Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Viale Teresa Michel 11, 15121 Alessandria
http://www.di.unipmn.it

A new symbolic approach for network reliability analysis

M. Beccuti, S. Donatelli, G. Franceschinis, R. Terruggia
(beccuti@di.unito.it, susi@di.unito.it,
giuliana.franceschinis@mfn.unipmn.it, roberta.terruggia@mfn.unipmn.it)

TECHNICAL REPORT TR-INF-2011-06-02-UNIPMN
(June 2011)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.unipmn.it/`. Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2011-01 *Spaced Seeds Design Using Perfect Rulers*, L. Egidi, G. Manzini, June 2011.

2010-04 *ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models*, D. Codetta Raiteri, L. Portinale, December 2010.

2010-03 *ICCBR 2010 Workshop Proceedings*, C. Marling, June 2010.

2010-02 *Verifying Business Process Compliance by Reasoning about Actions*, D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. Pozzato, D. Theseider Dupré, May 2010.

2010-01 *A Case-based Approach to Business Process Monitoring*, G. Leonardi, S. Montani, March 2010.

2009-09 *Supporting Human Interaction and Human Resources Coordination in Distributed Clinical Guidelines*, A. Bottrighi, G. Molino, S. Montani, P. Terenziani, M. Torchio, December 2009.

2009-08 *Simulating the communication of commands and signals in a distribution grid*, D. Codetta Raiteri, R. Nai, December 2009.

2009-07 *A temporal relational data model for proposals and evaluations of updates*, L. Anselma, A. Bottrighi, S. Montani, P. Terenziani, September 2009.

2009-06 *Performance analysis of partially symmetric SWNs: efficiency characterization through some case studies*, S. Baarir, M. Beccuti, C. Dutheillet, G. Franceschinis, S. Haddad, July 2009.

2009-05 *SAN models of communication scenarios inside the Electrical Power System*, D. Codetta Raiteri, R. Nai, July 2009.

2009-04 *On-line Product Configuration using Fuzzy Retrieval and J2EE Technology*, M. Galandrino, L. Portinale, May 2009.

2009-03 *A GSPN Semantics for Continuous Time Bayesian Networks with Immediate Nodes*, D. Codetta Raiteri, L. Portinale, March 2009.

2009-02 *The TAAROA Project Specification*, C. Anglano, M. Canonico, M. Guazzone, M. Zola, February 2009.

2009-01 *Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids*, C. Anglano, M. Canonico, February 2009.

2008-09 *Case-based management of exceptions to business processes: an approach exploiting prototypes*, S. Montani, December 2008.

2008-08 *The ShareGrid Portal: an easy way to submit jobs on computational Grids*, C. Anglano, M. Canonico, M. Guazzone, October 2008.

A new symbolic approach for network reliability analysis*

Marco Beccuti*, Susanna Donatelli*, Giuliana Franceschinis† and Roberta Terruggia†

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

†Dipartimento di Informatica
Università del Piemonte Orientale, Alessandria, Italy
giuliana.franceschinis, roberta.terruggia@mfn.unipmn.it

***Abstract*—In this paper we propose a new approach for network reliability analysis based on Binary Decision Diagrams (BDDs): the goal is to compute the s-t minpaths for a given network and pair of source-target (s-t) nodes, as well as the corresponding reliability measure. This approach comprises four steps: (1) a BDD B1 encoding the rules to generate the minpaths of a specific graph is built, (2) B1 is used to efficiently derive a BDD B2 encoding the minpaths, (3) B2 is expanded to obtain a new BDD B3 encoding the connectivity function: for efficiency reasons the last step takes advantage of an approach based on *saturation*. Finally, (4) the classical algorithm to compute the network reliability from the BDD encoding the connectivity function and the probability associated with the network graph edges is applied.**

As a result, the memory peak experienced in computing the connectivity function is significantly reduced, with respect to previous approaches which directly build the BDD (B3) encoding such function. A set of experiments is presented, comparing the memory utilization and computation time of the proposed approach with respect to previous ones.

I. Introduction

Many technological, economical and social systems can be abstracted as a network, an entity characterized by a set of nodes, the elements of the system, connected through a set of links, the relations between elements. An important aspect of these structures is represented by the existence of multiple redundant paths connecting each pair of nodes, this makes the systems intrinsically reliable. For this reason, the reliability of networks has been a major concern since the earliest times of reliability engineering [2], [16].

We can represent a network by means of a graph whose vertices and edges are considered as binary objects that can be in one of the following states: working (up) or failed (down). If a probability measure is assigned to the up or down condition of each element, a probability measure associated with the connectedness of the whole graph can be computed. The network *s-t reliability* is defined as the probability that a source node s can communicate with a target (or sink) node t through at least one path of up edges. The computational techniques proposed in the literature to solve the network reliability problem assume that the network elements are statistically independent and can be classified in two main categories; *i)* - approaches in which the desired network reliability measure is directly calculated (series-parallel reduction [4], pivotal decomposition using keystone components [13], [9]) 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], [11]) and then the reliability expression is evaluated, resorting to different techniques [2], like inclusion-exclusion method or sum of disjoint products [10], [17], [1].

In the last decades, Binary Decision Diagrams (BDD) [6] have provided an extraordinarily efficient method to represent and manipulate Boolean functions [7], and have also been exploited to model the connectivity of Boolean networks [5], [18].

This paper discusses a new approach based on BDD to compute qualitative and quantitative measures of network reliability. With respect to the previous works (e.g. [5], [18]) based on the algorithm proposed in [15] this new approach does not require to store the connectivity function in order to derive the minpaths of the net. This leads to a lower memory peak during their computation as reported in the experiments (Sec. VI). From this BDD, encoding only the minpaths, the network reliability can be computed either using a direct approach based on the sum of disjoint products (e.g. [1], [14]) or using a symbolic approach based on the Shannon's decomposition (e.g. [6]). Here we focus on the second choice: the BDD encoding the minpaths is extended to represent the BDD storing the s-t connectivity function: this is performed efficiently using an approach based on saturation, a strategy first proposed in [8] for the computation of the set of reachable states of a concurrent model. Even if this choice requires to store the whole connectivity function, thanks to the saturation strategy it is still possible to achieve a reduction in terms of memory peak with respect to other approaches

***Paper presented at Workshop InfQ 2011 (www.infq.it) June 27-29 2011 Lipari Italy**

based on BDD. Finally, to compare our approach with the existing methods we have developed a prototype framework implementing our approach and the one presented in [18].

The paper is organized as follows: Sec. II describes the main concepts of network reliability analysis, Sec. III introduces the BDDs and their operators. The previous algorithms based on BDD implemented in our framework to compute minpaths and network reliability are presented in Sec. IV. The new method is introduced in details in Sec. V; Sec. VI presents some numerical results comparing our approach with the other approach presented in Sec. IV. Finally, Sec. VII draws some conclusions and presents directions for future work.

II. Problem definition

A network can be described as a graph $G = (V, E)$, where V is the set of vertices (or nodes) and E the set of edges (or arcs). Each edge e is associated with a pair of (distinct) vertices that e connects; if the graph is oriented, then the pair of vertices associated with an edge is ordered.

Fig. 1 shows the graphical representation of a graph modeling a bridge network. 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. For directed graphs the connectivity property is more complex to establish since a connecting path must follow the direction of the arcs.

Let $G = (V, E)$ be a graph, and let s and t be two vertices called the source and the target vertex: we define a *path* as a subset of edges that, when in the up state, are sufficient to guarantee s and t connectivity. A *minpath* is a path that does not contain any other path. For instance the path $\langle e_1, e_2, e_4\rangle$ is not a minpath, while $\langle e_1, e_4\rangle$ is a minpath.

If the network elements are binary entities (up or down) the network connectivity (i.e. the set of all paths) can be expressed as a boolean function. By assigning a probability measure to each element of the graph to be up or down, we define point-to-point (or two-terminal) reliability as the probability that two nodes communicate through at least one path of working edges.

In general, networks can be assumed to have both vertices and edges as failure prone; however, in the sequel we assign a failure probability only to the edges since it is known that the case of networks with failing nodes can be reformulated into an equivalent one where only edges may fail.

III. Binary Decision Diagrams

Before presenting in details the algorithms to compute the minpaths and the network reliability, we introduce the basic characteristics of Binary Decision Diagrams (BDD) and associated operators.

BDDs [6] are well-known graph data structures used to efficiently represent and manipulate complex Boolean functions over boolean variables. They support efficient solutions for a large class of problems in different research fields (e.g. digital system design, combinatorial optimization, mathematical logic,. . .). For instance, in [18] the authors show that BDD can be efficiently used for reliability graph analysis directly encoding the s-t connectivity function on a BDD.

Formally a BDD is a directed acyclic graph that can represent 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*, the (unique) node without incoming arcs is called *root*, all the others are called *internal*; a node at level i has two outgoing arcs, one with label 0, and one with label 1. When using a BDD for representing a boolean function over N variables, the non terminal nodes are mapped onto the N variables, the two possible values of the function, *false* and *true*, are represented by two terminal nodes 0 and 1, and the arcs out of a node corresponding to variable $\mathcal{V}_i$ represent the possible assignments to such variable (true/1 or false/0). Each path in the graph from the root node to node 0 represents an assignment $v_N, \ldots, v_1$ of boolean values to the N variables for which $f(v_N, \ldots, v_1) = false$, and while each path from the root to node 1 represents an assignment for which $f(v_N, \ldots, v_1) = true$. The BDD encoding function $f()$ can also be interpreted as the representation of the set of N-tuples $\{v_N, \ldots, v_1\} : f(v_N, \ldots, v_1) = true$.

In Fig. 2 a BDD encoding the boolean function $e_2e_5 + e_2\overline{e_5}e_3e_4 + e_2\overline{e_5}\,\overline{e_3}e_1e_4 + \overline{e_2}e_1e_4$ is shown, an arc with label 0 starting from a node represents the assignment false for the corresponding variable while an arc with label 1 represents the assignment true.

A BDD in which nodes are organized into levels (one per variable) is called *ordered*. Two BDD nodes are *duplicate* if their outgoing edges go to the same set of nodes. Moreover, if the two arcs out of a given node n_i lead to the same destination node n_j, then node n_i is *redundant* and can be eliminated: the arcs originally directed towards n_i must be redirected towards n_j (so that after elimination of redundant nodes there may be arcs connecting nodes in non adjacent levels). An ordered BDD with no duplicate or redundant nodes is called *Reduced Ordered BDD* (ROBDD) and is *canonical*: if two boolean functions are equal (have the same value over all possible inputs), then they have the same same ROBDD representation. For instance, the BDD in Fig. 2 is a ROBDD where the variables are ordered as follows $e_2 \prec e_5 \prec e_3 \prec e_1 \prec e_4$: observe that the variable order can be arbitrarily chosen, however different variable ordering may lead to different BDD size. In the rest of this paper we will use the term BDD to indicate a ROBDD.

A BDD can be also used to express a mapping between the elements of one set (of boolean N-tuples) and the elements of a second set (of boolean N-tuples). The BDD encoding a relation of this kind has with $2N$ levels (i.e. it corresponds to a boolean function with 2N variables). We call the variables belonging to the first N-tuple *unprimed* while those of the second N-tuple are called *primed*. An example is shown in Fig. 3, where we encode a relation between paths in the bridge network in Fig. 1: this BDD will be discussed in details in Sec. V. Any permutation of the $2N$ variables can be used when establishing the variables ordering for this kind of BDD,

but here we assume that the unprimed and primed variables are interleaved, since it is often the case that interleaving is the most efficient order in terms of nodes required to store relations.

A very interesting aspect of BDDs is that they allow to reduce the basic boolean functions like *and* and *or* to basic operations like intersection (for the *and*) and union (for the *or*) over two BDDs: these operations can be easily defined recursively. 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 step of the algorithm (recursively) implementing the operator, so that it is never the case that the same operation is executed twice on the same pair of nodes, if our cache is large enough.

BDD provides also specific operators as the relational product (or *POST_IMAGE*) which allows to obtain from two BDDs, one encoding a generic set of N-tuples (B_S) and another one – with 2N levels – encoding a relation between N-tuples (B_R), a new BDD representing the result of the (simultaneous) application of the relation represented by B_R to the set of N-tuples represented by B_S. Considering the BDD in Fig.3 encoding a relation between the paths in the bridge network the relational product between it and the BDD representing the network state where all edges are disconnected ($\bigwedge_i e_i = false$) returns a BDD encoding all the minpaths of the bridge network.

IV. 2-TERMINAL RELIABILITY ALGORITHM

In this section the algorithm proposed in [18] for encoding of the connectivity function on BDD is summarized. To the best of our knowledge, this is one of the best algorithms in terms of execution time and memory utilization. This algorithm generates the BDD directly, via a recursive visit on the graph, without explicitly deriving the corresponding Boolean expression. It is sketched in Algorithm 1. Observe that all the Algorithms presented in this section must handle properly the cycles possibly appearing in G: this has been omitted in all the reported Algorithm sketches to keep the pseudo-code simpler, however it and (and actually it must) be introduced without problems.

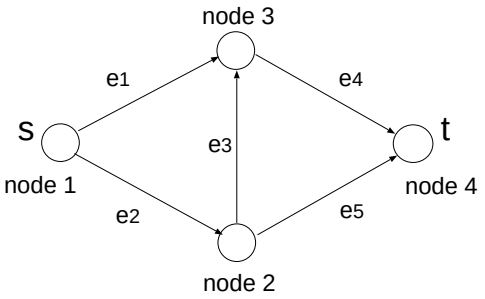

Fig. 1. A directed graph representing a bridge network

Let $G = (V, E)$ be a graph and $(Src, Trg) \in V$ be the source and the target vertices in G, There are $|E|$ variables

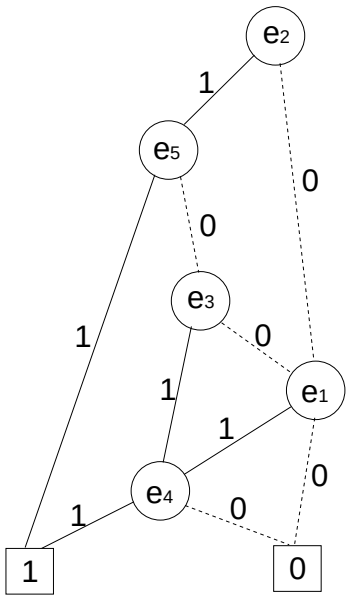

Fig. 2. BDD of the bridge network

Algorithm 1 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
```

in the BDD. The algorithm starting from node `Src` performs a depth first visit of the graph until reaches node `Trg`. The BDD construction starts once the node `Trg` is reached. The function *getBdd()* returns the BDD for the specific edge (i.e. the BDD representing the possible values of e_i encoding the fact that the corresponding edge can be in state up (branch *true*) or in state down (branch *false*).). The BDDs representing the edges along a path from `Src` to `Trg` are combined in AND (intersection operation) to form the BDD of a path, while the alternative paths starting from a given edge e are combined in OR (union operation). In this way it is possible to create the BDD representing the connectivity function.

Next we introduce two algorithms, that given a BDD encoding the connectivity function, compute the minpaths and the network reliability.

A. Algorithm for BDD minimization

Algorithm 1 derives directly the BDD of the Boolean connectivity function, however it is sometimes required to determine the list of the network minpaths. The minpaths provide a qualitative information about the connections between source and sink. Any path in a BDD linking the root with the terminal leaf 1, where only the list of events corresponding to the 1-labeled branches is included, is a path of the network, but not necessarily a minpath. To derive the minpaths the list of paths obtained from the BDD must be minimized. In [15] it is proposed an approach that transforms the original BDD into a new one embedding all and only the minpaths. Details of the transformation algorithm are in [15], while the algorithm is shown in Algorithm 2. The method *NodeDownPtr*(x) returns the sub-BDD (0-subtree or 1-subtree) reached assigning the value x to the root variable. The procedure *without()* (shown in details in Algorithm 3) implements the BDD operator *without* [15], that taking in input two BDDs, removes from the first BDD all the paths shared with the second one.

The method *createBDD*(x, T, E) returns a new BDD having the variable x as root and the BDDs T and E as its two branches (0-subtree and 1-subtree). Observe that a cache is used to avoid recomputing the same operations several times. Finally, the variable *flag* is used to choice between minpaths computation (*flag*=1) and the mincuts one (*flag*=0). Fig.4 shows the result of the minimization algorithm when applied to the connectivity BDD of the bridge network. It is important to observe that the BDD in Fig.4 must be interpreted in a non standard way to obtain the list of the minpaths: they are obtained selecting the paths from the root to terminal node 1 *setting to false all the edge variables that are not present in the path due to a jump between not adjacent levels*. The normal interpretation of such paths instead would consider both possible values for the edge variables not explicitly represented along the path.

B. Reliability algorithm

To compute the network reliability we need to assign to each variable x_i a probability p_i of being true ($1-p_i$ of being false), so that we can compute the probability $P\{F\}$ of function F (representing a set of paths, in our interpretation) by applying recursively Equation 1.

$$\begin{aligned} P\{F\} &= p_1 P\{F_{x_1=1}\} + (1-p_1) P\{F_{x_1=0}\} \\ &= P\{F_{x_1=0}\} + p_1 (P\{F_{x_1=1}\} - P\{F_{x_1=0}\}) \end{aligned} \quad (1)$$

In Algorithm 4 a sketch of this algorithm is shown.

It takes in input the BDD B encoding the connectivity function and the vector P storing the edge probabilities. 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 the subpaths originating in a given node x depends on the probability of the 0-subtree multiplied by the probability of x being false (edge being down) plus the probability of the 1-subtree multiplied by the probability of x being true (edge being up).

Algorithm 2 MinBDD algorithm

```
1: function MINBDD(B,flag)
   B = BDD encoding the connectivity function
2:     if (B==1) or (B==0) then
3:         return B
4:     else
5:         if Cache[B]!=NULL then
6:             return Cache[B]
7:         else
8:             B1=B.NodeDownPtr(0)
9:             B0=B.NodeDownPtr(1)
10:            x=B.NodeVar()
11:            K=MinBdd(B1,flag)
12:            T=without(K,B0,flag)
13:            E=MinBdd(B0,flag)
14:            if (T==E) then return T
15:            R=createBDD(x,T,E)
16:            Cache[B]=R;
17:            return R
18:        end if
19:    end if
20: end function
```

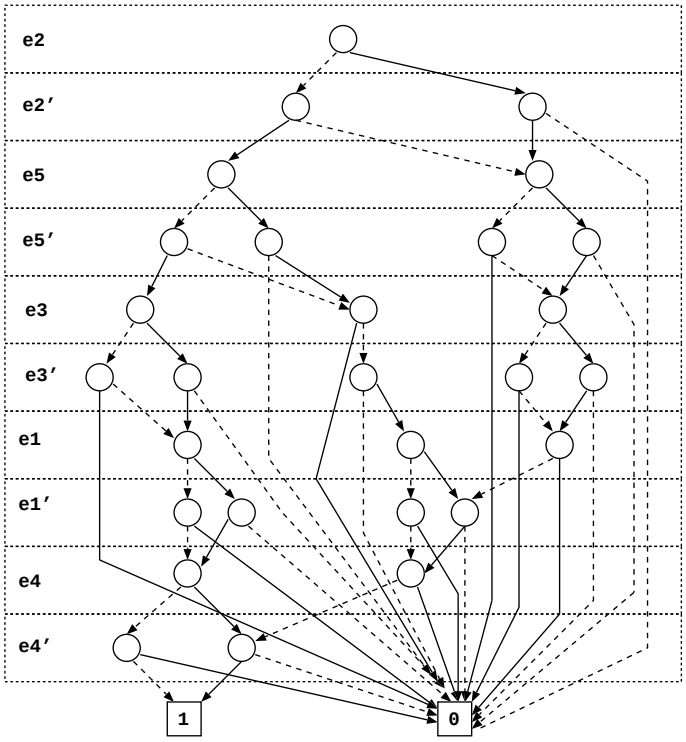

Fig. 3. BDD B1 expressing the rules to build the minpaths of the directed bridge network

V. The proposed algorithm

In this section the new approach is presented in details, first describing how the rules generating the minpaths of a specific graph can be derived and encoded in a BDD (B1). Then showing the algorithm to build the BDD (B2) encoding the minpaths; this is performed more efficiently than in previous

Algorithm 3 The Without algorithm, used by MinBDD

```
1: function WITHOUT(F,G,flag)
   G, F, R = BDDs
2:     if (flag==0) AND (F==0) AND (G==0) then
3:         return 1
4:     end if
5:     if (F==0) then
6:         return F
7:     else
8:         if (G==1) then
9:             if (flag==1) then return 0
10:            else return F
11:        else
12:            if (G==0) then
13:                if (flag==0) then return 1
14:                else return F
15:            end if
16:        end if
17:    end if
18:    if (computedtable has entry (flag,F,G),R) then
19:        return R
20:    else/* F=ite(x,F1,F2), G=ite(y,G1,G2)*/
21:        x= the top variable of F
22:        y= the top variable of G
23:        if (x>y) then
24:            if (flag==1) then return without(F,G2,flag)
25:            else return without(F,G1,flag)
26:        else
27:            if (x==y) then
28:                T=without(F1,G1,flag)
29:                E=without(F2,G2,flag)
30:            else/* x<y */
31:                T=without(F1,G,flag)
32:                E=without(F2,G,flag)
33:            end if
34:            if (T==E) then return T
35:            R=find_or_add_unique table(x,T,E)
36:            insert_computed_table(flag,F,G),R)
37:            return R
38:        end if
39:    end if
40: end function
```

Algorithm 4 Reliability algorithm

```
1: function RELBDD(B, P)
   B = BDD encoding the connectivity function
   P = vector storing the edges probabilities
   PF = reliability value
2:     if B == BddFalse then
3:         return 0;
4:     else
5:         if B == BddTrue then
6:             return 1;
7:         else
8:             if CACHE[B]!=NULL then
9:                 return CACHE[B];
10:            else
11:                PF1=RelBDD(B.NodeDownPtr(1));
12:                PF2=RelBDD(B.NodeDownPtr(0));
13:                PF=PF2 + P[Root(B)] * (PF1 - PF2);
14:            end if
15:        end if
16:        cache[B]=PF;
17:        return PF;
18:    end if
19: end function
```

Algorithm 5 Algorithm for encoding the rules generating the minpaths on BDD

```
1: procedure BDDGEN(Src, Trg, R, X, X')
   Src = source node
   Trg = target node
   R = BDD with 2L levels encoding the connectivity function
   E = list of edges
   X,X'= vectors indicating a set of BDD variables' assignments
2:     E=Src.getEdges()
3:     for e ∈ E do
4:         Dst = dest_node(e)
5:         X[e] = 0
6:         X'[e] = 1
7:         if Dst == Trg then
8:             R.insert(X, X')
9:         else
10:            BDDgen(Dst, Trg, R, X, X')
11:        end if
12:        X[e] = X'[e] = −2
13:    end for
14: end procedure
```

approaches that use Algorithm 1 and Algorithm 2 (recall that the minpath representation obtained by applying these algorithms is a non standard one). Finally a third BDD (B3) is derived, representing the network connectivity, from which the network reliability measure can be derived applying standard methods (this step can be omitted if we are only interested in the minpaths).

Before introducing the algorithm observe that each variable x_i of the boolean function encoded on BDD represents an edge in the graph, so that $x_i = 1$ models the edge i connected, while $x_i = 0$ means that is not connected. For instance the path $\langle 0,0,0,1,1\rangle$ in the BDD in Fig. 5 represents the minpath $\langle e_1, e_4\rangle$ of the bridge network in Fig. 1. Moreover, the rules generating the minpaths of a graph specified in terms of which disconnected edges have to be connected to reach the target node are encoded on a BDD with 2N levels, where N is the total number of edges in the graph. An example of this for the bridge network is shown Fig. 3 (recall that unprimed and primed edge variables are interleaved, evidenced through additional spacing in the 2N-tuples appearing in the text hereafter). The path $\langle 0,0,\ 0,0,\ 0,0\ ,0,1,\ 0,1\rangle$ means that e_1, e_4 should be connected to reach the target node, while $\langle 0,1,\ 0,1,\ 0,0,\ 0,0,\ 0,0\rangle$ that e_2, e_5 should be connected.

A. Encoding the rules generating the minpaths on BDD

The rules generating the minpaths of a graph are encoded through a depth first visit as shown in Alg. 5. It takes in input an unexplored node (initially the source node), the target node, the BDD (initially empty) encoding the minpath rules and two

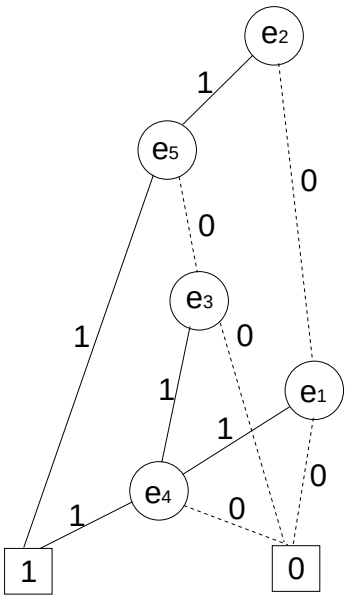

Fig. 4. BDD generated by Algorithm 2 representing a non standard encoding of the bridge network minpaths

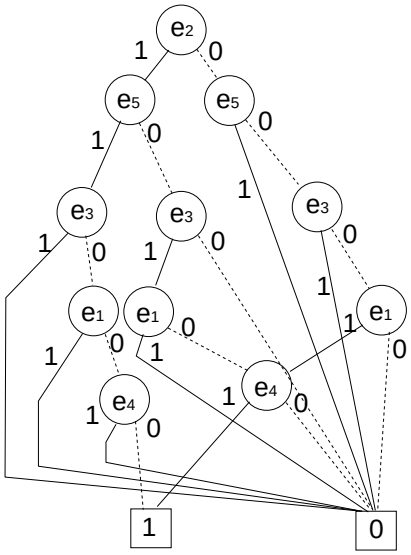

Fig. 5. BDD B2 encoding the set of minpaths of the bridge network

integer vectors with size equal to the total number of edges in the graph. Such two vectors (X, X') are used to insert a path in the BDD corresponding to a possible assignment for all the variables in the BDD (X for the unprimed variable and X' for the primed ones). In the beginning they are initialized to -2 meaning that the value of an unprimed variable (x) is the same of its corresponding primed variable (x'). In line 2 all the edges starting from Src are stored in E; then for each edge e in E its reached node is saved in Dst (line 4) and $X[e]$ and $X'[e]$ are updated with 0 and 1 respectively. This means that starting from a path where the edge e is not connected the edge must be connected to reach the target node.

If Dst is the target node (line 7 to 9) the generated path

Algorithm 6 Algorithm for minpath generation

1: **procedure** MINPAHS(Src, Trg, B, X, X')

Src = source node
Trg = target node
R = BDD encoding the connectivity function
X,X' = vectors indicating a set of BDD variables' assignments

2: Initialize(X,X',-2)
3: BDDgen(*Src*, *Trg*, R, X, X')
4: Initialize(X,0)
5: B.insert(X)
6: B=Apply(POST_IMAGE,R, B)
7: **end procedure**

is a minpath, hence its generation rule, specified by X and X', has to be inserted in the BDD R. Otherwise the function is recursively called on Dst. In the end, R encodes all the minpath generation rules of the input graph. Fig. 3 shows R for the bridge network in Fig. 1.

B. Computing minpaths

Algorithm 6 shows how it is possible to derive a BDD encoding the minpaths from the 2N level BDD R computed with Algorithm 5. This is performed applying the operator *POST_IMAGE* on the BDD B where all the network edges are disconnected and the BDD R encodes the minpath rules.

For instance, in Fig 5 the BDD encoding the minpaths for the bridge network in Fig. 1 is reported.

Observe that these two algorithms allow us to derive the minpaths of a graph more efficiently in terms of memory (as shown by the experiments in Sec. VI) than the classical algorithms presented in Sec.IV. Indeed, in this phase we do not need to store explicitly the connectivity function.

C. Computing the network reliability

Minpaths provide useful information, but if we are interested in computing the network reliability it is necessary to introduce a further step. As already explained in the introduction, in this step we are going to reuse Algorithm 4, hence we need to generate the BDD encoding the connectivity function. This can be efficiently produced from the minpaths BDD as shown in Algorithm 7.

In line 2 a BDD with 2N levels is generated, which encodes the rules to translate the minpath BDD into the BDD encoding the connectivity function. In details for each edge e a rule is inserted specifying that for each path containing edge $e = 0$ we want to generate two new paths one where $e = 0$ and a second one where $e = 1$: this is encoded as follows, for edge e_2: $X = \langle 0, -2, -2, -2, -2\rangle$ and $X' = \langle -1, -2, -2, -2, -2\rangle$ where the pair $0, -1$ associated to the unprimed and primed variable e_1, e_1' is a shorthand notation for the transformation described above. In line 3 we apply SAT_POST_IMAGE operator (a recursive version of the operator POST_IMAGE) on M and B to obtain the BDD encoding the connectivity function. Observe that this

Algorithm 7 Algorithm

1: **procedure** COMPUTERELIABILITY(B, P)

C = BDD encoding the connectivity function
B = BDD encoding the minpaths
M = BDD encoding the translation rule for minpaths BDD
P = vector storing the edge probabilities

2: TranslationRule(M)
3: C=Apply(SAT_POST_IMAGE,M, B)
4: RelBDD(C, P)
5: **end procedure**

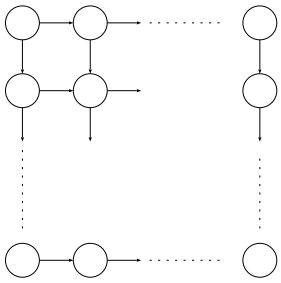

Fig. 6. Benchmark network

step is performed using the saturation strategy, so that first we update all the paths considering the edge at level i, then all the paths considering the edge at level $i-1$. Each time that considering level i we create a new path containing a 0-labeled arc at level $j > i$, then we come back to the level j. The algorithm halts when all the paths have been updated with respect to every level. This strategy, as shown in [8], can lead to significant time and memory savings. Finally in line 4 Algorithm 4 is called on the BDD encoding the connectivity function and the vector P storing edge probability to be up.

A possible extension of our approach to compute an approximate network reliability based on minpaths is now proposed. This can be useful in case of huge networks where it is impossible to apply the exact approach. Our idea is to compute a lower bound for the network reliability taking into account only the minpaths that are at most as long as a given threshold. This requires to update only Algorithm 5 so that it only encodes the rules for generating the minpaths that are at most as long as a given threshold. In next section some results computed with this approach are reported.

VI. RESULTS

In this section some experimental results are presented, which were computed thanks to a prototype implementation of our approach and of the previous ones presented in literature and described in Sec.IV; the implementation of BDD has been provided by an existing open–source library: Meddly [12]. The 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.

Two scalable benchmarks have been carried out to evaluate and to compare the efficiency of the two approaches. First, we take into account a regular $n \times n$ network topology of increasing complexity where each node is connected with two neighbors (the right one and the bottom one) in case of directed network, or with all the four neighbors (right, left, bottom and top ones) if the network is undirected. Fig.6 shows an example of directed network of this kind.
We shall also consider random undirected networks where the pair of vertices connected to each edge is chosen randomly among all nodes in the network (avoiding duplicates).

All the experiments have been executed on an 2.4 Ghz AMD Athlon 64-bit processor with 4 Gbytes memory, assuming a failure probability equal to $p = 0.9$ for all the edges.

A. Benchmark network reliability computation

Table I shows the results for only the minpath computation obtained on these benchmark networks increasing the value of n. For the regular network the upper left node and the lower right node are selected as Src and Trg, while for the random network they are randomly chosen. The "Algo. 1 + Algo 2" columns refer to the implementation of the first two algorithms in Sec. IV, and show its reached memory peak in bytes and its computation time in seconds. The other columns are related to our new approach for minpaths computation and report the reached memory peak, the computation time and the memory peak reduction factor with respect to the previous algorithm.

Considering the set of experiments on directed networks we observe that the computation time of the two approaches is comparable (same order of magnitude), but a high reduction in terms of memory peak occupation is obtained by our approach (e.g. for case 13×13 the total reduction factor is $\sim$4,523 for the regular directed network and $\sim$1935 for the regular undirected one). Unfortunately, for the random network the reduction level is lower, but this is justified by the fact that most of paths in the network are minpaths.

Table II shows the computation of the network reliability, so that we compare our approach (Algo. 5 + Algo. 6 + Algo.7) with the approach presented in Sec. IV (Algo. 1 + Algo. 2). A memory reduction is still obtained even if it is less relevant than the one of the minpath generation. These results suggest that it could be worthwhile to investigate the possibility to develop a new algorithm able to compute the network reliability without generating the BDD encoding the connectivity function.

B. Estimation of Reliability.

Finally, let us discuss a set of experiments performed using the extension of our approach to compute an approximate value (a lower bound) for network reliability based on minpaths.

Table III shows the results for a random network with 30 vertices and 31 edges (denoted $30n41e$ in the exact computation data Tables). The network has 575 minpaths and the exact reliability is 0.886926623780. In details Table III

	Algo 1 + Algo 2		Algo 5 + Algo 6		
Size	Mem.	T.	Mem.	T.	Red.
Regular Directed network					
8×8	2,951,776	0	29,836	0	71.81
9×9	8,806,500	0	41,048	0	214.54
10×10	25,935,088	5	54,704	3	471.1
11×11	71,895,036	12	71,044	12	1,011.98
12×12	194,233,280	47	90,308	50	2,150.79
13×13	509,980,836	171	112,736	213	4,523.67
Regular Undirected network					
5×5	1,989,288	0	185,132	4	10.75
6×6	112,374,344	41	267,772	129	419.66
7×7	1,994,572,944	19,490	1,030,632	20,150	1935.29
Random Undirected network					
$20n28e$	106,948	0	27,124	0	3.94
$30n36e$	125,064	1	27,200	0	4.6
$30n41e$	2,099,196	1	160,400	0	13.09

TABLE I
TIME (SECONDS) AND MEMORY (BYTES) REQUIRED FOR COMPUTING THE LIST OF MINPATHS OF THE NETWORK

	Algo 1 + Algo 3		Algo 5 + Algo 6 + Algo 7		
Size	Mem.	T.	Mem.	T.	Red.
Regular Directed network					
8×8	2,951,776	0	245,904	0	12
9×9	8,806,500	0	568,416	0	15.49
10×10	25,935,088	5	1,431,576	3	18.12
11×11	71,895,036	12	3,480,724	12	20.66
12×12	194,233,280	47	8,345,396	50	23.27
13×13	509,980,836	171	19,799,984	213	25.76
Regular Undirected network					
5×5	1,989,288	0	279,380	4	7.12
6×6	112,374,344	41	1,018,360	129	110.35
7×7	1,994,572,944	19,490	15,249,940	77,150	130.79
Random Undirected network					
$20n28e$	106,948	0	61,704	0	1.73
$30n36e$	125,064	1	52,528	0	2.38
$30n41e$	2,099,196	1	568,316	0	3.69

TABLE II
TIME (SECONDS) AND MEMORY (BYTES) REQUIRED FOR COMPUTING THE NETWORK RELIABILITY AND THE LIST OF MINPATHS

shows for each threshold value T (first column), the number of considered minpaths, the reliability approximation and the error computed as difference between the exact reliability value and the approximation.

In Fig.7 the different values of reliability increasing the value of threshold are depicted. Figure 8 shows the error.

VII. CONCLUSION AND FUTURE WORK

In this paper we have presented a new symbolic approach based on BDDs to compute the minpaths and network reliability. This approach has been compared with the others presented in literature; a framework including those algorithms has been implemented and used to perform several experiments. Such experiments have highlighted that minpaths generation achieves a high level of reduction in the memory utilization while the execution time only slightly increases. Moreover, our approach is also able to achieve a good reduction during the computation of the network reliability thanks to an efficient generation of the BDD encoding the connectivity function based on a saturation strategy.

Threshold (T)	Number of minpaths	Reliability	Error
1	0	0.000000000000	8.87E-001
2	0	0.000000000000	8.87E-001
3	0	0.000000000000	8.87E-001
4	1	0.656099930477	2.31E-001
5	1	0.656099930477	2.31E-001
6	5	0.839021942105	4.79E-002
7	7	0.859198261471	2.77E-002
8	22	0.884721786799	2.20E-003
9	41	0.886159710528	7.67E-004
10	81	0.886810475654	1.16E-004
11	131	0.886891663661	3.50E-005
12	189	0.886919860948	6.76E-006
13	261	0.886925749869	8.74E-007
14	324	0.886926443339	1.80E-007
15	397	0.886926571506	5.23E-008
16	452	0.886926613613	1.02E-008
17	510	0.886926622457	1.32E-009
18	555	0.886926623602	1.78E-010
19	569	0.886926623775	5.00E-012
20	575	0.886926623780	0.00E+000

TABLE III
APPROXIMATION OF THE RELIABILITY VALUE

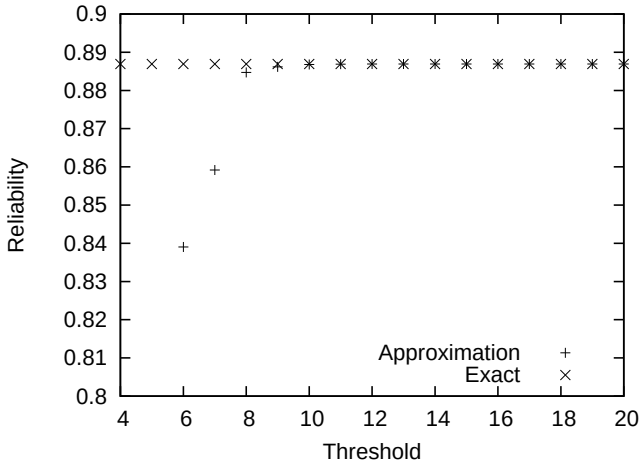

Fig. 7. Approximation of the reliability value varying the threshold

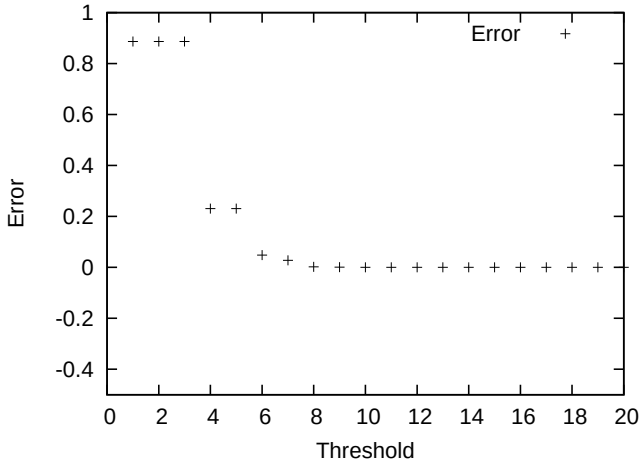

Fig. 8. Error of the approximation of the reliability value varying the threshold

Finally, we have proposed an extension of our approach to compute an approximation of the network reliability that may be useful in case of huge networks. Some experiments based on the implementation of such extension have been reported.

The future works concern two main aspects. First we are going to investigate the possibility to derive the network reliability without generating the whole connectivity function. The second future work consists to extend our approach in order to compute also the mincuts, and use them and together with the minpaths to obtain both upper and lower bounds on the network reliability.

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] A. Bobbio and A. Premoli. Fast algorithm for unavailability and sensitivity analysis of series-parallel systems. *IEEE Transactions on Reliability*, R-31:359–361, 1982.

[5] A. Bobbio and R. Terruggia. Binary decision diagram in network reliability analysis. In *1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07)*, pages 57–62, 2007.

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

[7] 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.

[8] Gianfranco Ciardo, Gerald Lüttgen, and Andrew S. Miner. Exploiting interleaving semantics in symbolic state–space generation. *Formal Methods in System Design*, 31(1):63–100, August 2007.

[9] 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.

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

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

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

[13] 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.

[14] S. Rai, M. Veeraraghavan, and K. S. Trivedi. A survey of efficient reliability computation using disjoint products approach. *Networks Journal*, 25(3):147–163, 1995.

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

[16] A. Rosenthal. Computing the reliability of complex networks. *SIAM Journal on Applied Mathematics*, 32:384–393, 1977.

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

[18] X. Zang and H. Sunand K.S. Trivedi. A BDD-based algorithm for reliability graph analysis. Technical report, Department of Electrical Engineering, Duke University, 2000.