©2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

A Memory Efficient Algorithm for Network Reliability

Johannes U. Herrmann and Sieteng Soh
Department of Computing
Curtin University of Technology, Perth, Australia
jherrmann@ieee.org, S.Soh@curtin.edu.au

***Abstract* – We combine the Augmented Ordered Binary Decision Diagram (OBDD-A) with the use of boundary sets to create a method for computing the exact *K*-terminal or all-terminal reliability of an undirected network with failed edges and perfect vertices. We present the results of implementing this algorithm and show that the execution time is comparable with the state of the art and the space requirement is greatly reduced. Indeed the space remains constant when networks increase in size but maintain their structure and maximum boundary set size; with the same amount of memory used for computing a 3×12 and a 3×1000 grid network.**

***Keywords* – binary decision diagram, boundary set, network reliability, all-terminal reliability, *K*-terminal reliability, space efficient.**

I. INTRODUCTION

WITH increasing reliance on communication networks (CN), measuring their reliability is an important aid in their design and analysis. When communication links between CN devices fail nodes may become isolated from each other. The reliability (REL) of CNs has been studied extensively [1-4] and measures the probability that the CN meets the relevant standard of connectivity. This paper focuses on *K*-terminal reliability (*K*-REL), the probability that all communication devices in *K* are connected to each other, and all-terminal reliability (ALL-REL), the probability that all devices in a CN are connected where the communication links in the CN are subject to independent failure and devices are perfect. The *K*-REL problem is the more general problem, and has been proven to be NP-Hard [5].

The literature contains two main approaches to computing REL; those that generate network paths or cuts and then manipulate them, and those that manipulate the network directly. For general networks, algorithms that apply factoring techniques to the network are more efficient than those that manipulate paths and/or cuts [3]. However even these systems can have difficulty analyzing large CNs.

The Ordered Binary Decision Diagram (OBDD) [6] has been successfully used to compute both *K*-REL[1-3, 7] and ALL-REL[1, 4] directly from the network. This approach reduces redundant computations by detecting and merging equivalent (isomorphic) diagram nodes, and thus reduces the number of nodes generated. For OBDDs it is important to efficiently represent and manipulate CN states. References [1, 4] use boundary sets [6] to efficiently represent OBDD states; we discuss boundary sets in detail in Section II.C. Even with the uses of isomorphism and boundary set notation, the number of OBDD nodes generated for very large networks (e.g., 5×15,000 grid) or networks with a large maximal boundary set (*e.g.*, K15, the 15-node fully connected network) is extremely large. In addition, the 2-step algorithms proposed in [1, 4] first generate the OBDD and then traverse it a second time to generate REL. This approach requires all diagram nodes to be stored. Thus, even for nodes with an efficient representation of the network, the amount of memory required is extremely large. As reported in [1], the 12×12 grid requires storing close to 65 million OBDD nodes, each contains at least two pointers and a numerical value which can exceed 4 bytes in size. Hence storing the nodes may require more than 780MB of memory.

In this paper we present an exact 1-step algorithm using an Augmented OBDD (OBDD-A) to compute *K*-REL or ALL-REL. Our algorithm significantly reduces the space complexity of the OBDD approach in [1] while generating an equivalent number of nodes and taking comparable computational time. The OBDD-A stores probability information in each diagram node, enabling us to compute the probability of child nodes directly. Each OBDD-A node is only processed once and can be discarded afterwards. At any time, our algorithm requires less than 2 complete levels of the diagram to be stored, in contrast to the OBDD algorithms [1, 4] which require all diagram nodes to be stored. Hence the OBDD-A greatly reduces the amount of nodes stored in memory and leads to constant memory use for problems of increasing size but identical connectivity structure.

This paper is organized into five sections. After the background and notation information in Section II we describe our OBDD-A algorithm in Section III. We present experimental results in Section IV and conclusions and the scope of future work in Section V.

II. BACKGROUND AND NOTATION

A. Network Model

We model a CN using a graph G=(V,E), where each vertex in V represents a communication device and every edge in E represents a communication link between the devices. An edge e_j is said to be UP (DOWN) if it is functioning (failed). Let p_j (q_j=1-p_j) be the operational (failure) probability of e_j and assume all failures are statistically independent. Assume that all vertices are always active and that all edges are undirected.

Let n=|V|, and let the vertices (v_0, v_1, …, v_{n-1}) of V be ordered in increasing distance (number of hops) from a source vertex, v_0. When two or more vertices have the same distance from v_0, they are ordered arbitrarily. Let $\{v_i,v_j\}$ denote an undirected edge between vertices v_i and v_j, with

978-1-4244-4785-5/09/$25.00 ©2009 IEEE

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on February 25,2010 at 22:05:37 EST from IEEE Xplore. Restrictions apply.

$i>j$ for each $\{v_i,v_j\}$. The sample network in Fig. 1 shows an example of such an ordering.

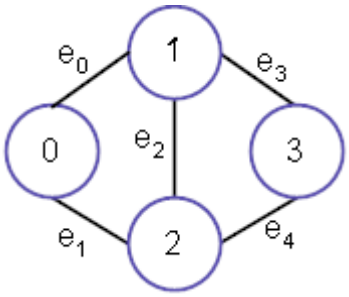

Figure 1: Sample Network

A network state $\Omega=E_\Omega$ of network G=(V,E) is a partition of E such that all edges in $E_\Omega \subseteq E$ are UP and all other edges in E are DOWN. The probability of state Ω is computed as:

$$\Pr(\Omega)= \prod_{e_i \in E_\Omega} p_i \prod_{e_i \notin E_\Omega} q_i \, .$$

Each Ω is associated with a sub-graph $G_\Omega = (V_\Omega,E_\Omega)$ where V_Ω are the vertices reachable from v_0 via the edges in E_Ω. For *K*-REL, Ω is a successful state if $K \subseteq V_\Omega$ and for ALL-REL, Ω is a successful state if $V_\Omega=V$. REL can be calculated by summing the probabilities of the success states of the network.

Since each edge can be UP or DOWN, there are $2^{|E|}$ states for network G., and therefore REL cannot be solved for large networks through state enumeration.

B. Ordered Binary Decision Diagrams

The OBDD is based on the Shannon decomposition [6]. Each level of the OBDD represents the evaluation of one variable; for this paper this is edge e_k at level k. Each diagram node N_i has two children; positive child N_{pi} representing the case when e_k is UP and negative child N_{ni} when it is DOWN. We say that an edge e_x is *decided* if it is known to be either UP or DOWN at the current level k of the OBDD; that is if $x < k$. Two diagram nodes N_i and N_j on level k are *isomorphic* if their sub-diagrams are identical. Isomorphic nodes are merged before being processed to eliminate redundant computation. The ordering of the variables affects the size of the OBDD, and finding the optimal variable ordering is itself an NP-Complete problem [8]. For this paper we use the breadth-first ordering described in Section A, which is equivalent to that used by Hardy *et al*. [1, 4]; edges $\{v_i,v_j\}$ are decided in increasing order of i and then j.

C. Boundary Sets

In order to calculate the reliability, the algorithm must record the effect of decided edges on the network state at any given level k of the diagram. In order to minimize memory usage, this information should be encoded as efficiently as possible. One approach is to use boundary sets which were introduced in [7].

For level k of the diagram, the Boundary Set, $F_k \subseteq V$, is composed of only those vertices required to encode the network state. For the vertices in the boundary set, the algorithm must record the connections (via paths of UP edges) to each other. If we are computing *K*-REL with $K<|V|$ the algorithm must also record whether each vertex in the boundary set is connected to any of the vertices in *K*.

Formally, the F_k is defined as:

$F_k = \{v_x |$ v_x is an endpoint of e_y and e_z with $y \le k$ and $z > k\}$.

The i^{th} element of F_k. is written as $F_k[i]$. The interconnectivity of elements of F_k is encoded by partitions of F_k. Two vertices are in the same partition if and only if they are connected to each other. For *K*-REL, if a partition is connected with one or more of the *K* vertices, then it is marked.

For example the boundary set of level 2 of the OBDD for the network given in Fig. 1 is {1,2}. If $K=\{0,3\}$ the possible partitions for OBDD nodes at this level are $[1\ 2]^*$, $[1]^*[2]$, $[1][2]^*$. Any partition that has no marked partitions (*e.g.* [1][2]) has no connection with vertex 0 and hence is failed. No partition can have two partitions marked since that would imply both are connected to 0 and hence each other. For a listing of the partitions at all levels of the diagram, see Fig. 5 in Section III.E.

Partitions are represented by vectors of size $|F_k|$ where the i^{th} position in the vector contains the number of the partition containing $F_k[i]$. The enumeration of partitions makes use of Stirling numbers of the second kind. These are calculated using $A_{i,j}= j \times A_{i-1,j}+A_{i-1,j-1}$ for $1 \le j \le i$, with $A_{i,1}=1$ and $A_{i,j}=0$ if $i<j$. This ordering is described in [1] and applies to ALL-REL. Because it does not consider marked partitions, it does not suffice for *K*-REL.

The algorithms given in [1] describe a pair of methods for ALL-REL that convert between the boundary set partition representation of the network and a unique partition number, and back again. Partitions (and hence the associated connectivity state) are efficiently stored as their associated partition number. However the methods in [1] do not differentiate between marked and unmarked partitions, and hence do not suffice for *K*-REL. Our modified methods, which can be used both *K*-REL and ALL-REL, are described in Section III.

III. The OBDD-A Algorithm

A. Introduction

This section details the structure of the OBDD-A and how it is used to solve ALL-REL and K-REL. Part B details the structure of the OBDD-A node, and hence the diagram itself. Part C extends the partition numbering scheme from [1] to K-REL. The method for constructing an OBDD-A and obtaining ALL-REL or K-REL from it is discussed in Part D, and illustrated with an example in Part E. Finally Part F discusses the space requirement of the OBDD-A algorithm..

B. OBDD-A

An OBDD-A is an OBDD whose nodes contain additional information [9] pertaining to the problem being computed. For example an OBDD-A used to compute the Expected Hop Count stores information on path length and state probability [9]. For REL, the OBDD-A stores only the probability of the node. An OBDD-A node does not store pointers to child or parent nodes since such linkages are never traversed. When a new node is created, its probability is calculated from the probability of the parent node; $\Pr(N_{ni})=\Pr(N_i)\times q_k$ and $\Pr(N_{pi})=\Pr(N_i)\times p_k$ for the negative

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on February 25,2010 at 22:05:37 EST from IEEE Xplore. Restrictions apply.

and positive child, respectively. When two OBDD-A nodes are found to be isomorphic, they are merged into a single node whose probability is the sum of the probabilities of both nodes. The success terminal node stores REL.

The connectivity of the network at level k is represented by partitions of F_k, with each OBDD-A node representing one partitioning of F_k. This provides an efficient means of storing the network state in the OBDD-A node and also of detecting isomorphic OBDD-A nodes.

C. Partition numbering for K-REL

Partitions of F_k are numbered consecutively as described in [1], ordered by the amount of partitions, the elements in each partition, and whether or not each partition is marked. These partition numbers can be calculated as shown in Fig. 2 and can be converted back into partitions as shown in Fig. 3; these functions are our modification of those in [1] to allow the computation of *K*-REL.

```
part_to_number (part[] Fi)
input: part[]: partition
Fi: size of boundary set of the level of part
output: number num of partition part[]

j=1, l, num=1: integer;
for all (l such that 2 ≤ l ≤ Fi ) do
      if ( part[l] = j + 1 ) then
            j = j + 1;
            num = num + j× A_{i-1,j};
      else
            num = j×( num -1) + part[l];
num = (num-1)×power(2, j)+1;
for all (l such that 1 ≤ l < j ) do
      num = num + A_{Fi,l}×2^l;
for all ( l such that 0 ≤ l < j ) do
      if (marked[l]) then
            num = num + power(2, l-1);
return num;
```

Figure 2: Computing *num* from *part*

When computing *K*-REL we must differentiate between partitions with differently marked partitions. This is done by firstly computing the partition number for the number of partitions j as described in [1] and then adding 2^{b-1} for each marked partition b of the partition. For example the partition $[1\ 2][3]^*$ has partition 2 marked and hence $2^1=2$ is added to its partition number.

The numbering for previous partition numbers is greater for *K*-REL than for ALL-REL, and hence we must increase the base of the partition number as well. We do this by applying the formula $num = (num\text{-}1)\times power(2, j)+1$. This allows for the previous partition numbers having been increased for different combinations of marked partitions. The function in Fig. 2 includes this formula, which is excluded from the original function in [1].

For example, consider the partition $[1\ 2][3]^*$. The first part of the method in Fig. 2 gives a base $num = 3$. This is because [1 2] [3] is the third partition of size 2 for $F_k=\{1,2,3\}$, with the first and second being [1 3] [2] and [1][2 3]. We next allow for previous partitions of size 2 to have permutations of marked partition giving us $num = 9$. We then allow for any partitions with fewer partitions, increasing *num* to 11. Lastly we allow for the marking of the partitions of the partition, adding 2 as discussed above to give a final partition number of 13. This agrees with the number from Fig. 6 in [1].

In order to restore the partition from the partition number, we reverse the process. First we find the number of partitions, j, by finding where the partition number, *num*, fits

```
number_to_part(num, size)
input: num: integer (partition number)
       size: integer (number of elements)
output: part[] of size elements with number num

i, j, n_ij: integer
part[]: partition
find j such that Σ_{x=1}^{j-1} 2^x A_{size,x} < num < Σ_{x=1}^{j} 2^x A_{size,x} ;
n_ij = (num - 1 - Σ_{x=1}^{j-1} A_{i,x} )%power(2, j);
i = 0;
while ( (n_ij > 0) && (i<size) ) do
      if ( (nij % 2) == 1) then
            marked[i] = true
      n_ij = n_ij /2;
      i = i + 1;
i = size;
nij = ((num - Σ_{x=1}^{j-1} (A_{i,x} + power(2,x)) - 1) / power(2,j))+1;
while (i ≠ 1 ) do
      if ( n_ij ≤ j × A_{i-1,j}) then
            part[i] = nij – floor((nij – 1)/j) + 1;
            n_ij = (n_ij - 1) / j +1;
      else
            n_ij = n_ij - j× A_{i-1,j};
            part[i] = j;
            j = j – 1;
      i = i – 1;
part[1] = 1;
return part;
```

Figure 3: Computing *part* from *num*

into the sums of Stirling numbers.

Secondly, the component of the partition number that encodes the marking of partitions is isolated. This number is repeatedly divided modulo 2, with a remainder indicating a marked partition.

The counting variable *nij* is then reduced to its equivalent for ALL-REL; reducing so that the *number_to_part* algorithm from [1] can be applied.

D. Constructing an OBDD-A for K-REL

Fig. 4 shows our OBDD-A algorithm for computing *K*-REL. The algorithm is initialized with $F_0 = \{v_0\}$ and the partition number for partition (v_0) stored in the root node of the BDD and on the current queue, Q_C. Note that for ALL-REL, the root node has partition number 1, and for *K*-REL the partition number is 2 because it is marked. At each level numbers are removed from Q_C, converted into partitions, and processed to produce two children. Non-terminal children are added to the next queue, Q_N and terminal children that represented a connected network have their probability added to REL.

E. OBDD-A Example

To illustrate the OBDD-A algorithm, we will apply it to the sample network in Fig. 1 with $K=\{0,3\}$. The root node is created with number 2 (which corresponds to $[0]^*$), and $F_0=\{0\}$.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on February 25,2010 at 22:05:37 EST from IEEE Xplore. Restrictions apply.

```
BDDAPart (G, K )
input: G = (V, E) (edges are ordered from 1 to m)
       K: set of K-terminal vertices
output: K-REL( G )

part[], part0[], part1[]: partition;
rel = 0: double;
create node (bdd root) with number 2 in level 1;
for all (k such that 1 ≤ k ≤ m ) do
    Compute F_{k+1};
    for each N_i on level k of the OBDD-A do
        part = number_to_part( i, |F_k|);
        create part0 from part to represent e_k DOWN;
        create part1 from part to represent e_k UP;
        if (all K-vertices are in the same partition in part1) then
            rel = rel + Prob(N_i)×p_k;
        else if ( no K-vertex is disconnected in part1 ) then
            j = part_to_number( part1 );
            if (N_j is not in the hash table on level k+1) then
                create N_j in level k+1;
                insert N_j in hash table;
            Prob(N_j) = Prob(N_j)×p_k;
        if ( no K-vertex is disconnected in part0 ) then
            j = part_to_number( part0 );
            if (N_j is not in the hash table on level k+1 ) then
                create N_j in level k+1;
                insert N_j in hash table;
            Prob(N_j) = Prob(N_i)×q_k;
        remove N_i from the hash table;
return rel;
```

Figure 4: Computing *K*-REL using OBDD-A

We enter the loop for level 1 (and edge {0,1}) and compute F_1={0,1}. The only node on level 0 is the root node, N_2, and this is translated back into the partition $[0]^*$. This partition is used to create children $[0]^*[1]$ and $[0\ 1]^*$, which translate to partition numbers 4 and 2 respectively. These are stored in child nodes N_4 and N_2.

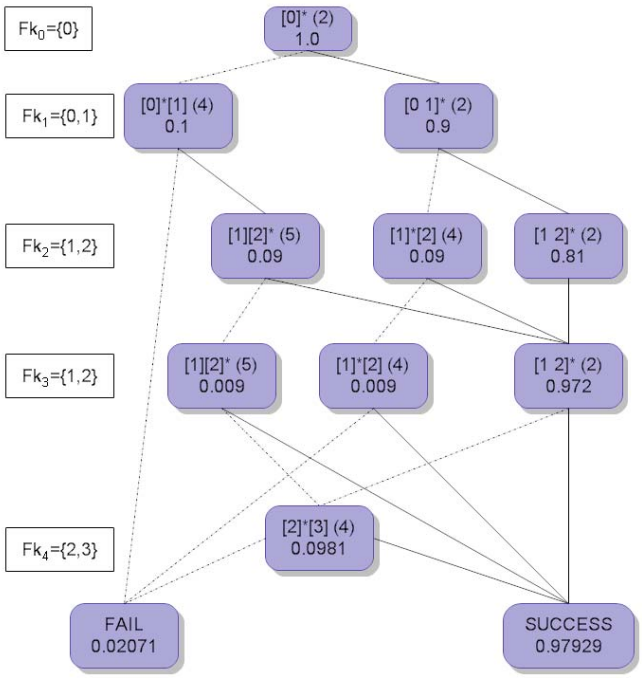

Figure 5: OBDD-A of *K*-REL for Sample Network

This is shown in the OBDD-A nodes of Fig. 5 with the partition numbers in parentheses and the probability of the node underneath. The partitions themselves are shown in the OBDD-A nodes as an aid to understanding. The boundary sets of the diagram are shown on the left-hand side. Note that the actual OBDD-A nodes are not linked; the links shown in the diagram are for ease of understanding only.

Note that *K*-REL is the probability stored in the success node; in this case 0.97929. For the sake of completeness the diagram shows that the failure node (which contains the probability 0.02071 = 1 – 0.97929) but the OBDD-A does not store failed partitions or their probability.

For this example, 10 non-terminal nodes are generated. The OBDD method [1] would store all 10 nodes, whereas the OBDD-A method stores at most 6 nodes (level 2 and 3) at any one time.

F. Space Required

As each node is processed, its child nodes are added to the level below and its parent node is deleted. Thus, the space required of OBDD-A is far less than that of the OBDD that stores all the nodes. When a new level of the OBDD-A is started the level below is still empty.

The maximum size of one level is $W_{th}(|F_k|)$, which is bounded by $(F_{max}^2+1)\times B_{Fmax}$[1] where W_{th} is the theoretical maximum number of marked partitions, F_{max} is the size of the maximum boundary set and B_{Fmax} is the Bell number of F_{max}[1]. Hence if a series of networks have a common connectivity structure (and hence constant F_{max}) then the width of any particular level (and hence the maximum number of nodes stored) will have a constant bound. Note that F_{max} depends on the structure of the network, and B_{Fmax} increases rapidly, so networks with a large boundary set will still require a large amount of memory.

For example, consider the 12×12 grid, which requires more than 780MB of memory using the OBDD method from [1]. Using the OBDD-A method, less than 235,000 nodes are stored at one time in two levels. Hence the OBDD-A method requires less than 3MB of memory. The processing time for both methods is comparable.

IV. Results

We implemented the OBDD-A algorithm in C++ using the GNU MP library for integers of arbitrary size, and tested it on a Pentium computer (2 Xeon 3.2GHz processors, 1MB cache, 2GB RAM). We firstly compared the reliability calculated with results in [1] to ascertain that the correct results were being calculated. We then computed 2-REL, *K*-REL and/or ALL-REL for each chosen network five times to generate an average CPU time recorded in seconds. The number of non-terminal nodes generated was recorded; terminal nodes were not recorded since their number is constant. All networks were taken from [1]; with K*x* denoting a fully connected network of *x* nodes, *W*×*L* denoting a grid network of width *W* and length *L*. The network *net.19* is Fig. 19 in [10].

The initial tests were designed to compare the performance of the algorithm for *K*-REL with |*K*| being 2, |V|/2 and V. However since [1] does not give information on how *K* is chosen for |*K*|=V/2, no reasonable comparison was possible and hence this is omitted. For |*K*| = 2, we chose *K* to be two nodes on the opposite ends of the network except for *net.19*, for which we used the source and target nodes given in [10].

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on February 25,2010 at 22:05:37 EST from IEEE Xplore. Restrictions apply.

TABLE 1: COMPARISON FOR 2-TERMINAL AND ALL-TERMINAL RELIABILITY

Network	2-Terminal					All-Terminal				
	OBDD		OBDD-A			OBDD		OBDD-A		
	Time	#Nodes	Time	#Nodes	Max	Time	#Nodes	Time	#Nodes	Max
net.19	0.09	5370	0.039	4916	488	0.12	4813	0.016	2060	159
3×12	0.05	440	0.005	475	12	0.04	290	0.004	270	7
5×5	0.05	2425	0.034	4948	226	0.07	1397	0.009	1256	54
2×100	0.02	792	0.009	1184	6	0.09	596	0.006	791	4

The results of these tests are shown in Table 1, with Max denoting the maximum number of nodes stored in memory for the network Note that the total number of nodes (#Nodes) generated varies between the implementations since, although both are ordered in a breadth-first manner, such an ordering is not unique. Despite this the time required for the OBDD-A is consistently smaller than those for the OBDD. Note also that the amount of memory required for the OBDD-A is far less than for the OBDD (Max vs. #Nodes for OBDD) since not all nodes are retained in memory at one time.

TABLE 2: CONSTANT MEMORY SIZE OF NETWORKS WITH IDENTICAL STRUCTURE

Network	|V|	|E|	F_{max}	Time	#Nodes	Max
3×12	36	57	4	0.004	270	7
3×100	300	496	4	0.014	2,470	7
3×1000	3000	4996	4	0.124	24,970	7
K7,7	7	21	6	0.007	759	129
K7,15	15	69	6	0.069	10,418	237
K7,50	50	279	6	0.346	53,048	237
K7,1000	1000	5979	6	7.824	1,210,148	237

In order to demonstrate that the maximum number of nodes stored in memory is constant for graphs of identical structure, we applied OBDD-A to 3×*L* grid networks and K*W*,*n* networks. Each K*W*,*n* is a network that has each of its *n* nodes connected to the following *W*-1 vertices. Since edges are undirected, each vertex is also connected to the preceding *W*-1 vertices. Note that if $n = W$ we have K*W*,*n* = K*n*, the fully connected graph of *n* vertices. It can be clearly seen in Table 2 that the number of vertices and edges does not affect the maximum number of nodes stored in memory at one time once the size has grown beyond a threshold (e.g., Max=237). Even the K7,1000 network with a total of over one million diagram nodes, only ever requires 237 to be stored at one time.

V. CONCLUSIONS

An improved method for computing *K*-REL and ALL-REL of a network using OBDD-A has been proposed in this paper. The method requires a comparable amount of processing time to the state of the art and uses exponentially less memory. For groups of networks that have the same connectivity structure the method uses a constant amount of memory regardless of the size of the network. This allows the computation of the reliability of extremely large networks.

For future research we will extend the application of boundary set notation to directed networks. We will also seek to allow multiple children per diagram node, creating an OMDD-A (multi-variate instead of binary) approach that can effectively compute networks for which both devices and links fail. Finally we will seek to apply the OBDD-A and/or OMDD-A to the problem of finding the expected hop count of a network.

REFERENCES

[1] G. Hardy, C. Lucet, and N. Limnios, "K-Terminal Network Reliability Measures With Binary Decision Diagrams," *IEEE Trans. Reliability,* vol. 56, pp. 506 - 515, Sept. 2007.

[2] F.-M. Yeh, H.-Y. Lin, and S.-Y. Kuo, "Analyzing network reliability with imperfect nodes using OBDD," in *Pacific Rim Int'l Symp. Dependable Computing*, 2002, pp. 89-96.

[3] F.-M. Yeh, S.-K. Lu, and S.-Y. Kuo, "OBDD-Based Evaluation of k-Terminal Network Reliability," *IEEE Trans. Reliability,* vol. 51, pp. 443-451, 2002.

[4] G. Hardy, C. Lucet, and N. Limnios, "Computing all-terminal reliability of stochastic networks with Binary Decision Diagrams," in *11th International Symposium on Applied Stochastic Models*, 2005.

[5] M. O. Ball, "Computational Complexity of Network Reliability Analysis: An Overview," *Reliability, IEEE Transactions on,* vol. 35, pp. 230 - 239, 1986 1986.

[6] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys,* vol. 24, pp. 293-318, 1992.

[7] J. Carlier and C. Lucet, "A decomposition algorithm for network reliability evaluation," *Discrete Applied Mathematics,* vol. 65, pp. 141–156, 1996.

[8] S. J. Friedman and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," in *24th ACM/IEEE Conf. Design Automation*, 1987, pp. 348-356.

[9] J. U. Herrmann, S. Soh, G. West, and S. Rai, "Using Multi-valued Decision Diagrams to Solve the Expected Hop Count Problem," in *IEEE 23rd Int. Conf. Advanced Information Networking and Applications Workshops*, Bradford, UK, 2009, pp. 419-424.

[10] S. Soh and S. Rai, "CAREL: Computer Aided Reliability Evaluation for Distributed Computing Networks," *IEEE Trans. Reliability,* vol. 2, pp. 199-213, 1991.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on February 25,2010 at 22:05:37 EST from IEEE Xplore. Restrictions apply.