A new symbolic approach for network reliability analysis

Marco Beccuti*, Andrea Bobbio †, Giuliana Franceschinis† and Roberta Terruggia†

*Dipartimento di Informatica,
Università di Torino, Torino, Italy
beccuti@di.unito.it

†DiSIT
Università del Piemonte Orientale, Alessandria, Italy
andrea.bobbio, giuliana.franceschinis, roberta.terruggia@mfn.unipmn.it

***Abstract*—In this paper we propose an improved BDD approach to the network reliability analysis, that allows the user to compute an exact solution or an approximation based on reliability bounds when network complexity makes the former solution practically impossible.**

To this purpose, a new algorithm for encoding the connectivity graph on a Binary Decision Diagram (BDD) has been developed; it reduces the computation memory peak with respect to previous approaches based on the same type of data structure without increasing the execution time, and allows us also to derive from a subset of the minpaths/mincuts a lower/upper bound of the network reliability, so that the quality of the obtained approximation can be estimated.

Finally, a fair and detailed comparison between our approach and another state of the art approach presented in the literature is documented through a set of benchmarks.

***Index Terms*—Network Reliability, Exact and Approximate Algorithms, Upper and Lower Bound, BDD**

Paper category: regular paper

Word count: about 8500 words
The material has been cleared through the affiliations of the author(s)

I. Introduction

Network structures have the property that each couple of nodes is usually connected by multiple paths making the systems intrinsically reliable. For this reason, the reliability of networks has always been a major concern in the reliability literature, as witnessed, for instance, by almost every issue of IEEE Transactions on Reliability.

A network is represented by means of a graph whose elements (both nodes and edges) are assumed to behave as binary objects, and they can be in one of the following two exhaustive and mutually exclusive states: working (up) or failed (down).

If a probability measure is assigned to the up or down state, a probability measure can be associated with the connectedness of the whole graph.

The *s-t network reliability* is defined as the probability that a source node s communicates with a sink node t through at least one path of working edges and has been the object of a continuing attention in the literature.

The many proposed algorithms can be roughly classified as exact or approximated algorithms. Exact algorithms pertain to two main categories; *i)* - pivotal decomposition using keystone components [18], [11]) and *ii)* - approaches where all the possibilities for which two specified nodes can communicate (or not communicate) are first enumerated (path/cut set search [3], [14]) and then the reliability expression is evaluated, resorting to different techniques [2], like inclusion-exclusion method or sum of disjoint products [13], [21], [1].

In the last decades, Binary Decision Diagrams (BDD) [7] have provided an extraordinarily efficient method to represent and manipulate Boolean functions [8], and have been also exploited to model the connectivity of networks and compute their reliability [22], [12], [6].

To the best of our knowledge, the algorithm presented in [22] is one of the best algorithms in terms of execution time and memory utilization for the computation of the network reliability and minpath/mincut set.

This algorithm generates the BDD encoding the connectivity function directly, via a recursive visit on the graph, without explicitly deriving the corresponding Boolean expression.

From this BDD it is possible to obtain, with some BDD manipulations taken from [19], the BDD encoding the minpaths and/or the BDD encoding the mincuts. With this approach even if only the list of minpaths/mincuts is necessary, the BDD encoding the connectivity function must be created.

The exact computation of the network reliability problem is known to be NP-complete [4], so that for large networks techniques providing approximate values appear to be the only possible solution. Natural and simple approximation methods are based on the computation of the reliability based on a subset of minpaths and mincuts.

Computing the reliability over a subset of minpaths provides a lower bound, while computing the reliability over a

subset of mincuts provides an upper bound, so that the total approximation error may be evaluated.
However, also with a reduced number of minpaths (mincuts) the computation of the probability of the bound using inclusion-exclusion formula or Sum of Disjoint Products (SDP that is known to perform better) may be very time consuming.
For this reason some authors [9] have used a further approximation by arresting the inclusion/exclusion formula to the first term.
Methods to find the mincuts that have major influence on the network reliability have been discussed [20], [9].

This paper proposes a new approach based on BDD to compute the exact network reliability or an approximation based on reliability bounds when the network complexity makes the exact solution intractable.
A known problem in the use of BDDs is that the peak occupation of the memory during the BDD construction may be orders of magnitude larger than the memory occupation of the BDD in its final form.
The present paper proposes a BDD manipulation algorithm that reduces the peak occupation with respect to approach presented in [22].
Furthermore, the proposed algorithm may compute the upper and lower bounds on the exact connectivity probability with a negligible additional cost.

With respect to the previous work presented in [22], in our approach the minpaths or the mincuts of the net are derived without storing the connectivity function.
Instead, if we need the network reliability then BDD encoding the whole connectivity function must be derived from the BDD encoding the minpaths or the mincuts through a symbolic algorithm.
Even if this requires to store the whole connectivity function, our approach is able to achieve a reduction in terms of memory peak and execution time with respect to other approach based on BDD as reported in the experiments of Sec. VI.

The paper is organized as follows: Sec. II describes the main concepts of network reliability analysis, Sec. III introduces BDD and its operators.
Sec. IV and Sec. V introduce our exact method and its extension to compute an approximation of network reliability value.
Sec. VI presents some numerical results comparing first our exact approach with our implementation of the one in [22]; second our exact approach with its approximated extension.
Finally, Sec. VII draws some conclusions and presents directions for future work.

II. Problem definition

A probabilistic network can be described as a labeled graph $\mathcal{G} = (V, E, P)$, where V is the set of vertices, $E \subseteq V \times V$ the set of edges (or arcs), and P a function that assigns a probability value to the elements of the graph (vertices and edges) of being in the up state.
In this paper we assume that only edges can fail and we use the notation e to denote a generic edge.

A network is undirected when the edges can be traversed in both directions, while a network is directed if the edges can be traversed only in one direction indicated by an arrow.
For undirected networks the presence of at least one edge per node guarantees that all the nodes are connected.
Usually real networks are much more connected than this minimal threshold thus allowing multiple paths among any pair of vertices. Figure 1 exemplifies the graphical representation of a bridge network.

In this paper, we concentrate on the evaluation of the s,t reliability, defined as the probability that a source node s is connected with a terminal node t by at least one path of working edges.
A path is a subset of arcs that guarantees the source s and sink t to be connected when all the arcs of the subset are working.
A path is a minpath if there does not exist any subset of the path that is also a path.
Furthermore, a path H_i can be also defined as a boolean formula of conjunction of terms indicating which arcs are in *up* state.
If H_1, H_2, ..., H_n are the n minpaths of a network, the connectivity function $C_{s,t}$ is defined as the disjunction of its minpaths:

$$C_{s,t} = H_1 \vee H_2 \vee \ldots \vee H_n = \bigvee_{i=1}^{n} H_i \tag{1}$$

The *disconnectivity* function is then defined as:

$$\overline{C_{s,t}} = \neg\, C_{s,t} \tag{2}$$

For instance, in the network example of Figure 1 the minpaths are $H_1 = e_1 \wedge e_4$, $H_2 = e_2 \wedge e_5$, $H_3 = e_2 \wedge e_3 \wedge e_4$. The path $\langle e_1 \wedge e_2 \wedge e_4 \rangle$ is a path but not a minpath.

Similarly, we can define a cut as a subset of arcs such that the source s and sink t are disconnected if all the arcs of the subset are down.
A cut is a mincut if there does not exist any subset of the cut that is also a cut.
A cut K_j is a boolean formula of conjunction of terms indicating which are are in *down* state. If K_1, K_2, ..., K_m are the m mincuts of a network, the disconnectivity function $\overline{C_{s,t}}$ is defined as as the disjunction of its mincuts:

$$\overline{C_{s,t}} = K_1 \vee K_2 \vee \ldots \vee K_m = \bigvee_{j=1}^{m} K_j \tag{3}$$

In the network example of in Figure 1 the mincuts are $K_1 = \overline{e_1} \wedge \overline{e_2}$, $K_2 = \overline{e_4} \wedge \overline{e_5}$, $K_3 = \overline{e_2} \wedge \overline{e_4}$, $K_4 = \overline{e_1} \wedge \overline{e_3} \wedge \overline{e_5}$.

If the function P assigns to edge e_i a probability p_i of being in the up state and $1 - p_i$ of being in the down state, the s,t network reliability can be computed as $R_{s,t} = Pr(C_{s,t})$.

Similarly the unreliability is computed as $U_{s,t} = 1 - R_{s,t} = Pr(\overline{C_{s,t}})$.

Suppose now that we take a subset of the mincuts and we define the Boolean function

$$C'_{s,t} = \bigvee_{i=1}^{n'} H_i \;\; \textit{with } n' < n \tag{4}$$

we have that:

$$Pr(C'_{s,t}) \leq Pr(C_{s,t}) = R_{s,t}. \tag{5}$$

Hence $Pr(C'_{s,t})$ provides a lower bound to the s,t reliability $R_{s,t}$. Similarly if we take a subset $m' < m$ of the m mincuts and we define the Boolean function

$$\overline{C''_{s,t}} = \bigvee_{j=1}^{m'} K_j \tag{6}$$

we have that:

$$P(\overline{C''_{s,t}}) \leq Pr(\overline{C_{s,t}}) = U_{s,t}. \tag{7}$$

Hence $P(\overline{C''_{s,t}})$ provides a lower bound to the s,t unreliability $U_{s,t}$ and $1 - Pr(\overline{C''_{s,t}})$ an upper bound to the s,t reliability $R_{s,t}$ and $1 - Pr(\overline{C'_{s,t}})$ is an upper bound to $U_{s,t}$.
Hence, taking any subset of minpaths and any subset of mincuts and applying Equations (5) and (7) we obtain a lower and an upper bound of $R_{s,t}$, respectively.

III. Binary Decision Diagram

Before presenting in detail the algorithms to compute the network reliability, we introduce the basic characteristics of Binary Decision Diagrams (BDD) and of the functions to manipulate them.

BDDs [7] are a well-known graph data structure used to represent and manipulate efficiently complex Boolean functions over Boolean variables.
They allow to find efficient solutions for a large class of problems in different research fields (e.g. digital system design, combinatorial optimization, mathematical logic,. . .).
For instance, in [22] the authors show that BDD can be efficiently used for reliability graph analysis encoding directly the connectivity function on BDD.

Formally a BDD is a directed acyclic graph that represents functions of type $f : \mathcal{V}_N \times \cdots \times \mathcal{V}_1 \rightarrow \{0, 1\}$, where $\mathcal{V}_i$ is a Boolean variable.
Nodes without outgoing arcs are called *terminal*, nodes without incoming arcs are called *root*, all the others are termed *internal*; a node at level i has two outgoing arcs.
Therefore, if we represent a Boolean function over N variables with a BDD then the internal nodes are mapped onto the N variables and their outgoing arcs correspond to the possible assignments to these variables (true/1 or false/0).
The two possible values of the function (i.e. *false* and *true*) are represented by two terminal nodes 0 and 1. A path in the BDD from a root node to node 0 represents an assignment to the N variables that evaluates to false, and viceversa for a path to 1.

Figure 2 shows a BDD encoding the Boolean function $e_2\, e_5\, \overline{e_3}\, \overline{e_1}\, \overline{e_4} + e_2\, \overline{e_5}\, e_3\, \overline{e_1}\, e_4 + \overline{e_2}\, e_5\, \overline{e_3}\, e_1\, e_4$. The edges labeled with 0 represent the assignment false for the corresponding variable while the edges labeled with 1 the assignment true.

A BDD in which nodes are organized into levels (one per variable) are called *ordered*.
Moreover, two BDD nodes are duplicate one of the other if their outgoing edges go to the same set of nodes.
An ordered BDD without duplicate nodes is called *canonical*, and it assures that if two boolean functions are equal (have the same value over all possible inputs), then they have the same canonical BDD representation.
Optionally, BDDs may also eliminate *redundant* nodes, which are non-terminal nodes with its edges pointing to the same node. The BDD keeping all redundant nodes are called *quasi–reduced*, while those that eliminate all redundant nodes are called *fully–reduced* and allow skipping levels.

For instance, the BDD in Fig. 2 is an ordered *fully–reduced* canonical BDD where the variables are ordered as follows $e_2 \prec e_5 \prec e_3 \prec e_1 \prec e_4$.
It is important to highlight that the variable order affects the BDD size (number of nodes), and finding the optimal order that minimizes the BDD size is a NP-complete problem.
In the rest of this paper we will use BDD to indicate an ordered *fully–reduced* canonical BDD.

A very interesting aspect of BDDs is that they allow to translate the basic boolean functions like *and* and *or* to basic operations like intersection (for the *and*) and union (for the *or*) over two BDDs.
A crucial factor in the efficiency of these operators is the utilization of a hash table, called *computed-table*, used to cache the result of each intermediate call to the algorithm, so that it is never the case that the same operation is executed twice on the same pair of nodes, if the cache is large enough.

IV. The proposed approach: exact method

In this section an extension of our previous work [5] is described. We present our approach in detail, first describing how to derive the set of minpaths/mincuts from the network and how to encode them on a BDD. Then, we describe our algorithm to derive from the BDD encoding the minpaths (mincuts) another BDD encoding the connectivity (disconnectivity) function. Finally we explain how to compute the network s,t reliability using the algorithm proposed in [22] from the connectivity $C_{s,t}$ (disconnectivity $\overline{C_{s,t}}$) function encoded in a BDD.

In the next section we show how this approach can be extended to provide an approximate solution in the form of upper and lower bounds on $R_{s,t}$/$U_{s,t}$.

A. Minpath search

In our approach the minpath search is implemented through a Depth First Search (DFS), an uninformed tree search algorithm that, starting from the root explores as far as possible

Algorithm 1 MinPath search

1: **procedure** MINPATHSEARCH(BDD_Q, Q, c, n, t)

BDD_Q = BDD encoding the minpaths
Q = set of edges belonging to the current minpath
c = current node
n = parent node of c (along the current path, NULL if c == source)
t = target node

```
2:     Q.insert(n,c);
3:     if (c == t) then
4:         BDD_Q.insertMinPath(Q);
5:     else
6:         for (j ∈ c.Child()) do
7:             MinPathSearch(BDD_Q,Q,j,c);
8:         end for
9:     end if
10: end procedure
```

along each branch before backtracking.
The found minpaths are encoded on a BDD, where each level is associated with an edge of the graph according to a predefined order.
Hence, each BDD path corresponds to a minpath *mp* where only the variables/edges included in *mp* are true.

For instance, this algorithm applied on the directed bridge network in Figure 1 returns the minpath BDD in Figure 2 assuming the following variables ordering $e_2 \prec e_5 \prec e_3 \prec e_1 \prec e_4$. The BDD encodes the minpaths $\langle e_1, e_4\rangle$, $\langle e_2, e_5\rangle$ and $\langle e_2, e_3, e_4\rangle$; corresponding respectively to the BDD paths $\langle 0,0,0,1,1\rangle$, $\langle 1,1,0,0,0\rangle$ and $\langle 1,0,1,0,1\rangle$ where the notation $\langle S_{e_2}, S_{e_5}, S_{e_3}, S_{e_1}, S_{e_4}\rangle$ represents the fact that edge e_i is up if $S_{e_i} = 1$ while it is down if $S_{e_i} = 0$

It is important to highlight that since in general a minpath involves only a subset of the network edges then a more efficient encoding of the minpath BDD could be obtained using Zero-suppressed DD (ZDD) [16], a BDD extension where a node is not explicitly represented if its positive edge points to terminal node 0.

The pseudo-code of minpath search is shown in Algorithm 1, where the BDD_Q, initially empty, encodes the found minpaths, the set Q, initially empty, stores the edges of the current minpath, the pointer c, initially pointing to the source node, identifies the current node and the pointer n, initialized to NULL, identifies a node reaching c.

Recursively, the algorithm updates Q inserting the edge connecting n to c (*insert()* method), and if c is the target node then Q is inserted in BDD_Q. Otherwise the algorithm explores all the children of c (*Children()* method).

The pseudo-code in Algorithm 1 can be extended to avoid loops introducing a further list storing all the nodes already visited, so that a node shall be explored iff it is not in such list.

Observe that the choice of implementing a DFS instead of Breadth First Search (BFS), another uninformed tree search algorithm that starting from the source node explores one level

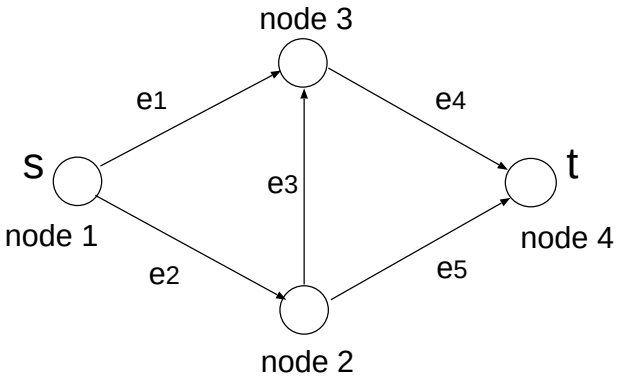

Fig. 1. A directed bridge network.

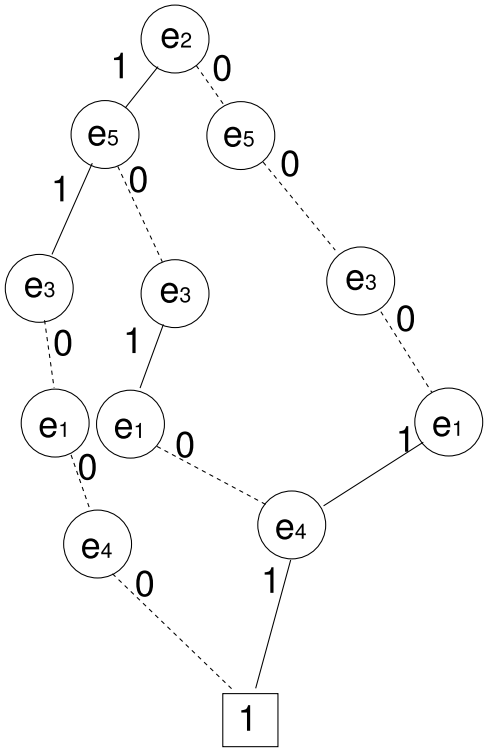

Fig. 2. Minpaths BDD of the directed bridge network.

of the graph at a time, is due to the fact that the latter requires to maintain in memory for each visited node in the frontier its corresponding "partial" minpath.

B. Mincut search

The mincut search is more complex, since it requires for each step to explore all the nodes in the current frontier. The found mincuts are then encoded on a BDD, where each level is associated with an edge of the graph according to a predefined order. Each BDD path corresponds to a mincut where only the variables/edges included in the mincut are false

For instance, this algorithm applied on the directed bridge network in Fig. 1 returns the mincut BDD in Fig. 3 with edge

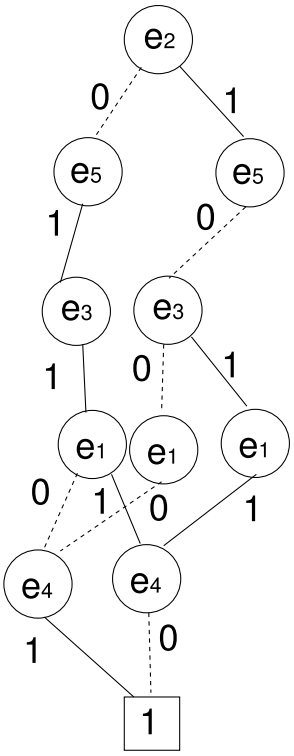

Fig. 3. Mincuts BDD of the directed bridge network.

order $e_2 \prec e_5 \prec e_3 \prec e_1 \prec e_4$. The BDD encodes the mincuts $\langle \overline{e_1}, \overline{e_2} \rangle$, $\langle \overline{e_4}, \overline{e_5} \rangle$, $\langle \overline{e_2}, \overline{e_4} \rangle$ and $\langle \overline{e_1}, \overline{e_3}, \overline{e_5} \rangle$ corresponding to the following BDD paths $\langle 0, 1, 1, 0, 1 \rangle$, $\langle 1, 0, 1, 1, 0 \rangle$, $\langle 0, 1, 1, 1, 0 \rangle$ and $\langle 1, 0, 0, 0, 1 \rangle$ using already explained for the minpaths.

The pseudo-code of the mincut search is shown in Algorithm 2, where the sets W and L, initially empty, are used to maintain the current mincut and the nodes in the frontier, while the pointer c, initially pointing to the source node, identifies the current node.

First the algorithm initializes the set W' as a copy of W where all the edges connected to c are inserted (operator $+$ and *Edge()* method) and then all the edges connecting it with a node in L are removed (operator $-$ and *EdgeConnecting()* method). Method *EdgeConnecting()* returns all the edges connecting directly c with a node in the current frontier L. BDD_C, the BDD encoding the mincuts, is updated (*InsertMinCut()* method) as follows: if the mincut in W' or any subset of it is not already in BDD_C then it is inserted and all its supersets are removed by BDD_C.

After that L' is created as a copy of L where all the children of c except the target node are inserted (*FindChildrenNotTarget()* method), and this procedure is recursively called on all the nodes in $L'' = L' - \{i\}$.

A cache can be used to improve the algorithm efficiency avoiding to re-compute the same mincuts, it can be implemented as a hash table using the node identifier and its W list as key.

This code, as the minpath one, can be extended to avoid loops introducing a further list storing all the nodes already visited.

Algorithm 2 MinCut search

```
1: procedure MinCutSearch(BDD_C, L, W, c, t)
   BDD_C = BDD encoding the mincuts
   W = set of edges in the current mincut
   L = set of nodes in the frontier
   c = current node
   t = target node
2:     W'=W +c.Edge()-c.EdgeConnecting(L);
3:     BDD_C.InsertMinCut(W');
4:     L'=L + c.FindChildrenNotTarget();
5:     for (i ∈ L') do
6:         L"=L'-{i}
7:         MinCutSearch(BDD_C,L",W',i);
8:     end for
9: end procedure
```

C. Connectivity (disconnectivity) function encoding

Before describing how to derive the BDD encoding the connectivity (disconnectivity) function from the minpath (mincut) BDD we introduce the following two functions, called $\mathcal{F}_{com}$ and $\mathcal{F}_{rem}$, as follows:

Definition 1: Let Q be a generic set of sets in 2^E, then $\mathcal{F}_{com}(Q)$ returns the smallest set Q' that satisfies:

- $\forall q \in Q \Rightarrow q \in Q'$
- $\forall q \in Q, q' \in 2^E : q \subset q' \Rightarrow q' \in Q'$

Definition 2: Let C be a generic set of sets in 2^E, then $\mathcal{F}_{rem}(C)$ returns the smallest set C' that satisfies:

- $\forall c \in C \Rightarrow c \in C'$
- $\forall c \in C, c' \in 2^E : c' \subset c \Rightarrow c' \in C'$

It is easy to prove that if Q encodes the whole set of the minpaths of a network then the function $\mathcal{F}_{com}$ applied on it returns a new set encoding the connectivity function; in the same way if C encodes the whole set of the mincuts of a network then the function $\mathcal{F}_{rem}$ applied on it returns a new set encoding the disconnectivity function.

These two functions can be easily implemented with an explicit algorithm that iteratively discovers and inserts all the correct supersets/subsets; however a symbolic implementation of such algorithm based on BDD can be much more efficient in terms of memory and execution time: only in the worst case it will require to add all the connectivity or disconnectivity states one by one.

Intuitively, these functions can be implemented on the minpath (mincut) BDD, visiting recursively all their nodes and when the true (false) edge of a node points to the terminal node 0 redirecting it to the same child node pointed by its false (true) edge.

This operation must always assure that no duplicate nodes are inserted in the BDD, so that when a node becomes a duplicate of another node in the same level then it is discarded and its

Algorithm 3 Symbolic implementation of $\mathcal{F}_{com}$

```
1: function F_com(B)
   B = BDD encoding minpath function
2:     if (B==True)||(B==False) then
3:         return B
4:     else
5:         R=Cache.search(B);
6:         if (R!=NULL) then
7:             return R;
8:         else
9:             B1=B.PtrTrue();
10:            B0=B.PtrFalse();
11:            E=F_com(B0)
12:            K=F_com(B1)
13:            T=Add(K,E)
14:            if (T==E) then return T
15:            R=createBDD(T,E)
16:            Cache.insert(B,R);
17:            return R
18:        end if
19:    end if
20: end function
```

Algorithm 4 Symbolic implementation of $\mathcal{F}_{rem}$

```
1: function F_rem(B)
   B = BDD encoding mincut function
2:     if (B==True)||(B==False) then
3:         return B
4:     else
5:         R=Cache.search(B);
6:         if (R!=NULL) then
7:             return R;
8:         else
9:             B1=B.PtrTrue();
10:            B0=B.PtrFalse();
11:            T=F_rem(B1)
12:            K=F_rem(B0)
13:            E=Add(K,T)
14:            if (T==E) then return T
15:            R=createBDD(T,E)
16:            Cache.insert(B,R);
17:            return R
18:        end if
19:    end if
20: end function
```

input edges redirected to other node.

The pseudocodes of these two methods are shown in Algorithms 3 and 4; where methods *PtrTrue()* and *PtrFalse()* return the nodes pointed by the true and false edges of the root node; and method *createBDD()* with input parameter T, E creates a new node with the true edge pointing to T and the false one to E. Observe that these algorithms are dual to the approach proposed by Rauzy in [19] to derive from a boolean function, encoded on BDD, a new BDD encoding only its minimal solutions.

In detail, the Algorithm 3 (4) starting from the BDD_Q (BDD_C) root recursively visits the subtrees pointed by the true and false edges of the current node (called in the rest of this paper 1-subtree and 0-subtree).
Then the function *Add*, shown in Algorithm 5 (where *RLevel()* returns the level of the BDD root) is applied on the updated subtrees, for $\mathcal{F}_{com}$ all the paths in the 0-subtree are recursively added in the 1-subtree, while viceversa for $\mathcal{F}_{rem}$.
Observe that Algorithm 5 is an instantiation of well-known "Apply" algorithm proposed by Bryant[7].
Finally, in line 15 we avoid to insert redundant nodes in the BDD, indeed if the new node is redundant (T==E) then this node is skipped and its parent is redirected to its unique child.

D. Reliability (unreliability) algorithm

To compute the network reliability (unreliability) first we have to assign to each variable x_i a probability p_i to be up ($1 - p_i$ to be down), so that we can compute the probability $Pr\{F\}$ of the function F by applying recursively Eq. 8.

$$Pr\{F\} = p_i Pr\{F_{1-subtree}\} + (1-p_i)Pr\{F_{0-subtree}\} = Pr\{F_{0-subtree}\} + p_i(Pr\{F_{1-subtree}\} - Pr\{F_{0-subtree}\}) \quad (8)$$

In Algorithm. 6 a sketch version of such algorithm is shown.
It takes in input the BDD B encoding the connectivity (disconnectivity) function and the vector P storing the edges probabilities and returns the corresponding reliability (unreliability) value. It starts from the B root node and recursively calls itself on the two branches until the terminal nodes are reached. Indeed, the probability of a node x depends on the probability of the 0-subtree multiplied by the probability of x being down plus the probability of the 1-subtree multiplied by the probability of x being up.

E. Comparison with previous approach

In [22] Trivedi at al. propose an algorithm that generates directly the connectivity function BDD, via a recursive visit on the graph, without explicitly deriving the Boolean expression. The algorithm starts from the source node s and visits the graph (according to a given but arbitrary visiting strategy) until the sink node t is reached. The BDD construction starts once the sink node t is reached. The BDDs of the nodes along a path from s to t are combined in AND, while if a node has more than one outgoing edge the BDDs of the paths starting from each edge are combined in OR. To derive the minpaths (mincuts) the list of paths (cuts) obtained from the BDD must be minimized. An alternative approach, proposed in [19], consists in transforming the original BDD into a new graph embedding all and only the minpaths (mincuts). Details

Algorithm 5 Add algorithm

```
1: function ADD(F,G)
   F, G = BDDs
2:     if ((F==G) || (F==True) || (G==False)) then
3:         return F
4:     else
5:         if ((G==True) || (F==False)) then
6:             return G
7:         else
8:             R=Cache.search(F,G);
9:             if (R!=NULL) then
10:                return R;
11:            else
12:                if (F.RLevel()>G.RLevel()) then
13:                    T=Add(F,G.PtrTrue());
14:                    E=Add(F,G.PtrFalse());
15:                else
16:                    if (F.RLevel()==G.RLevel()) then
17:                        T=Add(F.PtrTrue(),G.PtrTrue());
18:                        E=Add(F.PtrFalse(),G.PtrFalse());
19:                    else
20:                        T=Add(F.PtrTrue(),G);
21:                        E=Add(F.PtrFalse(),G);
22:                    end if
23:                    if (T==E) then return T
24:                    R=createBDD(T,E)
25:                    CACHE.insert(F,G,R)
26:                    return R
27:                end if
28:            end if
29:        end if
30:    end if
31: end function
```

Algorithm 6 Reliability algorithm

```
1: procedure PROBBDD(B, P)
   B = BDD encoding the connectivity function
   P = vector storing the edges probabilities
2:     if (B == True) then
3:         return 1;
4:     else
5:         if B == False then
6:             return 0;
7:         else
8:             PF=Cache.search(B);// return -1 if B is not in cache.
9:             if (PF!=-1) then
10:                return PF;
11:            else
12:                PF1=ProbBDD(B.PtrTrue(),P);
13:                PF2=ProbBDD(B.PtrFalse(),P);
14:                PF=PF2 + P[Root(B)] * (PF1 - PF2);
15:            end if
16:        end if
17:        cache[B]=PF;
18:        return PF;
19:    end if
20: end procedure
```

of the transformation algorithm are in [19].
With respect to this previous work our approach does not require to store the connectivity function in order to derive the minpaths or the mincuts of the net. Instead the connectivity function can be derived from the minpaths BDD leading to a lower memory peak during the computation. From these BDDs, encoding the minpaths (or the mincuts), the network reliability can be computed using a symbolic approach. Even if this choice requires to store the whole connectivity function, thanks to our strategy it is still possible to achieve a reduction in terms of memory peak as reported by the experiments (Sec. VI).

V. THE PROPOSED APPROACH:APPROXIMATE METHOD

The exact approach, described above, can be easily extended to compute an approximate solution, when the network complexity makes it practically impossible to compute the exact one.

The general idea behind this extension is that an approximate value of the network reliability can be generated considering only a subset of all the minpaths and mincuts. In this way, as already explained in section II, the reliability value derived by the subset of minpaths is a lower bound of the exact reliability value, while the one derived by the subset of mincuts is an upper bound; so that the gap between these two values provides an estimation for the accuracy of the approximation.

Practically, this requires to update the MinPath search and MinCut search algorithms allowing the user to specify constraints on their execution times, so that only the minpaths and the mincuts generated under these constraints are inserted in the minpath and mincut BDDs.
Moreover, for the minpaths, we allow the user to specify another constraint on the minpath probability (expressed as product of the probabilities to be up of all edges involved in the minpath): only the minpaths with such probability greater than this threshold are inserted in the minpath BDD. This is a good heuristics to assure that the more meaningful minpaths will be involved in the approximation.
Observe that, when all the edges have the same probability to be up this corresponds to impose a limitation on the minpath length (we shall call minpath length constraint). Under this assumption it is possible to define a more efficient minpath search algorithm that step by step increases the length of the minpaths to be searched until this becomes equal to the minpath length constraint or the time constraint is reached.
In this way this algorithm is able to find the most relevant minpaths thus improving the reliability lower bound.
Observe that all the experiments presented in section VI use this algorithm to compute the minpaths.

For the mincuts, the user can also specify a further constraint

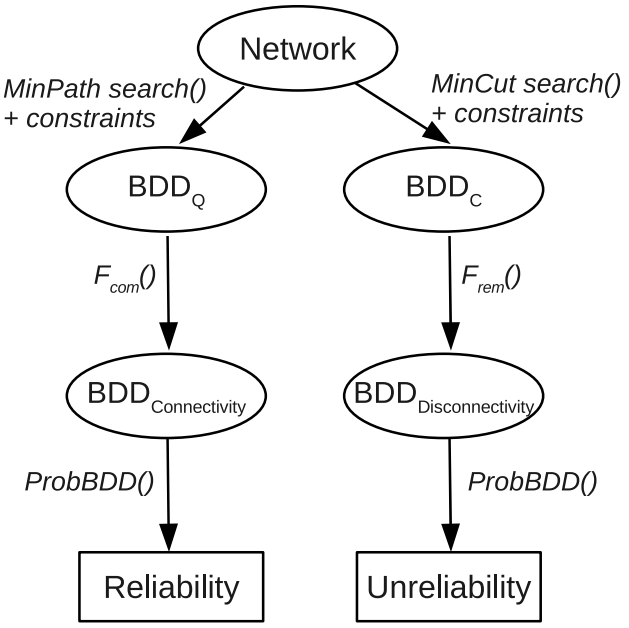

Fig. 4. General scheme of our approximate approach

on the mincut probability: only the mincuts that contribute more significantly (above a given threshold) to the unreliability are inserted in the mincut BDD.
Observe that, when all the edges have the same probability to be up this corresponds to impose a limitation on the mincut size (we shall call minpath size constraint).

Obviously, these constraints define a trade-off between accuracy of the approximation and computational resources (memory and time) required for the solution, hence an incremental iterative approach can be defined where the network reliability is iteratively improved relaxing the above constraints until a sufficient accuracy is reached or no more computational resources are available.

The general scheme of our approximate approach is reported in Fig. 4.

VI. Experimental results

In this section some experimental results are presented, which were computed thanks to a prototype implementation of the proposed approach where the BDD implementation has been provided by an existing open–source library (Meddly) [15].
Our choice to use this new open–source library was motivated mainly by the fact that it automatically handles the complex aspects of using BDDs (such as caching and garbage collection) and provides a simple interface for common BDD operations.

In order to test our approach we define a set of network benchmarks of different topologies.

1) Regular Networks : Regular networks are represented by graphs that can be described by means of defined geometrical properties.
In particular we have investigated networks where the nodes are located on a regular square grid of $N \times N$ nodes and where each node is connected with two neighbors (the up-right one and down-left one) in case of directed network, or with all the four neighbors (right, left, up and down ones) if the network is undirected.
Fig.5 shows an example of directed network.

2) Small world network: A small world network is a network where most nodes are not connected directly, but where it is possible to connect any two nodes in the network through few edges.
The average shortest distance between any two nodes increases logarithmically with the number of nodes (n).

$$L \sim log(n)$$

3) Random Network: Paul Erdős and Alfréd Rényi were the first to define a Random Graph (RG)[10] as N labeled nodes connected by n edges which are chosen randomly from the $E_n = \frac{N(N-1)}{2}$ possible edges for the undirected networks and from the $E_n = N(N-1)$ possible edges for the directed networks.
In total there are

$$\binom{E_n}{n}$$

different graphs with N nodes and n arcs.
An alternative way in order to obtain a RG is to start with N nodes and connect each pair of nodes with probability p. The total number n of edges is a random variable with expectation $E(n) = p\frac{N(N-1)}{2}$.

In RG networks, the degree distribution $P(k)$ follows a Poisson distribution;

$$Pr(k) \approx e^{-pN}\frac{(pN)^k}{k!} = e^{-<k>}\frac{< k >^k}{k!}$$

where $< k >$ is the average degree of the network and the distribution has a peak in $< k >$.

4) Scale Free Network: In Scale Free (SF) networks, the degree distribution P(k) follows a power-law [17]:

$$Pr(k) \sim k^{-\gamma}$$

where γ is a real positive constant. SF networks manifest when nodes are highly non-equivalent. Such networks have been named Scale Free because a powerlaw has the property of having the same functional form at all scales.
In fact, power- laws are the only functional forms that remain unchanged, apart from multiplicative factors, under a rescaling of the independent variable k. SF networks result in the simultaneous presence of a small number of very highly connected nodes (the hubs) linked to a large number of poorly connected nodes (the leaves).

In Table I we compare our approach, in terms of memory and execution time, with respect to an implementation developed by ourselves of the algorithm proposed by Trivedi at al in [22] and reported in Appendix, that as already explained

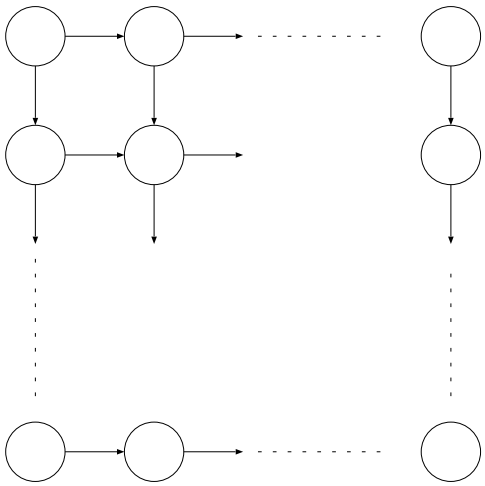

Fig. 5. Benchmark network

in the introduction, is a state of the art network unreliability computation algorithm based on BDD.

Two scalable benchmarks have been carried out to evaluate and to compare the efficiency of these two approaches: for the former we take into account a regular $n \times n$ network topology of increasing complexity.

Moreover, we consider random undirected network where the pair of nodes connected by each edge are chosen randomly among all nodes in the network.

These experiments have been executed on a 2.13 GHz Intel Xeon 64-bit processor with 132Gb RAM, and the network unreliability has been computed using only the minpaths and increasing the network complexity - i.e. for the regular network we increase the number of nodes, while for the random one we increase both the number of nodes and the number of edges. Moreover, for the regular network the upper left node and the lower right node are selected as source and target, while for the random network they are randomly chosen.

The reported experiments show that for the first two networks our approach achieves a good memory saving and a discrete execution time reduction with respect to the Trivedi at Al. approach. For instance, the memory reduction factor for the regular undirected network 7×7 is ~ 170; while the execution time is decreased by a factor of ~ 19.

For the last two benchmarks the achieved reduction factor still reaches a good memory saving increasing the model size (e.g. for the random directed network with 500 nodes and 1494 edges the memory reduction is 18, and for the random undirected network with 60 nodes and 160 edges the memory reduction is 20.33); while only for the first benchmark the execution time is decreased.

These experimental results suggest us to plan, as future work, a set of new experiments to better characterize for which types of networks our approach allows a higher reduction than the one proposed by Trivedi at Al.

In table II we compare in terms of memory, execution time and solution quality our exact approach with the one computing an approximation of the unreliability value on undirected/directed random networks.
These experiments have been executed on a 2.13 GHz Intel Xeon 64-bit processor with 132Gb RAM, and the exact value of network unreliability has been computed using only the minpaths.

The first three columns are related to the exact solution and report the used memory, the execution time and the exact unreliability value.
Instead, the last six columns are related to the approximate approach and show the used constraints (time and path/cut size) for computing the approximate solution, the total memory required, the total execution time, the upper (computed through a subset of minpaths) and lower (computed through a subset of mincuts) bounds for the network unreliability and the accuracy of the approximation in terms of the distance between the two bounds, divided by the average of the two bounds. In the table a line is reported instead of the accuracy value, when the bounds distance is orders of magnitude wide.

Observe that often a threshold $UnrelThr$ on the unreliability is part of the network reliability problem formulation: in this case if the upper bound lays below the $UnrelThr$ threshold, even if the bounds are very distant from each other the objective of the study is reached.
If instead the lower bound lays above the $UnrelThr$ then there is no hope that the system can satisfy the requirements. So the accuracy is interesting only in case no specific threshold is indicated, or when the indicated threshold lays between the two bounds.
The experiments reported in this table show how the approximate approach can be a good solution when the network complexity increases. Indeed, for the first two random undirected networks the approximate solution is able to derive that exact unreliability is lower than $2.070248e^{-06}$ and $2.145800e^{-08}$ decreasing the used memory by a factor of ~ 7.5 and ~ 18.5 and the execution time of 4 and 20 respectively.
This reduction in terms of memory and time is more notable for the last model where the memory saving is about 350 times and the execution time is reduced by a factor of 840 times reaching a solution accuracy equal to 3.74.
Finally, table III shows some experiments obtained applying our approximate approach on four different classes of networks: scale-free, small-world, random and regular.

In detail, the first three networks are scale-free and are obtained varying the number of nodes and edges, and the network diameter; the second two are small-world networks obtained varying the number of nodes and edges and the average degree of connectivity of each node.

Then, two random networks with different number of nodes

and edges are considered. Instead, the last benchmark is a regular network with 223 nodes and 252 edges connected as shown in Fig. 6.

For this model the exact reliability value can be easily computed by decomposing it into as many components as the number of repeating patterns with one source and one destination, solving each component in isolation, and then computing the reliability as the product of the reliabilities of each components (since they are connected in series).
In the special case when all edges have same probability p it is enough to compute with any method the reliability of any component (since they are all identical) and then compute the product.
We have thus computed the exact value, and in this case it is very close to the upper bound obtained with the third set of constraints.
The first four columns in table III are related to the minpath approximation; while the next four to the mincut approximation.
The last column represents the relative accuracy and is computed as for Table II.
For each type of approximation the number of minpaths/mincuts, the used constraints, the total execution time (i.e. minpaths/mincuts search + connectivity graph generation + approximation computation), and the approximate value of unreliability are reported.
The results in this last table highlight that our approximate approach is able to compute a good approximation for network that otherwise are intractable.
Two further considerations can be derived, first the minpath approximation is more efficient (in terms of execution time and solution quality) than the mincut one.
This is mainly due to the fact that the developed minpath search, as explained in Sec.IV and Sec. V, is able to find the most important minpaths that can greatly decrease the unreliability upper bound. For this reason, in the future we are going to study how to improve the mincut search to find first the most important mincuts.
The second consideration is related to the fact that in most of these experiments to further increase the solution quality quickly leads to an exponential growth of the execution time. For instance, for the Small-world network with 500 nodes and 5000 edges and average degree of connectivity equal to 20, a small increment in the number of considered minpaths (i.e. ~ 150) increases the computation time more than 3 hours.

VII. Conclusion

In this paper we have presented a new symbolic approach based on BDDs to compute the network reliability using the minpaths and/or the mincuts.
The performance of this approach was compared to a state of the art network reliability algorithm [22] through a set of experiments on different types of networks; the comparison has shown a generalized, and in same cases very consistent, memory saving and execution time reduction.

Moreover, we have shown how the proposed approach can be used for the computation of (un)reliability bounds: applying the same approach to any subset of minpaths we obtain a (upper)lower bound while applying the approach to any subset of mincuts we obtain an (lower)upper bound.
The upper and lower bounds have been compared with the exact solution in a first set of experiments, highlighting how a good accuracy can be obtained with a substantial saving in memory occupation and computational time.
In a second set of experiments we have computed the bounds for networks whose dimensions are far beyond the capabilities of any exact algorithm, showing again how a good accuracy can be attained with a reasonable computational effort.

From all these experiments two main considerations were derived: first, the implementation of the approximation based on the minpaths is more efficient (in terms of execution time and solution quality) than the one based on the mincuts; second, in most experiments a further increase in the solution quality quickly leads to an exponential growth of the execution time.

According to the obtained results two future developments will be pursued. The first one intends to investigate for which type of networks our approach reaches higher speed up factors with respect to traditional approaches. The second one, is directed toward the possibility of improving the mincut search finding a way to select the most important mincuts as we do for the minpaths.
Finally we will investigate how other reduction policies (e.g. zero-suppressed) may improve the performance of our approach.

References

[1] J.A. Abraham. An improved algorithm for network reliability. *IEEE Transaction on Reliability*, 28:58–61, 1979.

[2] A. Agrawal, , and R. E. Barlow. A survey of network reliability and domination theory. *Operations Research*, 32:478–492, 1984.

[3] A.O. Balan and L. Traldi. Preprocessing minpaths for sum of disjoint products. *IEEE Transaction on Reliability*, 52(3):289–295, September 2003.

[4] M.O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, R-35:230–239, 1986.

[5] M. Beccuti, S. Donatelli, G. Franceschinis, and R. Terruggia. A new symbolic approach for network reliability analysis. volume TR-INF-2011-06-02-UNIPMN. Ed. Computer Science Department, UPO, 2011.

[6] A. Bobbio and R. Terruggia. Reliability and quality of service in weighted probabilistic networks using algebraic decision diagrams. In *Proceedings IEEE Annual Reliability and Maintainability Symposium*, pages 19–24, Fort Worth, TX, 2009.

[7] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[9] E. S. Elmallah and H. M. F. AboElFotoh. Circular layout cutsets: An approach for improving consecutive cutset bounds for network reliability. *IEEE Transactions on Relibility*, 55:602–612, 2006.

[10] P. Erdös and A. Rényi. *On Random Graphs. I.* Publicationes Mathematicae, 1959.

[11] G. Hardy, C. Lucet, and N. Limnios. Computing all-terminal reliability of stochastic networks by Binary Decision Diagrams. In *Proceedings Applied Stochastic Modeling and Data Analysis*, 2005.

[12] G. Hardy, C. Lucet, and N. Limnios. K-terminal network reliability measures with Binary Decision Diagrams. *IEEE Transactions on Reliability*, 56:506–515, 2007.

	Existing Approach		Our Approach		
Size	Mem.	T.	Mem.	T.	Mem. Red.
$n \times n$	Regular Directed network				
8×8	2.9MB	0	245KB	0	12
9×9	8.8MB	0	568KB	0	15.49
10×10	25.9MB	5	1.4MB	1	18.12
11×11	71.8MB	12	3.4MB	5	20.66
12×12	194MB	47	8,3MB	24	23.27
13×13	509MB	171	19.7MB	108	25.76
14×14	**out of memory**	-	46MB	1089.52	-
15×15	**out of memory**	-	107MB	4797.68	-
16×16	**out of memory**	-	245.6MB	21707.78	-
$n \times n$	Regular Undirected network				
5×5	1.9MB	0	112KB	0	17.74
6×6	112MB	41	1.4MB	27.35	77
7×7	1.9GB	19,490	11.7MB	1,004.42	170.3
N, E	Random Directed network				
$500, 1494$	97.2GB	32,400	5.4GB	25,200	18
N, E	Random Undirected network				
$20, 28$	106KB	0	61.7KB	0	1.73
$30, 36$	125KB	1	52KB	0	2.38
$30, 41$	2MB	1	568KB	0	3.69
$50, 144$	1.7GB	49	100MB	48	17.45
$60, 160$	10GB	168	500MB	120	20.33

TABLE I
TIME (SECONDS) AND MEMORY REQUIRED FOR COMPUTING THE NETWORK UNRELIABILITY.

Exact solution			Aproximate solution					
Mem.	T.	Exact	Const.	Mem.	T.	Upper Bound	Lower Bound	Accuracy(%)
Random **undirected** network with 50 nodes and 144 edges								
132MB	8m	$1.020114e^{-06}$	10s. – 30	10MB	25s	$1.039438e^{-02}$	$9.999789e^{-13}$	-
			30s. – 30	35MB	80s	$1.020117e^{-04}$	$2.000078e^{-12}$	-
			45s. – 30	25MB	2m	$2.070248e^{-06}$	$2.000077e^{-12}$	-
Random **undirected** network with 60 nodes and 160 edges								
514MB	20m.	$2.123600e^{-08}$	5s. – 10	28MB	1m.	$2.145800e^{-08}$	$9.999789e^{-13}$	-
			1m. – 10	105MB	3m.	$2.145800e^{-08}$	$2.000078e^{-12}$	-
			3m. – 10	245MB	7m.	$2.145800e^{-08}$	$1.109100e^{-08}$	31.85
Random **directed** network with 500 nodes and 1494 edges								
5.4Gb	7h	$1.046068e^{-02}$	3s. – 20	6.5MB	6s.	$2.066508e^{-02}$	$1.000397e^{-02}$	69.52
			6s. – 22	8.4MB	15s.	$1.115888e^{-02}$	$1.010200e^{-02}$	9.94
			10s. – 25	15.4MB	30s.	$1.064958e^{-02}$	$1.025841e^{-02}$	3.74

TABLE II
TIME AND MEMORY REQUIRED FOR COMPUTING THE NETWORK UNRELIABILITY USING THE EXACT APPROACH AND THE APPROXIMATE ONE.

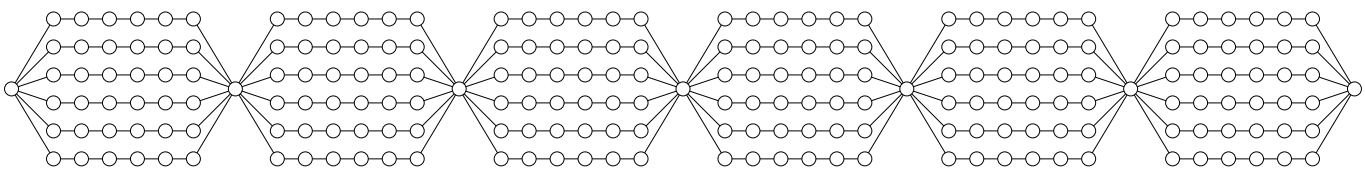

Fig. 6. Regular network used in the experiments in table III

Minpaths				Mincuts				Approx.
Num	Constraints	T.	Upper Bound	Num	Constraints	T.	Lower Bound	Accuracy (%)
Scale-free network with 500 nodes, 1949 edges and diameter equal to 5								
7	1s. - 4	1s.	$1.561550e^{-03}$	11	1s. - 3	1s.	$1.2e^{-07}$	-
62	1s. - 5	1s.	$1.34e^{-06}$	28	2s. - 3	1s.	$1.2e^{-07}$	167.12
77	1s. - 6	1s.	$9.8e^{-07}$	51	3s. - 3	3s.	$1.2e^{-07}$	156.36
Scale-free network with 1000 nodes, 1981 edges 5 and diameter equal to 8								
48	1s. - 7	3s.	$1.176971e^{-02}$	5	1s. - 5	1s.	$1.034201e^{-02}$	12.9
64	2s. - 8	6s.	$1.154293e^{-02}$	10	10s. - 5	10s.	$1.034292e^{-02}$	10.96
87	10s. - 8	3m.	$1.088859e^{-02}$	176	30s. - 5	120s.	$1.036431e^{-02}$	4.93
Scale-free network with 1000 nodes, 3938 edges and diameter equal to 5								
13	1s. - 5	1s.	$8.9064e^{-04}$	7	4s. - 3	1s.	$1.0001e^{-04}$	159.61
75	3s. - 6	4s.	$5.8489e^{-04}$	11	6s. - 4	20s.	$1.0001e^{-04}$	141.45
87	5s. - 6	3m.	$5.7889e^{-04}$	198	29s. - 4	10m.	$1.0002e^{-04}$	141.05
Small-world network with 500 nodes and 1000 edges and average degree of connectivity equal to 4								
1287	3s. - 14	4s.	$2.335444e^{-02}$	40	10. - 7	15s.	$1.6097e^{-04}$	197.26
4581	5s. - 15	6s.	$4.33667e^{-03}$	794	20s. - 9	487s.	$1.5858e^{-04}$	185.88
13564	10s. - 16	1m.	$1.36134e^{-03}$	987	60s. - 9	10m.	$1.03246e^{-03}$	25.29
Small-world network with 500 nodes and 5000 edges and average degree of connectivity equal to 20								
55	1s. - 5	2s.	$4.06e^{-06}$	32	1s. - 8	11s.	$1.0e^{-08}$	199.01
78	2s. - 5	8s.	$3.0e^{-08}$	47	2s. - 8	11s.	$1.0e^{-08}$	100
91	2s. - 6	8s.	$2.0e^{-08}$	81	3s. - 8	11s.	$1.0e^{-08}$	66.66
Random undirected network with 700 nodes and 1300 edges								
414	1s..- 11	0	$1.74641e^{-03}$	514	2s.- 25	2s.	$1.01e^{-06}$	199.76
1078	1s. - 12	1s	$7.7895e^{-04}$	2588	10s. - 30	10s.	$1.01e^{-06}$	199.48
6972	6s. - 14	7s	$3.0798e^{-04}$	3455	15s. - 30	50s.	$1.01e^{-06}$	198.69
Random undirected network with 715 nodes and 1000 edges								
18	1s. - 10	0	$1.01714e^{-01}$	14	10s. - 8	10s.	$2.2e^{-05}$	-
18	1s. - 10	0	$1.01714e^{-01}$	16	20s. - 8	20s.	$1.00101e^{-01}$	1.598
18	1s. - 10	0	$1.01714e^{-01}$	32	30s. - 8	28s.	$1.00105e^{-01}$	1.594
Regular directed network with 223 nodes and 252 edges								
522	0s.- 45	1s.	$1.315279e^{-01}$	18	0.01s.- 6	0	$1.0e^{-08}$	-
920	1s. - 45	6s.	$1.312557e^{-01}$	23	0.02s. - 6	0	$1.0e^{-08}$	-
46656	4s. - 45	32m.	$5.9e^{-07}$	19223	30s. - 6	37s.	$1.0e^{-08}$	193.33

TABLE III
A SET OF EXPERIMENTS USING OUR APPROXIMATE APPROACH

[13] K. Heidtmann. Statistical comparison of two sum-of-disjoint product algorithms for reliability and safety evaluation. In *Proceedings 21st International Conference SAFECOMP 2002*, pages 70–81. Springer Verlag, LNCS Vol 2434, 2002.

[14] T. Luo and K.S. Trivedi. An improved algorithm for coherent-system reliability. *IEEE Transaction on Reliability*, 47:73–78, 1998.

[15] MEDDLY webpage. http://sourceforge.net/projects/meddly.

[16] S. Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 3:156–170, 2001.

[17] M.E. Newman. Power laws, Pareto distributions and Zipfs laws. *Contemporary Physics*, 46:323–351, 2005.

[18] L.B. Page and J.E. Perry. A practical implementation of the factoring theorem for network reliability. *IEEE Transaction on Reliability*, R-37:259–267, 1988.

[19] A. Rauzy. New algorithms for fault tree analysis. *Reliability Engineering and System Safety*, 40:203–211, 1993.

[20] J. G. Shanthikumar. Bounding network-reliability using consecutive minimal cutsets. *IEEE Transaction on Reliability*, 37(1):45–49, 1988.

[21] M. Veeraraghavan and K. Trivedi. An improved algorithm for the symbolic reliability analysis of networks. *IEEE Transactions on Reliability*, 40:347–358, 1991.

[22] X. Zang, H. Sun, and K. Trivedi. A BDD-based algorithm for reliability graph analysis. Technical report, Department of Electrical Engineering, Duke University, 2000. http://www.ee.duke.edu/ hairong/workinduke/relgraph.ps.

APPENDIX

Algorithm 7 is the pseudo code of the algorithm presented in [22] and used for a comparison with our approach.

Algorithm 7 Algorithm for connectivity function

```
1:  procedure BDDGEN(Src, Trg)

    Src = source node
    Trg = target node
    B = BDD encoding the connectivity function
    E = list of edges

2:      E=Src.getEdges()
3:      for e ∈ E do
4:          Dst = dest_node(e)
5:          if Dst == Trg then
6:              sp=e.getBdd()
7:          else
8:              sp=BDDgen(Dst, Trg) * e.getBdd()
9:          end if
10:         B=B+sp
11:     end for
12: end procedure
```