International Journal of Performability Engineering, Vol. 6, No. 4, July 2010, pp. 305-316.

Printed in India

On Augmented OBDD and Performability for Sensor Networks

JOHANNES U. HERRMANN*, SIETENG SOH, SURESH RAI†, and GEOFF WEST

Curtin University of Technology, Perth, Australia
† Louisiana State University, Baton Rouge, L.A., USA

(Received on July 10, 2009, revised on September 2, 2009)

Abstract: The expected hop count (EHC) or performability of a wireless sensor network (WSN) with probabilistic node failures provides the expected number of operational nodes a message traverses from a set of sensors to reach its target station. This paper proposes a novel approach for computing the EHC of a practical communication model for WSN, *k*-of-all-sources to any-terminal (*k*-of-S,t). Techniques based on factoring and Boolean techniques solve the EHC when k=1 for $|S|\geq 1$ However, they fail to scale with large WSN and are not useful for computing the EHC with $k>1$. To overcome these problems, we propose an Augmented Ordered Binary Decision Diagram (OBDD-A) approach, which obtains the EHC for all cases of (*k*-of-S,t). We use randomly generated wireless networks and grid networks having up to 4.6×10^{20} (s,t)-minpaths to generate results. Results show that OBDD-A can obtain the EHC for networks that are unsolvable with existing approaches.

Keywords: *Binary decision diagram, expected hop count, many-to-one communication, network reliability, sensor network*

1. Introduction

Recently, wireless sensor networks (WSNs) have been proposed for various critical monitoring systems such as military, environment, and security [1-3]. Data dissemination in WSNs is categorized into *tasks* (a base station sends tasks to one or more sensors) and *events* (one or more sensor nodes send sensor data to the base station) [1] using various communication models. For *tasks*, the unicast (s,t), multicast (s,*k*-of-T), and broadcast (s,T) from a source base station s are typically used, for $t\in T$, all sensors of set T in the field, and $k\geq 1$. For *events*, the many-to-one communication model (*k*-of-S,t) is used, for all sensors of set S in the field and a target base station t. The model (s,t) is used for a single sensor node. Refer to [2] for multi-modal data acquisition details.

Sensor nodes in a WSN may be subject to random failures [4], or deliberate acts. We assume that communications will succeed when both communicating nodes are functioning. To address its reliability (REL), the directed diffusion paradigm [3] includes an event acquisition mechanism that is robust to node failures. However alternate paths may increase the number of hops and degrade the system responsiveness or Expected Hop Count (EHC). As most WSN applications require end-to-end delay-constraints, it is crucial to develop models to evaluate the performability of such critical systems for the various

* Communicating author's email: jherrmann@gmail.com

communication models. While two distinct reliability models exist for WSN, this paper focuses on EHC. This means that the infrastructure reliability [5] model is used.
AboElFotoh, *et al.* [4] employed the factoring theorem to obtain EHC(*k*-of-S,t), the EHC for (*k*-of-S,t) model for k=1 and $|S|\geq 1$ (*i.e.*, EHC(s,t) and EHC(1-of-S,t)) and showed the problem is #P-hard. Soh, *et al.* [6] proposed a Boolean technique to compute the EHC. Brooks, *et al.* [7] used random graph models to approximate EHC(s,t) in mobile WSN, but assume link failures. Note, none of the methods [4, 6, 7] are useful for computing the general REL and EHC metrics (*i.e.*, $k>1$) when a large number of paths are involved (*e.g.*, a 2×100 grid network that contains 4.6×10^{20} (s,t)-minpaths). Since (*k*-of-S,t) metrics are applicable in a range of critical applications, including smart houses, earthquake and tsunami detection as well as perimeter security, improved techniques for their solution are needed.

This paper proposes an Augmented Ordered Binary Decision Diagram (OBDD-A) to compute EHC(*k*-of-S,t) for $k\geq 1$ and $|S|\geq 1$. Our OBDD-A is an extension of the OBDD[8] in that it stores network state information in each diagram node. References [9, 10] use OBDD techniques to solve reliability problems such as REL(s,t), REL(s,*k*-of-T), and REL(s,T). The approaches in [9, 10] generate OBDD nodes to take advantage of isomorphism through hash table lookups, but they do not explicitly link them into a diagram. However these nodes contain no information on path length and as a result, these methods cannot be used to calculate the network EHC (refer to Section 3). Our OBDD-A method generates network information as part of the creation of each node. This additional information enables our OBDD-A to solve the EHC as well as REL.

The layout of the paper is as follows. Section 2 gives a definition of the network model and reviews OBDDs, especially their use for calculating the reliability of a network. In Section 3 we introduce the EHC problem. We propose an OBDD-A to solve the EHC problem in Section 4. In Section 5, we apply OBDD-A to a number of networks and present the results. Finally, Section 6 concludes the paper.

2. Background

2.1 Network Model and Terminology

We model a WSN using a graph G(V,E), where each vertex in V represents a computer, router or sensor, and every edge in E denotes a wired/wireless communication medium between the vertices. Communication occurs by a set S of source devices (*e.g.*, sensors) sending messages towards a target device (*e.g.*, a monitoring station). A vertex v_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 v_j. We assume: that vertex failures are statistically independent and that the edges are always functioning. If edges are also prone to failure the multivariate form [11] of the algorithm given in this paper should be used.

Let n=|V|, and let the vertices (v_0, v_1, …, v_{n-1}) of V be ordered in increasing distance to target vertex v_0, except that the source vertices, S, are always labelled $v_{n-|S|}$ to v_{n-1}. The width of G(V,E) is defined as W=MAX($|i-j|$: e=(v_i,v_j) or $\{v_i,v_j\}\in$E). For example, a 2×M grid WSN has W=2, and the network in Figure 1 has W=3. Larger width increases the number of nodes generated in OBDD-A, and reduces the efficiency of our approach. In this paper, directed and undirected edges are sorted in increasing order of v_j and, then, in increasing order of v_i. Such an ordering helps minimize W and the number of reverse directed edges (v_i,v_j) where $i>j$. Figure 1 shows such a vertex and edge ordering.

A (s,t)-minpath P_i between a source $v_s \in S$ and v_0 in G(V,E) traverses a loop-free path and is formed by a sequence of UP vertices. A *reaching path* from vertex v_x is a minpath, but leading from v_x to v_0, where v_x may not be a source vertex. We write a minpath or reaching path using its sub-scripts; *e.g.*, the minpath (v_9, v_6, v_3, v_1,$v_{0,}$) is written as 96310.

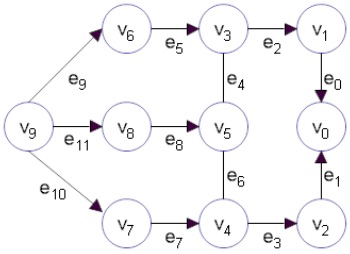

Figure 1: Sample Network

2.2 Ordered Binary Decision Diagram (OBDD)

The OBDD represents Boolean functions [8]. Figure 2(a) is an OBDD for function $x_1x_3 \cup x_2x_3$ where a solid (dashed) line denotes $x_i = 1$ ($x_i = 0$). Here circles (squares) are non-terminal (terminal) nodes. Note that a terminal with 1 (0) is a success (failure) node. To reduce the number of nodes, remove duplicate (isomorphic) nodes whose sub-trees are identical (Figure 2(b)). Utilizing isomorphism between nodes is one of the strengths of OBDD because it prevents sub-trees from being re-evaluated. Further, when the evaluation of a particular variable (node) does not affect the sub-tree, the redundant node can be removed as shown in Figure 2(c). The size of the OBDD that represents a function is dependant on the ordering of the variables [8]. Finding an optimal ordering for OBDDs is a NP-Complete problem [12], and thus several non-optimal ordering techniques are used [13].

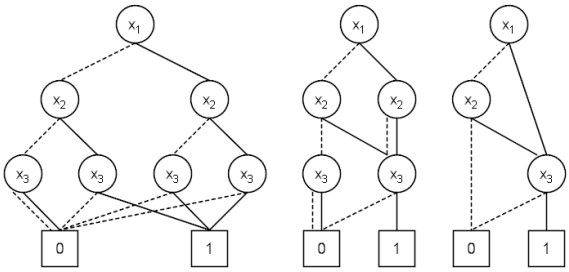

Figure 2: An Example for OBDD[8]

In the application of the OBDD technique to reliability [9], each variable (node) represents a vertex v_i or an edge e_i that is either UP with p_i or DOWN with $q_i = 1 - p_i$. The probability that the network is connected is then given by tracing paths upwards from the success terminal nodes and multiplying the reaching path probabilities by p_i (q_i) for a positive (negative) sub-tree. Since each traversed path represents a disjoint event the probability of each such path is added to give the network reliability.

3. Reliability and Expected Hop Count

Let Ω=(U) represent a state of a network G(V,E) when all vertices in $U \subseteq V$ (V-U) are UP (DOWN). The REL(*k*-of-S,t) is computed from the set of all success states $\Omega_g \subseteq \Omega$. This

model allows multiple source vertices and the system is successful only if at least *k* of the duplicate messages from *distinct* sources are received by the target station. In other words, a success state Ω_g contains at least *k* (s_i,t)-minpaths of G(V,E), for distinct $s_i \in S$. In addition to the success state information, computing the EHC requires the length of each Ω_g denoted as $1 \le L(\Omega_g) \le n\text{-}1$. We consider $L(\Omega_g)$ to be the k^{th} shortest *distinct* (s-t)-minpath; that is, the longest of the *k* shortest distinct (s,t)-minpaths. For example, consider $S=\{v_8,v_9\}$, $k=2$ and $t=v_0$ in Figure 1. The network state $\Omega_g=(v_0, v_1, v_2, v_4, v_5, v_6, v_7, v_8, v_9)$ contains the two (s,t)-minpaths 97420 and 985420 from v_9 and minpath 85420 from v_8. The shortest minpath from v_9 has length 4 and the one from v_8 also has length 4. It is a success state since it contains (s,t)-minpaths from $k=2$ distinct source vertices. Since $k=2$, $L(\Omega_g)=4$, the second-shortest of the two lengths.

EHC(k-of-S,t) is computed as:

$$\text{EHC} = \frac{\sum\left(L(\Omega_g) \times \Pr(\Omega_g)\right)}{\sum \Pr(\Omega_g)} \tag{1}$$

Note that $\Pr(\Omega_g)=\prod_{v_i \in U} p_i \prod_{v_i \in (V-U)} q_i$, and that the denominator in Eq. (1) is REL(*k*-of-S,t). The problem of computing EHC(s,t) and EHC(1-of-S,t), *i.e.*, EHC(*k*-of-S,t) with $k=1$ and $|S|=1$, and $k=1$ and $|S|>1$ respectively, have been shown #P-hard [4].

Continuing the above example with each $p_i = 0.9$ we find that the sum of all $\Pr(\Omega_g)$ with length 4 is 0.75051279, with length 5 is 0.027221589, and with length 6 is 0.001062882. Thus REL(2-of-S,t)=0.77879726, and

$$\text{EHC(2-of-S,t)} = \frac{4 \times 0.75051279 + 5 \times 0.027221589 + 6 \times 0.001062882}{0.77879726} = 4.0376829.$$

4. OBDD-A for Computing REL and EHC

4.1 The Mathematical Model of the OBDD-A

Let $\Psi(N,G)$ denote an OBDD-A for the graph G(V,E), where N is the set of OBDD-A nodes $\{N_0, N_1, \ldots, N_{2^{|E|}-1}\}$. Without loss of generality, let N_0 be the root node. Let N_{2i+1} (N_{2i+2}) be the left or negative (right or positive) child node of N_i, for $i=0, 1, 2, \ldots$. $\Psi(N,G)$ is divided into $n=|V|$ levels, where N_0 is on level 0, N_1 and N_2 on level 1, N_3, N_4, N_5, and N_6 on level 2, and so on. Thus any node N_i is on level *j* if and only if $2^j\text{-}1 \le i \le 2^{j+1}\text{-}2$. Each level *j* of $\Psi(N,G)$ represents a decision on the state (UP for each right child and DOWN for each left child) of v_j, and we say that a node on this level *decides* variable v_j and call it the *decision variable* (DV) for N_i.

Our OBDD-A is an OBDD in which each of the nodes $N_i \in N$ contains a pair $[VI_i, CI_i]$ representing information used to calculate REL and EHC. In contrast to the two-pass scheme in [9], our approach generates $\Psi(N,G)$, REL, and EHC directly from G(V,E). The VI/CI notation tracks which vertices have been reached by messages but has no record of where the messages originated. Hence, to make sure that messages from *k* distinct sources have reached the target, we start at the target vertex and backtrack messages to the sources.

The *condition information*, CI_i, is a set of conditions $\{C_0, C_1, \ldots C_{|CI|-1}\}$ of the form $C_x=(v_a,v_b,L_x)$ where L_x is the length of the shortest path of UP vertices from vertex v_a to vertex v_b. Each condition represents a path through the network that can be taken if its endpoint is reached.

The *vertex information*, VI_i, is a set of components $\{M^0, M^1, \ldots, M^{|VI|-1}\}$ storing path length information. Each $M^x=(\{(v_1, L^x_1), (v_2, L^x_2), \ldots, (v_k, L^x_k)\}, P^x)$ in VI_i contains a probability P^x and a set of ordered pairs of the form (v_a, L^x_a), where L^x_a is the length of the shortest reaching path from v_a to the target vertex v_0. If the set of pairs in M^x contains (v_j, L^x_j) then we write $(v_j, L^x_j)\in M^x$. VI_i has the property that if one component of VI_i contains a pair with vertex v_j then all components of VI_i contain a pair with this vertex.

Given a node N_i, let $VS_i = \{v_a: (v_a, L^x_a)\in M^x$ and $M^x\in VI_i\}$ be the set of undecided vertices that have known reaching paths to v_0. As an example, for $N_2=[VI_2=\{(\{(v_1,1), (v_2,1)\}, 0.9)\}, CI_1=\{\})]$, we have $VS_2=\{v_1, v_2\}$. Note that decided vertices need not be stored since the position of the node in $\Psi(N,G)$ implicitly encodes all decisions made at higher levels.

Definition: Components M^x and M^y are *equal* ($M^x = M^y$) if for every pair $(v_a,L^x_a)\in M^x$ there exists $(v_a,L^y_a)\in M^y$ such that $L^x_a=L^y_a$.

Let the probability that a message takes a path having L hops be denoted as $Pr(L)$. As successful components are found the probabilities $Pr(L)$ are calculated by the OBDD-A. When the generation of OBDD-A nodes is complete all $Pr(L)$ are used to calculate REL and EHC.

4.2 OBDD-A Node Type

An OBDD-A node is terminal (non-terminal) if it does not (does) have children. Our approach processes each non-terminal node in a breadth-first fashion to better take advantage of node isomorphism. When a terminal node represents only states that meet the requirements for the problem, it becomes a *success* node. The REL and EHC are computed from the reaching path probabilities contained in all success nodes. When the requirements cannot be met from the current state it is a *failure* node. A failure node has no sub-trees containing a success node. To avoid generating redundant information, it is favourable to detect failure nodes as early as possible. When $VS_i=\varnothing$, node N_i must be a failure node; however, a failure node N_i may have a non-empty VS_i, and detecting such nodes is computationally expensive. Hence the OBDD-A algorithm detects N_i failed only if $VS_i=\varnothing$.

A node N_i for which $VS_i\cap S\neq\varnothing$ (*i.e.* at least one minpath has been found) is not necessarily successful since the EHC calculation requires the shortest path to the target. Because one component of a node might be successful while another is not, individual components are tested for success. A success component representing a state of length L is removed from the node and its probability is added to $Pr(L)$.

A component is detected as successful only if it has at least k minpaths from distinct source vertices. If the longest of these k paths is L, no other reaching paths from a non-target vertex can have length less than L-1. The length of the state represented by the component is equal to the k^{th} longest of the minpaths. For example if k=2 and there are minpaths of length 4, 5 and 6 in a component, the component represents a state of length 5. Every reaching path in the component would be required to be length 4 or above.

4.3 Node Isomorphism

Isomorphic nodes have equivalent sub-trees. Merging them into one node avoids the need to process them separately. Each merge operation effectively prunes one of the sub-trees.

Definition: Nodes N_i and N_j at the same level in $\Psi(N,G)$ are *isomorphic* if $VS_i=VS_j$ and $CI_i=CI_j$. We write $N_i = N_j$.

Relaxing the definition by excluding CI increases the number of nodes found to be isomorphic but also greatly increases the computational complexity of processing each node. Similarly, strengthening the definition by requiring that the reaching path lengths be identical reduces the computational complexity per node at the cost of fewer nodes found to be isomorphic.

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, if $i<j$. When two isomorphic nodes N_i and N_j are merged, the VI are combined as follows. Every component M^x that is in only one node is present in the merged node with probability unchanged. If $M^x \in VI_i$ and $M^y \in VI_j$ are identical, then the merged node has a component that is identical to M^x but has probability P^x+P^y. Note that since $VS_i=VS_j$ we are guaranteed that for every pair $(v_a, L^x_a) \in M^x$ there exists a pair $(v_a, L^y_a) \in M^y$; for component equality it remains only to compare L^x_a and L^y_a for every $v_a \in VS_i$.

As an example, consider two isomorphic nodes N_{58}=[{({(v_5,3),(v_6,3),(v_7,5)}, 0.06561)}, {}], and N_{62}= [{({(v_5,3),(v_6,3),(v_7,3)}, 0.59049)}, {}]. To merge N_{62} into N_{58} we compare the single components in both nodes. Since the components are not equal (due to different reaching path lengths) we add the component from N_{62} into N_{58} giving N_{58}=[{ ({(v_5,3),(v_6,3),(v_7,5)},0.06561), ({(v_5,3),(v_6,3),(v_7,3)},0.59049) }, {}]. Further, consider isomorphic nodes N_{25}=[{({(v_4,2)},0.0081)}, {}] and N_{29}=[{({(v_4,2)},0.0729)}, {}]. Each has only one component, and they are equal. Hence the merged node has one component with the sum of the two probabilities; N_{25}=[{({(v_4,2)},0.081)}, {}].

4.4 Processing an Augmented OBDD Node

When processing a node N_i, the function Process in Figure 3 first creates the positive child, N_{2i+2} and then modifies it to represent the case of the decision vertex being UP. This involves modifying the probability of all components (Steps 2 and 3), following each edge entering the decision vertex, visiting its other endpoint (Steps 4 and 5) and then adding the edge to CI_{2i+2} (Step 6). Any condition with an endpoint on the decision variable is also followed and the other endpoint is visited (Steps 7 and 8). Lastly any pair of conditions with an end point on the decision variable are merged to form a new condition (Steps 9 and 10).

When a vertex v_a is added to VS_i we say that v_a has been visited. The visit(v_j, v_a, L_x) function represents a new path being added, extending a reaching path from v_j to a reaching path from v_a along a path segment of length L_x. This means if the existing shortest reaching path from v_j has length L_j we have a new reaching path from v_a of length $L_j + L_x$. The function checks to see whether a reaching path from v_a already exists. If no such reaching path exists or the existing reaching path has length greater than $L_j + L_x$ then the new length is recorded as the minimum length from vertex v_a. This is done for every component. Note that only the lengths of the paths are recorded.

If $p_i<1.0$, the function Process also creates a negative child N_{2i+1} which is initialized as a copy of N_i (Steps 11 and 12). The probabilities of all components are modified by multiplying them with the probability of failure of the decision variable (Steps 13 and 14) but the other updates applied to the positive child are not needed. Lastly all references to the decision variable are removed from the VI and CI of both nodes (Steps 15 to 20).

Process(N_i, v_j):
// Let N_i = (VI_i, CI_i) and v_j be the decision variable.
1. $N_{2i+2} \leftarrow N_i$.
2. **for** each $M^x = (\{\ldots(v_j,L^x_j), \ldots\}, P^x) \in VI_{2i+2}$ **do**
3. $P^x \leftarrow P^x \times p_j$.
4. **for** each edge e=(v_a, v_j), {v_a, v_j} or {v_j, v_a} in E **do** //$v_a \neq v_j$
5. visit (v_j, v_a, 1).
6. add (v_a, v_j, 1) to CI_{2i+2}; also add (v_a, v_j, 1) if e is undirected
7. **for** each C^y=(v_a, v_j, L^y)$\in CI_{2i+2}$ **do**
8. visit (v_j, v_a, L^y).
9. **for** each (v_a,v_j,L^y)$\in CI_{2i+2}$ and each (v_j,v_b,L^z)$\in CI_{2i+2}$ **do**
10. add (v_a, v_b, L^y+L^z) to CI_{2i+2}.
11. **if** p_j<1.0 **then**
12. $N_{2i+1} \leftarrow N_i$.
13. **for** each $M^x \in VI_{2i+2}$ **do**
14. $P^x \leftarrow P^x \times q_j$.
15. **for** each (v_j, L^x_j)$\in M^x$ with $M^x \in (VI_{2i+1} \cup VI_{2i+2})$ **do**
16. delete (v_j, L^x_j).
17. **for** each (v_a,v_j,L^y)$\in (CI_{2i+1} \cup CI_{2i+2})$ **do**
18. delete (v_a,v_j, L^y).
19. **for** each (v_j,v_b, L^y)$\in (CI_{2i+1} \cup CI_{2i+2})$ **do**
20. delete (v_j,v_b, L^y).

Figure 3: The Process function

4.5 OBDD-A Algorithm

Figure 4 shows our OBDD-A algorithm. Step 1 sets the root node N_0= [VI_0=({(v_0, L^0_0=0)}, P=1.0), CI_0={}]. We note that VS_0=v_0 for all networks. Hence the first level of Ψ(N,G) contains a single node deciding v_0. In Step 2, we initialize the current queue Q_C, the next queue Q_N={}, and set the decision variable, DV, to 0. We also initialize the probabilities of each possible reaching path length to zero. Q_C stores the unprocessed nodes on level DV, while Q_N contains the nodes to be processed on level DV+1. When Q_C=Q_N={}, the node generation is complete and the algorithm halts; if only Q_C={}, we set Q_C=Q_N and Q_N={}, and increment DV. We then remove the first node N_i from Q_C and call Process to create N_{2i+2} (and possibly N_{2i+1}).

In Step 8, each component is tested for success. Such components are removed and Pr(L) is updated accordingly. Finally, in Steps 9-14 we test each child node; if it is non-terminal we check for isomorphism with nodes on the next level of the OBDD-A and either merge the new node with an existing isomorphic node, or add it to Q_N if no isomorphic node exists. The algorithm then repeats from Step 3.

When both queues are empty (Step 3), we process the stored probabilities to generate REL and EHC as discussed in Section 3. One of the advantages of this approach is that we not only obtain REL and EHC, but the individual Pr(L) as well. For example we could use the results to calculate the probability that a message would arrive in five or less hops (calculated using $\Pr(\leq 5) = \sum_{i=1}^{5} \Pr(i)$).

OBDD-A Algorithm
1. Create root node $N_0 \leftarrow$ [VI_0={(v_0, $L^0{}_0$=0)}, P=1.0), CI_0={}].
2. $Q_C \leftarrow \{N_0\}$, $Q_N \leftarrow \{\}$, DV$\leftarrow$0, and Pr(L) $\leftarrow$0. // for L=1 to n-1
3. **if** Q_C={ } and Q_N={ } **then** compute REL and EHC. //from Pr(L), use Eq(1)
4. **else if** Q_C={ } and $Q_N \neq$ { } **then** $Q_C \leftarrow Q_N$, $Q_N \leftarrow$ { } and DV $\leftarrow$ DV + 1.
5. Remove the first node N_i from Q_C.
6. Call Process(N_i, DV) to create N_{2i+2} (and possibly N_{2i+1}).
7. **for** each child N created in step 6 **do**
8. **for** each success component M^x of length L **do** P(L) $\leftarrow$P(L)+P^x.
9. **if** N is non terminal **then**
10. **for** each $N_q \in Q_N$ **do**
11. **if** N = N_q **then**
12. merge N into N_q.
13. **break**.
14. **if** no N_q was isomorphic to N **then** add N to Q_N.
15. **goto** 3.

Figure 4: The OBDD-A Algorithm

4.6 Example

To illustrate our OBDD-A algorithm, we compute the EHC for the network in Figure 1. with S={v_9,v_8}, t=v_0 and k=2. Let p_i=0.9 for all vertices.

Step 1 of the algorithm in Figure 4 sets N_0 = [VI_0=({(v_0, $L^0{}_0$=0)}, P=1.0), CI_0={}]. Step 2 sets DV=0, Q_C={N_0} and Q_N={}. For a WSN, N_0 represents the state of G(V,E) when the target has received a message which we will track back towards the source(s). The target vertex has at this stage not been decided as UP, so the message has not propagated any further. Step 3 does not apply since Q_C is non-empty.

First, we call Process(N_0,v_0). This creates N_1 and N_2, which are initially copies of N_0. The nodes are multiplied by q_0=0.1 and p_0=0.9 respectively. Since DV=0, the state of N_2 is updated by following all edges entering v_0. The edges are e_0= (v_0, v_1), and e_1=(v_0, v_2), causing vertices v_1 and v_2 to be visited. In each case the instance of ({(v_0, 0)}, P) is copied to a new pair such as ({(v_x, 1)}, P), representing the message travelling one more hop to the next vertex. As a result, VI_2 = { ({(v_0,0), (v_1,1), (v_2,1)},0.9) }.

We next add conditions representing all of the adjacent edges (Step 6 of Process). The conditions added are (v_0, v_1, 1) and (v_0, v_2, 1) representing e_0 and e_1, respectively. We then combine any pairs of conditions (v_a, v_0, L^x) and (v_0, v_b, L^y), however N_2 does not contain any conditions (v_a, v_0, L) to match those just added and no action is taken. Lastly, we delete all elements of VI_1, VI_2, CI_1 and CI_2 that contain the decided vertex v_0. This gives N_1=[VI_1={({},0.1)}, CI_2 = { }], VS_1={ } N_2=[VI_2={({(v_1,1), (v_2,1)},0.9)}, CI_2 = {}] and VS_2 = {v_1, v_2}. Note that since VS_1 is empty, N_1 is a failure node and is not stored on Q_N. Since Q_N is empty, N_2 is not isomorphic to any existing nodes and is appended. We then return to Step 3, and continue repeating the loop until both queues are empty.

The successful components are ({(v_6,3),(v_8,4),(v_9,4)}, 0.59049), ({(v_7,3),(v_8,4),(v_9,4)}, 0.11219) and ({(v_7,3),(v_8,4),(v_9,4)}, 0.04783) of length 4, ({(v_8,4),(v_9,5)}, 0.017656) and ({(v_8,4),(v_9,5)}, 0.0095659) of length 5, and ({(v_8,4),(v_9,6)}, 0.00010629) and ({(v_8,4),(v_9,6)}, 0.00095659) of length 6. Each has information on source vertices v_8 and v_9, and some also have information on one of v_6 or v_7.

When a success component is detected, the reaching path length to the target and its reaching path probability are noted and the component is removed from the node. When a node contains only success components, it is a success terminal node. So for the given example, Pr(4)=0.75051279, Pr(5)= 0.027221589, and Pr(6)= 0.001062882. This gives REL = 0.75051279 + 0.027221589 + 0.001062882 = 0.778797261 and EHC = (4×0.75051279 + 5×0.027221589 + 6×0.001062882) / 0.778797261 = 4.037682918.

5. Simulation Results and Discussions

5.1 Simulation Environments

We have implemented our OBDD-A algorithm in C++ and run it on a Pentium computer (2 Xeon 3.2GHz processors, 1MB cache, 2GB RAM). Topologies WSN-1 through WSN-4 are generated by placing 50 devices randomly in a unit square, assuming a transmission radius of 0.5, and connecting any devices able to communicate directly as per the fixed radius model [14]. We assume p_i=0.9 for each vertex; source and target vertices have p_i = 1.0. For each simulation, the run time in CPU seconds is averaged over five runs[1]. Simulations using our OBDD-A and sum-of-disjoint products (SDP) techniques [6] for generating REL(s,t) and EHC(s,t) produced exactly the same results, verifying the correctness of our approach.

To see the effects of vertex and edge ordering (discussed in Section 2.1) on our OBDD-A performances, we considered two sets of input files: one with random ordering, and the other with the described ordering. Our simulation shows that the ordering significantly affects the performance of OBDD-A. As an example, computing REL(s,t) and EHC(s,t) for a 6×6 grid with (without) sorting generates 6592 (9548) OBDD-A nodes and takes 9(18) CPU seconds. Note that this ordering reduces its width, W, from 25 to 6, and hence reduces the number of conditions, CI, and vertices in VS of the OBDD-A. This reduces both the number of diagram nodes generated and their processing time. All input files used for the remaining simulations were ordered as described in Section 2.1.

5.2 Results for EHC(s,t)

The SDP approach has been shown to be more efficient than the factoring method [6], and therefore we compare the performance of our OBDD-A only with that of the solution [6] provided by the authors. As shown in Table 1, our OBDD-A is generally more efficient on the larger networks, especially the grid networks. Also, OBDD-A is able to compute the REL and EHC of large grid networks that contain up to $4.6x10^{20}$ (s,t)-minpaths. The SDP approach is not efficient for this network type because: (i) it is not feasible to generate the huge number of minpaths, and (ii) it requires large amounts of memory and CPU time to convert the paths into their disjoint terms. A simulation marked DNC in the Table 1 did not complete within 5 minutes of CPU time.

The OBDD-A approach is particularly efficient with networks with low width W. The performance of the OBDD-A approach is not directly related to the number of paths, as can be seen in Table 1. This is because reaching paths of the same length are merged, which is especially apparent in the grid networks with low W. Note that the performance of OBDD-A is better on the grids of width 2 than the grids of width 3. In particular, the 3×33 grid with 99 vertices takes considerably more processing time and generates more

[1] We used multiple runs to eliminate slight inconsistencies in CPU time generated. The standard deviation was extremely low in general (2% or lower) so it was decided that five runs were sufficient.

nodes than the 2×100 grid with 200 vertices. The SDP performs better for networks of larger width that have a small number of paths, such as the 6×6 grid.

Table 1: OBDD-A vs. Boolean Techniques

Network	W	#Paths	REL(s,t)	EHC(s,t)	OBDD-A		SDP
					|N|	Time	Time
Grid 2×18	2	3382	0.6629	18.6695	97	0.003	4.079
Grid 2×50	2	1.6×10^{10}	0.2928	52.9231	289	0.030	DNC
Grid 2×100	2	4.6×10^{20}	0.0817	106.4860	589	0.227	DNC
Grid 3×12	3	3652	0.9167	13.1953	355	0.015	6.028
Grid 3×33	3	2.3×10^{10}	0.7910	35.6753	2164	0.915	DNC
Grid 4×9	4	1949	0.9629	11.0336	1462	0.176	2.743
Grid 6×6	6	832	0.9828	9.01288	6778	9.411	2.449
WSN-1	24	28280	0.6905	10.3843	588	0.089	30.226
WSN-2	36	8548	0.8579	6.3497	1460	0.440	2.304
WSN-3	10	118440	0.3819	15.2223	1358	0.133	DNC
WSN-4	45	471	0.9897	2.1909	6592	30.412	0.265

The four WSNs shown were chosen as representative of the networks generated. WSN-1 and WSN-3 have a large number of paths while WSN-2 and WSN-4 have far fewer paths. The number of hops between the source and target vertices also varies between the networks, as evidenced by the EHC results shown in Table 1. The width of these networks is not as clear an indicator of OBDD-A performance as with grid networks because they are less evenly distributed, but it is noteworthy that the network with the highest width, WSN-4, is the network for which OBDD-A displayed the worst performance Note also that OBDD-A was able to solve WSN-3 (with the low W and large number of paths) while SDP could not.

5.3 Results for EHC(1-of-S,t) and EHC(k-of-S,t)

Table 2 shows the performance of OBDD-A for computing the REL and EHC of the 6x6 grid network shown in Figure 5. The vertex marked t is the target v_0 for all models shown, and the source(s) vary as shown in the table, chosen from the vertices marked 32 to 35. The number of OBDD-A nodes generated and the time in CPU seconds are shown in Table 2.

Table 2: OBDD-A Performance on Different Models

Model	S	Nodes	Time	REL	EHC
(1-of-S,t)	{35}	7712	9.4	0.9720	10.0012
(1-of-S,t)	{34,35}	7703	9.3	0.9828	9.0129
(2-of-S,t)	{34,35}	7714	9.4	0.9828	10.0129
(1-of-S,t)	{32,33,35}	8014	9.4	0.9856	7.0742
(2-of-S,t)	{32,33,35}	8136	9.5	0.9833	9.0198
(3-of-S,t)	{32,33,35}	8139	9.5	0.9818	10.0183

As can be seen, the EHC increases and the REL decreases as *k* increases. For equal *k*, the EHC decreases and REL increases for increasing |S|. Note that when |S|=1, (1-of-S,t) is an

(s,t), and as shown in Table 2, computing the model for |S|>1 is more efficient since there is more opportunity for a branch of the diagram to terminate earlier. Our OBDD-A approach computes the solution to all models in comparable time and number of nodes generated. The performance of OBDD-A for solving models EHC(*k*-of-S,t) is comparable with that of EHC(s, t).

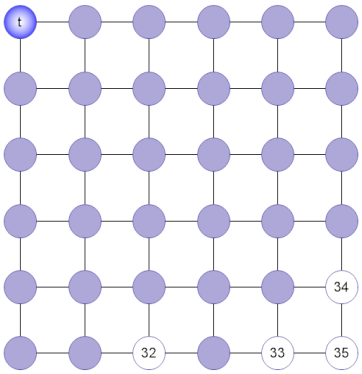

Figure 5: 6×6 Grid Network

6. Conclusions

This paper describes a model for the REL and EHC of a WSN, and presents an algorithm for solving it. Our OBDD-A approach is competitive on general networks, while being more efficient for networks with low width W, especially grid networks. Further, since the performance of the OBDD-A method is not directly related to the number of paths or cuts in the solution, our approach can solve problems with extremely large pathsets that the existing factoring [4] and SDP [6] approaches cannot. Our approach solves the (*k*-of-S,t) model for k>1 which was not addressed in [4] and [6].

The OBDD-A is equally applicable for network models with multiple targets, such as (s,*k*-of-T) by starting at the source and following the flow towards the target vertices. The approach can be generalized for (k_i-of-S_i,T) where each S_i is a group of source vertices and at least k_i messages from distinct vertices in group S_i are required to reach at least one of the target vertices in T. Similarly we can solve (S, k_i-of-T_i) by starting at the sources.

It is noted that for the case of vertex and edge failure, an OMDD-A is more efficient than the OBDD-A [11]. For the case of vertex failure and perfect edges, the grouping of variables in [11] is not applicable. Research will be undertaken to investigate whether a suitable ordering exists to allow the OMDD-A to be applied to the case of perfect edges. In addition the boundary set notation introduced by Carlier and Lucet [15] will be compared to the current VI/CI model, which requires extending it to EHC first. In addition other network models will also be investigated.

Acknowledgement: We thank the anonymous referees for their positive comments and constructive criticism.

References

[1] Soh, S., S. Rai, and R. R. Brooks. *Performability Issues in Wireless Communication Network*, in The Handbook on Performability Engineering, Editor: K. B. Misra, Springer Verlag, 2008, 1047-1067.

[2] Akyildiz, W. S., W. Su, Y. Sankarasubramaniam, and R. Cayirci. *Wireless sensor networks: a*

survey. Computer Networks, March 2002; 38, 393-422.
[3] Intanagonwiwat, C., R. Govindan, D. Estrin, J. Heidemann, and F. Silva. *Directed Diffusion for Wireless Sensor Networking*. IEEE/ACM Transactions on Networking, February 2003; 11(1), 2-16.
[4] AboElFotoh, H. M. F., S. S. Iyengar, and K. Chakrabarty. *Computing Reliability and Message Delay for Cooperative Wireless Distributed Sensor Networks Subject to Random Failures*. IEEE Trans. Reliability, 2005; 54(1), 145-155.
[5] Shrestha, A., and L. Xing. *Quantifying Application Communication Reliability of Wireless Sensor Networks*. Int'l J. Performability Engineering January 2008; 4(1), 43-56.
[6] Soh, S., W. Lau, S. Rai, and R. R. Brooks. *On Computing Reliability and Expected Hop Count of Wireless Communication Networks*. Int'l J. Performability Engineering, April 2007; 3(2), 267-279.
[7] Brooks, R. R., B. Pillai, S. Racunas, and S. Rai. *Mobile Network Analysis Using Probabilistic Connectivity Matrices*. IEEE Trans. Systems, Man, and Cybernetics —PART C: Applications and Reviews, 2007; 37(4), 1-9.
[8] Bryant, R. E. *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*. ACM Computing Surveys, 1992; 24(3), 293-318.
[9] Kuo, S.-Y., S.-K. Lu, and F.-M. Yeh. *Determining Terminal-Pair Reliability Based on Edge Expansion Diagrams using OBDD*. IEEE Trans. Reliability, 1999; 48(3), 234-246.
[10] Yeh, F.-M., S.-K. Lu, and S.-Y. Kuo. *OBDD-Based Evaluation of k-Terminal Network Reliability*. IEEE Trans. Reliability, 2002; 51(4), 443-451.
[11] Herrmann, J. U., S. Soh, G. West, and S. Rai. *Using Multi-valued Decision Diagrams to Solve the Expected Hop Count Problem*. IEEE 23rd Int. Conf. Advanced Information Networking and Applications Workshops, 2009; Bradford, UK.419-424.
[12] Friedman, S. J., and K. J. Supowit. *Finding the Optimal Variable Ordering for Binary Decision Diagrams*. IEEE Trans. Computers, 1990; 39(5), 710-713.
[13] Bollig, B., and I. Wegener. *Improving the Variable Ordering of OBDDs is NP-Complete*. IEEE Trans. Computers, 1996; 45(9), 993-1002.
[14] Krishnamachariy, B., S. B. Wickery, and R. Bejar. *Phase Transition Phenomena in Wireless Ad Hoc Networks*. Global Telecommunications Conference, 2001; San Antonio, TX, USA 2001.
[15] Carlier, J., and C. Lucet. *A Decomposition Algorithm for Network Reliability Evaluation*. Discrete Applied Mathematics, 1996; 65, 141-156.

Johannes U. Herrmann is a Ph.D. student at Curtin University of Technology, Western Australia. He is a member of the Institute for Multi-Sensor Processing and Content Analysis. He received his B.Sc. (Hons, Mathematics and Computer Science) and M.Sc. (Computer Science) from the University of Western Australia. His research interests include network reliability, artificial intelligence in online games, program synthesis and computers in education. He is a student member of the IEEE.

The biographies of **Sieteng Soh** and **Suresh Rai** can be found in [6].

Geoff West received the B.Sc. (Honors) in 1978 and the Ph.D. in 1982 from the City University, London. He was employed at the City University in the School of Engineering (1982-1990) and at the Department of Computing at Curtin University (1990-2008). His research interests include 3D object recognition, feature extraction, surveillance, real-time systems, automatic visual inspection, data mining, telemedicine and smart homes. He is currently Professor of Spatial Information in the Department of Spatial Sciences at Curtin University where he is focussing on object recognition in remotely sensed data, GIS and visualisation. Prof. West is a senior member of the IEEE, a member of the IET(UK), fellow of Engineers Australia, a member of the ACS (AUS), and is a Chartered Engineer.