the right (left) sub-tree is independent of component *k*. Then calculate $\Pr\{g_i(k)\}$ using (14).
$$\Pr\{g_i(k)\} = \Pr\{\bar{x}_i\}\Pr\{g_{\bar{x}_i=1}(k)\} + (1 - \Pr\{\bar{x}_i\})\Pr\{g_{\bar{x}_i=0}(k)\} \quad (14)$$
  - Otherwise, do nothing since $\Pr\{g_i(k)\}$ is equivalent to $\Pr\{g_i(0)\}$.

Finally, the probabilities, $\Pr\{g(k)\}$ for all $k \in \Omega$, at the root in the OBDD gives the Birnbaum importance measure of component *k*. Fig. 4 illustrates the OBDD-based algorithm for the calculation of Birnbaum importance measures of multiple components by traversing the OBDD only once.

5. Experimental Results

Our algorithm has been implemented on a Linux Red Hat 7.3 operating system with Pentium-III CPU and 128 Mbytes memory. All of our programs are written in C lan-

```
struct imp {          // Importance Measure
    double g[m];   // the set of components whose  Birnbaum's importance
}                       // measure is to be evaluated
main() {
    static imp bim, bdd_one, bdd_zero;
    for k = 0 to m
        bdd_one.g[k] = 1;
        bdd_zero.g[k] = 0;
    next
    bim = measure(root);
    //   bim.g[0] is the system unreliability;
    //   bim.g[k] is the Birnbaum's importance measure of component k;
}
Procedure imp measure(bdd $x_i$) {          // $x_i$ is a node in Fig. 2
    imp result, n_true, n_false;
    if ( $x_i$ = BDD_one ) then return (bdd_one);
    if ( $x_i$ = BDD_zero ) then return (bdd_zero);
    if ( result = get_computed_node($x_i$) is a hit ) then return (result);
    n_true = measure(sub_node_true($x_i$));
    n_false = measure(sub_node_false($x_i$));
    result.g[0] = q * n_true.g[0] + p * n_false.g[0];   // q = 1 – p
    for k = 1 to m                                  // the set of components whose  Birnbaum's importance
        if ($x_i$ is component $k$ ) then              // measure is to be evaluated
            result.g[k] = n_true.g[0] – n_false.g[0];
        elseif ( ordering($x_i$) < ordering(k) ) then
            sub_true = n_true.g[k];
            sub_false = n_false.g[k];
            if n_true is independent of component k then sub_true = 0;
            if n_false is independent of component k then sub_false = 0;
            result.g[k] = q * sub_true + p * sub_false;
        end if
    next
    insert_computed_node($x_i$, result);
    return (result);
}
```

Figure 4. The OBDD-based algorithm for calculating the Birnbaum importance measure of multiple components.

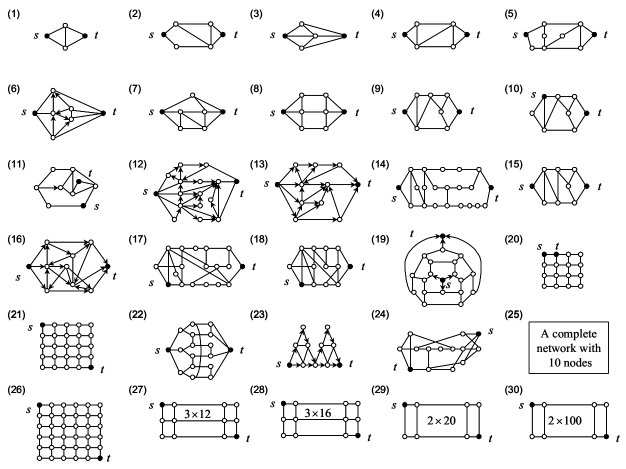

Figure 5. Benchmark networks #1 to #30.

guage. In the evaluation, we used 30 benchmark networks collected in [1-7][9] as shown in Fig. 5. All the unsuccess probabilities of links are 0.1. The results for terminal-pair network reliability match the results reported in [1-7][9]. It should be noted that our method does not assume the cuts or the minimum cutest to be given previously while the previous works [1-7] generally assumed that and finding them is very time-consuming.

Table 1 illustrates the reliability and the execution time obtained by our method as well as the comparison of the SDP methods [5][7] and the OBDD-based methods. The comparison of results between the SDP and the OBDD size is not straightforward because the OBDD representation represents a Boolean function as a graph-based set of disjoint products, which differs from the SDP of two-level forms. However, we still compare the number of disjoint cuts in [5] with the number of nodes in OBDD as a reference. The EED-ISO [9] column is the size of OBDD nodes based on path-set method. To the best of our knowledge, [5] has the best result in minimizing the number of disjoint products using an SDP generating method with a random and preprocessed list of cut-sets. For networks #17 to #19, the OBDD representations are more compact for large disjoint cut sets when the number of cuts > 200. However, compared with EED-ISO [9], the number of nodes using OBDD based on path-set method or cut-set method is of the same order.

The computation time in [5] is CPU time in seconds on an FPS 500 system and does not include the cut-set generation time. The approach in [7] is an efficient method to computing the terminal-pair reliability, but no disjoint cuts were generated. The time in [7] is the CPU time in IBM RISC System/6000. The EED-ISO [9] based on the path-set method using OBDD is run on a SPARC 20 workstation with 128 Mbytes memory. The execution time of our algorithm includes the times for the construction of OBDD and the reliability evaluation from the OBDD. For a large-scale network, especially network #19 and #30, the effectiveness of our algorithm becomes significant. Our cut-based approach has a great improvement over the previous works. Although, compared with EED-ISO [9], the performance is the same order as that based on the path-set method; however, we propose a new method based on the cut-set method using OBDD.

Table 2 shows the Birnbaum importance measure of each component in network #2 and #19. When the cut-based OBDD of a network have been constructed, we only need to traverse the OBDD once to get all the Birnbaum importance measures. Therefore, by our method, we can efficiently identify the critical components of a network for the sensitivity analysis.

6. Conclusions

This paper has two main contributions. First, we have proposed an algorithm to construct an OBDD representing the cut function of a network. The algorithm recognizes isomorphic sub-problems and thus avoids redundant computations. Therefore, the system reliability can be efficiently derived based on the OBDD. Second, we have proposed an approach to compute the importance measures for multiple components by traversing the OBDD only once. This technique could be applied to systems whose system structure function is represented by an OBDD. The experimental results showed that our method is very efficient and can handle very large complex terminal-pair networks. Based on this approach, researches on sensitivity analysis, importance measures, failure frequency analysis or optimal

Table 1. The comparison of the SDP and OBDD methods.

–: no data, Dcut: number of disjoint cuts, Nodes: size of OBDD nodes.
Lex: lexicographic ordering used to minimize Dcut, C&L: lexicographic ordering and cardinality used to minimize Dcut.
Time: CPU time, including the times required by Cut_BDD_Construct(), and Reliability(). 0.00 if the CPU time consumed < 0.01. The sampling period is 0.00195312 second.

Network	# of cuts	Reliability	Lex [5]	C&L [5]	EED-ISO [9]	Our Method	Lex [5]	CUT [7]	EED-ISO [9]	Our Method
			Dcut	Dcut	Nodes	Nodes	Time	Time	Time	Time
1	4	0.978480	5	4	10	9	0	0.00	0.00	0.00
2	9	0.968425	16	12	15	20	0	0.00	0.00	0.00
3	8	0.997632	13	10	26	24	0	0.00	0.00	0.00
4	9	0.977184	21	14	22	17	0	0.00	0.00	0.00
5	28	0.964855	82	50	50	68	0	0.00	0.00	0.00
6	18	0.996664	25	23	39	49	0	0.00	0.00	0.00
7	20	0.997494	62	43	51	61	0	0.00	0.00	0.00
8	29	0.996217	115	73	66	53	0.1	0.00	0.00	0.00
9	19	0.975116	76	42	36	31	0	0.00	0.00	0.00
10	20	0.984068	78	43	68	50	0	0.00	0.00	0.00
11	24	0.969112	120	38	48	67	0	0.00	0.00	0.00
12	396	0.997186	4496	1386	548	621	0.2	0.02	0.03	0.00
13	110	0.994076	315	150	157	153	0.1	0.01	0.00	0.00
14	528	0.904577	11443	3854	126	131	0.3	0.02	0.00	0.00
15	25	0.974145	157	81	40	43	0	0.00	0.00	0.00
16	78	0.997506	321	202	407	351	0.1	0.03	0.02	0.00
17	1300	0.985928	61651	16194	643	589	3.8	0.20	0.05	0.01
18	214	0.987390	11198	2319	292	260	0.2	0.15	0.03	0.00
19	7376	0.997024	1126719	281453	3591	3324	189.3	3.00	0.35	0.07
20	105	0.987831	–	–	177	179	–	0.02	0.02	0.01
21	8742	0.975557	–	–	1148	1148	–	0.55	0.43	0.16
22	1721	0.998059	–	–	1505	1681	–	0.20	0.08	0.02
23	16	0.959624	–	–	46	45	–	0.00	0.00	0.00
24	436	0.995744	–	–	250	441	–	0.03	0.03	0.01
25	256	1.000000	–	–	49785	46257	–	5.40	14.10	2.17
26	–	0.975224	–	–	4970	4970	–	–	15.75	4.74
27	34241	0.961730	–	–	317	317	–	–	3.65	0.98
28	–	0.956266	–	–	437	437	–	–	70.73	18.16
29	400	0.784482	–	–	115	115	–	–	0.02	0.01
30	10000	0.304317	–	–	595	595	–	–	2.52	0.24

design issues of multi-state systems will be the focus of our future works.

Table 2. The Birnbaum importance measure of network #2 and #19.

–: no data, BI: Birnbaum importance.

Network #2		Network #19			
Edge	BI	Edge	BI	Edge	BI
1	0.186567	1	0.012377	16	0.000656
2	0.097548	2	0.002033	17	0.001322
3	0.025377	3	0.001590	18	0.000877
4	0.018816	4	0.001625	19	0.000425
5	0.097548	5	0.001517	20	0.001507
6	0.010716	6	0.001490	21	0.001495
7	0.098277	7	0.002213	22	0.000422
8	0.107106	8	0.000484	23	0.001524
–	–	9	0.000548	24	0.001521
–	–	10	0.000650	25	0.000513
–	–	11	0.000654	26	0.001624
–	–	12	0.000753	27	0.001592
–	–	13	0.001186	28	0.012396
–	–	14	0.012377	29	0.012400
–	–	15	0.002033	30	0.012420

References

[1] S. Rai and K.K. Aggarwal, "An efficient method for reliability evaluation of a general network", *IEEE Trans. Reliability*, vol. R-27, pp. 206-211, Aug. 1978.

[2] J.A. Abraham, "An improved algorithm for network reliability", *IEEE Trans. Reliability*, vol. R-28, pp. 58-61, 1979.

[3] S. Rai, A. Kumar, and E.V. Prasad, "Computer terminal reliability of computer network", *Reliability Engineering*, vol. 16, pp. 109-119, Jan. 1986.

[4] S. Harri and C.S. Raghavendra, "SYREL: A symbol reliability algorithm based on path and cut-set methods", *IEEE Trans. Computers*, vol. C-36, pp.1224-1232, Oct. 1987.

[5] S. Soh and S.Rai, "Experimental results on preprocessing of path/cut term in the sum of disjoint products technique", *IEEE Trans. Reliability*, vol. 42, pp. 24-33, Mar. 1993.

[6] S. Soh and S. Rai, "CAREL: Computer Aided RELiability evaluator for distributed computing networks", *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 199-213, Apr. 1991.

[7] Y.G. Chen and M.C. Yuang, "A cut-based method for terminal-pair reliability", *IEEE Trans. Reliability*, vol. 45, pp. 413-41, Sept. 1996.

[8] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation", *IEEE Trans. Computers*, vol. C-35, pp. 677-691, Aug. 1986.

[9] S.Y. Kuo, S.K. Lu, and F.M. Yeh, "Determining Terminal-Pair Reliability Based on Edge Expansion Diagrams Using OBDD", *IEEE Trans. Reliability*, vol. 48, no. 3, pp. 234-246, Sept. 1999.

[10] A. Rauzy, "New algorithms for fault tree analysis", *Reliability Engineering and System Safety*, vol. 40, pp. 203-211, 1993.

[11] R.M. Sinnamon and J.D. Andrews, "Improved efficiency in qualitative fault tree analysis", *Quality and Reliability Engineering Int'l.*, vol. 13, pp. 293-298, 1997.

[12] R.M. Sinnamon and J.D. Andrews, "Fault Tree Analysis and Binary Decision Diagrams," *Proc. Ann. Reliability and Maintainability Symp.*, *(RAMS '96)*, pp.215-222, Jan. 1996.