An Effective Algorithm for Computing All-terminal Reliability Bounds

Jaime Silva

CISUC, Department of Informatics Engineering, University of Coimbra

Pólo 2, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

CFC, Department of Physics, University of Coimbra, 3004-516 Coimbra, Portugal

silva.jaime@gmail.com

Teresa Gomes

Department of Electrical and Computer Engineering, University of Coimbra

Pólo 2, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

INESC Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal

teresa@deec.uc.pt

David Tipper

Graduate Telecommunications and Networking Program

University of Pittsburgh, Pittsburgh, PA 15260

tipper@tele.pitt.edu

Lúcia Martins

Department of Electrical and Computer Engineering, University of Coimbra

Pólo 2, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

INESC Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal

lucia@deec.uc.pt

Velin Kounev

Graduate Telecommunications and Networking Program

University of Pittsburgh, Pittsburgh, PA 15260

vkounev@pitt.edu

Abstract

The exact calculation of all-terminal reliability is not feasible in large networks. Hence estimation techniques and lower and upper bounds for all-terminal reliability have been utilized. Here, we propose using an ordered subset of the mincuts and an ordered subset of the minpaths to calculate an all-terminal reliability upper and lower bound, respectively. The advantage of the proposed new approach results from the fact that it does not require the enumeration of all mincuts or all minpaths as re-

quired by other bounds. The proposed algorithm uses maximally disjoint minpaths, prior to their sequential generation, and also uses a binary decision diagram for the calculation of their union probability. The numerical results show that the proposed approach is computationally feasible, reasonably accurate and much faster than the previous version of the algorithm. This allows one to obtain tight bounds when it not possible to enumerate all mincuts or all minpaths as revealed by extensive tests on real-world networks.

Keywords: all-terminal network reliability; network availability; binary decision diagram (BDD); sum of disjoint products; bounds computation.

1 Introduction

Nowadays communication networks are ubiquitous in our daily life, being one of the critical infrastructures that our society depends on. This leads to concerns about the reliability and resilience of communication when subjected to failures and attacks [42]. A communication network failure is defined as an event where it is not possible to deliver communication services [42]. A network failure can typically occur due to cable cuts, natural disasters and physical/electronic attacks. Due to the importance of communication networks in today's society there is an interest in knowing the network resilience to a potential failure. According to [1], reliability is the "probability that an item will perform a required function under stated conditions for a given time interval". Thence the ability of a network to execute a desired network operation is related to network reliability [11].

In [41] a systematic architectural framework that unifies resilience disciplines, strategies, principles and analysis is presented. The ResiliNets strategy, according to the authors, leads to a set of design principles which can steer the analysis and design of resilient networks.

In communication networks, edges may fail, and are either in an operational or failed state. The problem of finding the probability that k specific vertices remain connected is termed, for k=2, the two-terminal reliability problem; when k is equal to the number of vertices in the network, the problem is designated as the all-terminal reliability problem. In the two-terminal reliability problem the main objective is the calculation of the probability of communication between two vertices, i.e., two vertices communicate if there exists at least one operational path that connects the two vertices. For the all-terminal reliability

problem the main objective is to calculate the probability of communication between all vertices in the network. These are well-known NP-hard combinatorial problems [5, 33].

Recently, new networked application domains have emerged that need mission critical communication networks with high all-terminal availability requirements. In particular, our work was motivated by the need to calculate the all-terminal availability for a potential smart grid communications network [23] (designated netvkk in Table 2), assumed to be installed in parallel to the California Power Grid. The availability of a network (or system) is related to the fraction of time the network or system is considered to be in its up state. Hence, if every vertex can communicate with every other vertex the network is considered to be in its up state. Reliability is a measure of how long the network is continuously in its up state, performing its intended function (for example ensuring communication between all its vertices). Furthermore, a network with a low reliability, because it is unable to ensure the desired continuous up time duration until a fail state occurs, may have high availability if the fail states are very short. For historical reasons, we adopt the usual designations in the literature: two-terminal reliability, k-terminal reliability, all-terminal reliability – which in fact coincide with the definition of two-terminal availability, k-terminal availability, all-terminal availability, respectively.

The main contribution of this work is a new algorithm to calculate all-terminal reliability upper and lower bounds, which uses an ordered subset of the mincuts to calculate the reliability upper bound and an ordered subset of minpaths to calculate the reliability lower bound. This algorithm builds on our earlier work [39], incorporating new heuristics that lead to significant improvement in the overall performance. Note that the proposed approach (as the algorithm in [35] for calculating two-terminal reliability bounds) does not need to enumerate all mincuts or minpaths and generally obtains bounds with the desired accuracy in a reasonable amount of time. It is important to stress that for networks with a large number of edges the number of mincuts is so large ($2^{N-1}-1$, for complete graphs with N vertices) that the time and memory needed to retrieve all the mincuts are infeasible and so exact calculation becomes impractical. Similarly the number of trees can grow very fast with the number of vertices and edges in a graph. For a complete graph the number of distinct trees with N vertices is N^{N-2}.

This article is organized as follows. In the next section some background information is presented and related work is reviewed. Then Section 3 starts by

discussing dynamic upper and lower bounds for all-terminal reliability; in Subsection 3.1 a preliminary study, which justifies the improved algorithm proposed in Subsection 3.2, is presented. Numerical results illustrating the performance of the algorithm on real-world sample topologies are discussed in Section 4, followed by the conclusions in Section 5.

2 Background and Related Work

There are several approaches to calculate network reliability as described in [37]. One of the most popular methods, due to its simplicity, is based on state space enumeration. Using the pivotal decomposition formula, also known as link factoring, the problem is iteratively decomposed into two smaller problems with the main drawback being the possibility of total state space enumeration. Nevertheless, the computational effort can be reduced by judicious selection of the edges for factoring [37]. Furthermore, there is the well-known factoring theorem that leads to an algorithm for the calculation of the network reliability proposed by Moskowitz [11, 29]. Nonetheless, the algorithm has an exponential time complexity.

Additional widely used methods for calculating all-terminal reliability focus either on the minimum pathsets (minpaths) or the minimum cutsets (mincuts) within a given network. A pathset is defined as a subset of components whose operation implies system operation [5]. A minpath is a pathset that does not contain another pathset. A cutset is a set of edges such that if all of them fail simultaneously, then the system fails. A mincut is a cutset that does not contain any other cutset [5]. Even if it is possible to enumerate all the minpaths or mincuts, the all-terminal network reliability usually becomes computational intractable, due to the fact that it requires the calculation of the probability of a union of events. Note that the probability of a union of events can be calculated by decomposing it into a union of disjoint events, i.e., disjoint products, whose probabilities can then be added.

One of the first algorithms for the calculation of a sum of disjoint products can be found in [2]. That algorithm significantly reduced the amount of computation required to obtain a disjoint sum, with respect to earlier approaches. Since then several other algorithms for tackling this problem have been proposed. An alpha-numerical ordering of the state variables, in order to obtain shorter disjoint sums, is discussed in [24]. The author of [45] claims to have

obtained a more efficient algorithm by changing the ordering rules of path lists and of the products in [24]. New forms of arranging the minpaths were also proposed in [6, 7]. In [19] an efficient algorithm, which inverts the products of variables instead of inverting single variables as done in previous works, is presented. An alternative method for calculating the probability of a union of events can be found in [18], where the proposed method is based on a recursive function which was originally developed for the calculation of the blocking probability in networks with alternative routing. In [18] the performance of the new method is compared with [19, 24, 45]. In [4] a new approach is proposed for pre-processing minpaths, but the authors also acknowledge that there are cases where the proposed pre-processing will not be able to reduce the number of disjoint products.

Another approach for the calculation of the probability of a union of events is the use of the binary decision diagram (BDD) method [9, 10]. The BDD is a directed acyclic graph with two sink vertices that represent the Boolean functions 0 and 1. The non-sink vertices are labeled with a Boolean variable ν and each has two outgoing edges labeled 1 (*then*) and 0 (*else*) [9]. The Boolean operations AND, OR and others can be easily implemented in a BDD and by transversing the BDD using the *then* and *else* vertices it is possible to calculate the result. The application of BDD trees to the calculation of all-terminal reliability was proposed in [21]. The authors use a BDD in conjunction with an edge contraction/deletion procedure to calculate the all-terminal reliability. The root of the BDD corresponds to the original graph G, and a new level is added to the BDD when an edge from the graph is contracted/deleted. Finally, any path from the root of the BDD to a leaf corresponds to a spanning tree of G.

An alternate approach to determining the all-terminal reliability is using Monte Carlo simulation which can be very accurate, but at the cost of a high computational effort [16]. Furthermore, special care must be taken under rare event scenarios [34]. In fact, the comparison in [15] of four different Monte Carlo sampling plans for estimating the two-terminal reliability of a network revealed that the sampling size can be very large, and the performance of the method depends on the edge probability. The authors of [40] propose the use of an artificial neural network to estimate the all-terminal network reliability. They considered two types of networks regarding edge reliability: identical and variable edge reliability; they conclude that their approach is suitable to be used in network design due its reasonable computational cost. The same methodol-

ogy was also used in [3] to calculate the all-terminal reliability of a network with identical edges reliabilities, improving the results obtained with earlier approaches. In [20] three estimation techniques, crude Monte Carlo, Permutation Monte Carlo and Merge Process, were considered and compared with their proposed Cross-Entropy method. By comparing the different strategies the authors conclude that the Cross-Entropy method is the fastest of all three techniques. Given the difficulty in determining the exact all-terminal reliability, there has been considerable work on determining bounds.

The classical approach to calculating upper and lower bounds on the network reliability is to consider minpaths, while for calculating upper and lower bounds on the network unreliability mincuts are used [37]. There are two well-known approaches for the calculation of all-terminal reliability bounds, the Bonferroni bounds [30, 37] and the Esary-Proschan bounds approximation [14]. These approaches are impractical for the evaluation of real large scale networks due to the fact that they need to calculate all the minpaths or mincuts which are in themselves NP-complete problems. Nevertheless, there are algorithms to approximate the all-terminal reliability of a given network. A bounding approximation algorithm, which uses the probability of failure for a union of events in minimal cutsets, can be found in [36]. In fact, to reduce the size of the problem, the authors in [36] use a small portion of the minimal cuts existing in the graph. Furthermore, the approximation to the probability of the union of events is calculated recursively and the algorithm can calculate upper and lower bounds on the all-terminal reliability of a given network.

A recent work [47] proposed a greedy factoring process. The bounds are updated by a network factoring procedure, first selecting the edges in the network, then enumerating all the states of the selected edges and finally evaluating the all-terminal reliability of the subnetworks associated with the states. A greedy approach is used for selecting the branches used in the recursively applied factoring process. Six greedy methodologies, all of them using a minimum spanning tree or a mincut for starting the factoring process, are proposed [47].

A more recent approach [35] computes two-terminal reliability bounds using a BDD for representing the reliability graph. It starts by considering only disjoint mincuts and minpaths, and if the desired bound accuracy is not achieved then it proceeds to an exhaustive search of mincuts and minpaths. The authors [35] also developed a new heuristic that selects only the most important minpaths or mincuts for reducing the gap between the upper and lower reliability bounds, in an iterative procedure. More details on this work are given in the

next section, as this work inspired some of the improvements in our algorithm originally presented in [39]. In [46] the authors seek to determine if a given network reaches a certain level of all-terminal reliability. Instead of calculating the all-terminal reliability the algorithm ends as soon as it can confirm if the required level can be reached (feasible solution) or is unattainable (infeasible solution). Their approach is based on network decomposition. The algorithm iteratively reduces the gap between the upper and lower reliability bounds, using a set of subnetworks derived from the original graph.

3 An improved procedure for the calculation of all-terminal reliability bounds

Consider a communication network, represented by an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_M\}$ is the set of edges. Each edge corresponds to an unordered pair of different elements belonging to V. We assume a random failure scenario, where vertices do not fail and edges fail independently. An edge e has a probability p_e of being operational and probability $q_e = 1 - p_e$ of being in the failed state. Let $A = \{p_{e_1}, p_{e_2}, \ldots, p_{e_M}\}$, where p_{e_i} is the probability of edge e_i being operational (that is the link availability). The all-terminal reliability $R(G)$ is the probability the graph G is connected. One approach for calculating all-terminal reliability [37] focuses on the minpaths within a given network. For all-terminal reliability a minpath is a spanning tree. Let the minpaths be denoted P_i, $i = 1, 2, \ldots, r$, where r is the number of spanning trees and let E_i be the event that all edges in P_i are simultaneously operational.

Then $R(G)$ is given by

$$R(G) = \Pr(E_1 \cup E_2 \cup \cdots \cup E_r). \tag{1}$$

The calculation of a union of events can be obtained as the sum of the probability of disjoint events:

$$\begin{aligned} R(G) &= \Pr(E_1 \cup E_2 \cup \cdots \cup E_r) && (2)\\ &= \Pr(E_1 \cup \bar{E}_1 E_2 \cup \cdots \cup \bar{E}_1 \bar{E}_2 \cdots \bar{E}_{r-1} E_r) && (3)\\ &= \Pr(E_1) + \Pr(\bar{E}_1 E_2) + \cdots + \Pr(\bar{E}_1 \bar{E}_2 \cdots \bar{E}_{r-1} E_r) && (4) \end{aligned}$$

where $\bar{E}_j$ represents the complement of event E_j. The disjoint formulation above can be used to iteratively define a dynamic lower bound for the all-terminal reliability:

- $R_{L_1}(G) = \Pr(E_1)$,
- $R_{L_2}(G) = R_{L_1}(G) + \Pr(\bar{E}_1 E_2)$,
- $\cdots$
- $R_{L_i}(G) = R_{L_{i-1}}(G) + \Pr(\bar{E}_1 \bar{E}_2 \bar{E}_3 \cdots \bar{E}_{i-1} E_i)$.

The network unreliability $U(G) = 1 - R(G)$ can be determined using a similar approach based on mincuts. Let $\{C_i,\, i = 1, 2, \ldots, s\}$ denote the set of mincuts where s is the number of mincuts in the network and let F_i be the event that all edges in C_i are simultaneously in a failed state.

Then $U(G)$ can be found from

$$U(G) = \Pr(F_1 \cup F_2 \cup \cdots \cup F_s). \tag{5}$$

In a fashion similar to the all-terminal reliability, one can develop a dynamic lower bound for the network unreliability using the events associated with the failure of all the edges in each of the mincuts:

- $U_{L_1}(G) = \Pr(F_1)$,
- $U_{L_2}(G) = U_{L_1}(G) + \Pr(\bar{F}_1 F_2)$,
- $\cdots$
- $U_{L_i}(G) = U_{L_{i-1}}(G) + \Pr(\bar{F}_1 \bar{F}_2 \bar{F}_3 \cdots \bar{F}_{i-1} F_i)$.

This results in a dynamic upper bound for the all-terminal reliability as $R_{U_i}(G) = 1 - U_{L_i}(G)$. Thus the bounds on all-terminal reliability are given by

$$R_{L_i}(G) \leq R(G) \leq R_{U_i}(G). \tag{6}$$

In the bound calculations each new term of order i, $(\bar{E}_1 \bar{E}_2 \bar{E}_3 \cdots \bar{E}_{i-1} E_i)$ or $(\bar{F}_1 \bar{F}_2 \bar{F}_3 \cdots \bar{F}_{i-1} F_i)$ can be expressed as a sum of disjoint products, by an iteration (the i-th) of Abraham's algorithm [2] or any of its improved versions [19, 24, 25, 45]. It can also be calculated using a BDD [9, 10] by iteratively adding the mincuts/minpaths (using the Boolean operations AND and OR) and transversing the tree using the Boolean vertices *then* and *else*.

Regarding the selection of the algorithm for the calculation of the reliability upper bound, the number of disjoint products and the fraction of added edges (number of edges added to the BDD divided be the number of edges of the network) are similar for sequential mincuts and for disjoint mincuts generation; this happens because the sequentially generated mincuts, albeit not disjoint, do not tend to share many edges. However, a BDD was not used for the calculation of the reliability upper bound because it was verified that the number of vertices of the BDD sometimes grew very fast with just a few additional mincuts. Hence, we adopted the algorithm KDH88 from [19] which we observed had, in general, better performance than BDD, especially when the number of relevant mincuts was relatively small, which was usually the case.

3.1 Heuristic study

In the context of the two-terminal reliability problem Sebastio *et al.* [35] proposed a set of heuristics which considers only the most important minpaths or mincuts for the calculation of the upper and lower bounds on the network reliability. One of the main ideas of Sebastio *et al.* [35] is to iteratively generate the minpaths (and mincuts) edge disjoint with the ones obtained in previous iterations, and use them to update the probability of the union of events. The main drawback of their heuristic is that it is an approximation since it ignores the probability of intersection of two events. If the desired bound accuracy is not achieved, then the heuristic proceeds to a process where they generate mincuts or minpaths until a fixed limit is reached or a maximum run time limit occurs. Their algorithm selectively includes in the BDDs only the mincuts (minpaths) which a prior calculation showed would contribute significantly to the upper (lower) reliability bound; the other mincuts (minpaths) are stored in a heap; if the desired bound accuracy is not achieved then the algorithm iteratively readjusts the relevance criteria for including a min-cut (minpath) and considers only candidates from the heap.

In our previous work [39] it was observed that the mincuts, generated in order of decreasing probability, contribute with decreasing amount to the reliability upper bound. Also, it was observed that the contribution of each spanning tree, sequentially generated, by decreasing probability, had a high variability. Hence, we investigated whether it would be advantageous to first consider disjoint cuts and trees, before their sequential enumeration in computation of lower and upper bounds.

For the generation of the disjoint mincuts an adaptation of the algorithm proposed by Vazirani and Yannakakis was used [44]. In the algorithm after the retrieval of a mincut, the cost of the graph edges that correspond to the retrieved mincut are changed to infinity and the heap is updated. The algorithm stops when the cost of the mincut is infinity, which signals that no more mincuts, which are disjoint with the previous ones, are present in the network. In this approach, if the desired reliability bound is not reached using all the obtained disjoint mincuts, then the algorithm continues generating mincuts by decreasing probability and without repeating disjoint mincuts.

Table 1 presents numerical results for thirteen networks illustrating the difference between the reliability lower bounds obtained using maximally disjoint trees and sequential trees ($R_L(\text{max dis}) - R_L(\text{seq})$ column) and the difference between the upper bounds obtained using disjoint mincuts and sequential mincuts ($R_U(\text{disj}) - R_U(\text{seq})$ column). The number of mincuts considered in Table 1 was equal to the total number of disjoint mincuts obtained for each network. Similarly the number of trees was equal to the total number of maximally disjoint trees obtained for each network. Regarding maximally disjoint tree generation, there is a clear advantage with respect to sequential tree generation, as can be seen by the positive difference in the corresponding reliability lower bounds. On the other hand, it can be seen that, with the exception of nobel-germany, where there is a tie, the disjoint mincut generation presents no advantage over the sequential mincut generation.

The results of using the procedure to upper bound the reliability are further illustrated in Figure 1 for the newyork and pioro40 networks (see Table 2) from the SNDlib library [31]. In computing the results of Figure 1, the same number of mincuts are added using two different approaches: disjoint mincuts ("Dis" in Figure 1) and sequentially generated mincuts ("Seq" in Figure 1). It is possible to observe that the two approaches have only a slight difference in the evolution of the reliability upper bound. Nevertheless, adding the mincuts using a sequential procedure results in a slightly lower reliability upper bound for the same number of mincuts. The latter observation can be explained due to the fact that the sequentially generated mincuts have a higher probability of failure than the disjoint ones which results in a greater decrease in the reliability upper bound for the same number of generated mincuts.

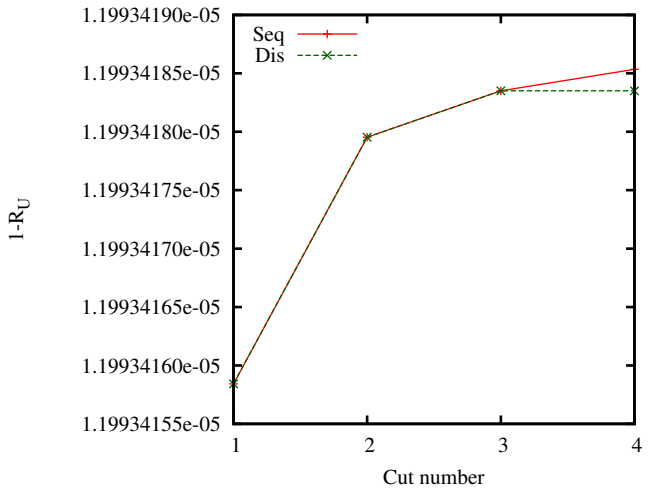
Now considering the construction of the dynamic lower bound on the reliability, we note that when the spanning trees are generated by decreasing probability, they often share a significant number of edges. Although this makes it

simpler to calculate their union probability, it usually results in a small additional contribution to the reliability lower bound. Hence using disjoint spanning trees, the approach equivalent to the disjoint path approach considered in the work of Sebastio *et al.* [35], would seem a promising technique. However, due to the number of edges in each spanning tree ($|N|-1$), the number of fully disjoint trees can be very small and in some topologies after generating the first tree, no second disjoint tree can be calculated. Therefore it was decided to evaluate the impact of considering maximally disjoint spanning trees, instead of using sequentially generated spanning trees, by decreasing probability.

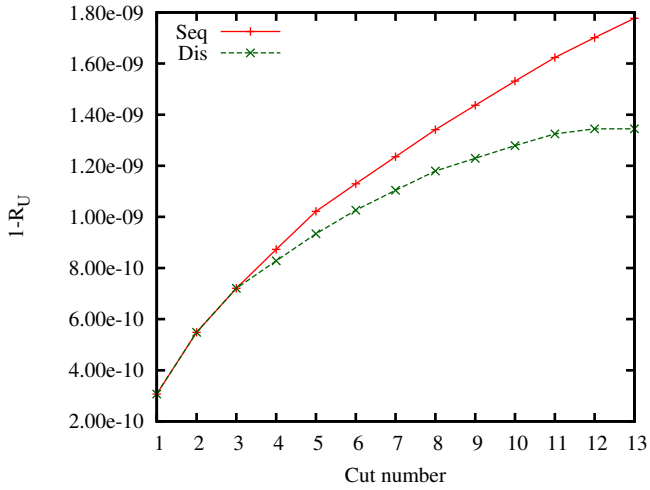
To generate maximally disjoint spanning trees an adaption of Prim's algorithm [12] was used. The adapted algorithm works iteratively by finding the minimum spanning tree and changing the corresponding edge weights to infinity in the graph. The algorithm stops when all the edge costs of the found minimum spanning tree are equal to infinity, which corresponds to the exhaustion of the maximally disjoint spanning trees in the graph – note that the set of obtained trees strongly depends on the first selected tree. The results for the two methods of generating the spanning trees are presented in Figure 2 for the newyork network from SNDlib. It can be observed that using maximally dis-

Table 1: Illustrating the advantage of considering maximally disjoint trees and sequential cuts.

Network	$R_L(\text{max dis}) - R_L(\text{seq})$	$R_U(\text{dis}) - R_U(\text{seq})$
polska	2.32e-04	3.77e-13
atlanta	6.96e-03	2.75e-08
newyork	1.83e-02	1.82e-13
nobel-germany	1.28e-04	0.0
geant	9.56e-04	6.81e-12
france	6.47e-03	1.40e-08
nobel-eu	8.92e-04	9.74e-12
pioro40	6.47e-02	4.33e-10
germany50	6.67e-04	1.14e-13
netvkk	2.18e-04	5.30e-14
ta2	3.93e-02	3.19e-08
italia	8.59e-04	4.20e-14
india	5.22e-03	3.31e-14



(a) newyork network



(b) pioro40 network

Fig. 1: Evolution of the upper bound for the reliability R_U when the mincuts are sequentially generated (Seq) and when the mincuts are disjointly generated (Dis) for the pioro40 and newyork network.

joint spanning trees results in a significant improvement in the calculation of the reliability lower bound, since for the same number of spanning trees a higher value for the lower bound is achieved earlier.

Note that in the results shown in Figure 2, the first three maximally disjoint trees are in fact fully disjoint while the fourth tree (in the line with the label

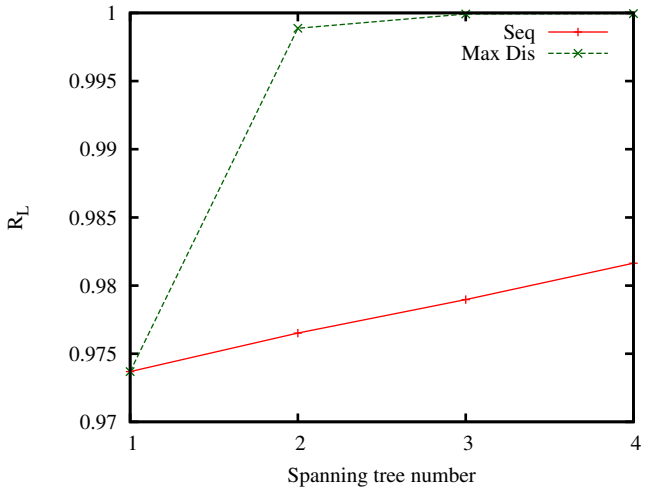


Fig. 2: Evolution of the reliability lower bound R_L when the added spanning trees are maximally disjoint with previously selected trees (Max Dis) and when the added spanning trees are sequentially generated by the order of their decreasing reliability (Seq) for the newyork network.

Max Dis) shares edges with the previously obtained trees. The improvement of the reliability lower bound when using maximally disjoint spanning trees to update the probability in the BDD can be explained if one considers the fraction of edges added and the number of disjoint products in the BDD. For the case of maximally disjoint spanning trees the fraction of added edges evolves very rapidly from 0.30 to 1.0 which corresponds to a number of disjoint products in the BDD from 1 to 4174. In contrast to the sequential case, where the spanning trees are added by the order of their reliability, the fraction of added edges increases slowly. This large difference between the number of disjoint products is the reason for the rapid increase of the reliability lower bound when using maximally disjoint spanning trees.

Figure 3 presents the evolution of the reliability lower bound (R_L) with the fraction of added edges, when trees are sequentially generated by the order of their decreasing reliability. It is possible to observe that the R_L changes in steps with the increase of added edges. Moreover, the increase of added edges is also related to the number of disjoint products in the BDD as can be seen in Figure 4. It is possible to observe in Figure 4 – note the logarithmic scale of the y axis – that there is an exponential increase of the number of disjoint products with the increase of added edges and thus revealing the NP-hard combinatorial

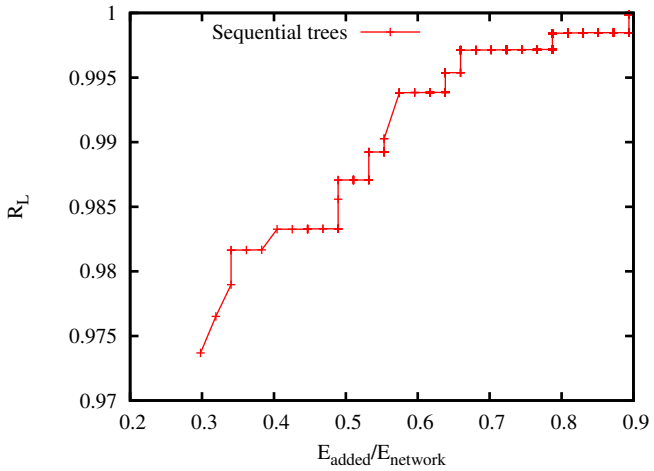


Fig. 3: Evolution of the reliability lower bound R_L with the fraction of added edges, that is the number of edges added to the BDD divided by the number of edges of the network, for the newyork network.

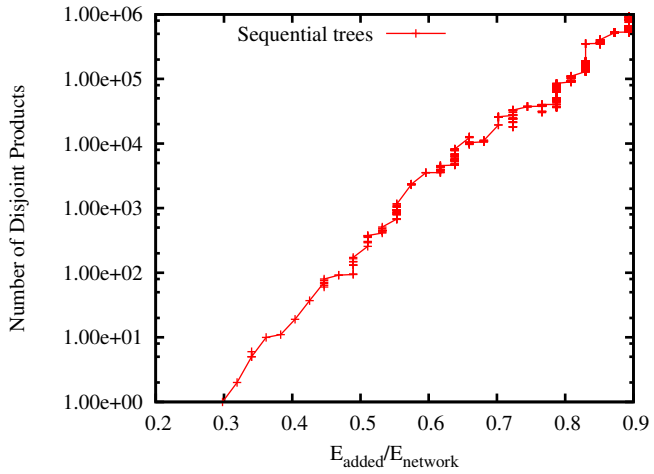


Fig. 4: Evolution of the number of disjoint products in the BDD with the fraction of added edges for the newyork network, when the spanning trees are sequentially generated by the order of their decreasing reliability.

nature of the problem.

3.2 A Novel Bounding Algorithm

Given the result above, we adopt a strategy of constructing bounds where for the upper bound the mincuts are examined by decreasing order of occurrence and for the lower bound trees are added in maximally disjoint fashion. Our goal is to obtain fast converging bounds in a computationally efficient fashion. The central ideas of the bounding procedure are as follows.

- First maximally disjoint trees are generated. Then spanning trees are iteratively generated, by order of their decreasing reliability – the minpaths corresponding to the maximally disjoint spanning trees, already considered, are ignored. This results in an increasing lower bound for the all-terminal reliability.
- Generating mincuts iteratively, by decreasing probability, in order to obtain a decreasing upper bound for the all-terminal reliability.
- Evaluating the contribution of a mincut or a minpath to the bounds improvement, and keeping only those whose contribution is considered significant – similar to what was proposed in [35].
- Taking advantage of the iterative nature of the algorithm for calculating the sum of disjoint products, in deciding whether to definitely add a new mincut.
- Taking advantage of the procedure proposed in [35] for deciding whether to insert a new minpath in a BDD.

These concepts were implemented in the bounding algorithm proposed here, which is termed Algorithm ATRB-SB. In order to make the description of ATRB-SB more concise, we define the following auxiliary functions, and algorithms.

- GraphReduction(R, A) which returns the triplet (G', A', Ω). This function makes pendant, series and possibly parallel reductions of the network, creating a network $G' = (V', E')$, with corrected edge probabilities of being operational given in set A', and where the multiplicative conditioning factor is $\Omega : R(G) = \Omega R(G')$ [38].
- UpperBound(G', A', C, R'_L, R'_U, α) which returns the triplet (C, C_i, R'_U), where C is the set of selected mincuts. If the mincut generated by UpperBound in the i-th iteration (C_i) was considered relevant, then C is

updated (C now contains C_i), and R'_U is also updated. The pseudo-code is given in Algorithm UpperBound.

- LowerBound(G',A', B, R'_L, R'_U, β) which returns the triplet (B, P_i, R'_L), where B is the BDD of the selected minpaths. If the minpath (spanning tree) generated by LowerBound in the i-th iteration (P_i) was considered relevant, then B is updated (B now contains P_i), and R'_L is also updated. The pseudo-code is given in Algorithm LowerBound.

Algorithm UpperBound Algorithm for possibly updating the all-terminal reliability Upper Bound, using mincuts and sum of disjoint products.

Require: A connected undirected graph $G' = (V', E')$, the edge reliabilities A', the already generated mincuts C, the current upper and lower bounds, R'_L and R'_U, and the α value.

Ensure: Returns the obtained mincut C_i, the possibly updated set of mincuts C, and the possibly updated reliability upper bound R'_U.

1: generate the i-th mincut C_i using an iterative mincut enumeration algorithm
2: calculate $\Delta U'_{L_i}$, calculating iteratively the sum of disjoint products
3: **if** $\Delta U'_{L_i} > \min((1 - R'_U), R'_L)\alpha$ **then**
4: update U'_L and then do $R'_U \leftarrow (1 - U'_L)$
5: $C \leftarrow C \cup \{C_i\}$
6: **end if**
7: **return** (C, C_i, R'_U)

Algorithm ATRB-SB starts with an undirected connected (otherwise $R(G) = 0$) graph and in order to reduce the size of the problem function GraphReduction is called. The network is pruned of all the spurs (or pendants); a series reduction is also performed to remove all edges incident on vertices of degree two [38]. This last modification may also result in the need to make parallel reductions. The reduction procedure is repeated until all network vertices are at least of degree three. The edge probabilities of being operational are dully adjusted and a reliability correction factor (Ω in Algorithm ATRB-SB) is also calculated as explained in [38]. Using this procedure, some networks can be completely reduced, as in the case of the abilene network [31] where $R(G) = \Omega$, while in other networks, like pioro40 [31], no reduction is possible and $\Omega = 1$. It is in the (hopefully) reduced network that the mincuts and spanning trees, required for the bounds calculations, are determined.

Algorithm LowerBound Algorithm for possibly updating the all-terminal reliability Lower Bound, using minpaths and a binary decision diagram.

Require: A connected undirected graph $G' = (V', E')$, the edge reliabilities A', the current binary decision digram B, the current upper and lower bounds, R'_L and R'_U, and the β value.

Ensure: Returns the obtained minpath (spanning tree P_i), the possibly updated binary decision diagram B, and the possibly updated reliability lower bound R'_L.

1: generate the i-th minpath P_i, starting by generating maximally disjoint spanning trees followed by an iterative spanning tree enumeration algorithm
2: calculate $\Delta R'_{L_i}$ according to [35]
3: **if** $\Delta R'_{L_i} > \min((1 - R'_U), R'_L)\beta$ **then**
4: update R'_L, and insert P_i into B
5: **end if**
6: **return** (B, P_i, R'_l)

In order to iteratively obtain the minpaths, by increasing probability, a k spanning trees enumeration algorithm is used. The probability P_i of a spanning tree being operational is given by $\prod_{e \in P_i} p_e$. Similarly the probability of all the edges in cutset C_i being simultaneously in a failed state is given by $\prod_{e \in C_i}(1-p_e)$. It is well known that these metrics can transformed into additive metrics, using logarithms. For minpath enumeration we define the cost of edge e, $c^p_e = -\log p_e$, and for mincut enumeration we define the cost $c^c_e = -\log(1 - p_e)$. The cost of the minpath P_i and mincut C_i become $c^p(P_i) = \sum_{e \in P_i} c^p_e$ and $c^c(C_i) = \sum_{e \in C_i} c^c_e$, respectively. Therefore, generating spanning trees and mincuts by increasing cost corresponds to generating spanning trees in order of decreasing total probability (of all edges in the spanning tree being operational) and to generating cuts by decreasing probability (of every edge in the cut being in a failed state), respectively.

In the final version of the algorithm maximally disjoint spanning trees are generated first using an adapted version of Prim's algorithm as explained previously. After finding all the maximally disjoint spanning trees, the minimum spanning trees are iterative generated.

In Step 1 of the auxiliary Algorithm UpperBound the mincuts are generated iteratively by increasing cost (that is by decreasing order of their probability) using the algorithm proposed by Vazirani and Yannakakis [44]. The algorithm

Algorithm ATRB-SB Algorithm for All Terminal Reliability Bounds using spanning trees, mincuts, sum of disjoint products and a binary decision diagram.

Require: A connected undirected graph $G = (V, E)$, the edge reliabilities A, the width of the interval between the bounds (ΔR), CPU time limit $cpu_{\max}$, the α and β values.

Ensure: Returns the obtained reliability lower and upper bounds: R_L and R_U.

1: $(G', A', \Omega) \leftarrow$GraphReduction($R$, A)
2: $U'_L \leftarrow 0$; $R'_L \leftarrow 0$
3: $k \leftarrow \lceil$average vertex degree of $G' \rceil$
4: $i \leftarrow 1$, $cut_{stop} \leftarrow 0$, $tree_{stop} \leftarrow 0$, $cpu_1 \leftarrow 0$
5: $C \leftarrow \emptyset$ (the current set of mincuts)
6: $B \leftarrow \emptyset$ (the current empty BDD)
7: **while** $((R'_U - R'_L) > \Delta R/\Omega) \wedge (cpu_i < cpu_{\max}) \wedge (cut_{stop} = 0 \vee tree_{stop} = 0)$ **do**
8: **if** $cut_{stop} = 0$ **then**
9: $(C, C_i, R'_U) \leftarrow$ UpperBound(G', A' C, R'_U, α)
10: **end if**
11: **if** $tree_{stop} = 0$ **then**
12: $(B, P_i, R'_L) \leftarrow$LowerBound($G'$,$A'$, B, R'_L, β)
13: **end if**
14: **if** $i \neq 1$ **then**
15: **if** $C_i \cap P_1 = k$ **then**
16: $cut_{stop} \leftarrow 1$
17: **end if**
18: **if** $C_1 \subset P_i \wedge P_i$ was sequentially generated **then**
19: $tree_{stop} \leftarrow 1$
20: **end if**
21: **end if**
22: $cpu_i \leftarrow$ total CPU time at the end of the i-th iteration
23: $i \leftarrow i + 1$
24: **end while**
25: $R_L \leftarrow \Omega R'_L$
26: $R_U \leftarrow \Omega(1 - U'_L)$
27: **return** (R_L, R_U)

proposed by Katoh *et al.* [17, 22] for the iterative generation of spanning trees by increasing cost (that is by decreasing order of their reliability) was used in Step 1 of the auxiliary Algorithm LowerBound. The contribution of cut C_i to the bounds is calculated in Step 2 of Algorithm UpperBound using algorithm KDH88, proposed in [19]. This algorithm was selected for two reasons: according to its author it is more efficient than the algorithms in [2, 6, 24], and in [18] it was also verified that this algorithm did perform better than [45]. Moreover, the performance of KDH88 is not strongly dependent on the order of the minpath/mincut, hence avoiding the need of ordering the $i-1$ minpaths/mincuts before making them disjoint with the i-th minpath/mincut.

The contribution of spanning tree P_i to the bounds is calculated in Step 2 of Algorithm LowerBound, using a BDD and the algorithm proposed in [35]. As illustrated in Subsection 3.1 the BDD is very effective for the calculation of the reliability lower bound when maximally disjoint spanning trees are considered first.

The purpose of Step 3 in Algorithm UpperBound is to select only the mincuts which contribute to a significant change in the probability, where α is a constant with value chosen between 0 and 1. Furthermore, the same procedure is used for each minpath – see Step 3 of Algorithm LowerBound – where β is a constant with value chosen between 0 and 1. If the contribution of mincut C_i ($\Delta U'_{L_i}$) and of minpath P_i ($\Delta R'_{L_i}$) for improving the bounds is considered significant, U'_L and R'_U are updated in Step 4 of Algorithm UpperBound and in Step 4 of of Algorithm LowerBound, respectively. The idea of considering only relevant minpaths and mincuts was inspired by [35], but presents the following different characteristics:

- The value of constants α and β are fixed; because the minpaths and mincuts are generated sequentially readjustment was not needed.
- The relevance of the contribution of a mincut or of a minpath is not considered only in relation with R_U or R_L, respectively, but with the minimum of $1-R_U$ and R_L. Hence, one selects candidate mincuts and candidate min-paths which are relevant for closing the reliability gap.

In Step 1 of Algorithm LowerBound the modified Prim algorithm is used to generate the maximally disjoint spanning trees, as described in Subsection 3.1; after the generation of all maximally disjoint spanning trees, a k minimum spanning tree enumeration algorithm is used. Relative to the algorithm proposed in

[39] the ceiling function was used in Step 3 instead of the floor function due to the fact that this allows one to include some additional mincuts, resulting in slight improvement of the final results. Therefore, Algorithm ATRB-SB iteratively reduces the width of the bounds using the generated mincuts (in Step 9) and spanning trees (see Step 12) in conjunction with a sum of disjoint products algorithm and a BDD, respectively.

Algorithm ATRB-SB has several stopping conditions. The first stopping condition holds when the difference between the upper and lower bounds achieves the desired error ΔR – this should be the main stopping condition of the algorithm. The second stopping condition is satisfied if the CPU time used exceeds $cpu_{\max}$ – in the experiments $cpu_{\max} = 3600$ seconds. The third stopping condition is of a topological nature. The first part of the third condition is related to the intersection of the current mincut with the spanning tree with the largest probability of being operational, that is, the first generated tree – see Step 15 of Algorithm ATRB-SB. If this intersection results in a set with a size equal to the ceiling function of the average degree k, this implies the following mincuts will not contribute significantly to diminishing the upper bound due to the fact that the mincut reliability will be very small. In fact, it was observed that U_L', in most cases, did not change significantly after considering the first few cuts. The second part of the third stopping condition tests if the first mincut, the one the largest probability, is contained in the current generated spanning tree – see Step 18 of Algorithm ATRB-SB. If the latter condition is met it implies that the following spanning trees (namely in networks with varying edge probability) will not contribute significantly to increase the lower bound due to the fact that their contribution to the reliability bound will be very small. In this case a large number of spanning trees would be required to effectively increase R_L'. Furthermore, during the generation of the maximally disjoint spanning trees the stopping condition is not tested ($tree_{stop}$ is not updated). After generating all the maximally disjoint spanning trees the stopping condition is tested as shown in Algorithm ATRB-SB. Note that the stopping conditions, associated with the variables $tree_{stop}$ and cut_{stop} in the algorithm, may result in limiting the maximum bound quality. In such a case one may choose to disable them.

4 Results and discussion

In order to evaluate the performance of the proposed algorithm we studied eleven real-world topologies taken from SNDlib [31], as well as two additional topologies representing the Italian telecommunications backbone network labeled italia from [43] and a topology representing a possible communication network for the California power grid labeled netvkk from [23]. The power grid network layout was obtained from California Energy Commission maps[1] and the communication network follows the power grid as detailed in [23]. Note that the power grid communication network needs to be designed to meet the 99.999% availability recommended by the U.S. Department of Energy (DOE) [13].

For each network topology, the probability that an edge e is operational p_e, which is equivalent to the edge availability a_e, was calculated using the following equation [28]:

$$a_e = 0.99987^{d_e/(250\times 1.6093)} \tag{7}$$

where d_e is the distance between the two end vertices of edge e in the network (calculated assuming that the coordinates of the vertices correspond to their GPS locations). In Eq. (7) the value 1.6093 converts *miles* to *km*. Structural properties of the considered networks are presented in Table 2. Note that the networks presented in Table 2 were reduced using the procedure proposed by Shooman [38]. The number of spanning trees was calculated using the determinant of the Kirchhoff matrix, after removing one column and row, using LU decomposition [8, 32]. It can be seen that even with this reduction the number of spanning trees in the network can be very large. Table 2 presents the number of vertices and edges of the reduced networks, the number of spanning trees, the average availability of the edges and the standard deviation of the edge reliabilities (σ_{Av}). Note that a wide range of networks was studied from sparse networks like polska to denser networks like newyork.

The results presented in this section were determined using three different algorithms:

- The **ATRB-SB** results correspond to the implementation of Algorithm ATRB-SB. In Step 12 Algorithm LowerBound is called. In Step 1 of LowerBound the *maximally disjoint* spanning trees, starting with the most probable one, are first generated. If the desired reliability bound is not reached, this is followed by the iterative generation of spanning trees cre-

[1] California Energy Commission: http://www.energy.ca.gov

Table 2: Network topological information for reduced networks.

Network	**Vertices**	**Edges**	**Number of spanning trees**	**Average Availability**	$\sigma_{\mathbf{Av}}$
polska	10	16	2501	0.999932	3.33E-5
atlanta	7	11	192	0.996398	0.001888
newyork	15	47	6.2391E5	0.996655	0.001336
nobel-germany	7	12	320	0.999942	2.91E-5
geant	10	21	3.8208E4	0.999501	0.000872
france	11	21	3.8909E4	0.997689	0.001448
nobel-eu	16	26	4.72554E5	0.999871	1.0E-5
pioro40	40	89	5.0612E20	0.996896	0.001345
germany50	39	73	9.0786E16	0.999968	1.60E-5
netvkk	14	26	4.8851E5	0.999961	2.54E-5
ta2	36	69	4.3905E15	0.997551	0.001442
italia	31	68	3.8011E15	0.999947	3.58E-5
india	31	75	8.1762E16	0.999660	1.566E-4

ated in order of increasing cost [17, 22] (that is, by decreasing probability). In Step 9 Algorithm UpperBound is called. In Step 1 of UpperBound the mincuts are iteratively generated by increasing cost [44]. Only relevant mincuts and minpaths are considered, according to Steps 3 and 3 of the auxiliary algorithms UpperBound and LowerBound, respectively. Recall that the KDH88 algorithm [19] is used in Step 2 of Algorithm UpperBound and the BDD[2] is used in Step 2 of Algorithm LowerBound.

- The **ATRB-2SDP** implementation corresponds to the algorithm [39] with the following improvements: only relevant mincuts and minpaths are considered, resulting in the inclusion of Steps 3 in Algorithm UpperBound and 3 in Algorithm LowerBound, both used in Algorithm ATRB-SB. The ceiling function in Step 3 is used instead of the floor function.

- The **ATRB-2BDD** implementation corresponds to an approach similar to the one proposed in [35], adapted for the all-terminal reliability problem.

  It corresponds to Algorithm ATRB-SB where a BDD is present in *both*

[2]The code used for the BDD implementation was retrieved from http://vlsi.colorado.edu/~fabio/CUDD/

Steps 2 and 2 of Algorithms UpperBound and LowerBound, respectively, for the calculation of bounds. Additionally, in Step 1 of UpperBound the *disjoint mincuts* are first generated, using a modified version of the Vazirani and Yannakakis algorithm [44], and when exhausted they are iteratively generated [44]. In Step 1 of Algorithm LowerBound the *fully disjoint* spanning trees, starting with the most probable one, are first generated, and when exhausted they are iteratively generated [17, 22]. As in **ATRB-SB** only relevant mincuts and minpaths are considered, according to Steps 3 of Algorithm UpperBound and 3 of Algorithm LowerBound.

The stopping conditions of the three algorithms are identical to the ones present in Algorithm ATRB-SB. All numerical results were obtained using a Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz processor with 16G of RAM. In all the results α and β were set to 1E-3.

As already mentioned, **ATRB-2BDD** is the implementation that is strongly related to the approach proposed by Sebastio *et al.* [35], taking into account the required adaptation for all-terminal reliability calculation. However after generating the disjoint mincuts and minpaths, the sequential generation of mincuts and minpaths is introduced, instead of some form of random selection. Note that in [35] the authors discarded using a k-shortest path enumeration algorithm, because they thought it would not allow the examination of each path as soon as it is found, when this is in fact possible using any of the following algorithms [26, 27, 48]. Also the conditions for including or excluding a mincut or a minpath are the ones in Steps 3 and 3 of Algorithms UpperBound and LowerBound, respectively, which differ (as already explained) from the equivalent condition in [35]. The cut_{stop} and $tree_{stop}$ conditions (not present in [35]) are only evaluated after all the disjoint mincuts and minpaths have been generated.

Figures 5-9 present the results for Algorithm ATRB-SB using the three different implementations. In the figures we used the keys 2SDP, 2BDD and SB for the algorithms ATRB-2SDP, ATRB-2BDD and ATRB-SB, respectively. In Figures 5-7 each CPU value corresponds to the average CPU of ten separate runs; error bars representing the standard deviation of the observed CPU times are also included, although in most cases they are barely visible. To avoid over loading the lines in the graphs not every value of ΔR, and corresponding CPU time, is represented in the graphics. In Figures 5-7 the first point in each line represents the ΔR value and execution time for the first iteration. The following

points shown in the graph satisfy the rule: if an iteration with ΔR_j is shown in the graph, the next iteration k ($k > j$) to be shown must satisfy the relation $\Delta R_j / \Delta R_k \geq \delta$. In the figures δ was considered equal to 1.005. The error bars are mostly barely visible, but can make the graphs difficult to read (filling the hollow symbols, and artificially extending the lines when they are more visible). So in the main graph we present the lines without the error bars and in a small window is presented the relevant part of the graph with the error bars (in most cases barely visible). As was previously stated the evolution of the bounds is done by steps (see Figure 3) and so the final ΔR can be smaller than the desired value.

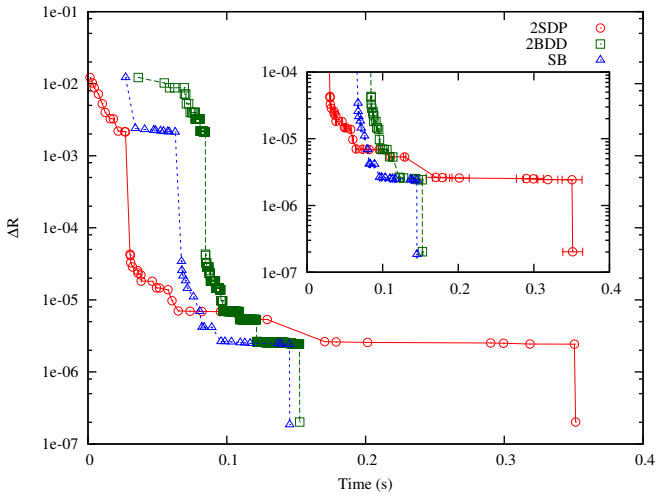


Fig. 5: Evolution of the execution time with ΔR, when the target reliability gap is 1E-6, for the france network.

In the figures it is possible to observe a CPU difference in the first iteration. This difference results from the cost associated with calling the BDD constructor, once in the case of **ATRB-SB** and twice in the case of **ATRB-2BDD**. For the france network, the results in Figure 5 show that **ATRB-2SDP** has a smaller execution time when ΔR is larger but for a more precise result **ATRB-SB** has the lowest CPU time. The performance of Algorithm **ATRB-SB** and Algorithm **ATRB-2BDD** becomes similar for ΔR less than 5E-6; after that point there is only a slight advantage using Algorithm **ATRB-SB**. It should be noted that for the three algorithms the value ΔR goes from 2.4E-06 to less than 1E-6 in a single iteration (the last one). This abrupt change in ΔR is visible

in the graph, where the last points correspond to ΔR =1.85E-07 in the case of **ATRB-SB**, and ΔR =2.027E-07 (in the case of the two other algorithms). The CPU time of the last iteration is too small to be noticeable in the graph.

Figure 6 presents the results for the netvkk network, which were obtained by setting ΔR equal 1E-10. It is possible to observe that **ATRB-SB** and **ATRB-**

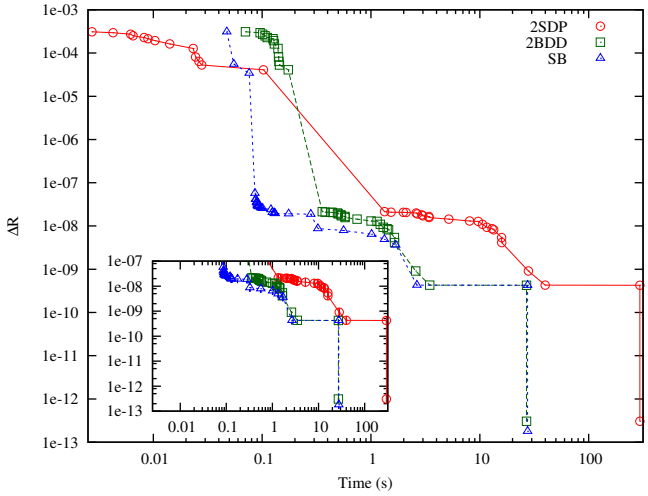


Fig. 6: Evolution of the execution time with ΔR, when the target reliability gap is 1E-10, for the netvkk network.

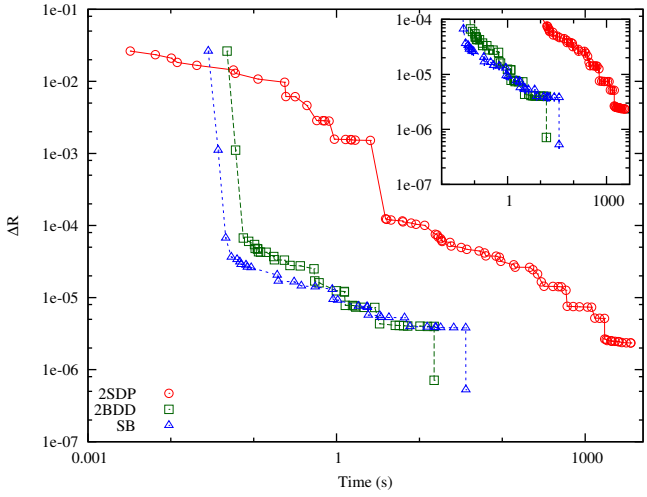


Fig. 7: Evolution of the execution time with ΔR, when the target reliability gap is 1E-6, for the newyork network.

2BDD have similar performance (especially for ΔR >1E-8) being the fastest in achieving the desired accuracy. As in the previous figure, a ΔR less than 1E-10 is abruptly achieved in the last iteration, where ΔR goes from 4.2E-6 to less than 4E-13. Note that many iterations were carried out without visibly reducing the reliability gap, hence the step like aspect of the final part of the graph.

In Figure 7, with ΔR = 1E-6, results are presented for the newyork network. In this case, **ATRB-SB** and **ATRB-2BDD** have almost the same performance, being the implementations with the lower execution time. Note that **ATRB-2SDP** does not reach ΔR = 1E-6 within the allowed maximum CPU time of 3600 seconds. As in the previous two figures, the final iteration results in an abrupt change in ΔR.

The results presented in Figures 5-7 are for relatively small networks. Nevertheless, they demonstrate that **ATRB-SB** and **ATRB-2BDD** are the implementations with best performance.

Figure 8 shows the results for larger networks when the target ΔR is set to 1E-5. It is possible to observe in Figure 8 that the desired error was not achieved in some networks. In the case of the india35 network, Algorithm **ATRB-SB** stopped with ΔR = 9.98E-6, overcoming the desired target of 1E-5, in less than 4 minutes. However the **ATRB-2BDD** and **ATRB-2SDP** algorithms stopped because of the $tree_{stop}$ and cut_{stop} stopping conditions, ending with with ΔR close to 1E-3. In this case the stopping conditions prevented these two algorithms from improving their solution.

For the other four networks the maximum allowed execution time of 3600 seconds was reached by both the **ATRB-2BDD** and **ATRB-2SDP** algorithms. However, it should be pointed out that in the case of the pioro40 network the execution time of **ATRB-2BDD** was on average 6345 seconds, due to the fact that it took on average 3600 seconds to complete the last iteration, and the CPU time limit is only verified per iteration. The results of Figure 8 demonstrate that **ATRB-SB** is the algorithm with the best performance for the networks under study. This can be explained by the use of the maximally disjoint trees in conjunction with the BDD for the calculation of the reliability lower bound and with the use of KDH88 for the calculation of the reliability upper bound.

The performance of the different versions can be analyzed further. Figure 9 presents the execution time versus the iteration number for the three methods when using a ΔR equal to 1E-6 in the france network. Each point is the average value of ten runs and error bars show the standard deviation of the CPU times.

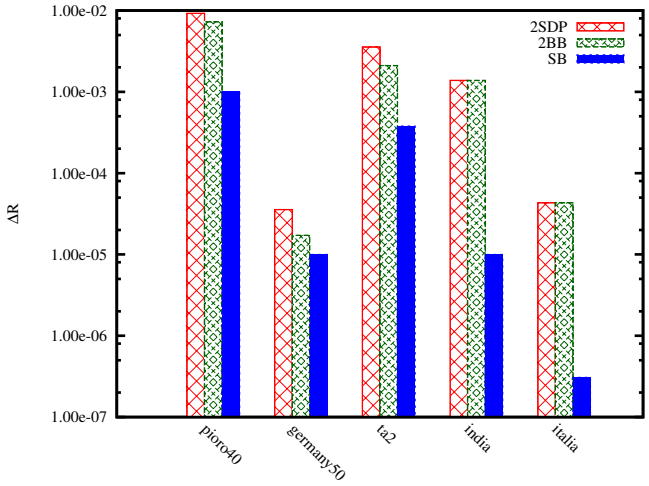


Fig. 8: Difference between the upper and lower reliability bounds for selected networks, with $|E| > 60$, when the target (ΔR) is set to 1E-5.

In Figure 9, for readability of the results, not every iteration is represented. In this case, after representing the initial 20 iterations, a point is marked for the 25-th iteration and from that point onwards, 25 iterations separate each point in the graphic. The results in Figure 9 demonstrate that the initial time for

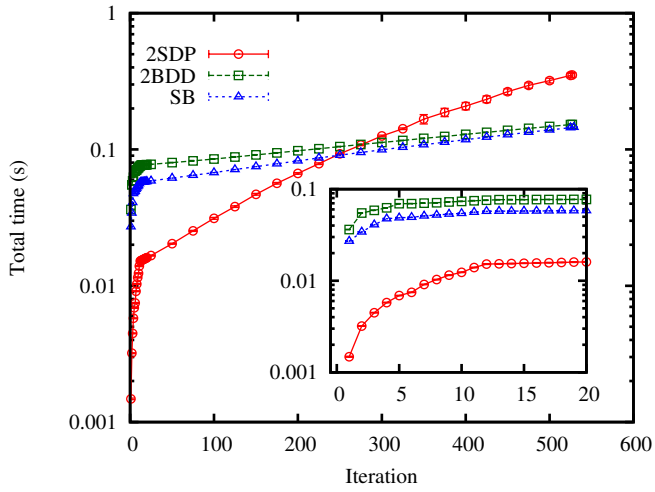


Fig. 9: Execution time after each iteration for the france network, when the target reliability gap is equal to 1E-6.

Table 3: Bounds for several networks with network reduction using Algorithm **ATRB-SB**, with ΔR =1E-6.

Network	$\mathbf{R_L}$	$\mathbf{R_U}$	**Time(s)**	σ_t(s)
polska	0.999999970	0.999999992	0.0548	0.0135
atlanta	0.999953259	0.999953409	0.0431	1.2E-4
newyork	0.999987480	0.999988007	36.339	0.045
nobel-germany	0.999999922	0.999999992	0.0273	1.5E-4
geant	0.999994046	0.999994692	0.0533	3.2E-4
france	0.999925386	0.999925572	0.1454	9.0E-4
nobel-eu	0.999998857	0.999999529	0.132	0.011
pioro40	0.999003391	0.999999998	3600.64	0.25
germany50	0.999999747	0.999999989	28.630	0.044
netvkk	0.999359484	0.999359542	0.0859	2.9E-3
ta2	0.998359248	0.998604593	3600.12	0.08
italia	0.999999686	0.999999993	0.9176	3.6E-3
india35	0.999992969	0.999999817	1918.34	0.98

methods **ATRB-2BDD** and **ATRB-SB** is larger than the time for **ATRB-2SDP**. This is related to the CPU time required by the BDD constructor, as already mentioned. Nevertheless, it is possible to observe for the three methods a rapid increase of the execution time in the first steps. The stopping condition for updating the reliability upper bound is achieved quite early (visible as the knee of the lines in Figure 9). After that point the algorithms only continue updating the reliability lower bound, which is related to the sequential generation of the spanning trees. Hence, one can verify that the execution time presents a lower slope when using the BDD (**ATRB-2BDD** and **ATRB-SB**) instead of KDH88 (**ATRB-2SDP**) for updating the reliability lower bound.

The results for the bounds calculated using Algorithm ATRB-SB, corresponding to the **ATRB-SB** implementation, presented in Subsection 3.2, using network reduction, and considering ΔR =1E-6, can be seen in Table 3. Please note that in columns R_L and R_U are the rounded down and rounded up values (to 9 digits) of the obtained lower and upper bounds, respectively. This table also contains the average execution time in seconds and the corresponding standard deviation (σ_t), considering 10 executions of the algorithm, for obtaining the presented upper and lower bounds. In Table 3 it is possible to observe that

for larger networks like germany50 and newyork there is a substantial reduction in the execution time when compared with the previous version of the Algorithm ATRB-SB in [39]. In the case of the india35 network, Algorithm **ATRB-SB** stops with ΔR equal to 6.846E-6 due to the $tree_{stop}$ and cut_{stop} conditions after 1918 seconds. The execution time of **ATRB-SB** for the germany50 and italia networks is less than 30 seconds (when ΔR =1E-6), where for the same networks the other implementations take 3600 seconds for a larger ΔR (1E-5), as presented in Figure 8.

It is also possible to observe in Table 3 that for the pioro40 and ta2 networks, the desired error was not achieved given a maximum execution time of 3600 seconds. In fact, it can be seen in Table 3 that the proposed algorithm calculates the bounds for networks pioro40, germany50, ta2 and italia which can not be calculated using the method described by Nelson [30] to compute the Bonferroni bounds as was demonstrated in [39] (for pioro40, germany50, and ta2). Note that although in the ta2 network the required error (1E-6) was not achieved, the obtained bounds show that the network reliability for this network has only two nines (it is larger than 99% but smaller than 99.9%). The precision achieved by the proposed algorithm is adequate for the all-terminal reliability problem with the exception of the pioro40 network. As expected, in the case of netvkk, the all-terminal availability (see Table 3) even if a perfectly available communication network is assumed within every sub-station, only achieves three nines quite below the required 99.999% reliability [13].

5 Conclusions

A new improved algorithm for computing all-terminal reliability bounds was proposed, suitable for networks where the use of the Bonferroni or the Esary-Proschan bounds is infeasible.

The advantages of using a set of procedures to decrease reliability bounds computational time was illustrated. Mainly the use of maximally disjoint spanning trees, followed by the sequential generation of the minimum spanning trees, was shown to improve the calculation of the reliability lower bound. It was also demonstrated that the conjunction of the BDD with the sum of disjoint products decreases the reliability bounds computational time.

The performance of the new algorithm **ATRB-SB** was compared with **ATRB-2SDP**, an improved version of the algorithm in [39], and it was shown

that **ATRB-SB** can calculate bounds with higher precision and with lower computational time for transport networks than **ATRB-2SDP**. Additionally, **ATRB-SB** was compared with **ATRB-2BDD** (an algorithm closely related to the approach proposed in [35]) and it was seen that considering maximally disjoint spanning trees is more effective than just fully disjoint trees, and that using KDH88 for calculating the union probability for the mincuts is a good alternative to the BDD.

In conclusion, the results show the proposed approach is computationally feasible and reasonably accurate. Hence the corresponding algorithm allows one to obtain bounds when it is not possible to enumerate all mincuts or all minpaths.

Acknowledgements

The work of Jaime Silva has been supported by ICIS Project CENTRO-07-0224-FEDER-002003. The work of Teresa Gomes and Lúcia Martins has been in part supported by the Portuguese Foundation for Science and Technology under project grant UID/MULTI/00308/2013 and ICIS Project CENTRO-07-0224-FEDER-002003.

References

[1] ITU-T recommendation E.800: Quality of service and dependability vocabulary September 2008.

[2] J.A. Abraham, An improved algorithm for network reliability, IEEE Trans Reliab R-28 (1979), 58–61.

[3] F. Altiparmak, B. Dengiz, and A. Smith, A general neural network model for estimating telecommunications network reliability, IEEE Trans Reliab 58 (2009), 2–9.

[4] A. Balan and L. Traldi, Preprocessing minpaths for sum of disjoint products, IEEE Trans Reliab 52 (2003), 289–295.

[5] M. Ball, Computational complexity of network reliability analysis: An overview, IEEE Trans Reliab 35 (1986), 230–239.

[6] F. Beichelt and L. Spross, An improved Abraham-method for generating disjoint sums, IEEE Trans Reliab R-36 (1987), 70–74.

[7] F. Beichelt and L. Spross, Comment on "An improved Abraham-method for generating disjoint sums", IEEE Trans Reliab 38 (1989), 422–424.

[8] N. Biggs, Algebraic graph theory, Cambridge University Press, 2nd edition, 1993.

[9] K. Brace, R. Rudell, and R. Bryant, Efficient implementation of a BDD package, Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE, June 1990, pp. 40–45.

[10] R. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans Comput C-35 (1986), 677–691.

[11] C.J. Colbourn, The combinatorics of network reliability, The International Series of Monographs on Computer Science, Oxford University Press, 1987.

[12] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson, Introduction to algorithms, McGraw-Hill Higher Education, 2nd edition, 2001.

[13] Department of Energy, Department of Energy Commission: Requirements for Smart Grid Technologies, DoE October 2010.

[14] J.D. Esary and F. Proschan, A reliability bound for systems of maintained, interdependent components, J American Stat Association 65 (1970), 329–338.

[15] G.S. Fishman, A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness, IEEE Trans Reliab 35 (1986), 145–155.

[16] G.S. Fishman, A Monte Carlo sampling plan for estimating network reliability, Oper Res 34 (1986), 581–594.

[17] H. Gabow, Two algorithms for generating weighted spanning trees in order, SIAM J Comput 6 (1977), 139–150.

[18] T. Gomes and J. Craveirinha, An alternative method for calculating the probability of an union of events, $\lambda\mu$13 - ESREL 2002, Eur Conference System Dependability Saf, Dec Making Risk Manageme, Lyon, France, Vol. 2, 19-21 March 2002, pp. 426–430.

[19] K.D. Heidtmann, Smaller sums of disjoint products by subproduct inversion, IEEE Trans Reliab 38 (1989), 305–311.

[20] K.P. Hui, N. Bean, M. Kraetzl, and D. Kroese, The cross-entropy method for network reliability estimation, Ann Oper Res 134 (2005), 101–118.

[21] H. Imai, K. Sekine, and K. Imai, Computational investigations of all-terminal network reliability via BDDs, IEICE Trans. Fundam 82 (1999), 714–721.

[22] N. Katoh, T. Ibaraki, and H. Mine, An algorithm for finding k minimum spanning trees, SIAM J Comput 10 (1981), 247–255.

[23] V. Kounev, M. Levesque, D. Tipper, and T. Gomes, On smart grid communications reliability, Design Reliable Commun Networks (DRCN), 2015 11th Int Conference , March 2015, pp. 33–40.

[24] M.O. Locks, A minimizing algorithm for sum of disjoint products, IEEE Trans Reliab R-36 (1987), 445–453.

[25] M.O. Locks and J.M. Wilson, Note on disjoint products algorithms, IEEE Trans Reliab 41 (1992), 81–92.

[26] E. Martins and M. Pascoal, A new implementation of Yen's ranking loopless paths algorithm, 4OR 1 (2003), 121–134.

[27] E. Martins, M. Pascoal, and J. Santos, Deviation algorithms for ranking shortest paths, Int J Foundations Comput Sci 10 (1999), 247–263.

[28] M. Mezhoudi and C.H.K. Chu, Integrating optical transport quality, availability, and cost through reliability-based optical network design, Bell Labs Tech J 11 (2006), 91–104.

[29] F. Moskowitz, The analysis of redundancy networks, AIEE Trans Commun Electronics, 77 (1958), 627–632.

[30] A. Nelson, J.R. Batts, and R.L. Beadles, A computer program for approximating system reliability, IEEE Trans Reliab R-19 (1970), 61–65.

[31] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, SNDlib 1.0–Survivable Network Design Library, Networks 55 (2010), 276–286, http://sndlib.zib.de.

[32] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, Numerical recipes 3rd edition: The art of scientific computing, Cambridge University Press, 3rd edition, 2007.

[33] J. Provan and M. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, SIAM J Comput 12 (1983), 777–788.

[34] G. Rubino and B. Tuffin (Editors), Rare event simulation using Monte Carlo methods, Wiley, 2009.

[35] S. Sebastio, K.S. Trivedi, D. Wang, and X. Yin, Fast computation of bounds for two-terminal network reliability, Eur J Oper Res 238 (2014), 810 – 823.

[36] A. Sharafat and O. Ma'rouzi, All-terminal network reliability using recursive truncation algorithm, IEEE Trans Reliab 58 (2009), 338–347.

[37] D.R. Shier, Network reliability and algebraic structures, Claredon Press - Oxford, 1991.

[38] A. Shooman, Algorithms for network reliability and connection availability analysis, Electro/95 International. Professional Program Proceedings, Jun 1995, pp. 309–333.

[39] J. Silva, T. Gomes, D. Tipper, L. Martins, and V. Kounev, An algorithm for computing all-terminal reliability bounds, Reliable Networks Design Modeling (RNDM), 2014 6th Int Workshop, Nov 2014, pp. 76–83.

[40] C. Srivareeratana, A. Konak, and A.E. Smith, Estimation of all-terminal network reliability using an artificial neural network, Comput & Oper Res 29 (2002), 849 – 868.

[41] J.P. Sterbenz, D. Hutchison, E.K. Çetinkaya, A. Jabbar, J.P. Rohrer, M. Schller, and P. Smith, Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines, Comput Networks 54 (2010), 1245 – 1265.

[42] D. Tipper, Resilient network design: Challenges and future directions, Telecommunication Syst 56 (2014), 5–16.

[43] M. Tornatore, G. Maier, and A. Pattavina, Availability design of optical transport networks, IEEE J Selected Areas in Commun 23 (2005), 1520–1532.

[44] V.V. Vazirani and M. Yannakakis, "Suboptimal cuts: Their enumeration, weight and number," Automata, languages and programming, W. Kuich (Editor), Springer, 1992, Vol. 623 of Lecture Notes in Computer Science, pp. 366–377.

[45] J.M. Wilson, An improved minimizing algorithm for sum of disjoint products, IEEE Trans Reliab 39 (1990), 42–45.

[46] J.M. Won and F. Karray, Cumulative update of all-terminal reliability for faster feasibility decision, IEEE Trans Reliab 59 (2010), 551–562.

[47] J.M. Won and F. Karray, A greedy algorithm for faster feasibility evaluation of all-terminal-reliable networks, IEEE Trans Syst, Man, Cybern B: Cybern 41 (2011), 1600–1611.

[48] J.Y. Yen, Finding the k shortest loopless paths in a network, Manage Sci 17 (1971), 712–716.