OBDD-Based Evaluation of k-Terminal Network Reliability

Fu-Min Yeh, Shyue-Kung Lu, and Sy-Yen Kuo, *Fellow, IEEE*

***Abstract*—An efficient approach to determining the reliability of an undirected k-terminal network based on 2-terminal reliability functions is presented. First, a feasible set of ($k - 1$) terminal-pairs is chosen, and the 2-terminal reliability functions of the ($k - 1$) terminal-pairs are generated based on the edge expansion diagram using an OBDD (ordered binary decision diagram). Then the k-terminal reliability function can be efficiently constructed by combining these ($k-1$) reliability expressions with the Boolean `and` operation. Because building 2-terminal reliability functions and reducing redundant computations by merging reliability functions can be done very efficiently, the proposed approaches are much faster than those which directly expand the entire network or directly factor the k-terminal networks. The effectiveness of this approach is demonstrated by performing experiments on several large benchmark networks. An example of appreciable improvement is that the evaluation of the reliability of a source-terminal 3 × 10 all-terminal network took only 2.4 seconds on a SPARC 20 workstation. This is much faster than previous factoring-algorithms.**

***Index Terms*—Factoring, network reduction, network reliability, Ordered Binary Decision Diagram (OBDD), terminal-pair reliability.**

ACRONYMS[1]

EED_BFS	algorithm based on edge expansion diagram with breadth-first-search ordering
OBDD	ordered binary decision diagram
SDP	sum of disjoint products
st	source-terminal.

NOTATION

$G = (V, E)$	a graph whose edges can fail s-independent of each other, with known probabilities
$\mathrm{Rel}(G_k)$	k-terminal reliability of network G
t_k	a node in a k-terminal node set
e_i	edge i in a network, $1 \le i \le n$
E_i	Boolean variable of e_i, $1 \le i \le k$
p_i, q_i	[success, failure] probability of e_i; $p_i + q_i = 1$
$*, -$	[contraction, deletion] operation on links
$G_k * e$	network G_k with contracted edge e
$G_k - e$	network G_k with deleted edge e
$f_x = 1, 0$	[positive, negative] cofactor of function f
$P(x_i)$	pathset of x_i
$P(G_k)$	k-working pathset function of G_k
$\lvert V \rvert$	number of nodes.

Manuscript received July 6, 1999; revised September 15, 2001. Responsible Editor: J. A. Nachlas.

F.-M. Yeh is with the Electronic System Research Division, Chung-Shan Institute of Science and Technology, Taoyuan, Taiwan - R.O.C. (e-mail: FMYeh@lion.ee.ntu.edu.tw; FMYeh@tpts5.seed.net.tw).

S.-K. Lu is with the Department of Electronic Engineering, Fu-Jen Catholic University, Taipei, Taiwan - R.O.C. (e-mail: SKLu@mars.ee.fju.edu.tw).

S.-Y. Kuo is with the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan - R.O.C. (e-mail: SYKuo@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TR.2002.804736

[1]The singular and plural of an acronym are always spelled the same.

I. INTRODUCTION

Definitions

- k-success path-set: the edges between the specified terminal-pairs
- K-direct: direct construction method
- fixed_source: the source node is fixed in a feasible set
- fixed_sink: the sink node is fixed in a feasible set
- merged_source: successive source merging for a feasible set
- feasible set: the k nodes in the set are covered and connected by ($k - 1$) terminal-pairs
- m-source to k-terminal reliability: $\Pr\{$at least 1 process can be successfully executed$\}$.

The most general network reliability problem in the literature is the k-terminal problem. Given a set of target nodes $k \in V$ with $k = |K|$, the k-terminal reliability is $\Pr\{$for a given node $s \in K$, there exists at least 1 working path from s to all other nodes in $k\}$. This reliability is the sum of the probabilities of disjoint success paths; but the complexity of identifying all disjoint success paths is exponential (a well-known NP-hard problem) [1], [11]. Therefore, the determining k-terminal reliability for a network is very time-consuming. Most existing researches on k-terminal reliability [2]–[5] focus on speeding up the calculations by reducing the computation efforts as much as possible. References [2]–[5] emphasize improving the factoring of a network with several "minimal sum of disjoint edges" based reduction rules. Generally, the factoring algorithms [2]–[5] with reduction rules are recognized, in the literature, as the most efficient algorithms in dealing with k-terminal reliability problem.

The factoring algorithms [2]–[5] use the cofactor-theorem to partition the network based on e_i in each step directly. Then they recursively factor $\mathrm{Rel}(G_k - e_i)$ and $\mathrm{Rel}(G_k * e_i)$ until the k terminals are fully contracted or disconnected. The edge cofactors are disjoint with respect to each other; therefore, the reliability can be obtained by summing all disjoint path products. They also use the network reduction rules to reduce the number of sub-problems. Although these methods are demonstrated with reasonable efficiency on small-scale networks, they have 3 inherent drawbacks:

1) determining the sum of disjoint product forms is inefficient for large Boolean functions,
2) tree-based factoring algorithm does not consider the merging of isomorphic sub-problems to avoid redundant computations,
3) factoring an entire network directly has exponential complexity.

Thus, factoring algorithms using tree-based partitioning are inevitably very time-consuming on large networks, even if a problem contains many isomorphic sub-graphs.

For determining k-terminal reliability, 4 assumptions are made:

1) The network is modeled as an undirected graph.
2) The p_i and q_i, $1 \leq i \leq n$, are known for each link.
3) Nodes are fault free.
4) All failure events are mutually s-independent.

To overcome these inherent drawbacks, this paper presents an efficient approach to determine the reliability of an undirected k-terminal network based on the 2-terminal reliability function.

1) A feasible set of $(k-1)$ terminal-pairs is chosen.
2) All the 2-terminal reliability functions for the $(k-1)$ terminal-pairs, respectively, are derived based on edge expansion diagram using OBDD.
3) The k-terminal reliability function can be efficiently obtained by combining these $(k-1)$ reliability expressions with Boolean $\mathtt{and}$ operations.

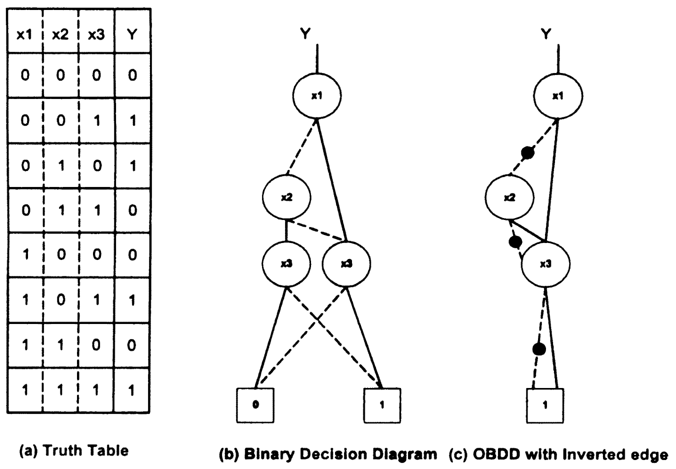
Because constructing 2-terminal reliability functions [6] and reducing the redundant computations by merging reliability functions as proposed in this paper can be done very efficiently, this approach is much faster than expanding or factoring the entire network directly [2]. The effectiveness of this approach is demonstrated by performing experiments on large networks. Experimental results show that the time required to evaluate the reliability of a st3 × 10 all-terminal network was only 2.4 seconds on a SPARC 20 workstation, which is appreciably better than the factoring algorithms.

Section II introduces the OBDD representation of a network, and proposes efficient procedures to calculate the probabilities of the OBDD-based functions. Section III describes a brute-force method that searches all working edges of a k-terminal network. Section IV presents efficient methods to evaluate k-terminal reliabilities based on terminal-pair reliability functions; and proposes an extension of this algorithm for evaluating m-source to k-terminal network reliability. Section V demonstrate the effectiveness of these algorithms by implementing them on large networks.

II. Preliminaries

The representation and manipulation of Boolean functions based on OBDD are introduced. Then the inherent drawbacks of the previous algorithms based on event-tree partition are illustrated by an example. The OBDD [7] is based on the decomposition of a Boolean function, the "Shannon expansion." A Boolean function f can be decomposed in terms of a Boolean variable x as:

$$f = x \cdot f_{x=1} + \overline{x} \cdot f_{x=0}. \tag{1}$$



x1	x2	x3	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Fig. 1. Various representations of a Boolean function.

A node and its descendants in an OBDD represent a Boolean function f, where or node label x, one outgoing edge is directed to the subgraph representing $f_{x=1}$, and the other to $f_{x=0}$. In following a path from the root to a terminal node, simply take successive cofactors of a function until it reduces to a constant. From the results in [8], the size of a OBDD with inverted-edges is 7% smaller than a noninverted one, and the speed of manipulation for the inverted one is about twice that for the noninverted one.

Fig. 1(a) shows Boolean representations of the truth table; Fig. 1(b) shows the OBDD, and Fig. 1(c) shows the OBDD with inverted edges of a Boolean function;

- a circle node represents the decision-variable node;
- a dashed line represents the value 0;
- a solid line represents the value 1;
- a terminal value (in rectangle) corresponds to True(1) or False(0);
- a dot on an edge represents: the function following this edge is complemented.

In Fig. 1, the size of OBDD is 6, with 4 nonterminal nodes and 2 terminal nodes. However, the size of OBDD with inverted edge is the most compact, which is only 4 with 3 nonterminal nodes and 1 terminal node. One useful property of OBDD is that all the paths are mutually disjoint. Hence, one can evaluate the probability of a Boolean function represented by the OBDD. An operation on 2 OBDD-based functions can be decomposed in terms of a variable x as follows:

$$f\langle op\rangle g = x \cdot (f_{x=1}\langle op\rangle g_{x=1}) + \overline{x} \cdot (f_{x=0}\langle op\rangle g_{x=0}). \tag{2}$$

These Boolean operations include AND, OR, NOT, and determining the size of the on-set for a function. Equation (2) can be realized by applying an algorithm which traverses the argument graphs to recursively-apply the operation to subgraphs. A more detailed description of the algorithms for constructing and manipulating an OBDD is in [7], [8].

Fig. 2 illustrates the construction and manipulation steps of the Boolean function

$$F = (x1 \text{ and } x3) \text{ or } (x2 \text{ and } x3).$$

First, for a given Boolean function, a global variable ordering must be determined. Then each step of evaluation and the resulting OBDD representations are depicted.

F = (x1 and x3) or (x2 and x3)

Variable Ordering : { x1, x2, x3 }

Evaluation steps:

```
x1 = declare_variable(x1, 1)
x2 = declare_variable(x2, 2)
x3 = declare_variable(x3, 3)
T1 = BDD_and( x1, x3 )
T2 = BDD_and( x2, x3 )
F  = BDD_or( T1, T2 )
```

Resulting Graphs

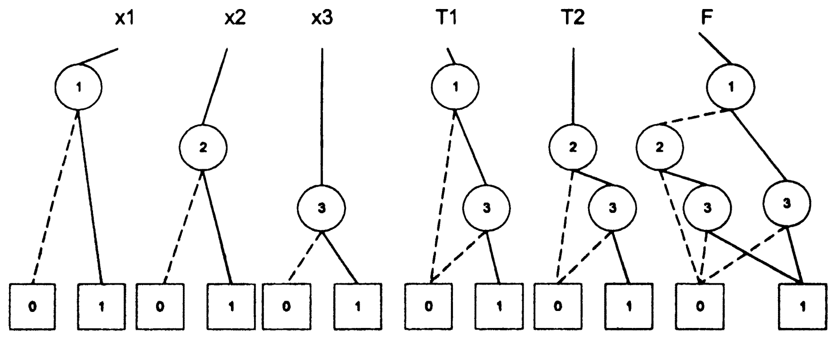


Fig. 2. OBDD generated from a Boolean equation.

The OBDD is based on a Shannon expansion: a set of disjoint decomposed functions. Thus, the OBDD can be recognized as a graph-based set of disjoint products. Given the probability of each variable, the reliability of an OBDD-based function f can be recursively evaluated by:

$$\Pr\{f\} = P(x_i) \cdot \Pr\{f_{x_i=1}\} + [1 - P(x_i)] \cdot \Pr\{f_{x_i} = 0\}; \quad (3)$$

x_i $\equiv$ a variable of f,

f is partitioned into 2 disjoint sets, $f_{x_i} = 1$ and $f_{x_i} = 0$.

Instead of modifying the original OBDD node structure, a hash table [9] is used to avoid the redundant computations of the isomorphic shared nodes in Procedure Prob(bdd_node) of Fig. 3.

If a terminal node is encountered, return the value of the terminal node. To avoid the redundant computation of the bdd_node, an isomorphism check is performed. If a node has been evaluated and found in the hash table, then directly return the value of the probability; else recursive call the procedure with a negative cofactor and a positive cofactor. The final probability is achieved by combining the results of the positive cofactor and the negative cofactor based on (3). If the top node contains the flag of the inverted edge, then swap the success probability and the failure probability of the variable and compute the probability based on (3). Fig. 4 shows the steps of the probability evaluation in every node of the OBDD. For each OBDD node, the values of the bdd_result are also shown in the boxes. This procedure uses 10 multiply operations and 5 additions overall. During the process of calculating the success probability, the number of multiply operations and additions is of the same order as the number of OBDD nodes. Therefore, the size of OBDD strongly dominates the efficiency of this algorithm. More detailed descriptions are in [6].

```
Procedure Prob(bdd_node)
bdd bdd_node;
{   p is the success probability of top variable of bdd_node;
    q = 1- p;
    if ( bdd_node == bdd_one ) return (1);
    if ( bdd_node == bdd_zero) return (0);
    if ( ( bdd_result = find_hash_table(bdd_node)) != NULL )   return(bdd_result);
    else { if   ( bdd_node is non-inverted edge )
              bdd_result = p  xProb(bdd_node->high)+ q  xProb(bdd_node->low);
           else
              bdd_result = q  xProb(bdd_node->high)+ p  xProb(bdd_node->low);
           insert_hash_table (bdd_node, result);
           }
     return(bdd_result);
}
```

Fig. 3. Algorithm for evaluating probability, based on OBDD.

III. Construction of the k-Terminal Reliability Functions

Several definitions and the concept of sharing isomorphic graphs are introduces. Then the k-terminal reliability can be evaluated by an edge expansion diagram using OBDD. An example illustrates this approach.

A graph $G = (V, E)$ consists of 2 sets: V is the node set, and E is the edge set. Two nodes u, v are connected if there exists a path from u to v. A set of k nodes are connected if there exists a path for each pair of these nodes. These connected paths are: k-success paths. The k-success path set is composed of the edges which are either directly connected to the k nodes or indirectly via a chain working path [11]. Therefore, the k-terminal reliability is the probability that at least 1 k-success path set exists.

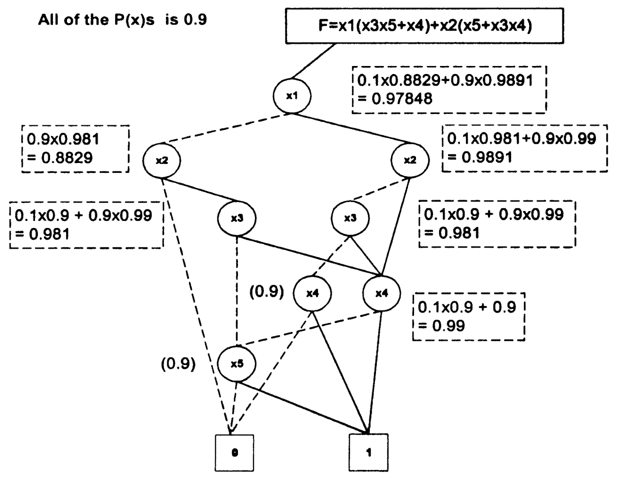

All of the P(x)s is 0.9
F=x1(x3x5+x4)+x2(x5+x3x4)
0.1x0.8829+0.9x0.9891 = 0.97848
x1
0.9x0.981 = 0.8829
x2
x2
0.1x0.981+0.9x0.99 = 0.9891
0.1x0.9 + 0.9x0.99 = 0.981
x3
x3
0.1x0.9 + 0.9x0.99 = 0.981
(0.9)
x4
x4
0.1x0.9 + 0.9 = 0.99
(0.9)
x5
0
1

Fig. 4. Determination of probability for an OBDD function.

Definition: Two networks are k-terminal reliability isomorphic if and only if their corresponding edges, nodes, k-terminal nodes, and success probability of any two corresponding edges are equivalent.

According to this definition, the calculations of k-terminal reliabilities of several networks can be reduced to the calculation for just 1 of these networks if these networks (graphs) are k-terminal reliability isomorphic: the reliability of 1 network is also the reliability of its isomorphic counterparts. Therefore, the computation effort can be reduced appreciably.

Based on the material in this paper so far, all k-success paths for a k-terminal network can be expressed as a k-terminal reliability function or a k-pathset function. Once the reliability function is obtained, the network reliability can be efficiently obtained by recursively evaluating all the OBDD-based reliability functions as stated in Section II. Instead of the conventional tree-based factoring partitions [2]–[5], a graph-based heuristic approach which can avoid the redundant computation of isomorphic subgraphs for the construction of all k-success paths is described in the following 4 steps.

1) To avoid redundant constructions of a k-pathset function, first examine if a graph contains isomorphic subgraphs. If isomorphic subgraphs exist in computing the hash table, then directly return the result. Otherwise, expand the graph with edges connected to the source. Let (e_1, e_2, …, e_k) be success edges incident to the source node. Expand the given network, G_k, into new sub-networks according to (e_1, e_2, …, e_k). Sub-network i corresponds to edge, e_i, for a total of k sub-networks.

2) For the sub-network i ($1 \le i \le k$), perform k-terminal contraction to obtain a new network, $G_k * e_i$, in which the nodes originally incident with e_i are lumped into a new node, and all the edges originally connected to the new node can not be deleted. Then the parallel edges are merged into a new edge by Boolean `or`. The internal k-working pathset function of G_k is determined by:

$$P(G_k) = \sum_{i=1}^{k} E_i \cdot P(G_k * e_i). \qquad (4)$$

3) Remove redundant-nodes to avoid redundant expansions. A node other than the terminal nodes is redundant if only 1 node is connected to it.

4) For each network, $G_k * e_i$, repeat steps 1–3 until all k terminals are reached, and then return `True`. The final reliability function of the network is obtained by recursively combining the results of intermediate k-working pathset functions.

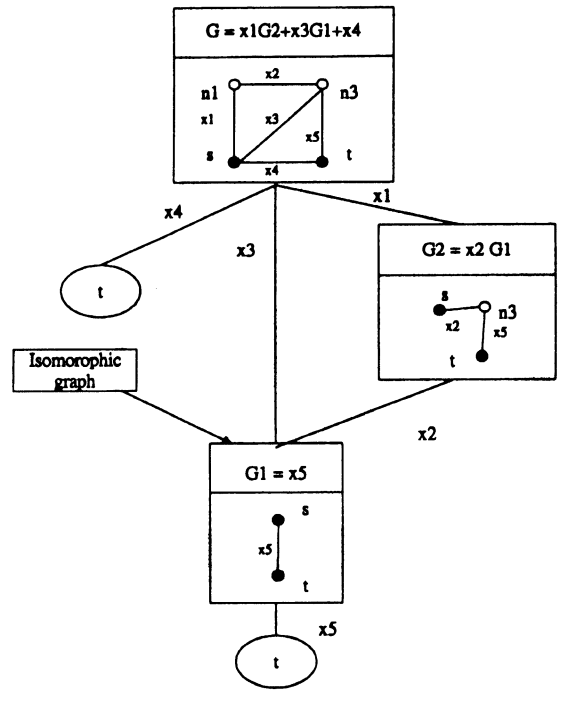
For network partitioning, k-terminal networks have higher complexity than 2-terminal networks. The main difference during the expansion is in step 2. For a 2-terminal network, there exist no simple paths which are parallel from 1 node. When an expansion edge is chosen, all the other edges originally connected to the expanded node can be deleted. Therefore, many sub-networks can be reduced. In contrast, in a k-terminal network, there might exist outgoing parallel paths from 1 node. By using the expanding approach directly, no edges can be further removed. Therefore, only a few edges can be removed, and the number of derived sub-networks might be still very large. The expansion approach for a k-terminal network still has a serious difficulty due to the huge number of subproblems generated, even if many isomorphic sub-networks have been identified. Another difficulty is that the edge functions change during the expansion. This change also dramatically reduces the number of isomorphic graphs, and increases the processing time.

Fig. 5(a) shows a 2-terminal network; Fig. 5(b) shows a 3-terminal network based on the edge expansion diagram. In the 2-terminal network, the expanded node s is chosen first. Then, according to its success edges x_1, x_3, x_4, network G is partitioned into sub-networks $G1$, $G2$ and a terminal node t. $G1$ is obtained by deleting x_1 and x_4 while merging nodes s and n_3. For $G1$ and $G2$, repeat steps 1–3 until a terminal node is reached. Then the logic value `True` is returned. Before expanding a network, first see if a graph contains isomorphic subgraphs; if they exist in computing the hash table, then return their results directly. The checking of isomorphism can be used to avoid redundant computations due to isomorphic reliability networks. The partitioning of the 2-terminal edge expansion diagram results in 3 nonterminal nodes and 1 isomorphic subgraph $G1$.

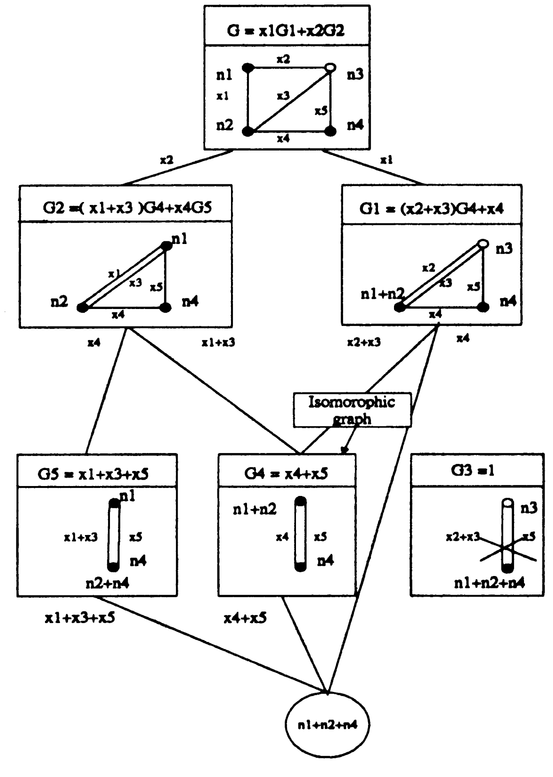
In Fig. 5(b), node n_1 is chosen first for expansion. According to its success edges, x_1 and x_2, network G is partitioned into $G1$ and $G2$. $G1$ is obtained by deleting x_1 and merging n_1 with n_2 in G. For $G1$ and $G2$, repeat steps 1–3 until terminal #3 is reached, and return the logic value `True`. Before expanding a network, isomorphism is first checked. The k-terminal edge-expansion diagram in Fig. 5(b) contains 5 nonterminal nodes and 1 isomorphic graph $G4$. A redundant node appears during the expansion of $G1$ with edge x_4. Sub-network $G3$ contains a redundant node, which can be eliminated to terminate the partitioning earlier. Although this approach has reasonable performance for small-scale networks, it generates an enormous number of sub-networks for large networks—because few edges can be deleted and the edge functions are not the same during the partitioning.

IV. Fast Construction of the k-Terminal Reliability Functions

Intuitively, the reliability function of an undirected k-terminal network can be derived through the reliability functions of its ($k-1$ edge) 2-terminal sub-networks: a set of k nodes requires $k-1$ edges in order to be connected. In this case,



(a)



(b)

Fig. 5. Edge-expansion diagrams. (a) 2–terminal edge-expansion diagram. (b) 3–terminal edge-expansion diagram.

"edges" between 2 terminal nodes are all the paths between those 2 terminal nodes. Instead of constructing the k-terminal reliability function by edge expansion diagram directly, this section presents a new approach.

1) Sufficient conditions for deriving the k-terminal reliability function by a set of $(k-1)$ terminal-pair reliability functions are derived.
2) Efficient algorithms for selecting a feasible set are presented.
3) The algorithm for a k-terminal network is extended to an "m-source to k-terminal" network.

A. A Feasible Set for an Undirected k-Terminal Network

Notation:

$F_k(G_k)$:	reliability function of an undirected k-terminal network.
$(t_i,\, t_j)$:	a terminal-pair in the set of k terminals
$F_2(t_i,\, t_j)$:	a terminal-pair reliability function
Π:	a Boolean and operation.

A feasible-set for an undirected k-terminal network is a set where all the k nodes are covered and connected by these $(k-1)$ terminal-pairs.

Theorem 1: $F_k(G_k)$ can be obtained by Boolean and of all the reliability functions of the $(k-1)$ 2-terminal networks in a feasible-set.

Proof: The reliability of an undirected k-terminal network is the probability that the network has at least 1 working k-success path, i.e., there must exist a success path between 2 arbitrary different nodes out of these k terminal-nodes. The reliability function for a 2-terminal network includes all the paths between 2 terminal-nodes. Therefore, the reliability function for

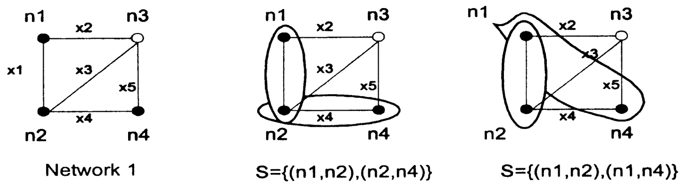


Fig. 6. Two feasible sets for a 3-terminal network.

an undirected k-terminal network can be derived by connecting all the $(k-1)$ 2-terminal networks if this terminal-pair set can cover all the k nodes. Therefore, the reliability function for an undirected k-terminal network can be denoted as

$$F_k(G_k) = \Pi_{(k-1)} F_2(t_i,\, t_j).$$

The set of $(k-1)$ terminal-pairs is a feasible-set for an undirected k-terminal network.

Fig. 6 is an example to illustrate theorem 1. Three nodes, n_1, n_2, n_4 are specified as terminal nodes. The simple working paths of this network are x_1x_4, $x_1x_2x_5$, $x_1x_3x_5$, $x_2x_3x_4$, $x_2x_4x_5$, $x_2x_3x_5$. The reliability function can be derived as Boolean or of these paths:

$$\begin{aligned} F_3(\text{Network 1}) =& F_3(1,\, 2,\, 4) \\ =& x_1x_4 + x_1x_2x_5 + x_1x_3x_5 + x_2x_3x_4 \\ &+ x_2x_3x_5 + x_2x_4x_5. \end{aligned}$$

This method of directly searching all working paths is a direct-construction-method. From theorem 1, the 2 terminal pairs $(n_1,\, n_2)$ and $(n_2,\, n_4)$ can be chosen to cover the 3 terminal nodes: n_1, n_2, n_4. The reliability functions for pairs $(n_1,\, n_2)$ and $(n_2,\, n_4)$ are

$$\begin{aligned} F_2(1,\, 2) =& x_1 + x_2x_3 + x_2x_5x_4, \\ F_2(2,\, 4) =& x_4 + x_3x_5 + x_1x_2x_5, \end{aligned}$$

respectively. The procedure to check for equivalence is:

$$\begin{aligned}
&F_2(1,\,2) \text{ and } F_2(2,\,4)\\
&\quad= (x_1 + x_2x_3 + x_2x_5x_4)(x_4 + x_3x_5 + x_1x_2x_5)\\
&\quad= x_1x_4 + x_1x_3x_5 + x_1x_2x_5 + x_2x_3x_4 + x_2x_3x_5\\
&\qquad + x_1x_2x_3x_5 + x_2x_3x_4x_5 + x_2x_4x_5 + x_1x_2x_4x_5\\
&\quad= x_1x_4 + x_1x_3x_5 + x_1x_2x_5 + x_2x_3x_4 + x_2x_3x_5\\
&\qquad + x_2x_4x_5\\
&\quad= F_2(\text{Network } 1) = F_2(1,\,2) \text{ and } F_2(1,\,4).
\end{aligned}$$

An alternative feasible set is $(n_1,\, n_2)$ and $(n_1,\, n_4)$. The function $[F_2(1,\,2)$ and $F_2(1,\,4)]$ is also equal to $F_3(\text{Network1})$. To simplify this discussion, the preceding expressions use sum-of-product form. Instead of "sum of product" form, OBDD for Boolean operations can be used.

B. The Fixed-Sink Algorithm

This section presents an efficient algorithm to generate the k-terminal reliability functions based on 2-terminal reliability functions. Instead of choosing the set based on a fixed source or arbitrary pairs, a fixed-sink algorithm is used to implement the whole idea. A hash table [9] is used to store the intermediate results of 2-terminal pair subgraphs and their OBDD-based reliability functions during the edge expansions [6]. Due to these $(k-1)$ 2-terminal pairs having the same sink, the hash table can be reused among different terminal-pair constructions. Thus this strategy avoids the redundant evaluation of isomorphic graphs during the process of building reliability functions of various terminal pairs; i.e., this method does not need to clear the intermediate hash table of subgraphs during edge expansions of all $(k-1)$ 2-terminal pairs because the sink is intact. All the isomorphic graphs can be identified and their corresponding computations can be reduced to just 1 calculation. The procedures to compute the k-terminal reliabilities are described here.

1) Use an heuristic approach to find a good variable ordering for the OBDD-based k-terminal reliability function.
2) Choose a feasible set by making the sink intact.
3) The reliability function for an undirected k-terminal network is derived through Boolean `and` of the reliability functions of all 2-terminal pairs in the feasible set.
4) Recursively evaluate the probabilities of all the nodes in the OBDD to obtain the k-terminal network reliability.

Given an undirected k-terminal network, this fixed sink algorithm `Procedure Fixed_sink()` for determining the reliability of a network is shown in Fig. 7. First, choose an $(s,\, t)$ pair, where $t \in k$ is an arbitrary node. Variable orderings for the network are obtained by the breadth-first-search approach [10]. The source node s, $s \in k$, has the longest distance from t by breadth-first-search traversal. Then make this sink node fixed and iteratively process all the $(s_j,\, t)$ pathset functions by the terminal-pair algorithm based on edge expansion diagrams, and Boolean `and` these $(k-1)$ expressions to generate the k-terminal reliability function. Once the OBDD-based reliability function has been derived, the reliability can be efficiently obtained by recursively applying `Procedure prob()`. For most cases, this algorithm avoids exponential expansion caused by the direct construction approach.

```
Procedure Fixed_sink( )
bdd bdd_result;
{
    bdd_result = BDD_one;
    Given a network G;
    Let K be a terminal set with k terminal ($s_1, s_2, .., s_k$);
    Arbitrarily choose a t from K;
    Select $s_j$ from K with the longest distance to s;
    Variable_Ordering(G, $s_j$, t);
    For all pairs ($s_i$, t)      /* $s_i \neq t$ */
    bdd_result = Pathset_Function_Construct(G, $s_i$, t) "and" bdd_result;
    Reliability = Prob(bdd_result);
}
```

Fig. 7. A `Fixed_sink` algorithm for a k-terminal network.

```
Procedure MK_ terminal( )
    bdd bdd_result;
{
    bdd_result = BDD_one;
    Let S-set be a source set with m source ($S_1, S_2, .., S_m$);
    Let T-set be a terminal set with k terminal ($t_1, t_2, .., t_k$);
    Given a undirected network G, arbitrarily select an S from S-set;
    Select $t_j$ from T-set with the largest distance to S;
    Variable_Ordering(G, S, $t_j$);
    For all $S_i$
    {   For all pairs ($S_i$, $t_i$)
            bdd_k_result=Pathset_Function_Construct(G, $S_i$, $t_i$) "and" bdd_result;
        bdd_mk_result = bdd_mk_result "or" bdd_k_result;
    }
    Reliability = Prob(bdd-result);
}
```

Fig. 8. Determination of the reliability of a $m \cdot k$-terminal network.

The longest 2-terminal pairs built first will have more isomorphic subgraphs and the corresponding OBDD functions, which can be shared with other 2-terminal constructions.

In the manipulations as well as the applications of OBDD, the efficiency is dominated by the size of OBDD for representing a given function. The OBDD size strongly depends on the input-variable ordering. However, the best algorithm for finding the optimal ordering has a complexity of $O(n^2 \cdot 3^n)$; but this approach is not realistic. Therefore, a simple, efficient BFS [10] method is used to obtain a good variable-ordering. This BFS method has the complexity $O(E)$, $E =$ number of edges in a network.

Although the fixed-sink algorithm is very efficient for calculating k-terminal reliability, there exist a few redundant calculations when processing these $(k-1)$ 2-terminal expressions. Therefore, an additional criterion for successively combining these $(k-1)$ 2-terminal reliability functions is applied to reduce the number of sub-networks. This additional criterion is: The selected nodes can be grouped, marked, and treated as a terminal node (referred as the Merged_source algorithm). This criterion can terminate network partitioning early when multiple terminals exist.

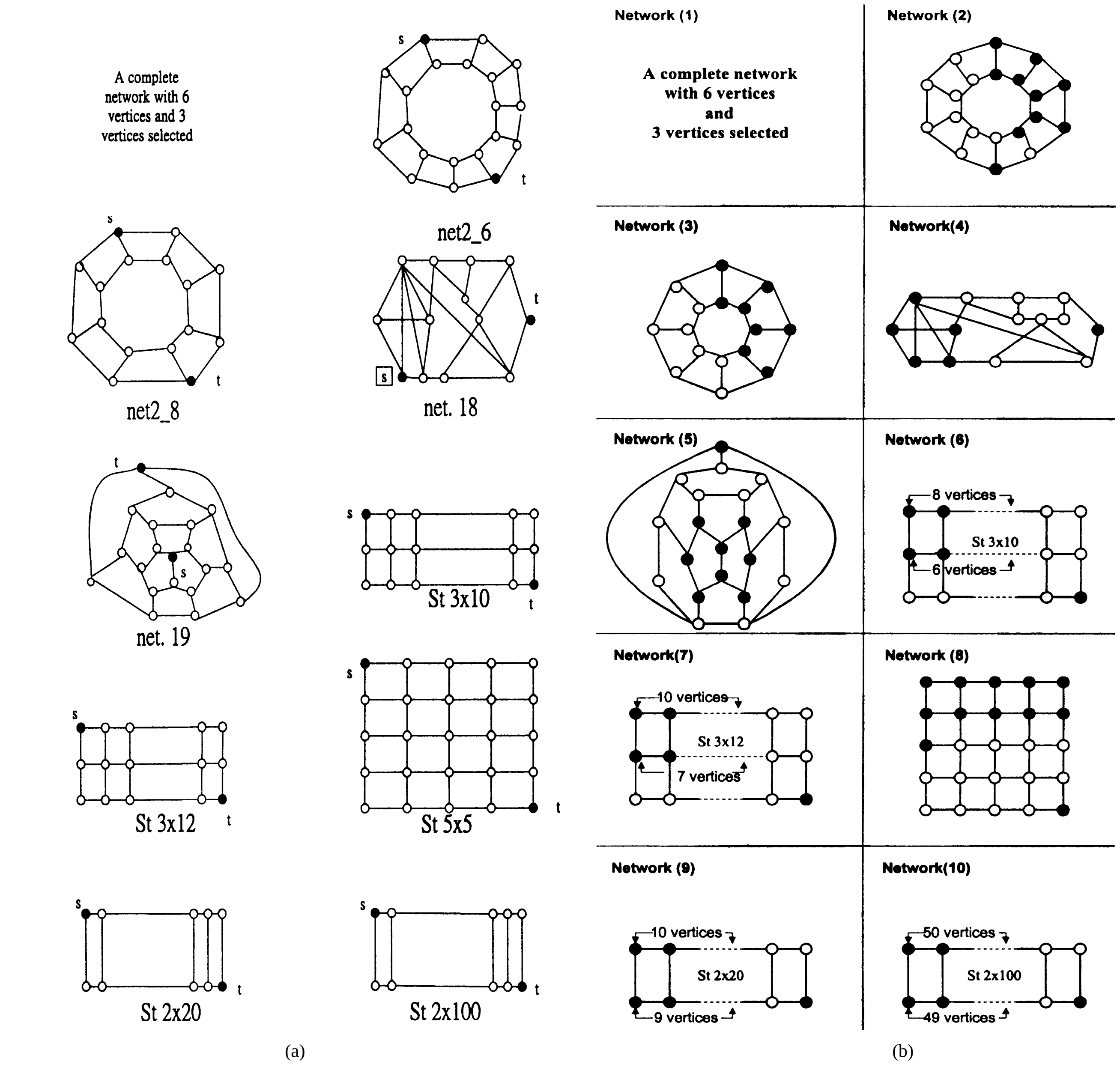


Fig. 9. K-terminal networks. (a) 2–terminal networks. (b) $|K| \approx |V|/2$

C. Evaluation of Reliabilities for the m-Source to k-Terminal Networks

The application of a distributed computer network for an "m-source to k-terminal" problem is that a process can duplicate on different nodes (m-source), which requires k-resource files distributed on different k nodes during program execution. The "m-source to k-terminal" reliability is the probability that at least 1 process can be successfully executed. This algorithm is similar to the k-terminal algorithm. The unique feature of this algorithm is that it requires the user to manipulate Boolean `or` operations on the reliability functions for each source to the k terminal nodes. The formulation is:

$$F(G_{m\cdot k}) = \sum_{s_i \subset S_m} \prod (s_i, T_k) P(G_2(t_i, t_j)); \qquad (5)$$

T_k	$\equiv$ set of k-terminal nodes,
S_m	$\equiv$ set of m sources,
$P(G_2(t_i, t_j))$	$\equiv$ reliability function for a $(s_i, t_1, t_2, \ldots, t_k)$ terminal network,
$F(G_{m\cdot k})$	$\equiv$ reliability function for an m-source to k-terminal network.

The complete algorithm is in Fig. 8: Procedure MK_terminal(). First, arbitrarily select 1 node s from S, and 1 terminal t from T, such that the (s, t) pair has the largest distance. Then the global variable ordering for an OBDD is obtained by breadth-first-search traversing. Equation (5) is applied for direct construction of an m-source to k-terminal reliability function based on the terminal-pair algorithm. Finally, the reliability is obtained by recursively evaluating all nodes of this OBDD.

V. Experimental Results

This approach for determining the reliability of an undirected k-terminal network has been implemented on a SPARC 20

TABLE I
COMPARISON RESULTS FOR ALL-TERMINAL NETWORK RELIABILITIES

Network	[2]	EED_BFS							
		K_direct		Fixed_source		Fixed_sink		Merged_source	
	Time	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
st3×6	10	28.9	100	1.8	100	0.8	100	0.5	100
st3×8	142	-	-	4.7	142	1.7	142	0.9	142
st3×10	1994	-	-	18.2	184	7.4	184	2.4	184
st3×12	-	-	-	92.7	226	32.9	226	7.73	226
st3×14	-	-	-	516	268	104	268	32.8	268
st3×16	-	-	-	2822	310	465	310	141	310
st3×18	-	-	-	-	-	-	-	656	352

K_direct: directly constructing method

Fixed_source: the source node is fixed in a feasible set

Fixed_sink : the sink node is fixed in a feasible set

[2]: on a Macintosh microcomputer

TABLE II
COMPARISON RESULTS FOR k-TERMINAL NETWORK RELIABILITIES

Network	k-terminal								
	$k = 2$ EED_BFS			$k = \|V\|/2$ Merged_source			$k = \|V\|$ Merged_source		
	Reli.	Time	Nodes	Reli.	Time	Nodes	Reli.	Time	Nodes
com.6	0.99998	0.02	148	0.99997	0.77	176	0.99994	0.82	173
net2_6	0.99440	0.08	423	0.98468	1.03	518	0.97350	0.95	221
net2_8	0.99569	0.02	302	0.98892	0.87	356	0.97966	1.08	295
fig_18	0.98739	0.02	292	0.98520	0.76	294	0.98076	0.83	232
fig_19	0.99712	0.33	3591	0.98768	2.18	5697	0.97713	2.50	1903
st3×10	0.96447	0.55	257	0.95493	2.50	310	0.92405	2.40	184
st3×12	0.96137	2.88	317	0.95108	7.80	383	0.91731	7.73	226
st5×5	0.97556	0.43	1149	0.95743	2.50	1633	0.93981	2.72	697
st2×20	0.78448	0.02	115	0.76525	1.33	136	0.74530	1.55	97
st2×100	0.30429	2.52	595	0.27663	1233	696	0.25107	1677	497

Reli.: reliability of a network

$|V|$: number of nodes

EED_BFS: algorithm based on edge expansion diagram with breadth-first-search ordering

workstation with 128 Mbytes memory. All of these programs are written in C language. Experimental results are compared with previous work [2]. Ten benchmark networks [Fig. 9(a) and (b)] are used in these experiments. The upper bound of the OBDD size is 1500k in terms of shared OBDD with inverted edges. The success probabilities of all nodes are 0.9.

Directly finding all k-success disjoint paths for a k-terminal network requires exponential computation-time. In general, this complexity is also much higher than that for a 2-terminal network. Only the approach in [2] seems to have dealt with large k-terminal networks. It uses a factoring theorem and network-reduction rules to calculate the reliability of a k-terminal network. Table I compares the results in this paper with those in [2]. The k_Direct column denotes direct construction of a k-terminal reliability network by edge expansion diagram. The Fixed_source column uses the proposed fixed source method based on Boolean and all $(k - 1)$ 2-terminal reliability functions. The results of the fixing-sink approach are shown in the column: Fixed_sink. An alternate reduction rule during the Boolean and of all $(k - 1)$ 2-terminal functions is applied to the fixing-sink approach; the results are in the Merged_source column. The columns of Time record the CPU times consumed; the columns of Nodes indicate the number of OBDD nodes generated for each method, respectively.

Another attractive property of these methods using OBDD is that the reliability expression is unique when these 4 reliability functions have the same variable ordering. This property can also be used to verify whether the new algorithm is correct or not. For example, to check the correctness of these reliability values obtained using various methods (k_direct, fixed_sink, fixed_source, and fixed_sink_MT), just check whether these 4 OBDD-based reliability functions are equivalent or not. The verification results show that these 4 OBDD are exactly equivalent; their OBDD sizes are shown in Table I.

Table I shows that the proposed methods are appreciably better than those in [2]. The computation time for a st3 $\times$ 10 all-terminal network is only 2.4 seconds on a SPARC 20 workstation. It is much faster than the 1994 seconds required by the factoring algorithm in [2] on a Macintosh microcomputer. The rate of time increase for every addition of a stage in this approach is about 2.5, which is only half of that in [2].

- Fixed_source approach: the hash table becomes invalid once the sink is changed for various terminal-pairs; therefore, the computation for the Fixed_source approach is inefficient.
- Fixed_Sink method: this can avoid these redundant computations and has better results.
- Merged_source strategy: this has the best time-performance among the 4 approaches, due to the effectiveness of increasing the hashing rate and reducing the number of sub-networks (as mentioned in Section IV-B).

Table II presents the reliability evaluation results for k-terminal networks with $k = 2$, $k = |V|/2$, and $k = |V|$. In general, the time overhead of determining the all-terminal reliabilities for the networks in Fig. 9 is sufficiently low. For example, the time required for a st3 $\times$ n network is only about 4 times that of the original 2-terminal network. These results are appreciably better than those of the direct construction method, which has exponential time complexity. The number of OBDD nodes for each method is limited within an acceptable range such that the proposed approaches can efficiently solve all the problems within a reasonable amount of time.

Table II shows that the probabilities of k-terminal networks are between the 2-terminal networks and all-terminal networks. The reliability of a 2-terminal network is the highest among $k = 2$, $k \approx |V|/2$, and $k = |V|$. In contrast, the all-terminal networks exhibit the lowest reliability. Although the number of 2-terminal reliability expressions for all-terminal networks is twice than that for $|V|/2$-terminal networks, the processing time only increases by 0.2% on average. The reason is that the evaluation of reliability functions of 2-terminal networks can be terminated immediately when the number of termination nodes increases.

REFERENCES

[1] M. O. Ball, "Computational complexity of network reliability analysis: An overview," *IEEE Trans. Reliability*, vol. R-35, pp. 230–239, Aug. 1986.

[2] L. B. Page and J. E. Perry, "A practical implementation of the factoring theorem for network reliability," *IEEE Trans. Reliability*, vol. 37, pp. 259–267, Aug. 1988.

[3] ——, "Reliability of directed networks using the factoring theorem," *IEEE Trans. Reliability*, vol. 38, pp. 556–562, Dec. 1989.

[4] R. K. Wood, "Factoring algorithms for computing K-terminal network," *IEEE Trans. Reliability*, vol. R-35, pp. 269–278, Aug. 1986.

[5] O. R. Theologous and J. G. Carlier, "Factoring & reductions for networks with imperfect vertices," *IEEE Trans. Reliability*, vol. 40, pp. 210–217, Apr. 1991.

[6] 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.

[7] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[8] K. S. Brace, R. L. Rudell, and R. E. Bryant, “Efficient implementation of an OBDD package,” in *Proc. 27th Design Automation Conf.*, 1990, pp. 40–45.
[9] R. Kruse, B. Leung, and C. L. Tondo, *Data Structures and Program Design in C*, 2nd ed: Prentice-Hall, 1996.
[10] F. M. Yeh and S. Y. Kuo, “OBDD-based network reliability calculation,” *Electronics Letters*, vol. 33, no. 9, pp. 759–760, Sept. 1997.
[11] J. A. Buzacott, “A recursive algorithm for finding reliability measures related to the connect of nodes in a graph,” *Networks*, vol. 10, pp. 311–327, 1980.
[12] S. J. Friedman and K. J. Supowit, “Finding optimal variable ordering for binary decision diagrams,” *IEEE Trans. Computers*, vol. 39, no. 5, pp. 710–713, May 1990.

Fu-Min Yeh received the B.S. in 1985 in electronic engineering from Chung-Yuan Christian University, the M.S. in 1992 in electrical engineering from National Taiwan University, and the Ph.D. in 1997 in electrical engineering from National Taiwan University. He is a deputy section-head at the Electronic System Research Division of Chung-Shan Research Institute of Science and Technology. His research interests include DSP system design, hardware verification, VLSI testing, and fault-tolerant computing.

Shyue-Kung Lu received his Ph.D. in 1995 in electrical engineering from National Taiwan University. From February 1995 to August 1998, he was an Associate Professor in the Department of Electrical Engineering at Lunghwa Junior College of Technology & Commerce. Since then, he has been with the Department of Electronic Engineering at Fu-Jen Catholic University, where he is an Associate Professor. His research interests include VLSI testing and fault-tolerant computing.

Sy-Yen Kuo received the B.S. in 1979 in electrical engineering from National Taiwan University, the M.S. in 1982 in electrical and computer engineering from the University of California at Santa Barbara, and the Ph.D. in 1987 in computer science from the University of Illinois at Urbana-Champaign. He is a professor and Chairman of the Department of Electrical Engineering, National Taiwan University, Taipei. He spent his sabbatical year as a visiting researcher at AT&T Labs-Research, New Jersey from 1999 to 2000. He was the Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan from 1995 to 1998, a faculty member in the Department of Electrical and Computer Engineering at the University of Arizona from 1988 to 1991, and an engineer at Fairchild Semiconductor and Silvar-Lisco, both in California, from 1982 to 1984. In 1989, he also worked as a summer faculty fellow at Jet Propulsion Laboratory of California Institute of Technology. His current research interests include mobile computing and networks, dependable distributed systems, software reliability, optical WDM networks.

Professor Kuo has published more than 180 papers in journals and conferences. He received the distinguished research award (1997–2001) from the National Science Council, Taiwan. He also received the Best Paper Award in the 1996 International Symposium on Software Reliability Engineering, the Best Paper Award in the simulation and test category at the 1986 IEEE/ACM Design Automation Conference (DAC), the National Science Foundation’s Research Initiation Award in 1989, and the IEEE/ACM Design Automation Scholarship in 1990 and 1991.