2010 24th IEEE International Conference on Advanced Information Networking and Applications

Improving Reliability Calculation with Augmented Binary Decision Diagrams

Johannes U. Herrmann
Department of Computing
Curtin University of Technology, Perth, Australia
jherrmann@ieee.org

***Abstract* – The Augmented Ordered Binary Decision Diagram (OBDD-A) has been shown to be extremely efficient for reliability calculations, especially when combined with the Boundary Set method of partition representation. The existing OBDD-A follows the Boundary Set method closely, requiring the calculation of partition numbers large enough to exceed the capacity of native storage types in languages such as C++. By omitting the use of partition numbers the execution speed of the algorithm is increased, while the low memory usage of an OBDD-A is maintained. We compare the new OBDD-A to the existing version on a number of networks, showing that processing time for large networks increases significantly.**

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

I. INTRODUCTION

WITH increasing reliance on communication networks, measuring their reliability is an important aid in design and analysis. When communication links between network devices fail, critical devices may become unable to communicate with each other. The reliability (REL) of a communication network has been studied extensively [1-9] and measures the probability that the network meets the relevant standard of connectivity. This standard depends on which nodes must communicate with each other.

A number of different REL metrics exist. The most common are two-terminal reliability (2-REL), *K*-terminal reliability (*K*-REL) and all-terminal reliability (ALL-REL). The difference between these metrics is the number of devices that are required to be able to communicate with each other. In 2-REL, a source device must be able to communicate with a target (or sink) device. With *K*-REL a given set of *K* devices must be able to communicate with each other; generally this is assumed to mean that one source device can communicate with a set of *K*-1 others.[1] Finally, ALL-REL requires all devices to be able to communicate with each other. It can be seen that 2-REL is a special case of *K*-REL. Similarly, *K*-REL is a special case of ALL-REL. These metrics have been shown to be NP-Hard [1, 5].

A number of other metrics have been considered, although these are less common. For example 1-of-*S*-REL requires that at least one of the source devices out of a group *S* of source devices can communicate with the target device. While some work has been done on these [6], this paper does not address them directly. The methods discussed in [6] apply equally to the OBDD-A introduced in this paper.

[1] When communication links are undirected this is equivalent to the general definition of *K*-REL.

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 networks.

The Ordered Binary Decision Diagram (OBDD) [10] has been successfully used to compute 2-REL[3], *K*-REL[1-3, 11] 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 network states. Hardy, Lucet and Limnios [1, 4] use boundary sets [6] to efficiently represent OBDD states; we discuss boundary sets in detail in Section II.C. Even with the use of isomorphism and boundary set notation, the number of OBDD nodes generated for very large networks (e.g., 7×1,000 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 boundary set algorithms [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 which may require more than 780MB of memory [8].

A more memory efficient solution [8] has been proposed, utilizing the partition numbering method [1] with an Augmented OBDD (OBDD-A). This solution combines the efficient partition notation of network states with the memory efficiency of the OBDD-A. In particular, for networks of varying sizes but identical connectivity structure the amount of diagram nodes generated has a constant bound. For example, for the 3×1000 grid the OBDD-A [8] stores at most 7 nodes at one time out of a total of 24,970 nodes generated.

The disadvantage of using partition numbering is that such numbers quickly grow to be larger than can be stored in the standard data types of languages such as C++. This requires the use of a library such as GNU MP [12] for dealing with numbers of arbitrary size and greatly slows the processing speed of the program. In addition, the partition numbers must be calculated whenever a new partition is created, and partitions must be recreated from numbers in

1550-445X/10 $26.00 © 2010 IEEE
DOI 10.1109/AINA.2010.148

order to be processed. This creates unnecessary processing overhead.

In this paper we present an exact algorithm using an Augmented OBDD (OBDD-A) which does not use partition numbering. The algorithm computes REL metrics for networks whose undirected communication links can fail but which have perfect devices. We compare this approach to the existing OBDD-A algorithm [8] that has been shown to perform comparably to the OBDD-based boundary set approach [1] in terms of processing time but use far less memory.

This paper is organized into five sections. We introduce the background of the REL metrics and introduce the notation used in Section II. The OBDD-A approach is discussed in Section III, with and without the use of partition numbering. The implementation of these methods and comparisons are discussed in Section IV and conclusions and the scope of future work in Section V.

II. Background and Notation

A. Network Model

We model a communication network 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 these 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 vertices V = {v_0, v_1, …, v_{n-1}} be ordered in increasing distance (number of hops) from a source vertex, v_0, except that the target vertex is labeled v_{n-1}. If the network has no distinct source, choose a vertex in K as v_0. If considering ALL-REL, any vertex can be chosen as v_0. If no distinct sink exists, v_{n-1} should be in K for K-REL, and can be any vertex for ALL-REL. 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 i>j. The sample network in Fig. 1 shows an example of the above ordering.

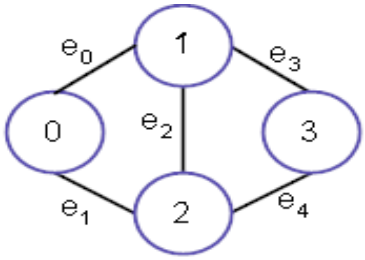

Figure 1: Sample Network

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

$$\Pr(\Omega)=\prod_{e_i\in E_\Omega} p_i \prod_{e_i\notin E_\Omega} q_i \,. \qquad (1)$$

Each Ω is associated with a sub-graph G_Ω = (V_Ω,E_Ω) where V_Ω are the vertices reachable from v_0 (or S) via the edges in E_Ω. Let Ω_G be the set of success states for graph G. For K-REL, Ω is a successful state if K⊆V_Ω and for ALL-REL, Ω is a successful state if V_Ω=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.

In this paper we use the words 'device' and 'communication link' to refer to the physical network. We use 'vertex' and 'edge' to refer to the mathematical model representing the network. Finally we use 'node' and 'link' to refer to the decision diagram representing the connectivity of the network.

B. Ordered Binary Decision Diagrams

The OBDD is based on the Shannon decomposition [10]. 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_{2i+2} representing the case when e_k is UP and negative child N_{2i+1} 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 $x < k$. We say that level k *decides* edge e_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 [13]. A breadth-first ordering is commonly used. We use the ordering of vertices as described in Section A of this chapter, and edges e_x=(v_i,v_j) are decided in increasing order of i and then j. This is equivalent to the ordering used by Hardy *et al*. [1, 4];

C. Boundary Sets

In order to calculate REL, an OBDD 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⊆V, is composed of only those vertices required to encode the current network state. For the vertices in the boundary set, the algorithm must record the connections (via paths of UP edges) between them. 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, F_k can be defined as F_k = {v_x| v_x is an endpoint of both e_y and e_z, with $y\le k$ and $z\ge k$}. 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*. We write both boundary sets and partitions using only the subscripts of the vertices.

For example the edges of the graph shown in Fig. 1 are ordered (0,1) (0,2) (1,2) (1,3) (1,4). The OBDD-A for this

network is shown in Fig. 2, including all partitions and boundary sets. For level k=2 of the diagram, the edge being decided is (1,2) and hence the boundary set is {1,2}.

If K={v_0,v_3} the possible partitions of the boundary set at this level are [1 2]*, [1]*[2], [1][2]*. We refer to [1] and [2] as the blocks of the partition [1]*[2] and denote the marking of the block [1] by an asterisk. Any partition that has no marked blocks (*e.g.* [1][2]) has no connection with vertex 0 and hence is failed.

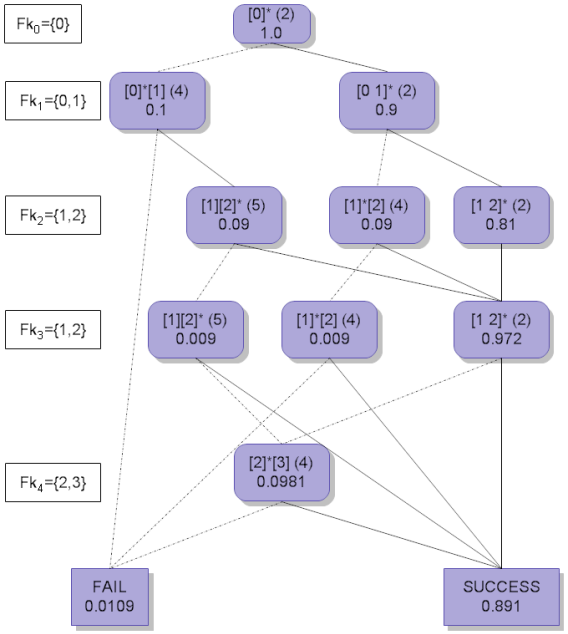

Figure 2: OBDD-A of Sample Network

Partitions are represented by vectors of size $|F_k|$ where the i^{th} position in the vector contains the number of the block 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. Modified methods that can be used for ALL-REL and K-REL (and hence 2-REL), are described in [8].

D. Augmented Decision Diagrams

The Augmented Ordered Binary Decision Diagram (OBDD-A) is an OBDD that stores network state information in each node. For an application such as REL only the probability of each state needs to be stored [8] with other applications requiring more information. For example, for the Expected Hop Count (EHC) the minimum path lengths to each active vertex are also stored [6]. The hybrid OBDD-A in Fig. 2 is for reliability calculation only.

Early versions [6, 9] of the OBDD-A stored the state of the network using a notation that tracked paths between source vertices and the vertices in the boundary set. While this notation allows directed networks to be analyzed, it is not as quick as a standard OBDD using boundary set numbering [1]. For this reason the hybrid OBDD-A [8] was created to store partition numbers instead of path information.

The hybrid OBDD-A was shown to be comparable in processing speed to the partition number OBDD but to use significantly less memory [8]. For the sake of brevity we will omit the 'hybrid' before each OBDD-A for the rest of this paper since we will discuss the hybrid OBDD-A exclusively.

III. The Hybrid OBDD-A Algorithm

A. Introduction

This section details the structure of the OBDD-A and how it is used to compute REL. Part B details the structure of the OBDD-A node, and hence the diagram itself. Part C discussed the different types of OBDD-A nodes and node isomorphism. The method for constructing an OBDD-A and obtaining REL from it is discussed in Part D, and illustrated with an example in Part E. Finally Part F briefly discusses the space required by the OBDD-A algorithm.

B. OBDD-A Node Structure

An OBDD-A is an OBDD whose nodes contain additional information [9]. For each metric, an OBDD-A node stores both the state represented by that node and information needed to compute the metric. An OBDD-A node does not store pointers to child or parent nodes since such linkages are never traversed.

For example an OBDD-A used to compute the Expected Hop Count stores information on path length and state probability [9] in addition to network state information. For REL, the OBDD-A stores only the network state information and its probability.

The state of the network at level k is represented by partitions of F_k, with each OBDD-A node representing one network state and hence having one unique 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; if two OBDD-A nodes have the same partition then they are isomorphic. For REL, the OBDD-A node N_i on level k of the diagram has the format ($part_i$, Pr_i).

The OBDD-A introduced in [8] uses the partition number to encode the network state. This means that $part_i = PN_i$. The OBDD-A introduced in this paper stores the partition directly. We refer to new OBDD as OBDD-A2 to avoid confusion and use the notation OBDD-A/2 when referring to both diagrams simultaneously. For OBDD-A2, $part_i$ = (B_i, M_i), where B_i is an array that allocates each member of F_k to a block of the partition, and M_i is a j-bit binary number that has a value of 1 for each of the j blocks that is marked.

For example, the node representing the network state of partition [0]*[1] on level 2 of the diagram in Fig. 2 is stored as N_1=(PN_1=4, 0.1) for OBDD-A and as N_1= ((B_1=[0 1], M_1=[1 0]), 0.1) for OBDD-A2 when computing K-REL. When computing ALL-REL partitions are not marked, hence N_1=(PN_1=1, 0.1) for OBDD-A and (B_1=[0 1], 0.1) for OBDD-A2. Note that the partition number of partition [0]*[1] is 4.

When a new node is created, its probability is calculated from the probability of the parent node. For node N_i on level k, the probability of the child nodes are $Pr_{2i+1}=Pr_i \times q_k$ and $Pr_{2i+2}=Pr_i \times p_k$ for the negative and positive child respectively. The negative child represents edge e_k being DOWN while the positive child represents e_k being UP.

When two OBDD-A/2 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 probability of the success terminal node is the appropriate REL metric.

C. OBDD-A Node Type and Isomorphism

An OBDD-A/2 node is either terminal or non-terminal and a terminal node is either successful or failed. For *K*-REL a node is failed if it contains a marked partition that is empty and is successful if it has a single marked partition containing v_{n-1}. For ALL-REL all nodes are considered marked and hence a node is failed if it has an empty partition and successful if it contains a single partition including v_{n-1}.

When boundary set notation was introduced [7], comparison between partitions was implemented using partition numbers. Unique methods exist for translating between partitions and partition numbers. Earlier versions of the hybrid OBDD-A [8] stored partition numbers instead of the partitions themselves. The advantage of this approach is that the partition numbers are relatively efficient to store and compare.

The main drawback of using partition is the translation between the partitions and partition numbers and back again. While node comparison is efficient with this method, the translation adds a large overhead. In addition, partition numbers grow quickly and thus require more space than the standard C++ long integers can store. This means that a special library must be used to handle integers of arbitrary size. Such a library adds its own processing overhead.

The disadvantage of storing partitions instead of partition numbers is that node comparisons are more complex. Two OBDD-A/2 nodes $N_{i,k}$ = ($part_i$, P_i) and $N_{j,k}$ = ($part_j$, P_j) are considered equal (isomorphic) if the partitions are equal ($part_i$ = $part_j$). For OBDD-A2 this requires up to $|F_k|$ comparisons between integers and a bitwise comparison of two binary numbers instead of the single comparison between large numbers required for OBDD-A.

D. Constructing an OBDD-A

Fig. 3 shows the OBDD-A/2 algorithm for computing REL. The algorithm is initialized with $F_0=\{0\}$ and the initial node N_0 = ([0]*, 1.0) stored in the root node of the BDD and on the current queue, Q_C. Note that for ALL-REL, all partitions are considered marked so we do not need to explicitly mark any; hence N_0 = ([0], 1.0).

The exact form of N_0 depends on whether OBDD-A or OBDD-A2 is used. For OBDD-A N_0 = (2, 1.0) for *K*-REL and (1, 1.0) for ALL-REL. For OBDD-A2, N_0= (([0],[1]), 1.0) for *K*-REL and ([0], 1.0) for ALL-REL.

At each level nodes are removed from Q_C and their partitions are processed to produce two child nodes. Successful child nodes have their results stored and non-terminal children are added to the next queue. When the loop exits, REL has been calculated.

```
Build (G )
k=0; F0={0};
Create node N0 = ([0]*, 1.0) and hash to QC;
Compute F1;
while (QC and QN are not both empty ) do
    if (QC is empty) then
        Increment k; // Working on new level
        Compute Fk+1;
        Move contents of QN into QC; // QC now non-empty
    Remove the first node, Ni from QC;
    create N2i+1 from Ni to represent ek DOWN;
    create N2i+2 from Ni to represent ek UP;
    for each ( N in {N2i+1 N2i+2} ) do
        if ( N is successful )
            Store Pr(N).
        else if ( N is not a failure ) do
            if ( N is isomorphic with M on QN ) do
                Merge N into M.
            else
                Add N to QN.
```

Figure 3: Computing REL using OBDD-A/2

Queues are used for ease of removing nodes from Q_C, however we use a hash table to link to nodes in Q_N to improve the performance of checking for node isomorphism. For OBDD-A, we use the partition numbers as a perfect hash. The hash for OBDD-A2 is not perfect, which means that several node comparisons may need to be performed.

As each success node is found, its probability is added to a running total. When the algorithm completes, REL has been calculated. This total may be stored in a success node, although our implementation keeps it separate.

E. OBDD-A Example

To illustrate the OBDD-A/2 algorithm, we apply it to the sample network in Fig. 1 with a single source (v_0) and target (v_3). We assume that each edge has a 0.1 probability of failure. The root node is initialized to be (2, 1.0) for OBDD-A and (([0],[1]), 1.0) for OBDD-A2, which represents the single vertex in $F_0=\{0\}$ being in block 0. This vertex is connected to the source (in fact it is the source) and hence the block is marked. The probability of this state is 1 since no decisions have been made yet. We compute $F_1=\{0,1\}$ from $e_0=\{0,1\}$.

We enter the loop and remove the only node from Q_C. The only node on level 0 is the root node, N_0, which contains the partition $[0]^*$. This partition is used to create children $[0]^*[1]$ and $[0\ 1]^*$, which are stored in child nodes N_1 and N_2 respectively. The probability of the child nodes is $q_0 \times 1.0 = 0.1$ for N_1 and $p_0 \times 1.0 = 0.9$ for N_2.

For OBDD-A we have N_1=(4, 0.1) and N_2=(2, 0.9). For OBDD-A2 we have N_1=(([0 1],[1 0]),0.1) and N_2=([0 0], [1], 0.9).

The OBDD-A/2 resulting from this process is shown in Fig. 2, with the partition numbers in parentheses and the probability of the node underneath. The boundary sets of the diagram are shown on the left-hand side. Note that the actual OBDD-A/2 nodes are not linked; the links shown in the diagram are for ease of understanding only.

TABLE 1: COMPARISON OF PROCESSING TIME

Network	2-Terminal			All-Terminal		
	OBDD-A	OBDD-A	OBDD-A2	OBDD-A	OBDD-A	OBDD-A2
net.19	0.09	0.039	0.042	0.12	0.016	0.039
5x5	0.05	0.034	0.09	0.07	0.009	0.08
2x100	0.02	0.009	0.005	0.09	0.006	0.006
K,7,1000	na	41.513	22.293	na	7.824	2.787
7x1000	na	495.33	59.159	85.18	53.51	23.168

The diagram in Fig. 2 shows nodes that contain both the partition and the partition number. It should be noted that the OBDD-A contains only the partition number and the OBDD-A2 contains only the partition itself.

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 the failure node (which contains the probability 0.02071 = 1 – 0.97929) but the OBDD-A/2 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/2 method stores at most 4 nodes at any one time.

F. Space Required

As each node on level *k* is processed, its child nodes are added to level *k*+1 and its parent node deleted. Thus the number of nodes kept in memory at any one time is always less than two full levels..

The details of this and experimental results for OBDD-A are discussed in [8]. The bound on the number of nodes per level depends on the structure of the network and increases rapidly for networks with larger boundary set sizes. Hence networks with a large boundary set may require a large amount of memory for a single level of the diagram.

Because the part of the algorithm that generates nodes is identical between OBDD-A and OBDD-A2, both algorithms generate the same number of nodes. Isomorphism is based on comparing partitions, whether directly or via partition numbering, so the same nodes are isomorphic for both algorithms. Hence the total number of nodes processed is the same for both algorithms.

This means that the OBDD-A2, like the OBDD-A, will store only a maximum of 7 out of 24,970 nodes for the 3×1000 grid and 237 out of 1,210,148 nodes for K7,1000 [8]. Similarly it has a constant upper bound on the number of diagram nodes generated for networks of constant internal connectivity and arbitrary size.

IV. RESULTS

We implemented the OBDD-A algorithm for solving *K*-REL and ALL-REL in gcc and tested it on a Pentium computer (2 Xeon 3.2GHz processors, 1MB cache, 2GB RAM). For each chosen network the test was run five times in order to generate an average CPU time, recorded in seconds.

We tested the implementation on a number of networks. Most networks were taken from [1] to enable a better initial comparison. Larger networks were taken from [8]; in particular, network K7,1000 denotes a 7-connected network of 1000 nodes and $W \times L$ is a grid network of width W and length L. A *K*-connected network is one where each vertex is connected to the next *K* vertices (and hence the previous *K* vertices for an undirected network). The network *net.19* is Fig. 19 in [5].

A KW,n network is one 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 KW,n = Kn, the fully connected graph of n vertices. We use KW,n and grid networks because we can generate networks of arbitrary size without changing the internal connectivity and hence F_{max}.

The results of a sample of these tests are shown in Table 1, with the OBDD results being from the algorithm by Hardy, Lucet and Limnios [1].

The initial testing on small networks showed that OBDD-A and OBDD-A2 were within an order of magnitude of each other. For these smaller networks, OBDD-A often outperformed OBDD-A2. In these networks, the library for large numbers is not required and the calculation of the partition numbers is not as costly due to smaller partitions.

The truth of this argument can be seen in the comparison on the 2x100 grid. The maximum number of nodes on one level of the OBDD-A is 3, so partition numbers are extremely low and conversion between partitions and partition numbers is fast. Hence the pattern of similar performance remains.

For larger networks with a greater number of partitions, the results are noticeably different. For such networks, the calculation of partition numbers and use of the GNU library greatly outweighs the hashing and node comparison. As can be seen for the 7x1000 grid and the K7,1000 networks, OBDD-A2 outperforms OBDD-A by a significant amount.

Both OBDD-A and OBDD-A2 outperform the state of the art OBDD-A algorithm [1] for all networks tested.

V. CONCLUSIONS

We have improved on our hybrid OBDD-A algorithm [8] by removing the need for large partition numbers. This improvement results in a significant improvement in processing speed for large networks. Such improvements are important in order to allow the analysis of more complex networks. The improved OBDD-A generates the same

amount of nodes as the original, and hence has a constant bound for families of networks with identical connectivity.

For future research we will extend the application of boundary set notation to directed networks. The hybrid OBDD-A will also be adapted to solve performability metrics such as the Expected Hop Count and Expected Message Delay. 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.[14]

REFERENCES

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

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

[3] F.-M. Yeh, *et al.*, "OBDD-Based Evaluation of k-Terminal Network Reliability," *IEEE Trans. Reliability,* vol. 51, pp. 443-451, 2002.

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

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

[6] J. U. Herrmann, *et al.*, "On Augmented OBDD and Performability for Sensor Networks," *Int'l J. Performability Engineering,* accepted for publication 2009.

[7] J. Carlier and C. Lucet, "A Decomposition Algorithm for Network Reliability Evaluation," *Discrete Applied Mathematics,* vol. 65, pp. 141-156, 1996.

[8] J. U. Herrmann and S. Soh, "A Space Efficient Algorithm for Network Reliability," presented at the 15th Asia-Pacific Conf. Communications (APCC2009), 2009.

[9] J. U. Herrmann, *et al.*, "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] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys,* vol. 24, pp. 293-318, 1992.

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

[12] Sept. 25). *The GNU MP Bignum Library*. Available: http://gmplib.org/

[13] 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.

[14] R. K. Ahuja, *et al.*, "Computational investigations of maximum flow algorithms," *European Journal of Operational Research,* vol. 97, pp. 509-542, 16 March 1997.