Using Multi-valued Decision Diagrams to Solve the Expected Hop Count Problem

Johannes U. Herrmann, Sieteng Soh, Geoff West
Department of Computing
Curtin University of Technology
Bentley, Western Australia
e-mail: jherrmann@gmail.com

Suresh Rai
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA, USA

Abstract—The Expected Hop Count (EHC) of a computer communication network has so far been computed for network models that consider only device or link failure, but not both. We introduce an Augmented Ordered Multi-valued Decision Diagram (OMDD-A) to obtain the EHC of a network in which both devices and links may fail. The OMDD-A approach can compute the EHC of a 2x100 grid network with 2^99 paths, which is unsolvable using existing techniques. We show that OMDD-A generates significantly fewer nodes than the corresponding ordered binary decision diagram, leading to large reductions in processing time.

Keywords- expected hop count, imperfect nodes, imperfect links, multi-value decision diagram, network reliability

I. INTRODUCTION

Reliability and performability are important for the design and maintenance of computer communication networks, road transport, power distribution and many other networks. Both links and nodes of the network are subject to failure, which impacts on performance. Network reliability (REL) has been studied extensively [1]-[5] but only considers the network connectivity. The performance of a network is often measured in terms of how long it takes messages to travel. The Expected Hop Count (EHC) [6]-[9] computes the number of vertices that a message is expected to pass through on the shortest active path from source to target.

AboElFotoh [6] has shown that computing EHC for the general network, where vertices may fail but edges are perfect, is #P-hard. A breadth-first-search combined with the factoring theorem was proposed to calculate EHC for networks with a single source and target; however the solution does not scale well with large networks. Soh, et al. [7] proposed a more efficient sum-of-disjoint products (SDP) technique that generates all network minpaths, sorts them in increasing cardinality, and then applies an SDP technique [1]. It is shown [7] that the SDP technique is significantly faster than the factoring approach. Neither of these approaches [6],[7] is feasible for computing EHC for networks with an extremely large number of paths, such as the 2x100 grid network with 2^99 paths. Brooks, et al. [8] use random graph models to approximate EHC in mobile WSN for EHC with a single source and target, but assume fallible edges and perfect vertices. Recently, AboElFotoh, et al. [9] extended the factoring approach [6] to solve EHC for multiple sources.

Currently no work considers the more general case where both links and devices can fail. Since EHC is a #P-Hard problem [6], existing solutions are exponential in the number of vertices or edges of the network graph. Ball, et al. [3] proposed transforming the graph model G(V,E) into one that considers only edge failures, G'(V',E'). Since |E'| = |V|+|E|, the complexity of computing EHC from G' is exponential in the sum of the edges and vertices of the original graph.

The Ordered Binary Decision Diagram (OBDD) [10] has been considered one of the most efficient methods for representing Boolean functions [11]. OBDDs have been used to solve REL [2]-[5] but do not store sufficient information to solve EHC. Decision Diagrams (DDs) with more than two output values were first suggested in [12] for the simulation of circuits. This concept was extended in [13], which presented a formalization of the Multi-variable Decision Diagram (MDD) and analyzed its properties. Both [14] and [15] produced formal and efficient implementations of the MDD. Research has shown that the MDD uses less space than the equivalent BDD over a wide variety of benchmarks [15],[16]. MDDs have been applied to a number of areas, including circuit design and verification [14], Petri nets [17] and fault tolerant systems [18]. However, to the best of our knowledge, MDDs have not yet been applied to REL or other performability measures such as EHC.

In this paper, we propose an Augmented Ordered MDD (OMDD-A) to compute the EHC of a network when both edges and vertices are susceptible to failure. Our simulations in Section IV show that its performance is far superior to that of the binary equivalent; the Augmented OBDD [19].

The layout of the paper is as follows. Section II discusses terminology and reviews DDs and their use in REL. Section III present the OMDD-A to compute EHC. We give results in section IV and conclusions and future work in Section V.

II. BACKGROUND

A. Network Model and Terminology

We model a computer communication network (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. A vertex v_i or edge e_j is said to be UP (DOWN) if it is functioning (failed). Let p_v (p_e) be the operational probability of vertex v_i (edge e_j) and assume all failures are statistically independent.

Let n=|V|, and let the vertices (v_0, v_1, ..., v_n) of V be ordered in increasing distance from the source vertex, v_0.
with the target vertex \( v_t \) always labeled as \( v_{n-1} \). When two or more vertices have the same distance from \( v_0 \), they are ordered arbitrarily. Let \((v_i,v_j)\) denote a directed (undirected) edge between vertices \( v_i \) and \( v_j \), with \( i \neq j \) for each \((v_i,v_j)\). Fig. 1 shows an example network that illustrates such an ordering.

![Sample Network](image)

A path \( P_i \) is a sequence of UP vertices \((v_{a_i},v_{b_i}, \ldots, v_{k_i})\) in \( V \) such that there exist UP edges \( e_{a_i} = (v_a,v_{b_i}) \) or \( \{v_a,v_b\} \), \( e_{b_i} = (v_{b_i},v_c) \) or \( \{v_{b_i},v_c\} \), etc. A reaching path \( P_i \) to \( v_c \) is a path from \( v_0 \) to \( v_c \) where each vertex in \( P_i \) is traversed only once. A minpath is a reaching path to \( v_c \). A diagram path to node \( N_i \) is a path in a DD starting at the root and leading to \( N_i \). In this paper, we use node and link to refer to the elements of a DD, and vertex and edge for those of the CN.

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

\[
Pr(\Omega) = \prod_{v \in V_U} \prod_{v \in V_U} (1-p_v) \prod_{e \in E_U} (1-q_e),
\]

since all failures are assumed to be statistically independent. A state is a success state if it contains at least one minpath. There are \( 2^{|V|+|E|} \) network states in \( G \), but REL and EHC are computed only from the set of all success states, \( \Omega_s \).

In addition to the success state information, computing the EHC requires the length of each success state, \( \Omega \in \Omega_s \), denoted as \( 1 \leq L(\Omega) \leq n-1 \). Thus, \( L(\Omega) \) is the length (the number of hops or hop count) of the shortest minpath contained in \( \Omega \). We assume that the routing protocol in the network always finds the shortest available minpath [9]. When this minpath is unavailable the router finds the next possible shortest minpath. Formally, the EHC is given by:

\[
EHC = \frac{\sum_{\Omega \in \Omega_s} (L(\Omega) \times Pr(\Omega))}{\sum_{\Omega \in \Omega_s} Pr(\Omega)}
\]

### B. Multi-valued Decision Diagrams

Figures 2(a) and 2(b) to 2(e) show the BDD and MDDs, respectively of a Boolean function \( f = e_0e_2 + e_1e_3 \). Each non-terminal node (a circle in Fig. 2) in a DD represents the evaluation of one or more Boolean variables, with one subtree representing each possible combination of values of these variables. In an Ordered DD (ODD), the variable order is fixed for all branches/paths of the diagram. Following a path from the root node, variables are decided in a given order until a value is returned. This value is stored in the terminal node of that path (a square in Fig. 2).

Links in Fig. 2 are labeled with the subscript of the variables that are UP, or are labeled with an ‘X’ in the case when no variables are UP. When one link represents multiple combinations of variables, these combinations are separated by commas. For example, in Fig. 2 (c) the label on the leftmost edge leaving the root node is “X,0,2”. This label represents three cases; all variables being DOWN (“X”), variable \( e_0 \) being UP and \( e_1 \) and \( e_2 \) being DOWN (“0”), and \( e_2 \) being UP and \( e_0 \) and \( e_1 \) being DOWN (“2”) respectively.

Each node in an Ordered BDD (OBDD) such as in Fig. 2(a) represents one Boolean variable, and thus has two children. By comparison, a node in an Ordered MDD (OMDD) may represent a group of several variables, and hence it can have more than two children. For example, the root node of the OMDD in Fig. 2(b) is evaluating a group of three variables \( (e_0,e_1,e_2) \) and thus it has 8 children.

An OMDD has a fixed variable grouping at each level of the diagram, although the number of variables in each grouping does not have to be identical. Nagayama and Sasao [16] showed that, over a wide variety of benchmarks, that such an OMDD uses less space than the equivalent OBDD. Note that each level of an OMDD represents the evaluation of one particular variable group.

The efficiency of a DD implementation is measured by its number of nodes [17] and its depth [16]. A good variable ordering can effectively reduce the number of nodes and depth of an OMDD [16]. A better variable ordering may result in more isomorphism between nodes in the diagram.

Two non-terminal (terminal) nodes are isomorphic if they have equivalent sub-trees (they produce identical outputs). Merging isomorphic nodes reduces the size of the ODD; this operation effectively prunes one of the sub-trees. For example, we obtain the smaller ODD in Fig. 2(c) by merging the three isomorphic nodes (labeled \( g_1 \)) and the terminal

![Figure 2: An OBDD (a) and OMDDs (b)-(e) representing \( f = e_0e_2 + e_1e_3 \)](image)
nodes in Fig. 2(b). Finding the optimal variable ordering has been shown an NP-Complete problem [20].

The MDD is a natural extension of the BDD, in that it has d terminal nodes labeled 0 to d-1, and similar labels are possible for the outputs of each MDD node. Each non-terminal node has a fixed number (d) of outputs, although some implementations remove redundant outputs. A MDD that has an equal number of outputs for every non-terminal node (before considering isomorphism) is referred to as homogeneous; otherwise it is a heterogeneous MDD.

Fig. 2 (e) shows the homogeneous OMDD with variable grouping g₀={e₀,e₁} and g₁={e₂,e₃}. The OMDD has four nodes, which is more efficient than the equivalent six node OBDD in Fig. 2(a). Fig. 2(c) shows a heterogeneous OMDD with groups g₀={e₀,e₁,e₂} and g₁={e₃}, and Fig. 2(d) shows another with g₀={e₀} and g₁={e₁,e₂,e₃}; these OMDDs have less nodes than the homogeneous OMDD in 2(e). In this example the OMDD with g₀={e₀,e₁,e₂} and g₁={e₃} gives the optimal result. Unfortunately, finding the optimal grouping is even more difficult than the NP-complete problem of finding the optimal variable ordering [20].

C. Decision Diagrams and Network Reliability

The Boolean function f=e₀e₂+e₁e₃ can represent the path set {e₀e₂, e₁e₃} of a network if only edges are considered. The OBDD in Fig. 2(a) is used to compute REL for this network with perfect vertices. In the application of the DD technique to REL [2] the probability that the network is connected is given by tracing paths upwards from the success terminal nodes and multiplying by the probability of the variable(s) being UP or DOWN as appropriate. Since each traversed path represents a disjoint event the probability of each such path is summed to give REL.

As an example, consider f = e₀e₂ + e₁e₃ as the reliability function of a network (for paths e₀e₂ and e₁e₃), with each edge having a probability of 0.9 of being UP. Thus, the REL of the network can be obtained from Fig. 2(a) by following each diagram path from the root to the success node (marked as 1), and multiplying by 0.9 for each positive edge and 0.1=(1-0.9) for each negative edge (marked with an 'X'). The right-hand diagram path is e₀e₁e₂ which has a probability of 0.729. The other diagram paths are e₀e₂e₁, e₀e₁e₂e₃ and e₁e₀e₁e₃. Note that the nodes of the DD do not contain any information other than what is inherent through their position in the diagram. Therefore the existing OBDD approaches [2],[4],[5] cannot be used to compute EHC, since this requires path length information (See Section II A).

OBD have been efficiently used for computing REL [2],[5], and fault covering and tolerance [18]. Kuo, et al. [2] have proposed a recursive EED-ISO algorithm to compute REL for a network with perfect vertices and failed edges. The use of node isomorphism in OBD makes EED-ISO able to compute REL for a 2x100 grid network with 2⁷⁰ minpaths. Yeh, et al. [5] use the OBDD for calculating REL for a one-to-many network, where one vertex must be connected with k-1 other vertices of the network. k-1 different REL are calculated, which are then combined to give the k-terminal reliability. Although these approaches [2],[5] are efficient for computing REL, they are not useful for computing EHC. The method in [2], for example, only generates OBDD nodes to take advantage of isomorphism through hash table lookups, but never explicitly links them into a diagram. The approach can compute REL by traversing the OBDD nodes, but cannot calculate the EHC, whose computation requires path length information.

III. AUGMENTED DECISION DIAGRAMS

A. Augmenting Decision Diagram Nodes

For computing EHC, each OMDD node requires more information than just its position in the diagram. In particular each node, Nᵢ, must store the state(s) of the CN that it represents, given the decisions that have been made in the diagram path(s) that lead to that node. We call the heterogeneous OMDD that comprises such nodes an Augmented OMDD (OMDD-A).

State information includes a set VIᵢ of vertex components, Mᵢ={(vᵢ, Lᵢₒ), (vᵢ, Lᵢᵢ), …, (vᵢ, Lᵢₙ), Pᵢ)}. Each pair (vᵢ, Lᵢₒ) ∈ Mᵢ denotes an undecided vertex vᵢ known to be reachable from vₒ along with the length Lᵢₒ of the shortest reaching path known. The probability Pᵢ is the probability of being in the network state represented by Mᵢ. Section III E describes how Pᵢ is computed. For each VI, we define a vertex set, VSᵢ={vᵢₒ, vᵢ₁, … vᵢₙ}. Each VI also contains a set CIᵢ of conditional paths of the form (vᵢₒ, vᵢᵢ, L) where L is the length of the shortest path between undecided vertices vᵢₒ and vᵢᵢ. When the first vertex, vᵢₒ, of a conditional path is reachable (i.e. vᵢₒ ∈ VSᵢ) and decided UP, (vᵢₒ, vᵢᵢ, L) is moved to VIᵢ as a minpath to vᵢᵢ by appending it to the minpath to vᵢₒ.

The augmenting information in each OMDD-A node is computed from its parent nodes (discussed in Section III E). Each separate component represents a different network state and the probability of being in this state. Because the OMDD-A stores state-based information, we can construct each level using only the nodes of the previous level. It also allows the tracking of information such as path length needed to compute EHC. Note that the number of children does not affect the size of a node since links are not explicitly stored. However, as the number of components in a node increases so does the size of the node. This size increase is generally manageable since less than two levels of nodes are kept in memory at any one time.

B. Variable Order and Grouping

The depth of an OMDD can be reduced by using variable partitioning [16]. For the OMDD-A algorithm, we group each vertex vᵢ with its adjacent edges. Note that both directions of undirected edges are considered at one time, hence any edge will only be in one variable grouping. Thus a node that decides a group consisting of vᵢ with d adjacent ungrouped edges has 2ᵈ⁻¹ children. An undirected edge is grouped with the first endpoint in the ordering; the use of conditions ensures that it only has to be considered once.

Note that those 2ᵈ children for which vᵢ is DOWN represent identical network states to the child for which all edges are DOWN and vᵢ is UP. To reduce the number of
nodes of the OMDD-A, we only generate one negative child, which represents the network state of the $2^d+1$ identical children. Hence an OMDD-A node deciding the grouping of $v_j$ and $d$ adjacent edges has only $2^d$ children. The link connecting a parent node to its negative child is referred to as the negative link and is marked with an ‘X’. For example, consider the network in Fig. 1. The OMDD-A partition for this network is $g_0 = \{(v_0, e_0, e_1), g_1 = \{v_1, e_2, e_3\}$ and $g_2 = \{v_2, e_4\}$. The negative child for $g_0$ represents the case when either $v_0$ is DOWN, or $e_0$ and $e_1$ are both DOWN.

C. OMDD-A Node Types

We consider two types of OMDD-A non-terminal: terminal, and non-terminal. Our approach processes each non-terminal node in a breadth-first fashion and completes when there are no more such nodes. A terminal node can be either a success node (whose value is a hop count of 1 or more) or a failure node (with a value of 0). The REL and EHC are computed from the reaching path probabilities contained in all success nodes (discussed in Section III E).

![TestNode function](image)

A failure node has no sub-trees containing a success node. It is favorable to detect failure nodes as early as possible, since REL are computed only from success nodes. However the processing cost of testing for failure must be kept to a minimum. $N_i$ is a failure node if $V S_i = \{\}$; if the node has no information on any undecided vertices then no new vertices (including the target) will be reached. However, a failure node may have a non-empty VS, and detecting such nodes is computationally expensive. Our TestNode function in Fig. 3 returns a value 1, 2, or 3 if $N_i$ is a failure, success, or non-terminal node, respectively. Note that a node that contains multiple components is not a success node if any of the components do not meet the criteria. The OMDD-A implementation removes successful components from the node as soon as they are detected, and stores their information to avoid unnecessary overhead.

![Unmerged OMDD-A](image)

D. Node Isomorphism

We consider non-terminal nodes $N_i$ and $N_j$ at the same level of an OMDD-A isomorphic iff $V S_i = V S_j$ and $C I_i = C I_j$. We check isomorphic nodes only from their equal VS and CI to tradeoff between the number of isomorphic nodes and the processing time complexity per node.

Two isomorphic nodes $N_i$ and $N_j$ can be merged into one node that keeps the VS and CI of merged nodes; without loss of generality let the resulting node be $N_i$. We say components $M_\alpha = \{(v_\alpha, L^\alpha_0), (v_\alpha, L^\alpha_1), \ldots, (v_\alpha, L^\alpha_i), P^\alpha\}$ and $M_j = \{(v_\alpha, L^\alpha_0), (v_\alpha, L^\alpha_1), \ldots, (v_\alpha, L^\alpha_i), P^\alpha\}$ are equal and write $M_j = M_j$ iff $L^\alpha_0 = L^\alpha_0$. Fig. 4 shows the Merge function.

![Merge function](image)

Consider the OMDD-A in Fig. 5, which computes the EHC of the network in Fig. 1; the diagram has not been reduced using isomorphism. Links between a node and its children have been labeled with the edges that are UP (e.g. 01 indicates that both $e_0$ and $e_1$ are UP, in addition to $v_0$ being UP). Negative links are labeled with an X. Terminal nodes are marked with the EHC, or 0 for a failure node. The shaded nodes deciding variable group $g_2$ are isomorphic, and thus are merged before being further processed to create sub-trees; the resulting OMDD-A is shown in Fig. 6.

While the top two levels of the diagram in Fig. 6 are unchanged, the third level now has only three non-terminal nodes instead of seven. Although each $g_i$ node in Fig. 6 could have a maximum of four sub-trees, two have less than this, and several of the sub-trees consist solely of a terminal node. This represents a reduction from a theoretical maximum of 53 ($1+4+16+32$) nodes to 19 nodes in this diagram, most of which are terminal nodes.

Terminal nodes are not processed to create sub-trees. Failure nodes are simply discarded, while success nodes have their probability contribution stored before being discarded. For this reason the OMDD-A algorithm does not merge terminal nodes, and it is more appropriate to compare diagrams by counting only non-terminal nodes. The OMDD-A in Fig. 6 has 7 non-terminal nodes.

An OMDD-A has $|V|$ levels, compared to $|V|+|E|$ levels for OBDD-A. The advantage of the OMDD-A having less depth is offset by the fact that more nodes are generated at each level. However, since augmented nodes are only tested for isomorphism with other nodes on the same level, the amount of isomorphism in the diagram increases. This leads to a far superior performance compared to deciding vertices and edges one at a time. For comparison, an OBDD-A equivalent to the OMDD-A in Fig. 6 would contain 31 non-terminal nodes in 9 levels. Hence, for this example OMDD-A uses only 23% ($\frac{7}{31}$) of the nodes of the OBDD-A.
E. Creating an OMDD-A for Computing EHC

The function in Fig. 7 constructs an OMDD-A to compute the EHC and REL. The function first creates a root node $N_0$, and places it in a queue $Q_C$, which is used to hold all nodes at the level currently being decided. The other queue, $Q_N$, is used to hold the child nodes created at the next lower level, and is initialized to null. The decision variable, $DV$, and all length probabilities, $Pr(L)$, are initialized to 0.

Lines 3 to 21 are a loop that first checks if $Q_C$ is empty; $Q_C$ being empty indicates that we have finished a level of the diagram. If $Q_C$ and $Q_N$ are both empty the algorithm terminates. If $Q_C$ is empty but $Q_N$ is non-empty, we generate the $DV$, and all length probabilities, $Pr(L)$, are initialized to 0.

The next step (line 8) removes the first $N$ from $Q_C$. We then call Merge($N_0$) for each children of $N$ based on UP edges. If $N$ is non-terminal then for each $N_i$ in $Q_N$ do if $N_i$ is isomorphic to $N$, then call Merge($N_i$, $N$). break. if no $N_i$ was isomorphic to $N$ then add $N$ to $Q_N$.

1. Create root node $N_0$
2. $Q_C ← \{N_0\}$, $Q_N ← \{}$, $DV ← 0$, and $Pr(L) ← 0$ (for all $0 ≤ i ≤ |E|$).
3. if $Q_C = \{}$ then
4. if $Q_N = \{}$ then
5. else
6. 7. $Q_C ← Q_N, Q_N ← \{}$ and $DV ← DV + 1$.
8. remove the first node $N$ from $Q_C$.
9. for each combination of unmarked edges $(v_{DV}, v_i), (v_i, v_{DV})$, or $(v_{DV}, v_{DV})$:
10. create child $N$ based on UP edges
11. if $N$ is non-terminal then
12. for each $N_i$ in $Q_N$ do
13. if $N$ is isomorphic to $N$, then
14. call Merge($N_i$, $N$).
15. break.
16. if no $N_i$ was isomorphic to $N$ then
17. add $N$ to $Q_N$.
18. else if $N$ is a success node then
19. store results in $Pr(L)$.
20. mark all edges used in step 9 above as decided
21. goto 3.

The next step (line 8) removes the first $N_i$ from $Q_C$. We generate the $D = 2^d$ children of $N_i$, and label them $N_{p[i]}$ to $N_{p+1}$ where $d$ is the number of unmarked edges adjacent to vertex $v_{DV}$, (i.e., all edges of the form $(v_{DV}, v_i), (v_i, v_{DV})$, or $(v_{DV}, v_{DV})$ where $x > DV$). The first of these child nodes, $N_{p[i]}$, is the negative child that represents the case where the $v_{DV}$ is DOWN or $v_{DV}$ is UP and all edges grouped with it are DOWN. All other nodes represent $v_{DV}$ being UP and some combination of the edges grouped with it being UP. Each $N_i$ contains information on the state of the CN, in particular the length of the shortest reaching path to each vertex in VS. All non-terminal child nodes which are isomorphic with the nodes in $Q_C$ are merged while others are added to the end of $Q_C$ (lines 11 to 17).

Failed terminal nodes are discarded. Success nodes have their information added to the relevant length probabilities, $Pr(L)$, before being discarded. For example if a component in the success node has a path of length 3 to the target vertex with a probability of 0.081, then $Pr(3)$ is increased by 0.081. The sum of these length probabilities produces REL, and can be used to calculate EHC using Equation (1).

IV. SIMULATION RESULTS AND DISCUSSIONS

OMDD-A has been implemented in C++ and tested on a Pentium computer (2 Xeon 3.2GHz processors, 1MB cache, 2GB RAM) for evaluating the REL and EHC of a variety of networks. Each reported CPU time is averaged over five runs for each simulation. Execution was halted after 5 hours (CPU time). Further, terminal nodes are excluded when stating the number of nodes since non-terminal nodes incur more of a processing cost and since the number of (merged) terminal nodes is fixed for the EHC of a given network.

Other than OBDD-A, we know of no other method that calculates EHC for networks with both node and edge failure. Thus, explicit comparison is made only between the OBDD-A and OMDD-A methods.

A. Computing the EHC using OMDD-A

Both the OMDD-A and OBDD-A were first applied to the the 19 benchmark networks from [1]. While OMDD-A was able to compute the EHC of the networks, OBDD-A failed to compute the metrics for networks 13, 17 and 19 in 5 hours of CPU time. We observed that the efficiency of both implementations is strongly affected by the maximum boundary set size of the network [4].

For those networks solvable by both, the OMDD-A requires less nodes than the OBDD-A. This result is consistent with that in [15],[16] for OMDD and OBDD. For these networks, the OMDD-A requires at most 22.6% of the nodes of the OBDD-A. The most telling difference was for the network from Fig. 7 of [1] ($|V|=7$, $|E|=21$), with the OBDD-A generating 292,504 nodes (out of a possible $29^7$) and the OMDD-A only 102 nodes. Further, OMDD-A reduces the height of OBDD-A (28 levels) to only 7.

The comparison of execution time yields a similar result. As an example, for the network from Fig. 18 from [1] (with 13 vertices and 22 edges) OMDD-A took just 0.5% of that of OBDD-A (0.7s to 159.6s).

In order to compare our results with Soh et. al.[7] we generated random networks using BRITE and applied the OMDD-A algorithm. Our algorithm took longer to complete as compared to [7], but the orders of magnitude were the same. In addition it must be noted that our algorithm considers both node and edge failure, which means the problem solved by OMDD-A is in the order of 150 variables instead of 50 for SDP. Further OMDD-A can compute the EHC of some networks (e.g., 2x100 grid) that cannot be solved using the SDP approach.
The implementation was also tested on other networks with comparable results. For the 2×100 grid network from [2] the OMDD-A generates 18.8% (593 compared to 3155) of the nodes of the OBDD-A. Similar results were produced for other 2×w grids tested. On networks with a larger maximum boundary set (Bmax) [4], such as the 4×4 grid, the OMDD-A was able to compute the answer (generating 398 nodes in 2.87 CPU seconds) while the OBDD-A failed to complete within a reasonable amount of time.

To demonstrate the effect of Bmax, we tested our algorithm on a number of grid networks with a similar number of vertices but varying Bmax. Table I shows that the number of OMDD-A nodes generated increases exponentially as Bmax increases. Note that the number of nodes is also influenced by other factors, such as the ordering of the vertices and edges in the network.

V. CONCLUSION

We have shown that OMDD-A is more time and space efficient than OBDD-A for computing the REL and EHC when network edges and vertices can fail. In our simulations, OMDD-A generates from under 1% to around 25% of the nodes of the OBDD-A. Since the complexity of OMDD-A is not directly related to the number of paths, our technique is suitable for networks (e.g., grid) with extremely large pathsets, not solvable by the existing techniques [6]-[9].

We are investigating more effective variable orderings and groupings for the OMDD-A. We will investigate other methods of storing network information in augmented nodes, including boundary sets [4] and the connectivity matrix. These approaches may reduce processing time and memory use, but it is not clear if they can be modified to efficiently record the path lengths of visited vertices.

REFERENCES