A Failure-Distance Based Method to Bound the Reliability of Non-Repairable Fault-Tolerant Systems without the Knowledge of Minimal Cuts

Víctor Suñé and Juan A. Carrasco
Departament d'Enginyeria Electrònica
Universitat Politècnica de Catalunya
Diagonal 647, plta. 9
08028 Barcelona, Spain
email: sunye,carrasco@eel.upc.edu

Technical report `DMSD_99_1`
Last revision: December 11, 2000

Summary & Conclusions

CTMC (continuous-time Markov chains) are a commonly used formalism for modeling fault-tolerant systems. One of the major drawbacks of CTMC is the well-known state-space explosion problem. This work develops and analyzes a method (SC-BM) to compute bounds for the reliability of non-repairable fault-tolerant systems in which only a portion of the state space of the CTMC is generated. SC-BM uses the failure distance concept as the method described in [1] but, unlike that method, which is based on the computation of exact failure distances, SC-BM uses lower bounds for failure distances, which are computed on the system fault tree, avoiding the computation and holding of all minimal cuts as required in [1]. This is important since computation of all minimal cuts is NP-hard and the number of minimal cuts can be very large. In some cases SC-BM gives exactly the same bounds as the method described in [1]; in other cases it gives less tighter bounds. SC-BM computes tight bounds for the reliability of quite complex systems with an affordable number of generated states for short to quite large mission times. The analysis of several examples seems to show that the bounds obtained by SC-BM appreciably outperform those obtained by simpler methods, *eg* [2], and, when they are not equal, are only slightly worse than the bounds obtained by the method in [1]. In addition, the overhead in CPU time due to computing lower bounds for failure distances seems to be reasonable.

Index terms: Fault-tolerant systems, Non-repairable systems, Reliability Bounds, State-space reduction.

1 Introduction

Acronyms[1]

CTMC	continuous-time Markov chain
DTMC	discrete-time Markov chain
BM-1	bounding method in [1]
SC-BM	bounding method in this work
T-BM	trivial bounding method, *eg* [2]
T-SC-BM	implementation of T-BM in this work
FIFO	first-in, first-out

Definitions

· bag: collection of possibly repeated elements; the notation $c_1[n_1]c_2[n_2]\cdots c_k[n_k]$ is for a bag a including $n_i > 0$ instances of element c_i, $i = 1, 2, \ldots, k$; each $c_i[n_i]$ is part of a. Notation for bags is from [3] except that: 1) a subbag b of bag a is $b \subset a$ whether b is strictly contained in a or not, and 2) bags are denoted as explained here

· minimal cut: minimal bag of component classes whose failure implies the system failure

· failure bag: bag of component classes that can fail simultaneously (in a single transition)

· failure distance from state a: minimum number of components that must fail, in addition to those already failed in a, to fail the system

Notation

$\Pr\{c\}$	probability of event c
$\mathrm{ur}(t)$	unreliability: $\Pr\{$system has failed by time $t\}$
$[\mathrm{ur}(t)]_{\mathrm{lb}}$	lower bound for $\mathrm{ur}(t)$ obtained with SC-BM, BM-1 or T-SC-BM
$[\mathrm{ur}(t)]_{\mathrm{ub}}$	upper bound for $\mathrm{ur}(t)$ obtained with SC-BM
$[\mathrm{ur}(t)]'_{\mathrm{ub}}$	upper bound for $\mathrm{ur}(t)$ obtained with BM-1
$[\mathrm{ur}(t)]''_{\mathrm{ub}}$	upper bound for $\mathrm{ur}(t)$ obtained with T-SC-BM
$X = \{X(t); t \geq 0\}$	acyclic CTMC modeling the system
$X' = \{X'(t); t \geq 0\}$	acyclic CTMC used in SC-BM for computing $[\mathrm{ur}(t)]_{\mathrm{lb}}$ and $[\mathrm{ur}(t)]_{\mathrm{ub}}$
$X'' = \{X''(t); t \geq 0\}$	acyclic CTMC used in T-SC-BM for computing $[\mathrm{ur}(t)]_{\mathrm{lb}}$ and $[\mathrm{ur}(t)]''_{\mathrm{ub}}$
λ_a	output rate of state a in X
$\lambda_{a,b}$	transition rate from state a to state b in X
$\lambda_{a,B}$	$\sum_{b\in B} \lambda_{a,b}$
Λ	randomization rate (greater than or equal to the maximum output rate of both X and X')
$Y = \{Y_n;\ n = 0, 1, \ldots\}$	DTMC obtained by randomizing [4] X with rate Λ
$Y' = \{Y'_n;\ n = 0, 1, \ldots\}$	DTMC obtained by randomizing [4] X' with rate Λ
O	up states: set of states of X in which the system is operational
f	down state: absorbing state that represents system failure
G	subset of O that is generated
o	state of O without failed components
U	$O - G$
$d(a)$	failure distance from state a

[1]The singular and plural of an acronym are always spelled the same.

L	$d(o)$
U_d	$\{a \in U \mid d(a) = d\}$
U^k	$\{a \in U \mid \text{number of failed components in } a \text{ is } k\}$
$U_{k,d}$	$U^k \cap U_d$
$\widetilde{d}(a)$	lower bound for $d(a)$
$\widetilde{L}$	$\widetilde{d}(o)$
$\widetilde{U}_{d,i}$	$\{a \in U_d \mid \widetilde{d}(a) = i\}$
$\widetilde{U}_d$	$\{a \in U \mid \widetilde{d}(a) = d\}$
$F(a)$	bag of failed component classes in state a
MC	set of minimal cuts of the system
$\lvert\cdot\rvert$	cardinality of a set or a bag [3]
E	set of failure bags of the system
E_i	$\{e \in E \mid \lvert e\rvert = i\}$
FC	set of different cardinalities of failure bags
$\lambda_{\text{ub}}(e)$	upper bound for the rate with which failure bag $e \in E$ is realized from any state
f_i	$\sum_{e\in E_i} \lambda_{\text{ub}}(e)$
$\wedge, \vee$	logical operator AND, OR

Modeling is important in the design and analysis of fault-tolerant systems. These systems exhibit a stochastic behavior and, therefore, probabilistic measures are adequate for their quantitative assessment. An important class of such systems are those whose components cannot be repaired. For these systems, the reliability, Pr{system is operational by time t}, or its complement, ur(t), are suitable measures. Non-repairable fault-tolerant systems can be modeled using combinatorial methods and, more generally, hierarchical methods [5, 6]. Hierarchical methods require the behavior of components and subsystems to be mutually s-independent. Recently, combinatorial methods have been improved, allowing some complex dependencies such as lack of coverage [7, 8] to be dealt with. However, when the failure rate of a component depends on the global state of the system, then state-level modeling techniques such as CTMC are required. A major drawback of CTMC, especially of those modeling complex systems, is that the size of their state space is typically so large that it goes far beyond the available computing resources. This well-known problem is referred to as the state-space explosion. One approach to attack this problem is the use of bounding methods, in which only a subset G of O is generated. Typically, G includes the states with up to a given number K of failed components. Bounds for ur(t) can be trivially derived (*eg* [2]) by modifying X so that exits of X from G not going to f are directed to an absorbing state u_0. The probability of the modified CTMC X'' being in state f by time t is a lower bound for ur(t), and the probability of X'' being in $\{f, u_0\}$ by time t is an upper bound for ur(t). This lower bound is usually good, but the upper bound is not, because it is equivalent to assuming that the system is non-operational in all the states in U, which can be far from reality. A recent paper [1] proposed the BM-1 method, in which the behavior of the system out of the generated portion is bounded using the failure distance concept, resulting in an improved upper bound for ur(t).

BM-1 requires computing the failure distances from the states of U that are reachable from G in a single transition. These computations can be done knowing MC. We have developed an algorithm [9] that computes MC efficiently in many cases. However, computing

MC is NP-hard [10] and in some cases the algorithm [9] can break down. In addition, $|MC|$ can be very large, causing a large memory overhead due to the need to hold MC and related data structures for efficient failure distances computation.

This work develops and analyzes SC-BM, our new bounding method for $\text{ur}(t)$ using lower bounds for failure distances that are obtained on the fault tree, avoiding the computation and holding of MC.

Section 2 describes the class of models assumed in SC-BM and shows how bounds for $\text{ur}(t)$ can be computed in SC-BM from lower bounds for failure distances satisfying some conditions.

Section 3 defines the lower bounds for failure distances used in SC-BM, proves that they satisfy the required conditions, describes the procedures used in SC-BM to compute such lower bounds, and describes how the CTMC X' is generated in SC-BM.

Section 4 describes how the CTMC X'' is generated in T-SC-BM and proves that the cost (in terms of CPU time) of SC-BM is at most identical to the cost of T-SC-BM when $\widetilde{L} = 1$.

Section 5 analyzes SC-BM using two examples, and compares it with T-SC-BM and BM-1.

2 Class of Models and Unreliability Bounds

We consider acyclic CTMC X modeling non-repairable fault-tolerant systems. We assume that the system is made up of components that can be grouped into classes, the components of the same class being indistinguishable from a dependability view-point. The operational/down state of the system is determined by the unfailed/failed state of its components by a fault tree. The fault tree of the system is constructed using AND and OR gates and inputs. Inputs have associated with them different bags of the form $c[n]$. Input x with associated bag $c[n]$ has the value 1 if and only if at least n components of class c are failed. The value of the fault tree is computed as usual from the values of its inputs. The system is down if and only if the value of the fault tree is 1. To avoid trivialities, we assume that no inputs x, y with associated bags $c[n]$, $c[n']$, $n \neq n'$ feed the same gate. This is not a true restriction because, for $n' > n$ and an OR gate, x, y can be replaced by x and for an AND gate by y. Each state $a \in O$ has associated with it a bag of failed component classes $F(a)$. There is a single state, state o, with $F(o) = \emptyset$. Each transition of X has associated with it a failure bag $e \in E$, including the components that are failed when the transition is followed. Imperfect coverage can be modeled by introducing fictitious components that do not fail by themselves and to which uncovered faults are propagated. This point is illustrated in the following example.

Figure 1 shows the architecture of an example system, adapted from [6], which is used for illustration. The system consists of two memory modules MM_1 and MM_2, three s-identical CPU chips CPUC and two s-identical port chips PTC. In addition, to model

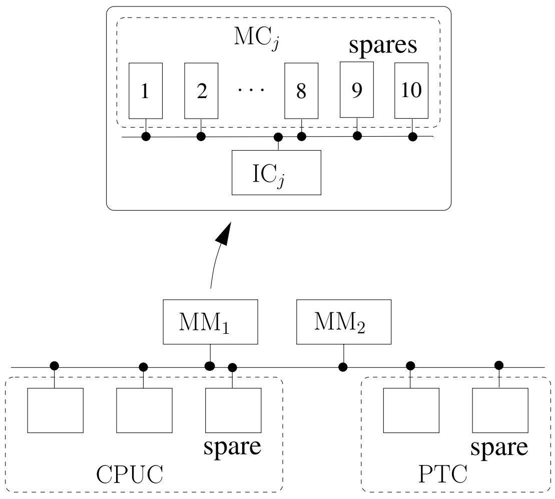

Figure 1: Architecture of the example system.

imperfect coverage, one fictitious component RMM$_j$, $j = 1, 2$ and two fictitious components RCM are added to the system. One CPUC and one PTC are spares. Each MM$_j$ has ten memory chips MC$_j$, two of which are spares, and one interface chip IC$_j$. The IC$_j$ and active MC$_j$, PTC, and CPUC fail, respectively, with rate λ_{IC_j}, λ_{MC_j}, λ_{PTC}, and λ_{CPUC}. Spare chips fail with rates $\nu \cdot \lambda_{\mathrm{MC}_j}$, $\nu \cdot \lambda_{\mathrm{PTC}}$, and $\nu \cdot \lambda_{\mathrm{CPUC}}$, where ν, $0 < \nu < 1$ is a dormancy factor. Recovery is hierarchical. A fault in a MC$_j$ is covered with probability C_{MC}. Failure of MM$_j$ and faults of CPUC and PTC are covered with probabilities C_{MM}, C_{CPUC} and C_{PTC}, respectively. To model imperfect coverage, an uncovered fault in a MC$_j$ is propagated to the fictitious component RMM$_j$, and an uncovered failure of MM$_j$, and an uncovered fault of a CPUC or a PTC are propagated to the two fictitious components RCM. The MM$_j$ is operational if at least eight MC$_j$, the IC$_j$ and the RMM$_j$ are unfailed. The system is operational if at least one memory module is operational, and at least two CPUC, one PTC and one RCM are unfailed.

Table 1 gives the failure bags of the example system and, for each failure bag e, a suitable upper bound $\lambda_{\mathrm{ub}}(e)$ expressed in terms of the above failure rates, coverage probabilities and the dormancy factor. Thus, *eg*, failure bag e_1 is the fault of a memory chip of the first memory module which is covered at memory module level, e_2 is the fault of that chip which is uncovered at memory module level and covered at system level, and e_3 is the uncovered fault of the chip. For the example system, $FC = \{1, 2, 3, 4\}$ and $f_1 = \lambda_{\mathrm{ub}}(e_1) + \lambda_{\mathrm{ub}}(e_4) + \lambda_{\mathrm{ub}}(e_6) + \lambda_{\mathrm{ub}}(e_9) + \lambda_{\mathrm{ub}}(e_{11}) + \lambda_{\mathrm{ub}}(e_{13})$, $f_2 = \lambda_{\mathrm{ub}}(e_2) + \lambda_{\mathrm{ub}}(e_7)$, $f_3 = \lambda_{\mathrm{ub}}(e_5) + \lambda_{\mathrm{ub}}(e_{10}) + \lambda_{\mathrm{ub}}(e_{12}) + \lambda_{\mathrm{ub}}(e_{14})$, and $f_4 = \lambda_{\mathrm{ub}}(e_3) + \lambda_{\mathrm{ub}}(e_8)$.

2.1 SC-BM

SC-BM computes $[\mathrm{ur}(t)]_{\mathrm{lb}}$ and $[\mathrm{ur}(t)]_{\mathrm{ub}}$ by solving the transient regime of the CTMC X'. The CTMC X' has state space $G \cup \{f\} \cup \{u_0, \ldots, u_{\widetilde{L}}\}$. Although other selections for G are possible, we assume that G includes all the up states of the model with up to K failed components. We also assume $\Pr\{X(0) \in G\} = 1$. The states u_d, $0 \le d \le \widetilde{L}$ pessimistically

Table 1: Failure bags of the example system and, for each failure bag e, a suitable upper bound $\lambda_{\mathrm{ub}}(e)$.

	description	$\lambda_{\mathrm{ub}}(e)$
e_1	$\mathrm{MC}_1[1]$	$(8+2\nu)\lambda_{\mathrm{MC}_1}C_{\mathrm{MC}}$
e_2	$\mathrm{MC}_1[1]\,\mathrm{RMM}_1[1]$	$(8+2\nu)\lambda_{\mathrm{MC}_1}(1-C_{\mathrm{MC}})C_{\mathrm{MM}}$
e_3	$\mathrm{MC}_1[1]\,\mathrm{RMM}_1[1]\,\mathrm{RCM}[2]$	$(8+2\nu)\lambda_{\mathrm{MC}_1}(1-C_{\mathrm{MC}})(1-C_{\mathrm{MM}})$
e_4	$\mathrm{IC}_1[1]$	$\lambda_{\mathrm{IC}_1}C_{\mathrm{MM}}$
e_5	$\mathrm{IC}_1[1]\,\mathrm{RCM}[2]$	$\lambda_{\mathrm{IC}_1}(1-C_{\mathrm{MM}})$
e_6	$\mathrm{MC}_2[1]$	$(8+2\nu)\lambda_{\mathrm{MC}_2}C_{\mathrm{MC}}$
e_7	$\mathrm{MC}_2[1]\,\mathrm{RMM}_2[1]$	$(8+2\nu)\lambda_{\mathrm{MC}_2}(1-C_{\mathrm{MC}})C_{\mathrm{MM}}$
e_8	$\mathrm{MC}_2[1]\,\mathrm{RMM}_2[1]\,\mathrm{RCM}[2]$	$(8+2\nu)\lambda_{\mathrm{MC}_2}(1-C_{\mathrm{MC}})(1-C_{\mathrm{MM}})$
e_9	$\mathrm{IC}_2[1]$	$\lambda_{\mathrm{IC}_2}C_{\mathrm{MM}}$
e_{10}	$\mathrm{IC}_2[1]\,\mathrm{RCM}[2]$	$\lambda_{\mathrm{IC}_2}(1-C_{\mathrm{MM}})$
e_{11}	$\mathrm{CPUC}[1]$	$(2+\nu)\lambda_{\mathrm{CPUC}}C_{\mathrm{CPUC}}$
e_{12}	$\mathrm{CPUC}[1]\,\mathrm{RCM}[2]$	$(2+\nu)\lambda_{\mathrm{CPUC}}(1-C_{\mathrm{CPUC}})$
e_{13}	$\mathrm{PTC}[1]$	$(1+\nu)\lambda_{\mathrm{PTC}}C_{\mathrm{PTC}}$
e_{14}	$\mathrm{PTC}[1]\,\mathrm{RCM}[2]$	$(1+\nu)\lambda_{\mathrm{PTC}}(1-C_{\mathrm{PTC}})$

approximate the behavior of X in U from the instant in which X enters U from G. The transition rates in X' from a to b, a, $b \in G$ and from a to f, $a \in G$ are as in X. The transition rates from states $a \in G$ to u_d, $1 \le d \le \widetilde{L}$ have values $\lambda_{a,\widetilde{U}_d}$, and for each $1 \le d \le \widetilde{L}$ and each $i \in FC$, there is a transition rate f_i from u_d to $u_{\max\{0,d-i\}}$. The initial probability distribution of X' in G is the same as the initial probability distribution of X in G. Section 3.1 shows that $\widetilde{L} = 2$ for the example system. Figure 2 shows the structure of X' for the example system. The bounds are:

$$\begin{aligned}
[\mathrm{ur}(t)]_{\mathrm{lb}} &= \Pr\{X'(t) = f\}\,,\\
[\mathrm{ur}(t)]_{\mathrm{ub}} &= \Pr\{X'(t) \in \{u_0, f\}\}\,.
\end{aligned} \tag{1}$$

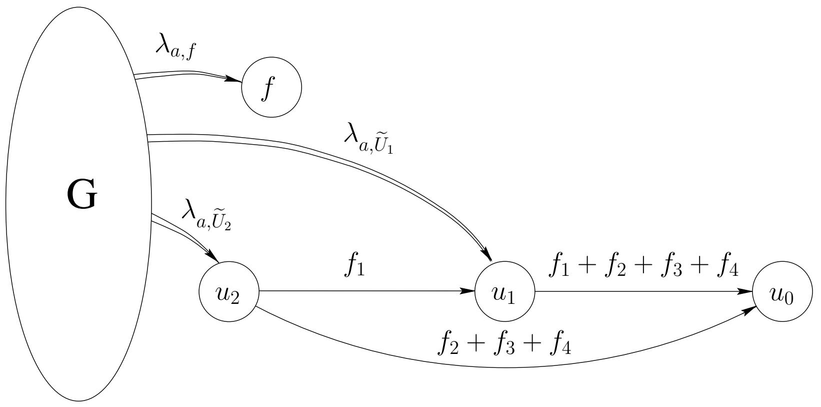

Figure 2: State transition diagram of X' for the example system ($\widetilde{L} = 2$, $FC = \{1, 2, 3, 4\}$).

The correctness of $[\mathrm{ur}(t)]_{\mathrm{lb}}$ is trivial. The correctness of $[\mathrm{ur}(t)]_{\mathrm{ub}}$ is proved in Section 2.2. Given the relationships between X' and the CTMC used in BM-1 [1], it is easy to conclude that SC-BM and BM-1 give exactly the same bounds when $\widetilde{d}(a) = d(a)$, $a \in U$. Otherwise, BM-1 gives, in general, tighter bounds than SC-BM.

2.2 Correctness of $[\mathrm{ur}(t)]_{\mathrm{ub}}$

This section establishes the correctness of $[\mathrm{ur}(t)]_{\mathrm{ub}}$ under the conditions $1 \le \widetilde{d}(a) \le d(a), a \in U$, and $\widetilde{d}(a) \le \widetilde{L}$.

The proof is constructed with the aid of the DTMC Y and Y'. Since [11] $X = \{X(t); t \ge 0\}$ is probabilistically identical to $\{Y_{N(t)}; t \ge 0\}$ and $X' = \{X'(t); t \ge 0\}$ is probabilistically identical to $\{Y'_{N(t)}; t \ge 0\}$, where $N = \{N(t); t \ge 0\}$ is a Poisson process with arrival rate Λ independent of both Y and Y':

$$\Pr\{X(t) = a\} = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Pr\{Y_n = a\}\,, \tag{2}$$

$$\Pr\{X'(t) = a\} = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Pr\{Y'_n = a\}\,. \tag{3}$$

Let

$$R'_m(d) = \Pr\{Y'_m = u_0 \,|\, Y'_0 = u_d\}\,, \tag{4}$$

$$R_m(a) = \Pr\{Y_m = f \,|\, Y_0 = a\}\,. \tag{5}$$

Then, we have the following two results. Lemma 1 is formally identical to Lemma 1 of [1].

Lemma 1 $R'_m(d)$, $m > 0$, $d > 0$ *is decreasing on* d.

Proof. From the structure of Y',

$$R'_m(0) = 1,\ m > 0\,, \tag{6}$$

$$R'_1(d) = \sum_{\substack{i \in FC \\ i \ge d}} \frac{f_i}{\Lambda},\ d > 0\,. \tag{7}$$

Also, for $m > 1$, $d > 0$,

$$R'_m(d) = \Big(1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i\Big) R'_{m-1}(d) + \sum_{i \in FC} \frac{f_i}{\Lambda} R'_{m-1}\big(\max\{0, d-i\}\big)\,. \tag{8}$$

The proof is by induction on m.

Base case ($m = 1$): We show that $R'_1(d) \le R'_1(d-1)$, $d > 0$. For $d = 1$, using (7), $\Lambda \ge \sum_{i \in FC} f_i$ and (6):

$$R'_1(1) = \sum_{i \in FC} \frac{f_i}{\Lambda} \le 1 = R'_1(0)\,.$$

For $d > 1$, using (7),

$$R'_1(d) = \sum_{\substack{i \in FC \\ i \geq d}} \frac{f_i}{\Lambda} \leq \sum_{\substack{i \in FC \\ i \geq d-1}} \frac{f_i}{\Lambda} = R'_1(d-1)\,.$$

Induction step: Let $m > 0$ and assume $R'_m(d)$, $d > 0$ is decreasing on d; it has to be shown that $R'_{m+1}(d) \leq R'_{m+1}(d-1)$, $d > 0$. For $d = 1$, using (8), $\Lambda \geq \sum_{i \in FC} f_i$, $R'_m(1) \leq 1$, and (6),

$$\begin{aligned} R'_{m+1}(1) &= \Big(1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i\Big) R'_m(1) + \sum_{i \in FC} \frac{f_i}{\Lambda} R'_m(0) \\ &\leq 1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i + \sum_{i \in FC} \frac{f_i}{\Lambda} = 1 = R'_{m+1}(0)\,. \end{aligned}$$

For $d > 1$, using (8), $\Lambda \geq \sum_{i \in FC} f_i$ and the induction hypothesis,

$$\begin{aligned} R'_{m+1}(d) &= \Big(1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i\Big) R'_m(d) + \sum_{i \in FC} \frac{f_i}{\Lambda} R'_m\big(\max\{0, d-i\}\big) \\ &\leq \Big(1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i\Big) R'_m(d-1) + \sum_{i \in FC} \frac{f_i}{\Lambda} R'_m\big(\max\{0, d-i-1\}\big) = R'_{m+1}(d-1)\,. \square \end{aligned}$$

Proposition 1 *$R_m(a) \leq R'_m(\min\{d, \widetilde{L}\})$, $a \in U_d$, $m > 0$, $d > 0$.*

Proof. Let $\lambda^i_{a,f}$ be the contribution to $\lambda_{a,f}$ associated with failure bags $e \in E_i$. Then, $\lambda^i_{a,f} \leq f_i$. If $i < d$, $\lambda_{a,f}$ does not have any contribution $\lambda^i_{a,f}$ because $a \in U_d$ and a failure bag $e \in E_i$ reduces the failure distance by at most i. Therefore,

$$R_1(a) = \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f}}{\Lambda}\,. \tag{9}$$

Let k be the number of failed components of the system in state a. Since f is absorbing, for $m > 1$,

$$\begin{aligned} R_m(a) = \Big(1 - \frac{\lambda_a}{\Lambda}\Big) R_{m-1}(a) &+ \sum_{\substack{i \in FC \\ i \geq d}} \Big[\frac{\lambda^i_{a,f}}{\Lambda} + \sum_{d'=1}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R_{m-1}(b)\Big] \\ &+ \sum_{\substack{i \in FC \\ i < d}} \sum_{d'=d-i}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R_{m-1}(b)\,. \end{aligned} \tag{10}$$

The proof is by induction on m.

Base case ($m = 1$): We show that $R_1(a) \leq R'_1(\min\{d, \widetilde{L}\})$, with $a \in U_d$, $d > 0$. Using (9), $\lambda^i_{a,f} \leq f_i$, and (7),

$$R_1(a) = \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f}}{\Lambda} \leq \sum_{\substack{i \in FC \\ i \geq d}} \frac{f_i}{\Lambda} \leq \sum_{\substack{i \in FC \\ i \geq \min\{d, \widetilde{L}\}}} \frac{f_i}{\Lambda} = R'_1(\min\{d, \widetilde{L}\})\,.$$

Induction step: Let $m > 0$ and assume $R_m(a) \le R'_m(\min\{d, \widetilde{L}\})$, $a \in U_d$, $d > 0$; it has to be shown that $R_{m+1}(a) \le R'_{m+1}(\min\{d, \widetilde{L}\})$, $a \in U_d$, $d > 0$. Using (10), $\Lambda \ge \lambda_a$ (which implies $1 - \lambda_a/\Lambda \ge 0$) and the induction hypothesis,

$$R_{m+1}(a) \le \Big(1 - \frac{\lambda_a}{\Lambda}\Big) R'_m(\min\{d, \widetilde{L}\}) + \sum_{\substack{i \in FC \\ i \ge d}} \Big[\frac{\lambda^i_{a,f}}{\Lambda} + \sum_{d'=1}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R'_m(\min\{d', \widetilde{L}\})\Big]$$

$$+ \sum_{\substack{i \in FC \\ i < d}} \sum_{d'=d-i}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R'_m(\min\{d', \widetilde{L}\})\,.$$

Using $R'_m(\min\{d', \widetilde{L}\}) \le 1$, Lemma 1 and that for $i \in FC$ and $d > 1$

$$\sum_{d'=d-j}^{d} \sum_{b \in U_{k+i,d'}} \lambda_{a,b} \le \lambda_{a,U^{k+i}}\,,\ 0 \le j < d\,,$$

we have

$$R_{m+1}(a) \le \Big(1 - \frac{\lambda_a}{\Lambda}\Big) R'_m(\min\{d, \widetilde{L}\}) + \sum_{\substack{i \in FC \\ i \ge d}} \Big[\frac{\lambda^i_{a,f}}{\Lambda} + \sum_{d'=1}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda}\Big]$$

$$+ \sum_{\substack{i \in FC \\ i < d}} \Big[R'_m(\min\{d-i, \widetilde{L}\}) \sum_{d'=d-i}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda}\Big]$$

$$\le \Big(1 - \frac{\lambda_a}{\Lambda}\Big) R'_m(\min\{d, \widetilde{L}\}) + \sum_{\substack{i \in FC \\ i \ge d}} \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda}$$

$$+ \sum_{\substack{i \in FC \\ i < d}} R'_m(\min\{d-i, \widetilde{L}\}) \frac{\lambda_{a,U^{k+i}}}{\Lambda}\,. \tag{11}$$

Since $\lambda_a = \lambda_{a,f} + \sum_{i \in FC} \lambda_{a,U^{k+i}}$, then

$$\frac{\lambda_a}{\Lambda} = \sum_{\substack{i \in FC \\ i \ge d}} \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda} + \sum_{\substack{i \in FC \\ i < d}} \frac{\lambda_{a,U^{k+i}}}{\Lambda}\,. \tag{12}$$

Combining (11) and (12),

$$R_{m+1}(a) \le R'_m(\min\{d, \widetilde{L}\}) + \sum_{\substack{i \in FC \\ i \ge d}} \big[1 - R'_m(\min\{d, \widetilde{L}\})\big] \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda}$$

$$+ \sum_{\substack{i \in FC \\ i < d}} \big[R'_m(\min\{d-i, \widetilde{L}\}) - R'_m(\min\{d, \widetilde{L}\})\big] \frac{\lambda_{a,U^{k+i}}}{\Lambda}\,.$$

For $i \ge d$, $\lambda^i_{a,f} + \lambda_{a,U^{k+i}} \le f_i$; for $i < d$, $\lambda_{a,U^{k+i}} < f_i$; then, using $R'_m(\min\{d, \widetilde{L}\}) \le 1$, Lemma 1 (which guarantees $R'_m(\min\{d-i,\widetilde{L}\}) - R'_m(\min\{d,\widetilde{L}\}) \geq 0$, $i > 0$), (6), and (8),

$$\begin{aligned}
R_{m+1}(a) &\leq R'_m(\min\{d,\widetilde{L}\}) + \sum_{\substack{i\in FC\\ i\geq d}} \left[1 - R'_m(\min\{d,\widetilde{L}\})\right]\frac{f_i}{\Lambda} \\
&\quad + \sum_{\substack{i\in FC\\ i<d}} \left[R'_m(\min\{d-i,\widetilde{L}\}) - R'_m(\min\{d,\widetilde{L}\})\right]\frac{f_i}{\Lambda} \\
&= \Big(1 - \frac{1}{\Lambda}\sum_{i\in FC} f_i\Big)R'_m(\min\{d,\widetilde{L}\}) + \sum_{i\in FC}\frac{f_i}{\Lambda}R'_m\big(\max\{0,\min\{d-i,\widetilde{L}\}\}\big) \\
&\leq \Big(1 - \frac{1}{\Lambda}\sum_{i\in FC} f_i\Big)R'_m(\min\{d,\widetilde{L}\}) + \sum_{i\in FC}\frac{f_i}{\Lambda}R'_m\big(\max\{0,\min\{d,\widetilde{L}\}-i\}\big) \\
&= R'_{m+1}(\min\{d,\widetilde{L}\})\,.\square
\end{aligned}$$

Using Lemma 1 and Proposition 1, it is possible to prove the following proposition, which establishes that Y' "upper bounds" Y.

Proposition 2 *Let $1 \leq \widetilde{d}(a) \leq d(a)$, $a \in U$, $\widetilde{d}(a) \leq \widetilde{L}$, and $\Pr\{X(0) \in G\} = 1$. Then, $\Pr\{Y_n = f\} \leq \Pr\{Y'_n \in \{u_0, f\}\}$, $n > 0$.*

Proof. The following notation is used in the proof:

$$\begin{aligned}
\psi(m,x) &\equiv Y_{m-1} \in G \wedge Y_m = x\,, \\
\psi'(m,x) &\equiv Y'_{m-1} \in G \wedge Y'_m = x\,.
\end{aligned}$$

Y can enter f through U or directly from G. Because f is absorbing, conditioning the entry of Y in f through U to the step in which Y enters U and the entry state, and using (5),

$$\begin{aligned}
\Pr\{Y_n = f\} &= \sum_{m=1}^{n-1}\sum_{a\in U}\Pr\{\psi(m,a)\}\Pr\{Y_n = f \mid Y_m = a\} + \sum_{m=1}^{n}\Pr\{\psi(m,f)\} \\
&= \sum_{m=1}^{n-1}\sum_{a\in U}\Pr\{\psi(m,a)\}R_{n-m}(a) + \sum_{m=1}^{n}\Pr\{\psi(m,f)\}\,.
\end{aligned}$$

Since, for $a \in U$, $1 \leq \widetilde{d}(a) \leq d(a)$ and $\widetilde{d}(a) \leq \widetilde{L}$, then U_d can be partitioned as:

$$U_d = \bigcup_{i=1}^{\min\{d,\widetilde{L}\}} \widetilde{U}_{d,i}\,.$$

Then, since $\widetilde{L} = \widetilde{d}(o) \leq d(o) = L$,

$$U = \bigcup_{d=1}^{L} U_d = \bigcup_{d=1}^{L}\bigcup_{i=1}^{\min\{d,\widetilde{L}\}} \widetilde{U}_{d,i} = \bigcup_{i=1}^{\widetilde{L}}\bigcup_{d=i}^{L}\widetilde{U}_{d,i}\,.$$

Using this partition of U and Proposition 1,

$$\Pr\{Y_n = f\} = \sum_{m=1}^{n-1}\sum_{i=1}^{\widetilde{L}}\sum_{d=i}^{L}\sum_{a\in\widetilde{U}_{d,i}}\Pr\{\psi(m,a)\}R_{n-m}(a) + \sum_{m=1}^{n}\Pr\{\psi(m,f)\}$$

$$\leq \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \sum_{d=i}^{L} \sum_{a \in \widetilde{U}_{d,i}} \Pr\{\psi(m,a)\} R'_{n-m}(\min\{d, \widetilde{L}\}) + \sum_{m=1}^{n} \Pr\{\psi(m,f)\} .$$

Using Lemma 1, $\widetilde{U}_i = \bigcup_{d=i}^{L} \widetilde{U}_{d,i}$, the relations between Y and Y', and (4):

$$\begin{aligned}
\Pr\{Y_n = f\} &\leq \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \sum_{d=i}^{L} \sum_{a \in \widetilde{U}_{d,i}} \Pr\{\psi(m,a)\} R'_{n-m}(i) + \sum_{m=1}^{n} \Pr\{\psi(m,f)\} \\
&= \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \sum_{a \in \widetilde{U}_i} \Pr\{\psi(m,a)\} R'_{n-m}(i) + \sum_{m=1}^{n} \Pr\{\psi(m,f)\} \\
&= \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \Pr\{\psi'(m,u_i)\} R'_{n-m}(i) + \sum_{m=1}^{n} \Pr\{\psi'(m,f)\} \\
&= \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \Pr\{\psi'(m,u_i)\} \Pr\{Y'_n = u_0 \,|\, Y'_m = u_i\} + \sum_{m=1}^{n} \Pr\{\psi'(m,f)\} \\
&= \Pr\{Y'_n = u_0\} + \Pr\{Y'_n = f\} = \Pr\{Y'_n \in \{u_0, f\}\} . \square
\end{aligned}$$

Proposition 1 allows us to prove the correctness of $[\mathrm{ur}(t)]_{\mathrm{ub}}$. This result is Theorem 1.

Theorem 1 *Let* $1 \leq \widetilde{d}(a) \leq d(a)$, $a \in U$, $\widetilde{d}(a) \leq \widetilde{L}$, *and* $\Pr\{X(0) \in G\} = 1$. *Then,* $\mathrm{ur}(t) \leq [\mathrm{ur}(t)]_{\mathrm{ub}}$.

Proof. Using (2) and $\Pr\{Y_0 = f\} = \Pr\{X(0) = f\} = 0$,

$$\mathrm{ur}(t) = \Pr\{X(t) = f\} = \sum_{n=1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Pr\{Y_n = f\} .$$

Using Proposition 2, $\Pr\{Y'_0 \in \{u_0, f\}\} = \Pr\{X'(0) \in \{u_0, f\}\} = 0$, (3), and (1),

$$\begin{aligned}
\mathrm{ur}(t) &\leq \sum_{n=1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Pr\{Y'_n \in \{u_0, f\}\} = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Pr\{Y'_n \in \{u_0, f\}\} \\
&= \Pr\{X'(t) \in \{u_0, f\}\} = [\mathrm{ur}(t)]_{\mathrm{ub}} . \square
\end{aligned}$$

3 Computation of Lower Bounds for Failure Distances and Model Generation in SC-BM

Definitions

- node (also referred to as event): gate or input of the fault tree
- related: two inputs x, y are related if $\mathrm{b}(x) = c[n]$ and $\mathrm{b}(y) = c[n']$, $n \neq n'$
- realized: an event x is realized if $\mathrm{val}(x) = 1$
- path: a sequence of nodes $x_1 \cdots x_k$ such that $x_i \in \mathrm{fo}(x_{i+1})$, $i = 1, \ldots, k-1$
- reachable node: a node x is reachable from node y if there exists a path from y to x

· module: $x \in I \cup P$ is a module if and only if every path $z \cdots y$, $z \notin \mathrm{Reach}(x)$, $y \in \mathrm{Reach}(x)$ contains node x, and for each input $y \in \mathrm{Support}(x)$, no related inputs exist outside $\mathrm{Support}(x)$

· independent: two nodes $x, y \in I \cup P$ are said to be independent if $\mathrm{Support}(x) \cap \mathrm{Support}(y) = \emptyset$ and there do not exist related inputs $z \in \mathrm{Support}(x)$, $t \in \mathrm{Support}(y)$

Notation

C	set of component classes
I	set of inputs (basic events) of the fault tree; each input x has associated with it a different bag, $\mathrm{b}(x)$, of the form $c[n]$, $c \in C$, $n \geq 1$
P	set of gates (complex events) of the fault tree
g_r	root gate (top event) of the fault tree
$\mathrm{type}(g)$	type of gate g: AND or OR
$\mathrm{fo}(x)$	fanout of (set of nodes fed by) node x
$\mathrm{fi}(x)$	fanin of (set of nodes that feed) node x
$\mathrm{val}(\cdot)$	value (1 or 0) of an input or a gate
$\mathrm{lev}(x)$	level of node x; if $x \in I$, $\mathrm{lev}(x) = 0$; if $x \in P$, $\mathrm{lev}(x) = 1 + \max_{y \in \mathrm{fi}(x)} \mathrm{lev}(y)$
$\mathrm{Reach}(x)$	node x plus set of nodes reachable from x
$\mathrm{Support}(x)$	$I \cap \mathrm{Reach}(x)$, $x \in I \cup P$
$S(F, x)$	$\left\lvert F \cap \left(\bigcup_{y \in \mathrm{Support}(x)} \mathrm{b}(y) \right) \right\rvert$, where F is a bag of failed component classes and x is a node
$d_{\mathrm{b}}(F, x)$	distance from F to x: minimum number of components which must fail in addition to those in the bag of component classes F to realize event x
$\widetilde{d}_{\mathrm{b}}(F, x)$	lower bound for $d_{\mathrm{b}}(F, x)$
$\widetilde{\eta}(e)$	$\widetilde{d}_{\mathrm{b}}(e, g_r)$: lower bound for the distance from a failure bag $e \in E$ to g_r

This section obtains the lower bounds $\widetilde{d}(a) = \widetilde{d}_{\mathrm{b}}(F(a), g_r)$ for the failure distance from states a used in SC-BM. The bounds are proved to fulfill the conditions which, according to Theorem 1, guarantee the correctness of $[\mathrm{ur}(t)]_{\mathrm{ub}}$ and the condition $\widetilde{d}(a) = 0$ if and only if $d(a) = 0$, which eases the generation of X' (see Section 3.4). The section also gives sufficient conditions for $d(a) = \widetilde{d}(a)$. Finally, it describes procedures that can be used to compute the lower bounds for the failure distances from the successors of a state, and describes how the CTMC X' is generated using those procedures.

3.1 Recursive Definition of Lower Bounds for Failure Distances

The lower bounds for failure distances $d(a) = d_{\mathrm{b}}(F(a), g_r)$ used in SC-BM are defined on the fault tree of the system using the concept of module, which generalizes to component classes the definition in [12, 13], in the sense that a module is a node such that the subtree hanging from it has that node as only entry point and every input of the subtree does not have related inputs outside the subtree. To determine which gates or inputs of the fault tree are modules, the algorithm LTA/DR of [13] is used with a small modification to deal with component classes: during the first depth-first left-most traversal of the fault tree (step no. 2 of the algorithm LTA/DR), visit to $x \in I$ implies simultaneous visit (*viz* with the same time-stamp as for x) to all inputs related to it.

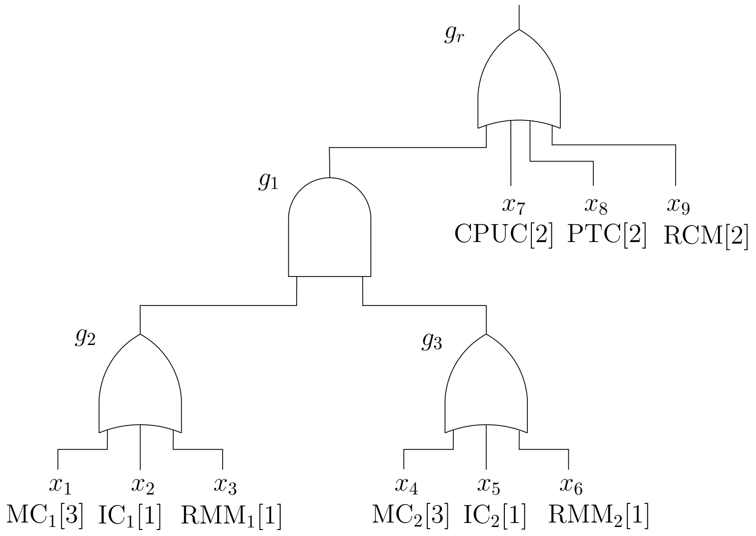

Figure 3: Fault tree of the example system. The bag associated with an input is given next to that input.

Given a bag of component classes F, $\widetilde{d}_{\mathrm{b}}(F,x)$, $x \in I \cup P$ is defined recursively by:

For $x \in I$, $\mathrm{b}(x) = c[n]$:

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \begin{cases} n & \text{if no } c[n'] \text{ is part of } F \\ \max\{0, n - n'\} & \text{if } c[n'] \text{ is part of } F \end{cases} . \tag{13}$$

For $x \in P$, $\mathrm{type}(x) = \mathrm{OR}$:

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \min_{y \in \mathrm{fi}(x)} \widetilde{d}_{\mathrm{b}}(F,y) . \tag{14}$$

For $x \in P$, $\mathrm{type}(x) = \mathrm{AND}$:

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\left\{ \sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F,y) \, , \, \max_{y \in C(x)} \left\{0, \widetilde{d}_{\mathrm{b}}(F,y)\right\} \right\} , \tag{15}$$

where $A(x) \equiv \{y \in \mathrm{fi}(x) | \, y \text{ is a module} \wedge |\mathrm{fo}(y)| = 1\}$, $B(x) \equiv \{y \in \mathrm{fi}(x) | \, y \text{ is a module} \wedge |\mathrm{fo}(y)| > 1 \vee y \text{ is not a module} \wedge y \in I\}$, $C(x) \equiv \{y \in \mathrm{fi}(x) | \, y \text{ is not a module} \wedge y \in P\}$.

Expressions (13)–(15) allow computation of $\widetilde{d}_{\mathrm{b}}(F, g_r)$ traversing the fault tree depth-first left-most, starting at g_r. Figure 3 shows the fault tree of the example system. Table 2 shows how $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$ is computed for that fault tree. All gates and inputs of the fault tree are modules and, therefore, (15) reduces to

$$\widetilde{d}_{\mathrm{b}}(F,x) = \sum_{y \in \mathrm{fi}(x)} \widetilde{d}_{\mathrm{b}}(F,y) .$$

3.2 Correctness of the Lower Bounds for Failure Distances and Related Results

Let F be a bag of failed component classes and $x \in I \cup P$. This section:

Table 2: Computation of $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$ traversing depth-first left-most the fault tree of the example system, starting at g_r and using (13)–(15).

step	1	2	3	4	5	6	7	8	9
node x	g_r	g_1	g_2	x_1	x_2	x_3	g_2	g_3	x_4
$\widetilde{d}_{\mathrm{b}}(\emptyset, x)$	–	–	–	3	1	1	1	–	3
step	10	11	12	13	14	15	16	17	
node x	x_5	x_6	g_3	g_1	x_7	x_8	x_9	g_r	
$\widetilde{d}_{\mathrm{b}}(\emptyset, x)$	1	1	1	2	2	2	2	2	

a) Proves that $0 \le \widetilde{d}_{\mathrm{b}}(F, x) \le d_{\mathrm{b}}(F, x)$; also, $\widetilde{d}_{\mathrm{b}}(F, x) = 0$ if and only if $d_{\mathrm{b}}(F, x) = 0$ (Lemma 2, Propositions 3 and 4, and Theorem 2).

b) Gives a sufficient condition for $\widetilde{d}_{\mathrm{b}}(F, x) = d_{\mathrm{b}}(F, x)$ (Theorem 3).

c) Gives a lower and upper bound for $\widetilde{d}_{\mathrm{b}}(F, x)$, $F = F' + F''$ in terms of $\widetilde{d}_{\mathrm{b}}(F', x)$ (Theorem 4).

We start by proving Lemma 2, Propositions 3 and 4, and Theorem 2.

Lemma 2 *Let $x, y \in I \cup P$, and let x be a module. Then, if $x, y \in I$, $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \emptyset$; otherwise, $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) \neq \emptyset$ if and only if either $x \in \mathrm{Reach}(y)$ or $y \in \mathrm{Reach}(x)$.*

Proof. If $x, y \in I$, the result is trivial. The three remaining cases that have to be dealt with are: $x \in I, y \in P$; $x \in P, y \in I$; and $x, y \in P$. The if implication for these cases is also trivial since $x \in \mathrm{Reach}(y)$ or $y \in \mathrm{Reach}(x)$ implies (recall that $x \in \mathrm{Reach}(x)$ and $y \in \mathrm{Reach}(y)$) $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) \neq \emptyset$. Regarding the only if implication, consider first the case $x \in I, y \in P$. $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \{x\} \cap \mathrm{Reach}(y) \neq \emptyset$ implies $x \in \mathrm{Reach}(y)$. The case $x \in P, y \in I$ is analogous: $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \mathrm{Reach}(x) \cap \{y\} \neq \emptyset$ implies $y \in \mathrm{Reach}(x)$. Now, consider the case $x, y \in P$. Assume $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) \neq \emptyset$ and neither $x \in \mathrm{Reach}(y)$ nor $y \in \mathrm{Reach}(x)$, and take $z \in \mathrm{Reach}(x) \cap \mathrm{Reach}(y)$. Then, since $x \notin \mathrm{Reach}(y)$, the path $y \ldots z$ does not contain event x, which contradicts the fact that x is a module because $y \notin \mathrm{Reach}(x)$ and $z \in \mathrm{Reach}(x)$. $\square$

Proposition 3 *Let $x, y \in I \cup P$, $z \in P$, $x, y \in \mathrm{fi}(z)$, and let x be a module. Then, x and y are independent if one of the following conditions holds:*

a) $y \in I$,

b) $y \in P$ and y is a module,

c) $y \in P$ and $|\mathrm{fo}(x)| = 1$.

Proof. If $\mathrm{Support}(x) \cap \mathrm{Support}(y) = \emptyset$, the inputs in $\mathrm{Support}(x)$ are not related to those in $\mathrm{Support}(y)$ because x is a module and, thus, x and y are independent. Therefore, it suffices to prove $\mathrm{Support}(x) \cap \mathrm{Support}(y) = \emptyset$ or, equivalently, $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \emptyset$.

Condition a: If $x \in I$, $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \emptyset$ by Lemma 2. If $x \in P$, $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) = \mathrm{Reach}(x) \cap \{y\} \neq \emptyset$ if and only if $y \in \mathrm{Reach}(x)$ by Lemma 2 ($x \in \mathrm{Reach}(y)$ is not possible). But $y \in \mathrm{fi}(z)$ implies the existence of the path zy not containing x, and, then, $y \in \mathrm{Reach}(x)$ would contradict the fact that x is a module.

Condition b: Assume $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) \neq \emptyset$. Using Lemma 2, either $x \in \mathrm{Reach}(y)$ or $y \in \mathrm{Reach}(x)$. $x \in \mathrm{Reach}(y)$ and the existence of the path zx contradicts the assumption that y is a module. If $x \in I$, $y \in \mathrm{Reach}(x)$ is not possible. If $x \in P$, $y \in \mathrm{Reach}(x)$ and the existence of the path zy contradicts the assumption that x is a module.

Condition c: Assume again $\mathrm{Reach}(x) \cap \mathrm{Reach}(y) \neq \emptyset$. From Lemma 2, either $y \in \mathrm{Reach}(x)$ or $x \in \mathrm{Reach}(y)$. If $x \in I$, $y \in \mathrm{Reach}(x)$ is not possible. If $x \in P$, $y \in \mathrm{Reach}(x)$ and the existence of the path zy contradicts the assumption that x is a module. $x \in \mathrm{Reach}(y)$ implies $|\mathrm{fo}(x)| > 1$ since $x \in \mathrm{fi}(z)$, in contradiction with $|\mathrm{fo}(x)| = 1$. □

Proposition 4 *Let $x \in P$, $\mathrm{type}(x) = \mathrm{AND}$. Let the partition $\mathrm{fi}(x) = A(x) \cup B(x) \cup C(x)$, where $A(x) \equiv \{y \in \mathrm{fi}(x) | \, y$ is a module $\wedge$ $|\mathrm{fo}(y)| = 1\}$, $B(x) \equiv \{y \in \mathrm{fi}(x) | \, y$ is a module $\wedge$ $|\mathrm{fo}(y)| > 1 \vee y$ is not a module $\wedge\, y \in I\}$ and $C(x) \equiv \{y \in \mathrm{fi}(x) | \, y$ is not a module $\wedge\, y \in P\}$. Then,*

a) all $y \in A(x)$ are mutually independent,

b) all $y \in A(x)$ are independent from all $y' \in B(x) \cup C(x)$, and

c) all $y \in B(x)$ are mutually independent.

Proof. To show part a, consider $y, y' \in A(x)$. Obviously, y is a module. If $y' \in I$, condition a of Proposition 3 is satisfied. If $y' \in P$, condition b of Proposition 3 is satisfied because y' is also a module. To show part b, consider first $y \in A(x)$, $y' \in B(x)$. Clearly, y is a module and y' is an input or a gate. If $y' \in I$, condition a of Proposition 3 is satisfied; if $y' \in P$, y' is a module and condition b of Proposition 3 is satisfied. The case $y \in A(x)$, $y' \in C(x)$ is dealt with as follows. We have that y is a module, $|\mathrm{fo}(y)| = 1$ and $y' \in P$. Therefore, condition c of Proposition 3 is fulfilled. Regarding part c, let $y, y' \in B(x)$. The following four cases have to be considered: 1) y, $y' \in I$, 2) $y \in I$, $y' \in P$, 3) $y \in P$, $y' \in I$, and 4) y, $y' \in P$. In case 1 the result holds trivially since, as assumed (see Section 2 on page 3), two inputs of a gate cannot be related. In case 2, y' must be a module and y, y' satisfy condition a of Proposition 3. Case 3 is symmetric to case 2. Finally, in case 4 both y and y' are modules and condition b of Proposition 3 is satisfied. □

Theorem 2 *Let F be a bag of component classes and $x \in I \cup P$. Then, the $\widetilde{d}_{\mathrm{b}}(F, x)$, defined recursively by (13)–(15), verify:*

a) $0 \leq \widetilde{d}_{\mathrm{b}}(F, x) \leq d_{\mathrm{b}}(F, x)$.

b) $\widetilde{d}_{\rm b}(F,x)=0$ *if and only if* $d_{\rm b}(F,x)=0$.

Proof. By complete induction over $\mathrm{lev}(x)$.

Base case ($\mathrm{lev}(x)=0$): In this case, $x\in I$, $\mathrm{b}(x)=c[n]$. From (13) and the definition of $d_{\rm b}(F,x)$, $0\leq \widetilde{d}_{\rm b}(F,x)=d_{\rm b}(F,x)$, showing both a and b.

Induction step: Assume that the theorem holds for all $x\in I\cup P$, $\mathrm{lev}(x)\leq l$, $l\geq 0$; it has to be shown that the theorem also holds for $x\in P$, $\mathrm{lev}(x)=l+1$ (x cannot be an input since $\mathrm{lev}(x)\geq 1$).

Part a: Consider first the case $\mathrm{type}(x)=\mathrm{OR}$. Using the definition of $d_{\rm b}(F,x)$, the fact that x is realized if and only if some $y\in \mathrm{fi}(x)$ is realized, the induction hypothesis for $y\in\mathrm{fi}(x)$ since $\mathrm{lev}(y)\leq l$, the monotonicity of $\min\{\cdot\}$, and (14),

$$d_{\rm b}(F,x)=\min_{y\in\mathrm{fi}(x)}\Big\{d_{\rm b}(F,y)\Big\}\geq \min_{y\in\mathrm{fi}(x)}\Big\{\widetilde{d}_{\rm b}(F,y)\Big\}=\widetilde{d}_{\rm b}(F,x)\geq 0\,.$$

Consider now the case $\mathrm{type}(x)=\mathrm{AND}$. Let the partition $\mathrm{fi}(x)=A(x)\cup B(x)\cup C(x)$ defined in Proposition 4 and let $t=\bigwedge_{y\in B(x)}y$ and $u=\bigwedge_{y\in C(x)}y$ (if some of the subsets $B(x)$ or $C(x)$ is empty, the corresponding logical variable is equal to the logical constant 1 and $\widetilde{d}_{\rm b}(F,1)=0$). Using the fact that x is realized if and only if all $y\in\mathrm{fi}(x)$ are realized, the definition of $d_{\rm b}(F,x)$ and parts a and b of Proposition 4,

$$d_{\rm b}(F,x)=\sum_{y\in A(x)}d_{\rm b}(F,y)+d_{\rm b}(F,t\wedge u)\geq \sum_{y\in A(x)}d_{\rm b}(F,y)+\max\Big\{d_{\rm b}(F,t),d_{\rm b}(F,u)\Big\}\,.$$

Using the definition of failure distance from a bag to an event, the fact that t is realized if and only if all $y\in B(x)$ are realized and u is realized if and only if all $y\in C(x)$ are realized, part c of Proposition 4, the induction hypothesis, the monotonicity of $\max\{\cdot\}$, and (15),

$$\begin{aligned}d_{\rm b}(F,x)&\geq \sum_{y\in A(x)}d_{\rm b}(F,y)+\max\Bigg\{\sum_{y\in B(x)}d_{\rm b}(F,y),\ \max_{y\in C(x)}\Big\{0,d_{\rm b}(F,y)\Big\}\Bigg\}\\&\geq \sum_{y\in A(x)}\widetilde{d}_{\rm b}(F,y)+\max\Bigg\{\sum_{y\in B(x)}\widetilde{d}_{\rm b}(F,y),\ \max_{y\in C(x)}\Big\{0,\widetilde{d}_{\rm b}(F,y)\Big\}\Bigg\}\\&=\widetilde{d}_{\rm b}(F,x)\geq 0\,,\end{aligned}$$

where the use of $\max_{y\in C(x)}\{0,\widetilde{d}_{\rm b}(F,y)\}$ instead of $\max_{y\in C(x)}\widetilde{d}_{\rm b}(F,y)$ allows to deal correctly with the case $C(x)=\emptyset$.

Part b: The if implication follows from $0\leq\widetilde{d}_{\rm b}(F,x)\leq d_{\rm b}(F,x)$. The only if implication is proved as follows. If $\mathrm{type}(x)=\mathrm{OR}$, $\widetilde{d}_{\rm b}(F,x)=0$ implies (14) the existence of $y\in\mathrm{fi}(x)$ with $\widetilde{d}_{\rm b}(F,y)=0$. From the induction hypothesis, this implies $d_{\rm b}(F,y)=0$, which leads to $d_{\rm b}(F,x)=0$ by the definition of distance from a bag to an event and the fact that x is realized if and only if some $y\in\mathrm{fi}(x)$ is realized. If $\mathrm{type}(x)=\mathrm{AND}$, using (15), $\widetilde{d}_{\rm b}(F,x)=0$ requires $\widetilde{d}_{\rm b}(F,y)=0$ for all $y\in\mathrm{fi}(x)$. As before, the induction hypothesis implies $d_{\rm b}(F,y)=0$ for all $y\in\mathrm{fi}(x)$, and, hence, $d_{\rm b}(F,x)=0$ by the definition of $d_{\rm b}(F,x)$ and the fact that x is realized if and only if all $y\in\mathrm{fi}(x)$ are realized. $\square$

Theorem 3 gives a sufficient condition for $\widetilde{d}_{\rm b}(F,x)=d_{\rm b}(F,x)$.

Theorem 3 *Let $x \in I \cup P$ and let F be a bag of component classes. Then, $\widetilde{d}_{\mathrm{b}}(F,x) = d_{\mathrm{b}}(F,x)$ if for every $z \in P$ with* $\mathrm{type}(z) = \mathrm{AND}$ *one of the following two conditions holds:*

a) Every $y \in \mathrm{fi}(z)$ is a module or an input.

b) There exists only one $u \in \mathrm{fi}(z)$ which is neither a module nor an input, and every $y \in \mathrm{fi}(z)$, $y \neq u$ is a module with $|\mathrm{fo}(y)| = 1$.

Proof. By complete induction over $\mathrm{lev}(x)$.

Base case ($\mathrm{lev}(x) = 0$): In this case, $x \in I$, $\mathrm{b}(x) = c[n]$. From the definition of $d_{\mathrm{b}}(F,x)$ and (13), $\widetilde{d}_{\mathrm{b}}(F,x) = d_{\mathrm{b}}(F,x)$.

Induction step: Assume that the theorem holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$; it has to be shown that the theorem also holds for $x \in P$, $\mathrm{lev}(x) = l+1$ (x cannot be an input since $\mathrm{lev}(x) \geq 1$). Consider first the case $\mathrm{type}(x) = \mathrm{OR}$. Using the definition of $d_{\mathrm{b}}(F,x)$, the fact that x is realized if and only if some $y \in \mathrm{fi}(x)$ is realized, the induction hypothesis, and (14),

$$d_{\mathrm{b}}(F,x) = \min_{y \in \mathrm{fi}(x)} \left\{ d_{\mathrm{b}}(F,y) \right\} = \min_{y \in \mathrm{fi}(x)} \left\{ \widetilde{d}_{\mathrm{b}}(F,y) \right\} = \widetilde{d}_{\mathrm{b}}(F,x)\,.$$

Consider next the case $\mathrm{type}(x) = \mathrm{AND}$. Let the partition $\mathrm{fi}(x) = A(x) \cup B(x) \cup C(x)$ defined in Proposition 4. If condition a of the theorem holds, $C(x) = \emptyset$. Then, using the fact that x is realized if and only if all $y \in \mathrm{fi}(x)$ are realized, the definition of $d_{\mathrm{b}}(F,x)$, the fact that $C(x) = \emptyset$, and that, according to Proposition 4, all $y \in \mathrm{fi}(x)$ are mutually independent,

$$d_{\mathrm{b}}(F,x) = \sum_{y \in A(x) \cup B(x)} d_{\mathrm{b}}(F,y)\,.$$

Using the induction hypothesis, the fact that $C(x) = \emptyset$ and (15),

$$\begin{aligned} d_{\mathrm{b}}(F,x) &= \sum_{y \in A(x) \cup B(x)} \widetilde{d}_{\mathrm{b}}(F,y) \\ &= \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\Big\{ \sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F,y),\, \max_{y \in C(x)} \left\{0, \widetilde{d}_{\mathrm{b}}(F,y)\right\} \Big\} \\ &= \widetilde{d}_{\mathrm{b}}(F,x)\,. \end{aligned}$$

Assume now that condition b of the theorem holds. Then, $B(x) = \emptyset$ and $C(x) = \{u\}$. Using the fact that x is realized if and only if all $y \in \mathrm{fi}(x)$ are realized, the definition of $d_{\mathrm{b}}(F,x)$, the fact that $B(x) = \emptyset$ and $C(x) = \{u\}$, and that, according to parts a and b of Proposition 4, all $y \in \mathrm{fi}(x)$ are mutually independent,

$$d_{\mathrm{b}}(F,x) = \sum_{y \in A(x)} d_{\mathrm{b}}(F,y) + d_{\mathrm{b}}(F,u).$$

Finally, using the induction hypothesis, the fact that $B(x) = \emptyset$ and $C(x) = \{u\}$, and (15):

$$\begin{aligned} d_{\mathrm{b}}(F,x) &= \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \widetilde{d}_{\mathrm{b}}(F,u) \\ &= \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\Big\{ \sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F,y),\, \max_{y \in C(x)} \left\{0, \widetilde{d}_{\mathrm{b}}(F,y)\right\} \Big\} \\ &= \widetilde{d}_{\mathrm{b}}(F,x)\,. \square \end{aligned}$$

Theorem 4 gives a lower and an upper bound for $\widetilde{d}_{\mathrm{b}}(F,x)$, $F = F' + F''$ in terms of $\widetilde{d}_{\mathrm{b}}(F',x)$.

Theorem 4 *Let F, F', F'' be bags of component classes with $F = F'+F''$ and let $x \in I \cup P$. Then,*

$$\widetilde{d}_{\mathrm{b}}(F',x) \geq \widetilde{d}_{\mathrm{b}}(F,x) \geq \widetilde{d}_{\mathrm{b}}(F',x) - S(F'',x)\,.$$

Proof. By complete induction over $\mathrm{lev}(x)$.

Base case ($\mathrm{lev}(x) = 0$): Since $\mathrm{lev}(x) = 0$, then $x \in I$, $\mathrm{b}(x) = c[n]$. The following three cases cover all possibilities: a) $\widetilde{d}_{\mathrm{b}}(F',x) = 0$, b) $\widetilde{d}_{\mathrm{b}}(F',x) > 0$ and no $c[n']$ is part of F'', and c) $\widetilde{d}_{\mathrm{b}}(F',x) > 0$ and there exists $c[n']$ part of F''. In case a, there exists (13) $c[n']$ part of F' with $n' \geq n$. Since $F = F' + F''$, then $c[n'']$, $n'' \geq n'$ part of F exists. Therefore, $\widetilde{d}_{\mathrm{b}}(F,x) = 0 = \widetilde{d}_{\mathrm{b}}(F',x)$ showing both inequalities because $S(F'',x) \geq 0$. In case b, clearly $\widetilde{d}_{\mathrm{b}}(F,x) = \widetilde{d}_{\mathrm{b}}(F',x)$ and both inequalities are also shown. In case c, let $\widetilde{d}_{\mathrm{b}}(F',x) = n''$, $0 < n'' \leq n$. Since $\mathrm{Support}(x) = \{x\}$ and $\mathrm{b}(x) = c[n]$, it follows that $S(F'',x) = \min\{n,n'\}$. Let $n''' = n - n''$. Then (13), if $n''' > 0$ we have that $c[n''']$ is part of F' and if $n''' = 0$, no $c[m]$ is part of F'. Since $F = F' + F''$, then $c[n''' + n']$ is part of F. Therefore, using (13),

$$\begin{aligned}\widetilde{d}_{\mathrm{b}}(F',x) &= n'' = n - n''' \geq \max\bigl\{0, n - n''' - n'\bigr\} \\ &= \widetilde{d}_{\mathrm{b}}(F,x) \geq n - n''' - \min\{n,n'\} = n'' - \min\{n,n'\} = \widetilde{d}_{\mathrm{b}}(F',x) - S(F'',x)\,.\end{aligned}$$

Induction step: Assume that the theorem holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$; it has to be shown that the theorem also holds for $x \in P$, $\mathrm{lev}(x) = l+1$ (x cannot be an input since $\mathrm{lev}(x) \geq 1$). Since $\mathrm{Support}(x) = \bigcup_{y \in \mathrm{fi}(x)} \mathrm{Support}(y)$, it is immediate to show that for any $x \in P$,

$$S(F'',x) \geq S(F'',y),\ y \in \mathrm{fi}(x)\,, \tag{16}$$

$$S(F'',x) \geq \max_{y \in \mathrm{fi}(x)} \bigl\{S(F'',y)\bigr\}\,. \tag{17}$$

Consider first the case $\mathrm{type}(x) = \mathrm{OR}$. Using (14), the induction hypothesis, the monotonicity of $\min\{\cdot\}$, and (16),

$$\begin{aligned}\widetilde{d}_{\mathrm{b}}(F',x) &= \min_{y \in \mathrm{fi}(x)} \Bigl\{\widetilde{d}_{\mathrm{b}}(F',y)\Bigr\} \geq \min_{y \in \mathrm{fi}(x)} \Bigl\{\widetilde{d}_{\mathrm{b}}(F,y)\Bigr\} \\ &= \widetilde{d}_{\mathrm{b}}(F,x) \geq \min_{y \in \mathrm{fi}(x)} \Bigl\{\widetilde{d}_{\mathrm{b}}(F',y) - S(F'',y)\Bigr\} \geq \min_{y \in \mathrm{fi}(x)} \Bigl\{\widetilde{d}_{\mathrm{b}}(F',y) - S(F'',x)\Bigr\} \\ &= \min_{y \in \mathrm{fi}(x)} \Bigl\{\widetilde{d}_{\mathrm{b}}(F',y)\Bigr\} - S(F'',x) = \widetilde{d}_{\mathrm{b}}(F',x) - S(F'',x)\,.\end{aligned}$$

Assume now that $\mathrm{type}(x) = \mathrm{AND}$. Let the partition $\mathrm{fi}(x) = A(x) \cup B(x) \cup C(x)$ defined in Proposition 4 and let $t = \bigwedge_{y \in A(x)} y$, $u = \bigwedge_{y \in B(x)} y$, and $v = \bigwedge_{y \in C(x)} y$ (if some of the subsets into which $\mathrm{fi}(x)$ is partitioned is empty, the corresponding logical variable is equal to the logical constant 1 and $S(F'',1) = 0$). Let $\alpha = S(F'',x)$, $\beta = \sum_{y \in A(x)} S(F'',y)$, $\gamma = \sum_{y \in B(x)} S(F'',y)$, and $\delta = \max_{y \in C(x)}\bigl\{S(F'',y)\bigr\}$. From part b of Proposition 4, t

and $u \wedge v$ are independent, which, taking into account the definition of $S(F'', \cdot)$, implies $S(F'', x) = S(F'', t) + S(F'', u \wedge v)$. Then, using (16) and (17),

$$\begin{aligned} S(F'', x) &= S(F'', t) + S(F'', u \wedge v) \geq S(F'', t) + \max\bigl\{S(F'', u), S(F'', v)\bigr\} \\ &\geq S(F'', t) + \max\Bigl\{S(F'', u), \max_{y \in C(x)} \bigl\{S(F'', y)\bigr\}\Bigr\} = S(F'', t) + \max\bigl\{S(F'', u), \delta\bigr\}\,. \end{aligned}$$

¿From parts a and c of Proposition 4, $S(F'', t) = \beta$ and $S(F'', u) = \gamma$. Then, using the definition of α the last inequality becomes

$$\alpha \geq \beta + \max\bigl\{\gamma, \delta\bigr\}\,. \tag{18}$$

Using (15), the induction hypothesis, the definition of α, β, γ and δ, the monotonicity of $\max\{\cdot\}$, and (18),

$$\begin{aligned}
&\widetilde{d}_{\mathrm{b}}(F', x) = \\
&\quad \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y)\Bigr\}\Bigr\} \\
&\geq \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F, y) + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F, y), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F, y)\Bigr\}\Bigr\} = \widetilde{d}_{\mathrm{b}}(F, x) \\
&\geq \sum_{y \in A(x)} \Bigl(\widetilde{d}_{\mathrm{b}}(F', y) - S(F'', y)\Bigr) \\
&\quad + \max\Bigl\{\sum_{y \in B(x)} \Bigl(\widetilde{d}_{\mathrm{b}}(F', y) - S(F'', y)\Bigr), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y) - S(F'', y)\Bigr\}\Bigr\} \\
&= \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \sum_{y \in A(x)} S(F'', y) \\
&\quad + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \sum_{y \in B(x)} S(F'', y), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y) - S(F'', y)\Bigr\}\Bigr\} \\
&\geq \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \sum_{y \in A(x)} S(F'', y) \\
&\quad + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \sum_{y \in B(x)} S(F'', y), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y) - \max_{y \in C(x)} \bigl\{S(F'', y)\bigr\}\Bigr\}\Bigr\} \\
&\geq \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \beta + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \gamma, \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y)\Bigr\} - \delta\Bigr\} \\
&\geq \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \beta \\
&\quad + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y) - \max\{\gamma, \delta\}, \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y)\Bigr\} - \max\{\gamma, \delta\}\Bigr\} \\
&= \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F', y) + \max\Bigl\{\sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F', y), \max_{y \in C(x)} \Bigl\{0, \widetilde{d}_{\mathrm{b}}(F', y)\Bigr\}\Bigr\} - \bigl(\beta + \max\{\gamma, \delta\}\bigr) \\
&= \widetilde{d}_{\mathrm{b}}(F', x) - \bigl(\beta + \max\{\gamma, \delta\}\bigr) \geq \widetilde{d}_{\mathrm{b}}(F', x) - \alpha = \widetilde{d}_{\mathrm{b}}(F', x) - S(F'', x)\,. \Box
\end{aligned}$$

Let a be a state. With $F = F(a)$ and $x = g_r$, part b of Theorem 2 implies $\widetilde{d}(a) = 0$ if and only if $d(a) = 0$, part a implies $0 \leq \widetilde{d}(a) \leq d(a)$, and both results imply $1 \leq \widetilde{d}(a) \leq d(a)$ for $a \in U$. In addition, taking $x = g_r$, $F' = \emptyset$, $F'' = F(a)$ and $F = F' + F'' = F(a)$, the left inequality of Theorem 4 states that $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r) \geq \widetilde{d}_{\mathrm{b}}(F(a), g_r) = \widetilde{d}(a)$. Thus, the derived $\widetilde{d}(a)$ satisfies the requirements of Theorem 1, guaranteeing that the upper bound $[\mathrm{ur}(t)]_{\mathrm{ub}}$ obtained by SC-BM is correct. In addition, we have $\widetilde{d}(a) = 0$ if and only if $d(a) = 0$. With $F = F(a)$ and $x = g_r$, Theorem 3 gives a sufficient condition for $\widetilde{d}(a) = d(a)$ which can be checked inexpensively. Finally, when $L = 1$, Theorem 2 with $F = F(a)$ and $x = g_r$ implies $\widetilde{d}(a) = d(a)$. Thus, when the condition of Theorem 3 is satisfied or $L = 1$, $\widetilde{d}(a) = d(a)$ and SC-BM gives the same bounds as BM-1.

3.3 Computation of the Lower Bounds for Failure Distances

Section 3.4 shows that the generation of the CTMC X' can be done knowing $\widetilde{d}(b)$ for all successors b of each state $a \in G$. Each $\widetilde{d}(b)$ could be computed from $F(b)$ by traversing the fault tree as described in Section 3.1. However, that procedure is expensive if the fault tree is large. This section describes a typically much more efficient procedure, *comp_all_d*, which can compute $\widetilde{d}(b)$ for all successors b of a state.

The procedure *comp_all_d* is built on top of three procedures named *update_d*, *comp_d* and *restore_d*.

Each node x of the fault tree holds a distance variable $\mathrm{dv}(x)$ properly initialized. The procedure *update_d* takes as inputs a bag of component classes F, a positive integer ub and a stack CS, and processes the fault tree as follows. For each $c[n]$ that is part of F, the procedure makes $\mathrm{dv}(x) = \max\{0, \mathrm{dv}(x) - n\}$ for each input x, $\mathrm{b}(x) = c[n']$ and follows a recursive processing from x. The recursive processing from a node z of the fault tree involves computing for each $y \in \mathrm{fo}(z)$ a potential new value for $\mathrm{dv}(y)$ using (14) for OR gates and (15) for AND gates with $\mathrm{dv}(t)$, $t \in \mathrm{fi}(y)$ instead of $\widetilde{d}_{\mathrm{b}}(F, t)$. If the potential new value for $\mathrm{dv}(y)$ is $< ub$ and smaller than the current value of $\mathrm{dv}(y)$, the variable $\mathrm{dv}(y)$ is updated and the processing of the fault tree continues recursively from node y. When a distance variable is updated, the corresponding node and the old value of the variable are put in CS.

The procedure *comp_d* takes as inputs a bag of component classes F, two non-negative integers lb and ub, with $lb \leq ub$, and a stack CS. If $lb = ub$, the procedure returns lb without doing anything else. If $lb < ub$, the procedure makes the call *update_d*(F, ub, CS) and returns $\min\{\mathrm{dv}(g_r), ub\}$.

The procedure *restore_d* takes as input a stack CS and simply restores the distance variable of the nodes kept in CS to their old values. After the call to *restore_d*, CS becomes empty.

The procedure *comp_all_d* takes as inputs a bag of failed component classes F, the lower bound $\widetilde{d} = \widetilde{d}_{\mathrm{b}}(F, g_r)$, a subset E' of the set of failure bags E, and $\widetilde{\eta}(e)$, $e \in E'$, and computes $\widetilde{d}_{\mathrm{b}}(F + e, g_r)$ for each $e \in E'$. For the procedure to work properly, $\widetilde{d}$ must be > 0 and the distance variable $\mathrm{dv}(x)$ of each node x of the fault tree must have been initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$. The procedure is:

```
1. Let $CS$ and $CS'$ be empty stacks.
2. $comp\_d(F, 0, \widetilde{d}, CS)$
3. for (each $e \in E'$) {
4.     $s = \max\{0, \widetilde{d} - |e|, \widetilde{\eta}(e) - |F|\}$
5.     $t = \min\{\widetilde{d}, \widetilde{\eta}(e)\}$
6.     $\widetilde{d}_{\mathrm{b}}(F + e, g_r) = comp\_d(e, s, t, CS')$
7.     $restore\_d(CS')$
   }
8. $restore\_d(CS)$
```

After calling the procedure, the distance variable $\mathrm{dv}(x)$ of each node x of the fault tree holds the value $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$.

Next, the procedure *comp_all_d* is illustrated using the example system of Section 2, whose fault tree is given in Figure 3. The procedure is illustrated for the inputs $F = \mathrm{MC}_1[1]$, $\widetilde{d}_{\mathrm{b}}(F, g_r) = 2$ and $E' = \{e_4, e_7\}$ with $e_4 = \mathrm{IC}_1[1]$, $e_7 = \mathrm{MC}_2[1]\mathrm{RMM}_2[1]$, $\widetilde{\eta}(e_4) = 1$, and $\widetilde{\eta}(e_7) = 1$ (see Table 1). The procedure begins by creating empty stacks CS and CS' and making the call $comp\_d(F = \mathrm{MC}_1[1], lb = 0, ub = 2, CS)$. Since $lb < ub$, that call to *comp_d* results in the call $update\_d(F = \mathrm{MC}_1[1], ub = 2, CS)$. Figure 4 illustrates the processing of the fault tree during that call. The distance variable $\mathrm{dv}(x)$ of each node x is initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ before the call. The only input having associated with it a bag related to $\mathrm{MC}_1[1]$ is x_1. Using (13), $\mathrm{dv}(x_1)$ is updated to $\max\{0, 3-1\} = 2$ and the pair $(x_1, 3)$ is stored in CS. The change in $\mathrm{dv}(x_1)$ is not propagated up the fault tree because the new value of $\mathrm{dv}(x_1)$ is not smaller than $ub = 2$. Next, the procedure processes the failure bag $e_4 = \mathrm{IC}_1[1]$. The values of the variables s and t are $s = 1$ and $t = 1$; the procedure continues by making the call $comp\_d(F = \mathrm{IC}_1[1], lb = 1, ub = 1, CS')$ followed by the call $restore\_d(CS')$. The first call returns $lb = 1$ without processing the fault tree because $lb = ub$. This results in $\widetilde{d}_{\mathrm{b}}(\mathrm{MC}_1[1]\mathrm{IC}_1[1], g_r) = 1$. The second call does not restore any distance variable because $CS' = \emptyset$. The procedure continues by processing the failure bag $e_7 = \mathrm{MC}_2[1]\mathrm{RMM}_2[1]$. The s, t become $s = 0$ and $t = 1$ and the procedure continues by making the call $comp\_d(F = \mathrm{MC}_2[1]\mathrm{RMM}_2[1], lb = 0, ub = 1, CS')$. Since $lb < ub$, the call to *comp_d* results in the call $update\_d(F = \mathrm{MC}_2[1]\mathrm{RMM}_2[1], ub = 1, CS')$. Figure 5 illustrates the processing of the fault tree during that call. The only inputs having associated with them a bag related to some part of $\mathrm{MC}_2[1]\mathrm{RMM}_2[1]$ are x_4 and x_6. Using (13), $\mathrm{dv}(x_4)$ is updated to 2 and the pair $(x_4, 3)$ is stored in CS'. The change in $\mathrm{dv}(x_4)$ is not propagated up the fault tree because the new value of $\mathrm{dv}(x_4)$ is not smaller than $ub = 1$. Next, the input x_6 is dealt with. Using (13), $\mathrm{dv}(x_6)$ is updated to 0 and the pair $(x_6, 1)$ is stored in CS'. Since the new value of $\mathrm{dv}(x_6)$ is smaller than $ub = 1$, that change is propagated up the fault tree to the gate g_3. Using (14), the potential new value for $\mathrm{dv}(g_3)$ is 0. Since that potential new value is smaller than both the old value of the variable and $ub = 1$, then $\mathrm{dv}(g_3)$ is updated, the pair $(g_3, 1)$ is stored in CS' and the change

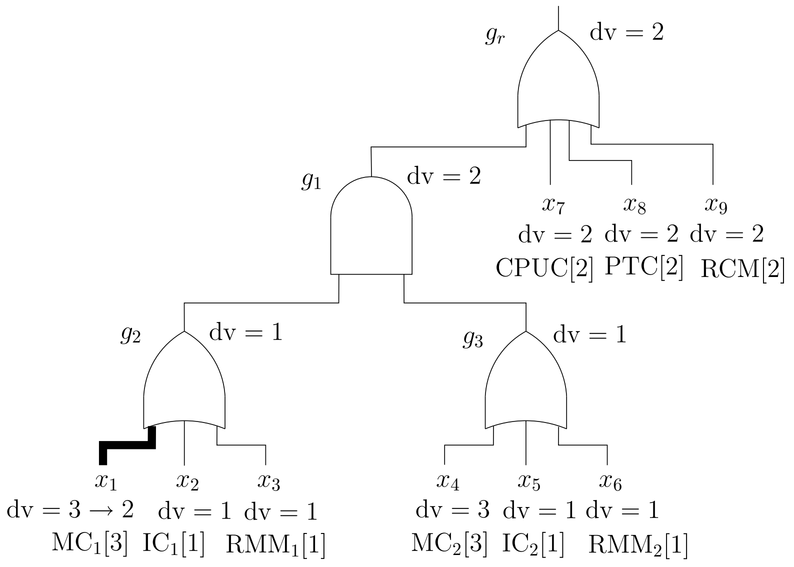

Figure 4: Processing of the fault tree of the example system during the call *update_d*($F = \mathrm{MC}_1[1], ub = 2, CS$). The distance variable values are given next to each node. The '$\rightarrow$' shows their changes.

is propagated up the fault tree to gate g_1. Using (15), the potential new value for $\mathrm{dv}(g_1)$ is 1, which is not smaller than $ub = 1$. Then, $\mathrm{dv}(g_1)$ is not updated and because there is no pending processing of the fault tree, the call *update_d*($F = \mathrm{MC}_2[1]\mathrm{RMM}_2[1], ub = 1, CS'$) ends and the call *comp_d*($F = \mathrm{MC}_2[1]\mathrm{RMM}_2[1], lb = 0, ub = 1, CS'$) returns the value $\min\{\mathrm{dv}(g_r), ub\} = \min\{2,1\} = 1$. This results in $\widetilde{d}_{\mathrm{b}}(\mathrm{MC}_1[1]\mathrm{MC}_2[1]\mathrm{RMM}_2[1], g_r) = 1$. The procedure finishes by making the calls *restore_d*(CS') and *restore_d*(CS), which leave the fault tree with the distance variable $\mathrm{dv}(x)$ of each node x equal to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$.

The remaining of this section proves the correctness of the procedures *comp_d* (which will be used to justify the algorithm for the generation of X' in SC-BM described in Section 3.4) and *comp_all_d*. The proof consists of Propositions 5, 6, 7, and 8, and Theorems 5 and 6.

Proposition 5 *Let F be a bag of component classes, ub a positive integer and CS a stack. Let the distance variable* $\mathrm{dv}(x)$ *of each node x of the fault tree be initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$. Then, after the call* update_d(F, ub, CS)*, for each $x \in I \cup P$ we have $\widetilde{d}_{\mathrm{b}}(F, x) = \mathrm{dv}(x)$ if $\mathrm{dv}(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$ if $\mathrm{dv}(x) \geq ub$.*

Proof. We start by noting that, during the call, distance variables $\mathrm{dv}(x)$ can only decrease. This is trivially true for $x \in I$ and follows for $x \in P$ by complete induction on $\mathrm{lev}(x)$ noting that (14) and (15) with $\mathrm{dv}(y)$ instead of $\widetilde{d}_{\mathrm{b}}(F, y)$ are monotonic. The proof is by complete induction on $\mathrm{lev}(x)$, $x \in I \cup P$.

Base case ($\mathrm{lev}(x) = 0$): Since $\mathrm{lev}(x) = 0$, then $x \in I$. Given the way in which the call processes the inputs, it is clear that, after the call, $\mathrm{dv}(x) = \widetilde{d}_{\mathrm{b}}(F, x)$, showing that $\widetilde{d}_{\mathrm{b}}(F, x) = \mathrm{dv}(x)$ if $\mathrm{dv}(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$ if $\mathrm{dv}(x) \geq ub$.

Induction step: Assume that the proposition holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$;

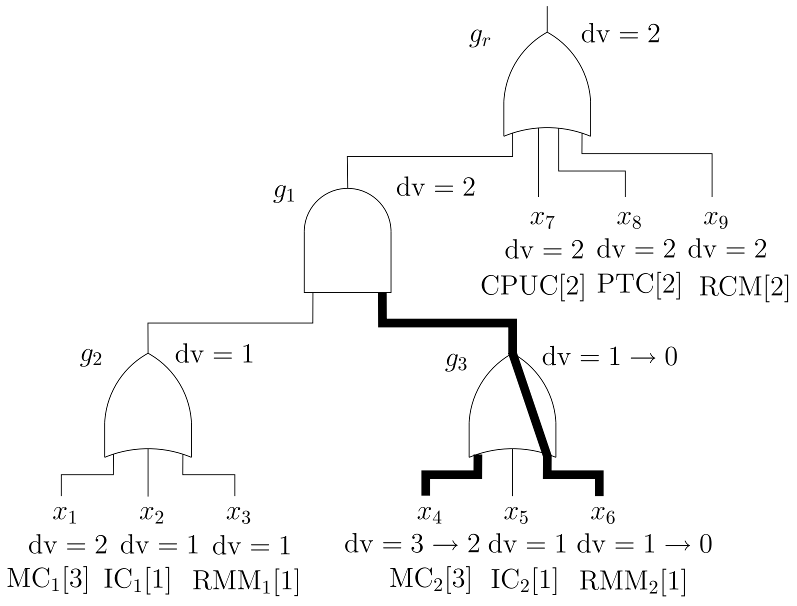

Figure 5: Processing of the fault tree of the example system during the call *update_d*($F = \mathrm{MC}_2[1]\,\mathrm{RMM}_2[1], ub = 1, CS'$). The distance variable values are given next to each node. The '→' shows their changes.

it has to be shown that the proposition also holds for all $x \in P$, $\mathrm{lev}(x) = l+1$ ($x \notin I$ because $l+1 > 0$). Let $\mathrm{dv}^{(n)}(x)$, $x \in I \cup P$ denote the value of $\mathrm{dv}(x)$ after performing the n^{th} distance variable update. (the $\mathrm{dv}^{(0)}(x)$ is the value of $\mathrm{dv}(x)$ at the beginning of the call.) Let N be the number of distance variable updates during the call. Then, it has to be shown that $\widetilde{d}_{\mathrm{b}}(F,x) = \mathrm{dv}^{(N)}(x)$ if $\mathrm{dv}^{(N)}(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F,x) \geq ub$ if $\mathrm{dv}^{(N)}(x) \geq ub$ for all $x \in P$, $\mathrm{lev}(x) = l+1$ assuming $\widetilde{d}_{\mathrm{b}}(F,x) = \mathrm{dv}^{(N)}(x)$ if $\mathrm{dv}^{(N)}(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F,x) \geq ub$ if $\mathrm{dv}^{(N)}(x) \geq ub$ for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$.

Let $x \in P$, $\mathrm{lev}(x) = l+1$ and consider the partition $\mathrm{fi}(x) = \alpha \cup \beta$ with $\alpha = \{y \in \mathrm{fi}(x) \,|\, \mathrm{dv}^{(N)}(y) < ub\}$ and $\beta = \mathrm{fi}(x) - \alpha = \{y \in \mathrm{fi}(x) \,|\, \mathrm{dv}^{(N)}(y) \geq ub\}$. Since distance variables can only decrease

$$\mathrm{dv}^{(n')}(y) \leq \mathrm{dv}^{(n)}(y), \quad n' \geq n, y \in \mathrm{fi}(x)\,. \tag{19}$$

Then,

$$\mathrm{dv}^{(n)}(y) \geq \mathrm{dv}^{(N)}(y) \geq ub, \quad n = 0,1,\ldots,N, y \in \beta\,. \tag{20}$$

Also, using the induction hypothesis,

$$\widetilde{d}_{\mathrm{b}}(F,y) = \mathrm{dv}^{(N)}(y) < ub, \quad y \in \alpha \tag{21}$$

and

$$\widetilde{d}_{\mathrm{b}}(F,y) \geq ub, \quad y \in \beta\,. \tag{22}$$

We consider four cases:

Case 1: $\mathrm{type}(x) = \mathrm{OR}$ and $\alpha = \emptyset$. We show $\widetilde{d}_{\mathrm{b}}(F,x) \geq ub$ and $\mathrm{dv}^{(N)}(x) \geq ub$. We have $\mathrm{fi}(x) = \beta$. Then, using (14) and (22), $\widetilde{d}_{\mathrm{b}}(F,x) = \min_{y\in\beta} \widetilde{d}_{\mathrm{b}}(F,y) \geq ub$. Also, $\mathrm{dv}(x)$ is not updated during the call because (20) $\min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(n)}(y) = \min_{y\in\beta} \mathrm{dv}^{(n)}(y) \geq ub$. Therefore, $\mathrm{dv}^{(N)}(x) = \mathrm{dv}^{(0)}(x) = \min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(0)}(y) = \min_{y\in\beta} \mathrm{dv}^{(0)}(y)$, which (20) is $\geq ub$.

Case 2: $\mathrm{type}(x) = \mathrm{OR}$ and $\alpha \neq \emptyset$. We show $\mathrm{dv}^{(N)}(x) = \widetilde{d}_{\mathrm{b}}(F,x) < ub$. We start by showing $\mathrm{dv}^{(N)}(x) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$. We consider two cases: a) $\mathrm{dv}(x)$ has not been updated during the call, and b) $\mathrm{dv}(x)$ has been updated during the call. In case a, $\mathrm{dv}^{(N)}(x) = \mathrm{dv}^{(0)}(x) = \min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(0)}(y)$. But, if $\mathrm{dv}(x)$ has not been updated, then $\min_{y\in\alpha} \mathrm{dv}(y)$ cannot have decreased, because, using (20) and (21), this would imply that sooner or later $\min_{y\in \mathrm{fi}(x)} \mathrm{dv}(y)$ would have decreased to a value $< ub$ and that $\mathrm{dv}(x)$ would have been updated. Then, $\min_{y\in\alpha} \mathrm{dv}^{(N)}(y) = \min_{y\in\alpha} \mathrm{dv}^{(0)}(y)$. But, since (21) $\min_{y\in\alpha} \mathrm{dv}^{(N)}(y) < ub$, this implies that $\min_{y\in\alpha} \mathrm{dv}^{(0)}(y) < ub$ and, using (20) and (21), that $\min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(0)}(y) = \min_{y\in\alpha} \mathrm{dv}^{(0)}(y) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$, which, combined with $\mathrm{dv}^{(N)}(x) = \min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(0)}(y)$, implies $\mathrm{dv}^{(N)}(x) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$. In case b, taking into account (20), some $y \in \alpha$ must have been updated to a value $< ub$ such that $\min_{y\in\alpha} \mathrm{dv}(y)$ decreases to a value $< ub$. Let n be the step at which $\min_{y\in\alpha} \mathrm{dv}^{(n)}(y)$ reaches the value $\min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$ and let $y' \in \alpha$ be the node whose distance variable is updated at that step. Taking into account (20) and (21), $\min_{y\in \mathrm{fi}(x)} \mathrm{dv}^{(n')}(y) = \min_{y\in\alpha} \mathrm{dv}^{(n)}(y) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$, $n' \geq n$ and $\mathrm{dv}(x)$ will be last updated when following recursively the processing of node y' and will be updated to $\min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$, implying $\mathrm{dv}^{(N)}(x) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$. This completes case b and, therefore, proves $\mathrm{dv}^{(N)}(x) = \min_{y\in\alpha} \mathrm{dv}^{(N)}(y)$. Then (21), $\mathrm{dv}^{(N)}(x) = \min_{y\in\alpha} \widetilde{d}_{\mathrm{b}}(F,y)$. On the other hand, using (14), (21) and (22), we have $\widetilde{d}_{\mathrm{b}}(F,x) = \min_{y\in\alpha} \widetilde{d}_{\mathrm{b}}(F,y) < ub$, and both results imply $\mathrm{dv}^{(N)}(x) = \widetilde{d}_{\mathrm{b}}(F,x) < ub$.

Case 3: $\mathrm{type}(x) = \mathrm{AND}$ and $\beta \neq \emptyset$. We show $\mathrm{dv}^{(N)}(x) \geq ub$ and $\widetilde{d}_{\mathrm{b}}(F,x) \geq ub$. Note that it suffices that some $\mathrm{dv}^{(n)}(y)$, $y \in \mathrm{fi}(x) = A(x) \cup B(x) \cup C(x)$ is $\geq ub$ for

$$\sum_{y\in A(x)} \mathrm{dv}^{(n)}(y) + \max\left\{\sum_{y\in B(x)} \mathrm{dv}^{(n)}(y), \max_{y\in C(x)} \{0, \mathrm{dv}^{(n)}(y)\}\right\} \geq ub.$$

Then, using $\beta \neq \emptyset$, (19) and (20),

$$\begin{aligned} \mathrm{dv}^{(0)}(x) &= \sum_{y\in A(x)} \mathrm{dv}^{(0)}(y) + \max\left\{\sum_{y\in B(x)} \mathrm{dv}^{(0)}(y), \max_{y\in C(x)} \{0, \mathrm{dv}^{(0)}(y)\}\right\} \\ &\geq \sum_{y\in A(x)} \mathrm{dv}^{(n)}(y) + \max\left\{\sum_{y\in B(x)} \mathrm{dv}^{(n)}(y), \max_{y\in C(x)} \{0, \mathrm{dv}^{(n)}(y)\}\right\} \geq ub, \\ &\qquad n = 1, 2, \ldots, N. \end{aligned}$$

Therefore, $\mathrm{dv}(x)$ is never updated and $\mathrm{dv}^{(N)}(x) = \mathrm{dv}^{(0)}(x) \geq ub$. Also, using (15), (22) and $\beta \neq \emptyset$,

$$\widetilde{d}_{\mathrm{b}}(F,x) = \sum_{y\in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\left\{\sum_{y\in B(x)} \widetilde{d}_{\mathrm{b}}(F,y), \max_{y\in C(x)} \{0, \widetilde{d}_{\mathrm{b}}(F,y)\}\right\} \geq ub.$$

Case 4: $\mathrm{type}(x) = \mathrm{AND}$ and $\beta = \emptyset$. In this case, $A(x) \cup B(x) \cup C(x) = \alpha$. Then, using

(15) and (21),

$$\begin{aligned}\widetilde{d}_{\mathrm{b}}(F,x) &= \sum_{y\in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\Big\{\sum_{y\in B(x)} \widetilde{d}_{\mathrm{b}}(F,y),\ \max_{y\in C(x)}\{0,\widetilde{d}_{\mathrm{b}}(F,y)\}\Big\}\\ &= \sum_{y\in A(x)} \mathrm{dv}^{(N)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(N)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(N)}(y)\}\Big\} = dv'. \quad (23)\end{aligned}$$

We consider three cases: a) $dv' \geq ub$, b) $dv' < ub$ and $\mathrm{dv}(x)$ has been updated, and c) $dv' < ub$ and $\mathrm{dv}(x)$ has not been updated. In case a we show $\widetilde{d}_{\mathrm{b}}(F,x) \geq ub$ and $\mathrm{dv}^{(N)}(x) \geq ub$. The first inequality follows trivially from (23) and $dv' \geq ub$. Using (19) and (23),

$$\begin{aligned}&\sum_{y\in A(x)} \mathrm{dv}^{(n)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(n)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(n)}(y)\}\Big\}\\ &\geq \sum_{y\in A(x)} \mathrm{dv}^{(N)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(N)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(N)}(y)\}\Big\}\\ &= dv' \geq ub\,, \qquad 0\leq n\leq N\,, \quad (24)\end{aligned}$$

and $\mathrm{dv}(x)$ is never updated, implying

$$\mathrm{dv}^{(N)}(x) = \mathrm{dv}^{(0)}(x) = \sum_{y\in A(x)} \mathrm{dv}^{(0)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(0)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(0)}(y)\}\Big\} \geq ub$$

by (24). This completes case a. In case b we show $\mathrm{dv}^{(N)}(x) = \widetilde{d}_{\mathrm{b}}(F,x) < ub$. Since (23) $\widetilde{d}_{\mathrm{b}}(F,x) = dv' < ub$, it suffices to show $\mathrm{dv}^{(N)}(x) = dv'$. But, the last value at which $\mathrm{dv}(x)$ has been updated must be the minimum value of $\sum_{y\in A(x)} \mathrm{dv}^{(n)}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}^{(n)}(y), \max_{y\in C(x)}\{0,\mathrm{dv}^{(n)}(y)\}\}$, which, using (19) is $\sum_{y\in A(x)} \mathrm{dv}^{(N)}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}^{(N)}(y), \max_{y\in C(x)}\{0,\mathrm{dv}^{(N)}(y)\}\} = dv'$ (by (23)), implying $\mathrm{dv}^{(N)}(x) = dv'$. Finally, in case c we also show $\mathrm{dv}^{(N)}(x) = \widetilde{d}_{\mathrm{b}}(F,x) < ub$. Again, since (23) $\widetilde{d}_{\mathrm{b}}(F,x) = dv' < ub$, it suffices to show $\mathrm{dv}^{(N)}(x) = dv'$. Since $\mathrm{dv}(x)$ has not been updated, $\mathrm{dv}^{(N)}(x) = \mathrm{dv}^{(0)}(x)$. Then, it is enough to prove $\mathrm{dv}^{(0)}(x) = dv'$. The fact that $\mathrm{dv}^{(0)}(x) \leq dv'$ is trivial since, considering (23) and $dv' < ub$, $\mathrm{dv}(x)$ would otherwise have been updated. But, using (19) and (23),

$$\begin{aligned}\mathrm{dv}^{(0)}(x) &= \sum_{y\in A(x)} \mathrm{dv}^{(0)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(0)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(0)}(y)\}\Big\}\\ &\geq \sum_{y\in A(x)} \mathrm{dv}^{(N)}(y) + \max\Big\{\sum_{y\in B(x)} \mathrm{dv}^{(N)}(y),\ \max_{y\in C(x)}\{0,\mathrm{dv}^{(N)}(y)\}\Big\} = dv'\,,\end{aligned}$$

which, with $\mathrm{dv}^{(0)}(x) \leq dv'$, implies $\mathrm{dv}^{(0)}(x) = dv'$. □

Proposition 6 *Let F, F' and F'' be bags of component classes with $F = F' + F''$ and let ub, ub' be positive integers with $ub' \leq ub$; let CS, CS' be stacks. Let $\mathrm{dv}_1(x)$, $x \in I \cup P$ be the values of the distance variables that will result after initializing them to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ and next making the call* update_d(F, ub', CS')*; let $\mathrm{dv}_2(x)$, $x \in I \cup P$ be the values of the distance variables that result if after performing the same initialization, the call* update_d(F', ub, CS) *is made followed by the call* update_d(F'', ub', CS')*. Then, either $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$ or $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$, $x \in I \cup P$.*

Proof. For the sake of conciseness, let scenario 1 stand for "the distance variable of each node $x \in I \cup P$ has been initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ and next the call *update_d*(F, ub', CS') has been made" and let scenario 2 stand for "the distance variable of each node $x \in I \cup P$ has been initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ and next the call *update_d*(F', ub, CS) has been made followed by the call *update_d*(F'', ub', CS').

Base case ($\mathrm{lev}(x) = 0$): Since $\mathrm{lev}(x) = 0$, then $x \in I$. Given the way in which the fault tree is processed in both cases, it is clear that $\mathrm{dv}_1(x) = \widetilde{d}_{\mathrm{b}}(F, x)$ and, since $F = F' + F''$, $\mathrm{dv}_2(x) = \widetilde{d}_{\mathrm{b}}(F, x)$, implying $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$.

Induction step: Assume that the result holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$; it has to be shown that the result also holds for all $x \in P$, $\mathrm{lev}(x) = l + 1$ (x cannot be an input since $l + 1 > 0$). Let $x \in P$, $\mathrm{lev}(x) = l + 1$ and consider the partition $\mathrm{fi}(x) = \alpha + \beta$ with $\alpha = \{y \in \mathrm{fi}(x) \,|\, \mathrm{dv}_1(y) = \mathrm{dv}_2(y)\}$ and $\beta = \{y \in \mathrm{fi}(x) \,|\, \mathrm{dv}_1(y) \neq \mathrm{dv}_2(y)\}$. Using the induction hypothesis, both $\mathrm{dv}_1(y)$ and $\mathrm{dv}_2(y)$, $y \in \beta$ are $\geq ub'$. This together with the fact that the distance variables cannot increase yields the following two properties:

P1. $\mathrm{dv}(y) \geq ub'$, $y \in \beta$ throughout the call *update_d*(F, ub', CS') (scenario 1).

P2. $\mathrm{dv}(y) \geq ub'$, $y \in \beta$ throughout the consecutive calls *update_d*(F', ub, CS) and *update_d*(F'', ub', CS') (scenario 2).

We consider four cases:

Case 1: $\mathrm{type}(x) = \mathrm{OR}$ and $\alpha = \emptyset$. We show $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$. Since $\alpha = \emptyset$, then $\mathrm{fi}(x) = \beta$. In scenario 1, property P1 implies $\mathrm{dv}_1(x) \geq ub'$, since, initially, $\mathrm{dv}(x) = \min_{y \in \beta} \mathrm{dv}(y) \geq ub'$ and $\mathrm{dv}(x)$ can be made $< ub'$ only if $\min_{y \in \beta} \mathrm{dv}(y)$ is made $< ub'$. Likewise, in scenario 2, property P2 implies $\mathrm{dv}_2(x) \geq ub'$.

Case 2: $\mathrm{type}(x) = \mathrm{OR}$ and $\alpha \neq \emptyset$. We consider two cases: a) there exist $y \in \alpha$ such that $\mathrm{dv}_1(y) = \mathrm{dv}_2(y) < ub'$, and b) for all $y \in \alpha$, $\mathrm{dv}_1(y) = \mathrm{dv}_2(y) \geq ub'$. In case a we show $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$. Given the way the procedure *update_d* works and considering that distance variables can only decrease and that $ub \geq ub'$, it is clear that if $\min_{y \in \mathrm{fi}(x)} \mathrm{dv}_1(y) < ub'$, then $\mathrm{dv}_1(x) = \min_{y \in \mathrm{fi}(x)} \mathrm{dv}_1(y)$, and if $\min_{y \in \mathrm{fi}(x)} \mathrm{dv}_2(y) < ub'$, then $\mathrm{dv}_2(x) = \min_{y \in \mathrm{fi}(x)} \mathrm{dv}_2(y)$. But, in case a we clearly have $\min_{y \in \mathrm{fi}(x)} \mathrm{dv}_1(y) < ub'$ and $\min_{y \in \mathrm{fi}(x)} \mathrm{dv}_2(y) < ub'$ and, taking into account properties P1 and P2 and $\mathrm{dv}_1(y) = \mathrm{dv}_2(y)$, $y \in \alpha$,

$$\begin{aligned}
\mathrm{dv}_1(x) &= \min_{y \in \mathrm{fi}(x)} \mathrm{dv}_1(y) = \min_{\substack{y \in \alpha \\ \mathrm{dv}_1(y) < ub'}} \mathrm{dv}_1(y) = \min_{\substack{y \in \alpha \\ \mathrm{dv}_2(y) < ub'}} \mathrm{dv}_2(y) \\
&= \min_{y \in \mathrm{fi}(x)} \mathrm{dv}_2(y) = \mathrm{dv}_2(x).
\end{aligned}$$

In case b we show $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$. In that case, $\mathrm{dv}(y) \geq ub'$, $y \in \alpha$ throughout the calls to *update_d* in both scenario 1 and scenario 2. But, using properties P1 and P2, this implies $\mathrm{dv}(y) \geq ub'$, $y \in \mathrm{fi}(x)$ throughout the calls to *update_d* in both scenarios and, hence, $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$, since, in both scenarios, initially $\mathrm{dv}(x) = \min_{y \in \mathrm{fi}(x)} \mathrm{dv}(y)$ and $\mathrm{dv}(x)$ can be made $< ub'$ only if $\min_{y \in \mathrm{fi}(x)} \mathrm{dv}(y)$ is made $< ub'$.

Case 3: type(x) = AND and $\beta \neq \emptyset$. We show $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$. Since it is enough $\mathrm{dv}(y) \geq ub'$ for some $y \in \mathrm{fi}(x)$ for $\sum_{y\in A(x)} \mathrm{dv}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}(y), \max_{y\in C(x)}\{0, \mathrm{dv}(y)\}\} \geq ub'$, properties P1 and P2 imply $\sum_{y\in A(x)} \mathrm{dv}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}(y), \max_{y\in C(x)}\{0, \mathrm{dv}(y)\}\} \geq ub'$ throughout the calls to *update_d* in both scenarios, implying that $\mathrm{dv}(x)$ has not been updated to a value $< ub'$ in either scenario. But, initially, $\mathrm{dv}(x) = \sum_{y\in A(x)} \mathrm{dv}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}(y), \max_{y\in C(x)}\{0, \mathrm{dv}(y)\}\} \geq ub'$ in both scenarios by properties P1 and P2 and, then, $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$.

Case 4: type(x) = AND and $\beta = \emptyset$. In this case we have $\mathrm{fi}(x) = \alpha$ and $\mathrm{dv}_1(y) = \mathrm{dv}_2(y)$, $y \in \mathrm{fi}(x)$. We consider three cases: a) $\mathrm{dv}(x)$ is updated in scenario 1, b) $\mathrm{dv}(x)$ is not updated in scenario 1 and is updated in scenario 2, and c) $\mathrm{dv}(x)$ is not updated in either scenario. In case a we show $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$. Since $\mathrm{dv}(x)$ is updated in scenario 1, necessarily $\sum_{y\in A(x)} \mathrm{dv}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}(y), \max_{y\in C(x)}\{0, \mathrm{dv}(y)\}\}$ has been made, in that scenario, smaller than ub' and smaller than the initial value of $\mathrm{dv}(x)$. Then, since distance variables can only decrease,

$$\mathrm{dv}_1(x) = d = \sum_{y\in A(x)} \mathrm{dv}_1(y) + \max\left\{ \sum_{y\in B(x)} \mathrm{dv}_1(y), \max_{y\in C(x)} \{0, \mathrm{dv}_1(y)\} \right\} < ub' . \qquad (25)$$

Hence, at this point we have:

F1. $d = \sum_{y\in A(x)} \mathrm{dv}_2(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}_2(y), \max_{y\in C(x)}\{0, \mathrm{dv}_2(y)\}\} < ub'$ since $\mathrm{dv}_2(y) = \mathrm{dv}_1(y)$, $y \in \mathrm{fi}(x)$ and (25);

F2. $ub \geq ub'$.

F3. the initial value of $\mathrm{dv}(x)$ in scenario 1 is $> d$;

F4. the initial value of $\mathrm{dv}(x)$ is the same in both scenarios;

Then, F3 and F4 imply that the initial value of $\mathrm{dv}(x)$ in scenario 2 is $> d$, F1 and F2 imply that $d < ub' \leq ub$, and both results along with the way *update_d* works and the fact that distance variables can only decrease result in $\mathrm{dv}_2(x) = \mathrm{dv}_1(x)$.

In case b we show $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$. Since $\mathrm{dv}(x)$ is updated in scenario 2, necessarily $\sum_{y\in A(x)} \mathrm{dv}(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}(y), \max_{y\in C(x)}\{0, \mathrm{dv}(y)\}\}$ has been made, in that scenario, smaller than the initial value of $\mathrm{dv}(x)$. Then, given that distance variables can only decrease and that $ub \geq ub'$, we have

$$\mathrm{dv}_2(x) = \sum_{y\in A(x)} \mathrm{dv}_2(y) + \max\left\{ \sum_{y\in B(x)} \mathrm{dv}_2(y), \max_{y\in C(x)} \{0, \mathrm{dv}_2(y)\} \right\},$$

smaller than the initial value of $\mathrm{dv}(x)$, with either $\mathrm{dv}_2(x) < ub'$ or $ub' \leq \mathrm{dv}_2(x) < ub$. If $\mathrm{dv}_2(x) < ub'$, using $\mathrm{dv}_1(y) = \mathrm{dv}_2(y)$, $y \in \mathrm{fi}(x)$, we have $\sum_{y\in A(x)} \mathrm{dv}_1(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}_1(y), \max_{y\in C(x)}\{0, \mathrm{dv}_1(y)\}\} = \sum_{y\in A(x)} \mathrm{dv}_2(y) + \max\{\sum_{y\in B(x)} \mathrm{dv}_2(y), \max_{y\in C(x)}\{0, \mathrm{dv}_2(y)\}\} < ub'$ and, since the initial value of $\mathrm{dv}(x)$ in scenario 1 (equal to the initial value of $\mathrm{dv}(x)$ in scenario 2) is greater than this value, $\mathrm{dv}(x)$ would have been updated in scenario 1, a contradiction. Therefore, necessarily, $\mathrm{dv}_2(x) \geq ub'$. Since $\mathrm{dv}(x)$

has not been updated in scenario 1, the only way in which $\mathrm{dv}_1(x)$ could be $< ub'$ is that initially $\mathrm{dv}(x)$ were $< ub'$ in scenario 1, but the initial value of $\mathrm{dv}(x)$ is the same in both scenarios and we have shown that value to be $> \mathrm{dv}_2(x) \geq ub'$. Therefore, $\mathrm{dv}_1(x) \geq ub'$. Finally, in case c both $\mathrm{dv}_1(x)$ and $\mathrm{dv}_2(x)$ are equal to the common initial value of $\mathrm{dv}(x)$ in scenarios 1 and 2, implying $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$. $\square$

Proposition 7 *Let F, F' and F'' be bags of component classes with $F = F' + F''$ and let ub, ub' be positive integers with $ub' \leq ub$; let CS, CS' be stacks. Let the distance variable* $\mathrm{dv}(x)$ *of each node x of the fault tree be initialized to $\widetilde{d}_\mathrm{b}(\emptyset, x)$. Then, after the call* $\mathrm{update\_d}(F', ub, CS)$ *followed by the call* $\mathrm{update\_d}(F'', ub', CS')$*, for each $x \in I \cup P$ we have $\widetilde{d}_\mathrm{b}(F, x) = \mathrm{dv}(x)$ if $\mathrm{dv}(x) < ub'$ and $\widetilde{d}_\mathrm{b}(F, x) \geq ub'$ if $\mathrm{dv}(x) \geq ub'$.*

Proof. Let $x \in I \cup P$ and let $\mathrm{dv}_1(x)$ and $\mathrm{dv}_2(x)$ be as in Proposition 6. It has to be shown that $\widetilde{d}_\mathrm{b}(F, x) = \mathrm{dv}_2(x)$ if $\mathrm{dv}_2(x) < ub'$ and $\widetilde{d}_\mathrm{b}(F, x) \geq ub'$ if $\mathrm{dv}_2(x) \geq ub'$. Using Proposition 6, either $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$ or $\mathrm{dv}_1(x) \geq ub'$ and $\mathrm{dv}_2(x) \geq ub'$. This implies that $\mathrm{dv}_1(x) = \mathrm{dv}_2(x)$ if $\mathrm{dv}_2(x) < ub'$ and $\mathrm{dv}_1(x) \geq ub'$ if $\mathrm{dv}_2(x) \geq ub'$. Using Proposition 5 with $ub = ub'$ and $CS = CS'$ we have $\widetilde{d}_\mathrm{b}(F, x) = \mathrm{dv}_1(x)$ if $\mathrm{dv}_1(x) < ub'$ and $\widetilde{d}_\mathrm{b}(F, x) \geq ub'$ if $\mathrm{dv}_1(x) \geq ub'$, implying $\widetilde{d}_\mathrm{b}(F, x) = \mathrm{dv}_2(x)$ if $\mathrm{dv}_2(x) < ub'$ and $\widetilde{d}_\mathrm{b}(F, x) \geq ub'$ if $\mathrm{dv}_2(x) \geq ub'$. $\square$

Proposition 8 *Let F, F' and F'' be bags of component classes with $F = F' + F''$; let lb and ub' be non-negative integers and let ub be a positive integer with $lb \leq \widetilde{d}_\mathrm{b}(F, g_r) \leq ub' \leq ub$; let CS and CS' be stacks. Let the distance variable* $\mathrm{dv}(x)$ *of each node x of the fault tree be initialized to $\widetilde{d}_\mathrm{b}(\emptyset, x)$. Then, after making the call* $\mathrm{comp\_d}(F', 0, ub, CS)$*, the call* $\mathrm{comp\_d}(F'', lb, ub', CS')$ *returns $\widetilde{d}_\mathrm{b}(F, g_r)$.*

Proof. If $lb = ub'$, the call *comp_d*(F'', lb, ub', CS') returns lb, which is equal to $\widetilde{d}_\mathrm{b}(F, g_r)$ because, as assumed, $lb \leq \widetilde{d}_\mathrm{b}(F, g_r) \leq ub'$. If $lb < ub'$, the call *comp_d*(F'', lb, ub', CS') results in the call *update_d*(F'', ub', CS') and returns $\min\{\mathrm{dv}(g_r), ub'\}$. Two cases are possible: a) $\mathrm{dv}(g_r) < ub'$ and b) $\mathrm{dv}(g_r) \geq ub'$. The inequality $ub > 0$ implies that the call *comp_d*$(F', 0, ub, CS)$ results in making the call *update_d*(F', ub, CS). Also, $lb \geq 0$ and $lb < ub'$ implies $ub' > 0$ and Proposition 7 can be invoked. In case a, the call *comp_d*(F'', lb, ub', CS') returns $\min\{\mathrm{dv}(g_r), ub'\} = \mathrm{dv}(g_r)$, which, according to Proposition 7, is equal to $\widetilde{d}_\mathrm{b}(F, g_r)$; in case b, the call *comp_d*(F'', lb, ub', CS') returns $\min\{\mathrm{dv}(g_r), ub'\} = ub'$, but Proposition 7 asserts $\widetilde{d}_\mathrm{b}(F, x) \geq ub'$, which with, as assumed, $\widetilde{d}_\mathrm{b}(F, g_r) \leq ub'$ implies $ub' = \widetilde{d}_\mathrm{b}(F, g_r)$. $\square$

Theorem 5 *Let F be a bag of component classes, let lb and ub be non-negative integers with $lb \leq \widetilde{d}_\mathrm{b}(F, g_r) \leq ub$, and let CS be a stack. Let the distance variable* $\mathrm{dv}(x)$ *of each node x of the fault tree be initialized to $\widetilde{d}_\mathrm{b}(\emptyset, x)$. Then, the call* $\mathrm{comp\_d}(F, lb, ub, CS)$ *returns $\widetilde{d}_\mathrm{b}(F, g_r)$.*

Proof. If $lb = ub$, the call *comp_d*(F, lb, ub, CS) returns lb, which is equal to $\widetilde{d}_\mathrm{b}(F, g_r)$ because, as assumed, $lb \leq \widetilde{d}_\mathrm{b}(F, g_r) \leq ub$. If $lb < ub$, the call *comp_d*(F, lb, ub, CS) results in the call *update_d*(F, ub, CS) and returns $\min\{\mathrm{dv}(g_r), ub\}$. Two cases are possible: a) $\mathrm{dv}(g_r) < ub$ and b) $\mathrm{dv}(g_r) \geq ub$. Since $lb \geq 0$, then $ub > 0$ and Proposition 5 can be invoked. In case a, the call *comp_d*(F, lb, ub, CS) returns $\min\{\mathrm{dv}(g_r), ub\} = \mathrm{dv}(g_r)$, which, according to Proposition 5, is equal to $\widetilde{d}_{\mathrm{b}}(F, g_r)$; in case b, the call *comp_d*(F, lb, ub, CS) returns $\min\{\mathrm{dv}(g_r), ub\} = ub$, but Proposition 5 asserts $\widetilde{d}_{\mathrm{b}}(F, g_r) \geq ub$, which with, as assumed, $\widetilde{d}_{\mathrm{b}}(F, g_r) \leq ub$ implies $ub = \widetilde{d}_{\mathrm{b}}(F, g_r)$. □

Theorem 6 *Let F be a bag of component classes, let $\widetilde{d} = \widetilde{d}_{\mathrm{b}}(F, g_r) > 0$, and let $E' \subset E$. Let the distance variable* $\mathrm{dv}(x)$ *of each node x of the fault tree be initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$. Then, the call* $\mathrm{comp\_all\_d}(F, \widetilde{d}, E', \widetilde{\eta}(e), e \in E')$ *computes correctly the lower bounds $\widetilde{d}_{\mathrm{b}}(F + e, g_r)$, $e \in E'$.*

Proof. The call *restore_d*(CS') done in step 7 of *comp_all_d* restores the distance variables to the values they had before the previous call *comp_d*(e, s, t, CS') done in step 6. Therefore, the value returned by the call to *comp_d* in that step is the same as the value that would be returned by making, for each $e \in E'$ and with the distance variable $\mathrm{dv}(x)$ of each node x of the fault tree initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$, the call to *comp_d* of step 2 followed by the call to *comp_d* of step 6.

Next, it is shown that the s and t computed, respectively, in steps 4 and 5 of *comp_all_d* satisfy $s \leq \widetilde{d}_{\mathrm{b}}(F + e, g_r) \leq t \leq \widetilde{d}$. Regarding the first inequality, Theorem 4 with $F' = F$, $F'' = e$ and $x = g_r$ gives $\widetilde{d}_{\mathrm{b}}(F + e, g_r) \geq \widetilde{d} - S(e, g_r)$; the same theorem with $F' = e$, $F'' = F$ and $x = g_r$ yields $\widetilde{d}_{\mathrm{b}}(F + e, g_r) \geq \widetilde{\eta}(e) - S(F, g_r)$; in addition, from the definition of $S(\cdot)$ it is clear that $S(e, g_r) \leq |e|$ and $S(F, g_r) \leq |F|$. Combining these results, taking into account that $\widetilde{d}_{\mathrm{b}}(F + e, g_r) \geq 0$, we have $\widetilde{d}_{\mathrm{b}}(F + e, g_r) \geq \max\{0, \widetilde{d} - |e|, \widetilde{\eta}(e) - |F|\} = s$. To show the second inequality, it is enough to recall that $\widetilde{d} = \widetilde{d}_{\mathrm{b}}(F, g_r)$ and $\widetilde{\eta}(e) = \widetilde{d}_{\mathrm{b}}(e, g_r)$, and to note that, according to (13–15), $F' \supset F$ implies $\widetilde{d}_{\mathrm{b}}(F', x) \leq \widetilde{d}_{\mathrm{b}}(F, x)$. The third inequality is trivial: $t = \min\{\widetilde{d}, \widetilde{\eta}(e)\} \leq \widetilde{d}$. Then, the result follows invoking Proposition 8 with $F' = F$, $F'' = e$, $lb = s \geq 0$, $ub' = t \geq 0$, and $ub = \widetilde{d} > 0$. □

3.4 Generation of the CTMC X' in SC-BM

To describe with detail the generation of the CTMC X' in SC-BM we assume that the failure behavior of the fault-tolerant system is described by a high-level specification made up of a set of rules. Each rule is composed of a guard condition that determines when the rule is enabled in a given (current) state, an execution part that allows to obtain a next state from the current one, and a rate specification that determines the rate with which the next state is reached. Each rule r has associated with it a failure bag $e(r)$ meaning that the rule models the failure of the components in $e(r)$. Different rules can have associated with them the same failure bag. We assume the availability of three procedures named *enabled*, *rate* and *successor*, which provide the interface with the high-level specification required for model generation. The procedure *enabled* has as inputs a state a and returns the set of rules $R(a)$ enabled in a. The procedure *rate* takes as inputs a state a and a rule r and returns the rate $\lambda^{r,a}$ with which the next state is reached from a following rule r. The procedure *successor* takes as inputs a state a and a rule r and returns the state b reached from a following r. Although the interface with the high-level specification could be provided by other sets of procedures, the assumed one is reasonable, matches our implementation, and leads to an efficient generation of X', allowing a fair comparison between SC-BM and T-SC-BM.

The $\widetilde{L}$ and $\widetilde{\eta}(e)$, $e \in E$ are computed before the generation of X'. To do that, first the distance variable $\mathrm{dv}(x)$ of each node x of the fault tree is initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ by traversing the fault tree depth-first left-most, starting at g_r and using (13)–(15). Since $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$, after the initialization $\mathrm{dv}(g_r)$ holds $\widetilde{L}$. The $\widetilde{\eta}(e)$, $e \in E$ are computed using *comp_d* and *restore_d*. Note that $\widetilde{\eta}(e) = \widetilde{d}_{\mathrm{b}}(e, g_r) \geq 0$ and, according to the definition of $\widetilde{d}_{\mathrm{b}}(\cdot)$, $\widetilde{\eta}(e) \leq \widetilde{d}_{\mathrm{b}}(\emptyset, g_r) = \widetilde{L}$. This allows the computing of $\widetilde{\eta}(e)$, $e \in E$ by creating an empty stack *CS* and, for each $e \in E$, making the call *comp_d*$(e, 0, \widetilde{L}, CS)$ followed by the call *restore_d*(CS). Using Theorem 5 with $F = e$, $lb = 0$ and $ub = \widetilde{L}$, the value returned by the call *comp_d*$(e, 0, \widetilde{L}, CS)$ is $\widetilde{\eta}(e)$. After computing $\widetilde{\eta}(e)$, $e \in E$, the distance variable of each node x of the fault tree holds its initial value $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$.

The CTMC X' is generated breadth-first using $\widetilde{L}$, $\widetilde{\eta}(e)$, $e \in E$, and a FIFO queue Q, by calling the procedures *comp_all_d*, *enabled*, *rate*, and *successor*. The queue Q holds triplets $(s, F(s), \widetilde{d}(s))$ corresponding to the states s that have to be processed. The generation of X' starts by creating the states f and u_i, $0 \leq i \leq \widetilde{L}$, adding the transition rates f_i from u_d to $u_{\max\{0,d-i\}}$ (see Section 2.1) and making $Q = \{(o, \emptyset, \widetilde{L})\}$ and $G = \{o\}$. Then, while $Q \neq \emptyset$, a triplet $(a, F(a), \widetilde{d}(a))$ is pulled out of Q and processed as follows. First, $R(a)$ is obtained by making the call *enabled*(a). Next, letting $E(a) = \{e(r)\}_{r \in R(a)}$, the lower bounds $\widetilde{d}(a, e) = \widetilde{d}_{\mathrm{b}}(F(a) + e, g_r)$ for the failure distance from the states with bag of failed component classes $F(a) + e$, $e \in E(a)$ are obtained by making the call *comp_all_d*$(F(a), \widetilde{d}(a), E(a), \widetilde{\eta}(e), e \in E(a))$. The lower bound for the failure distance from the state reached from a following rule $r \in R(a)$ is $\widetilde{d}(a, e(r))$. Then, each $r \in R(a)$ is processed as follows. First, $\lambda^{r,a}$ is computed by making the call *rate*(a, r). Second, if $\widetilde{d}(a, e(r)) = 0$, the state reached from a following r is a down state and a transition rate $\lambda^{r,a}$ from a to f is added. If $\widetilde{d}(a, e(r)) > 0$ and $|F(a) + e(r)| > K$, the state reached from a following r belongs to $\widetilde{U}_d$, $d = \widetilde{d}(a, e(r))$ and a transition rate $\lambda^{r,a}$ from a to u_d is added. Finally, if $\widetilde{d}(a, e(r)) > 0$ and $|F(a)+e(r)| \leq K$, the state reached from a following r belongs to the subset of states that have to be generated. In that case, the reached state, b, is obtained by making the call *successor*(a, r) and, if $b \notin G$, b is added to G, a transition rate $\lambda^{r,a}$ from a to b is added and the triplet $(b, F(a)+e(r), \widetilde{d}(a, e(r)))$ is put into Q; if $b \in G$, a transition rate $\lambda^{r,a}$ from a to b is added.

4 Generation of X'' in T-SC-BM and Comparison of SC-BM with T-SC-BM

This section describes T-SC-BM and shows that, when $\widetilde{L} = 1$, the cost (in terms of CPU time) of SC-BM is at most identical to the cost of T-SC-BM.

T-BM requires knowledge of the operational/down state of the successors b of an operational state a. T-SC-BM makes use of a procedure, *eval_all_ft*, to determine, using the fault tree, whether the states reached from a given state following the rules enabled in it are operational or down. The procedure *eval_all_ft* is built on top of two procedures: *eval_ft* and *restore_ft*.

Each input x of the fault tree has associated with it a distance variable $\mathrm{dv}(x)$, and each node z of the fault tree has associated with it a value variable $\mathrm{vv}(z)$. The procedure *eval_ft* takes as inputs a bag of failed component classes F and a stack CS, and works as follows. For each input x, $\mathrm{b}(x) = c[n]$ for which $c[n']$ is part of F, $\mathrm{dv}(x)$ is set to $\max\{0, \mathrm{dv}(x) - n'\}$ and, if $\mathrm{dv}(x)$ becomes 0, x is implied to 1 (*ie* $\mathrm{vv}(x)$ is set to 1). Each implication to 1 of an input x is propagated up the fault tree while the visited gate becomes implied to 1. When a distance variable changes, the corresponding input and the old value of the variable are put in CS. Gates that become implied to 1 are put in CS too. The procedure returns $\mathrm{vv}(g_r)$.

The procedure *restore_ft* has as input a stack CS. For each input x of the fault tree held in CS, the procedure restores $\mathrm{dv}(x)$ to its old value, setting $\mathrm{vv}(x)$ to 0 if the restored value is > 0. For each gate x kept in CS, the procedure sets $\mathrm{vv}(x)$ to 0. After the call to *restore_ft*, CS becomes empty.

The procedure *eval_all_ft* takes as inputs a bag of failed component classes F and a subset $E' \subset E$, and for each $e \in E'$, computes the value, v_{F+e}, of the root gate of the fault tree when the bag of failed component classes is $F + e$. For the procedure to work properly, the distance variable $\mathrm{dv}(x)$ of each input x, $\mathrm{b}(x) = c[n]$ of the fault tree must have been initialized to n, and the value variable $\mathrm{vv}(x)$ of each node of the fault tree must have been set to 0. The procedure is as follows.

```
P1. Let $CS$ and $CS'$ be empty stacks.
P2. $eval\_ft(F, CS)$
P3. for (each $e \in E'$) {
P4.     $v_{F+e} = eval\_ft(e, CS')$
P5.     $restore\_ft(CS')$
    }
P6. $restore\_ft(CS)$
```

Note that after calling the procedure, $\mathrm{dv}(x) = n$ for each input x, $\mathrm{b}(x) = c[n]$ of the fault tree and $\mathrm{val}(z) = 0$ for each node z of the fault tree.

The CTMC X'' is generated breadth-first using a FIFO queue Q and calling the procedure *eval_all_ft* and the procedures *enabled*, *rate* and *successor* described in Section 3.4. The queue Q holds pairs $(s, F(s))$ corresponding to the states s that have to be processed. The generation of X'' starts by creating the states f and u_0, and making $Q = \{(o, \emptyset)\}$ and $G = \{o\}$. Then, while $Q \neq \emptyset$, a pair $(a, F(a))$ is pulled out of Q and processed as follows. First, the set of rules enabled in a, $R(a)$, is obtained by making the call *enabled*(a). Next, denoting by $e(r)$ the failure bag associated with rule r and letting $E(a) = \{e(r)\}_{r \in R(a)}$, the values of the root gate of the fault tree, $v_{F(a)+e}$, for the states with bag of failed component classes $F(a)+e$, $e \in E(a)$ are obtained by making the call *eval_all_ft*$(F(a), E(a))$. The value of the root gate of the fault tree for the state reached from a following rule $r \in R(a)$ is $v_{F(a)+e(r)}$. Then, each $r \in R(a)$ is processed as follows. First, the rate $\lambda^{r,a}$ with which the next state is reached from a following rule r is computed by making the call *rate*(a, r). Second, if $v_{F(a)+e(r)} = 1$, the state reached from a following r is a down state and a transition rate $\lambda^{r,a}$ from a to f is added. If $v_{F(a)+e(r)} = 0$ and $|F(a) + e(r)| > K$, the state reached from a following r belongs to U and a transition rate $\lambda^{r,a}$ from a to u_0 is added. Finally, if $v_{F(a)+e(r)} = 0$ and $|F(a) + e(r)| \leq K$, the state reached from a following r belongs to the subset of states that have to be generated. In that case, the reached state, b, is obtained by making the call *successor*(a, r) and, if $b \notin G$, b is added to G, a transition rate $\lambda^{r,a}$ from a to b is added, and the pair $(b, F(a) + e(r))$ is put into Q; if $b \in G$, a transition rate $\lambda^{r,a}$ from a to b is added.

This section finishes by comparing the cost (in terms of CPU time) of SC-BM with the cost of T-SC-BM for the particular case $\widetilde{L} = 1$. The comparison is done with the aid of the following theorem.

Theorem 7 *Let F be a bag of component classes, let $\widetilde{d} = \widetilde{d}_{\mathrm{b}}(F, g_r) > 0$, and let $E' \subset E$. If $\widetilde{L} = 1$, then the cost of the call* $\mathrm{comp\_all\_d}(F, \widetilde{d}, E', \widetilde{\eta}(e), e \in E')$ *after setting* $\mathrm{dv}(x) = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$*, $x \in I \cup P$ is at most equal to the cost of the call* $\mathrm{eval\_all\_ft}(F, E')$ *after setting* $\mathrm{dv}(x) = n$*, $x \in I$,* $\mathrm{b}(x) = c[n]$*, and* $\mathrm{vv}(x) = 0$*, $x \in I \cup P$.*

Proof. The proof starts by showing that $\widetilde{L} = 1$ implies $\widetilde{d} = 1$ and $0 \leq \widetilde{\eta}(e) \leq 1$, $e \in E'$. Let F' and F'' with $F' \subset F''$ be bags of component classes and let $x \in I \cup P$. Then, $\widetilde{d}_{\mathrm{b}}(F'', x) \leq \widetilde{d}_{\mathrm{b}}(F', x)$. Also, it has been assumed $\widetilde{d} > 0$ and $\widetilde{L} = 1$. Then, $0 < \widetilde{d} = \widetilde{d}_{\mathrm{b}}(F, g_r) \leq \widetilde{d}_{\mathrm{b}}(\emptyset, g_r) = \widetilde{L} = 1$ and, thereby, $\widetilde{d} = 1$. Regarding $\widetilde{\eta}(e) = \widetilde{d}_{\mathrm{b}}(e, g_r)$, $e \in E'$, they are always ≥ 0, and, therefore, $0 \leq \widetilde{d}_{\mathrm{b}}(e, g_r) \leq \widetilde{d}_{\mathrm{b}}(\emptyset, g_r) = \widetilde{L} = 1$.

Since (13)–(15), then $\mathrm{dv}(x) = n$ for inputs x with $\mathrm{b}(x) = c[n]$, and $\mathrm{dv}(x) > 0$, $x \in P$ before the call *comp_all_d*$(F, \widetilde{d}, E', \widetilde{\eta}(e), e \in E')$. Therefore, the distance variable of each input of the fault tree has the same value before both the call to *comp_all_d* and the call to *eval_all_ft*.

Comparison of the costs of the calls *comp_all_d*$(F, \widetilde{d}, E', \widetilde{\eta}(e), e \in E')$ and *eval_all_ft*(F, E') is done in four steps:

P1. Comparison of the cost of the call *comp_d*$(F, 0, \widetilde{d}, CS)$ done in step 2 of *comp_all_d* with the cost of the call *eval_ft*(F, CS) done in step P2 of *eval_all_ft*.

P2. Comparison of the cost of each call *comp_d*(e, s, t, CS') done in step 6 of *comp_all_d* with the cost of each call *eval_ft*(e, CS') done in step P4 of *eval_all_ft*.

P3. Comparison of the cost of each call *restore_d*(CS') done in step 7 of *comp_all_d* with the cost of each call *restore_ft*(CS') done in step P5 of *eval_all_ft*.

P4. Comparison of the cost of the call *restore_d*(CS) done in step 8 of *comp_all_d* with the cost of the call *restore_ft*(CS) done in step P6 of *eval_all_ft*.

In case 1, $\widetilde{d} = 1$ implies that the call *comp_d*$(F, 0, \widetilde{d}, CS)$ results in the call *update_d*$(F, 1, CS)$. That call processes the inputs x of the fault tree that have associated with them a bag $\mathrm{b}(x)$ related to some part of F, and updates the distance variable $\mathrm{dv}(x)$ of the gates x such that $\widetilde{d}_{\mathrm{b}}(F, x) = 0$. The call *eval_ft*$(F, CS)$ processes the same inputs as the call *update_d*$(F, 1, CS)$, and sets $\mathrm{vv}(x) = 1$ for the nodes x that become implied. The $\mathrm{dv}(x)$, $x \in I$ is the same before either call, and (13) is updated alike in both calls. Also, $\mathrm{vv}(x) = 1$, $x \in I \cup P$ if and only if $\widetilde{d}_{\mathrm{b}}(F, x) = 0$. Then, the set of nodes whose value variable is updated in the call *eval_ft*(F, CS) is the same as the set of nodes whose distance variable is updated in the call *update_d*$(F, 1, CS)$. Therefore, the cost of the call *comp_d*$(F, 0, \widetilde{d}, CS)$ done in step 2 of *comp_all_d* is essentially the same as the cost of the call *eval_ft*(F, CS) done in step P2 of *eval_all_ft*.

In case 2, $\widetilde{d} = 1$ and $0 \leq \widetilde{\eta}(e) \leq 1$, $e \in E'$ imply that the variables s and t computed in steps 4 and 5 of *comp_all_d* satisfy $0 \leq s, t \leq 1$. Because $s \leq t$, the only possibilities are $s = 0$, $t = 0$; $s = 0$, $t = 1$; and $s = 1$, $t = 1$. In cases $s = 0$, $t = 0$ and $s = 1$, $t = 1$, the call *comp_d*(e, s, t, CS') does not make any processing on the fault tree, whereas the call *eval_ft*(e, CS') may or may not result in processing the fault tree. In case $s = 0$, $t = 1$, the call *comp_d*$(e, 0, 1, CS')$ results in the call *update_d*$(e, 1, CS')$. Reasoning as it has been done in case 1, the cost of that call is essentially the same as the cost of the call *eval_ft*(e, CS'). Therefore, the cost of each call *comp_d*(e, s, t, CS') done in step 6 of *comp_all_d* is at most equal to the cost of each call *eval_ft*(e, CS') done in step P4 of *eval_all_ft*.

In cases 3 and 4, the sizes of the stacks CS and CS' in *comp_all_d* are not larger than the sizes of the corresponding stacks in *eval_all_ft*. Therefore, the cost of each call *restore_d*(CS') done in step 7 of *comp_all_d* is at most equal to the cost of each call *restore_ft*(CS') done in step P5 of *eval_all_ft*; and the cost of the call *restore_d*(CS) done in step 8 of *comp_all_d* is essentially the same as the cost of the call *restore_ft*(CS) done in step P6 of *eval_all_ft*.

In summary, the cost of the call *comp_all_d*$(F, \widetilde{d}, E', \widetilde{\eta}(e), e \in E')$ is at most equal to the cost of the call *eval_all_ft*(F, E') when $\widetilde{L} = 1$. $\square$

The subset G generated by both methods is the same. Then, taking into account the description of the generation of X' in SC-BM done in Section 3.4 and the description of the generation of X'' in T-SC-BM done in this section, the former differs basically from the latter only in that a call to *comp_all_d* is done during the generation of X' whenever a call to *eval_all_ft* is done during the generation of X''. Then, since solution of X' and X'' takes negligible time compared with their generation, comparison of the costs of both methods can be roughly done by comparing, for a given $a \in G$ and $E(a) \subset E$, the cost of the call *comp_all_d*$(F(a), \widetilde{d}(a), E(a), \widetilde{\eta}(e), e \in E(a))$ done in SC-BM with the cost of the corresponding call *eval_all_ft*$(F(a), E(a))$ done in T-SC-BM. Therefore, invoking Theorem 7 with $F = F(a)$, $\widetilde{d} = \widetilde{d}_{\mathrm{b}}(F(a), g_r) = \widetilde{d}(a) > 0$ and $E' = E(a)$, it follows that, for the particular case $\widetilde{L} = 1$, the cost of SC-BM is at most equal to the cost of T-SC-BM.

5 Analysis and Comparison

This section analyzes SC-BM and compares it with BM-1 and T-SC-BM by means of two large examples representing two scenarios:

1. The fault tree satisfies the condition of Theorem 3 and $\widetilde{L} > 1$.

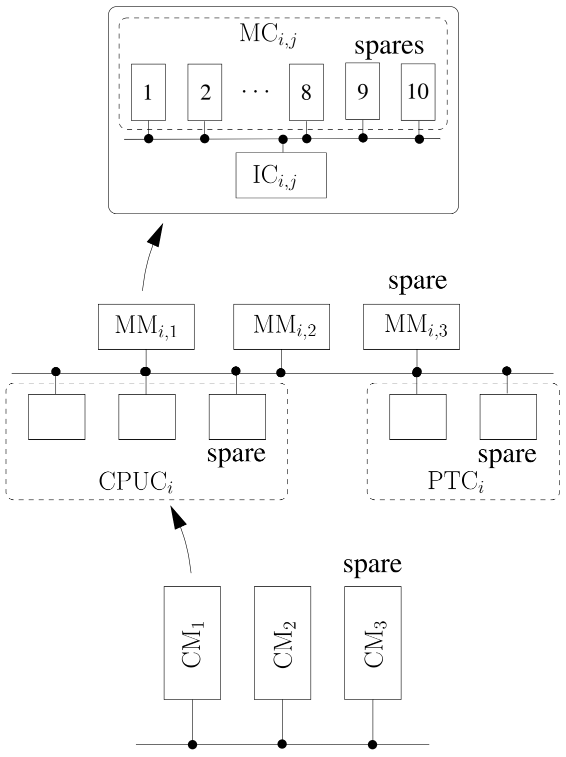

Figure 6: Architecture of example 1.

2. The fault tree does not satisfy the condition of Theorem 3 and $\widetilde{L} > 1$.

In both examples, the state o is the initial state of X. The CTMC are solved in all methods using the randomization method [14]. In example 1, SC-BM and BM-1 give exactly the same bounds because Theorem 3 guarantees $\widetilde{d}(a) = d(a)$. In example 2, SC-BM gives less tighter bounds than BM-1. Since $\widetilde{L} > 1$ for both examples, SC-BM could have a higher cost (in terms of CPU time) than T-SC-BM. The parameter R of the algorithms for the computation of failure distances used in BM-1 [1] was set to 2 for both examples.

5.1 Example 1 (Scenario 1)

The system, adapted from [6], has 114 components, and its architecture is shown in Figure 6. The system has three computing modules CM_i, $1 \leq i \leq 3$, one of which is spare. The CM_i includes three memory modules $\mathrm{MM}_{i,j}$, $1 \leq j \leq 3$, three s-identical CPU chips CPUC_i and two s-identical port chips PTC_i. One $\mathrm{MM}_{i,j}$, one CPUC_i and one PTC_i are spares. The $\mathrm{MM}_{i,j}$ has ten s-identical memory chips $\mathrm{MC}_{i,j}$, two of which are spares, and one interface chip $\mathrm{IC}_{i,j}$. Memory modules $\mathrm{MM}_{i,j}$ and $\mathrm{MM}_{i',j}$ are s-identical (*eg* $\mathrm{MM}_{1,1}$ and $\mathrm{MM}_{2,1}$).

Active $\mathrm{MC}_{i,j}$ and active $\mathrm{IC}_{i,j}$ fail, respectively, with rates λ_{MC_j} and λ_{IC_j}. Active PTC_i and active CPUC_i fail, respectively, with rates λ_{PTC_i} and $\lambda_{\mathrm{CPUC}_i}$. Spare chips fail with rates $\nu \times \lambda_{\mathrm{MC}_j}$, $\nu \times \lambda_{\mathrm{IC}_j}$, $\nu \times \lambda_{\mathrm{PTC}_i}$, and $\nu \times \lambda_{\mathrm{CPUC}_i}$, where ν, $0 < \nu < 1$ is a dormancy factor. Components of non-operational memory modules and non-operational computing modules do not fail.

Recovery is hierarchical. A fault in a $\mathrm{MC}_{i,j}$ is covered with probability C_{MC}. Failure of $\mathrm{MM}_{i,j}$, a CPUC_i and a PTC_i is successfully covered with probabilities C_{MM}, C_{CPUC} and C_{PTC}, respectively. Failure of CM_i is covered with probability C_{CM}.

Coverage faults are modeled by introducing fictitious components as explained in Section 2. An uncovered fault in a $\mathrm{MC}_{i,j}$ is propagated to a fictitious component $\mathrm{RMM}_{i,j}$. An uncovered failure in $\mathrm{MM}_{i,j}$, a CPUC_i or a PTC_i is propagated to two fictitious components RCM_i. An uncovered failure in CM_i is propagated to four fictitious components RSYS.

The $\mathrm{MM}_{i,j}$ is operational if at least eight $\mathrm{MC}_{i,j}$, the $\mathrm{IC}_{i,j}$ and the $\mathrm{RMM}_{i,j}$ are unfailed. The CM_i is operational if at least two memory modules are operational and two CPUC_i, one PTC_i, and one RCM_i are unfailed. The system is operational if at least two computing modules are operational and one RSYS is unfailed.

The fault tree has 37 inputs, all of which are modules, 25 gates, 13 of which are modules, and 73 edges. The fault tree is defined by the logical expressions:

$$\begin{aligned}
FM_{i,j} &= T_{i,j} \vee U_{i,j} \vee V_{i,j}\,, \quad 1 \le i,j \le 3\,;\\
FMM_{i,l,k} &= FM_{i,l} \wedge FM_{i,k}\,, \quad 1 \le i \le 3,\ 1 \le l < k \le 3\,;\\
FC_i &= FMM_{i,1,2} \vee FMM_{i,1,3} \vee FMM_{i,2,3} \vee W_i \vee X_i \vee Y_i\,, \quad 1 \le i \le 3\,,;\\
FCC_{i,j} &= FC_i \wedge FC_j\,, \quad 1 \le i < j \le 3\,;\\
g_r &= Z \vee FCC_{1,2} \vee FCC_{1,3} \vee FCC_{2,3}\,.
\end{aligned}$$

The bags associated with the inputs of the fault tree are: $\mathrm{b}(T_{i,j}) = \mathrm{MC}_{i,j}[3]$, $\mathrm{b}(U_{i,j}) = \mathrm{IC}_{i,j}[1]$, $\mathrm{b}(V_{i,j}) = \mathrm{RMM}_{i,j}[1]$, $\mathrm{b}(W_i) = \mathrm{CPUC}_i[2]$, $\mathrm{b}(X_i) = \mathrm{PTC}_i[2]$, $\mathrm{b}(Y_i) = \mathrm{RCM}_i[2]$, and $\mathrm{b}(Z) = \mathrm{RSYS}[4]$. Also, $L = \widetilde{L} = 4$ and $|MC| = 2{,}701$.

The numerical results have been obtained for the parameter values: $\lambda_{\mathrm{MC}_1} = 10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{MC}_2} = 2\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{MC}_3} = 3\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{IC}_1} = 2\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{IC}_2} = 4\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{IC}_3} = 6\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{CPUC}_1} = 6\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{CPUC}_2} = 1.2\times10^{-6}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{CPUC}_3} = 1.8\times10^{-6}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{PTC}_1} = 6\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{PTC}_2} = 1.2\times10^{-6}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{PTC}_3} = 1.8\times10^{-6}\,\mathrm{h}^{-1}$, $\nu = 0.2$, $C_{\mathrm{MC}} = 0.99$, $C_{\mathrm{MM}} = 0.95$, $C_{\mathrm{CPUC}} = 0.99$, $C_{\mathrm{PTC}} = 0.97$, and $C_{\mathrm{CM}} = 0.95$.

5.2 Example 2 (Scenario 2)

The system has 60 components and its architecture is sketched in Figure 7. The system has four processing clusters that communicate through two independent double-ring networks, A and B; both networks have the same structure.

Processing cluster i, $0 \le i \le 3$ includes three s-identical processing units PU_i. Network A includes eight nodes NA_i, $0 \le i \le 7$. Nodes NA_i and $\mathrm{NA}_{(i+1)\ \mathrm{mod}\ 8}$ communicate through direct (clockwise) and reverse (counter-clockwise) links DA_i and RA_i, respectively. Network B includes eight nodes NB_i, $0 \le i \le 7$. Nodes NB_i and $\mathrm{NB}_{(i+1)\ \mathrm{mod}\ 8}$ communicate through direct and reverse links DB_i and RB_i, respectively.

The operational configuration of the system includes two processing units from the processing clusters with two or three unfailed processing units, one processing unit from the

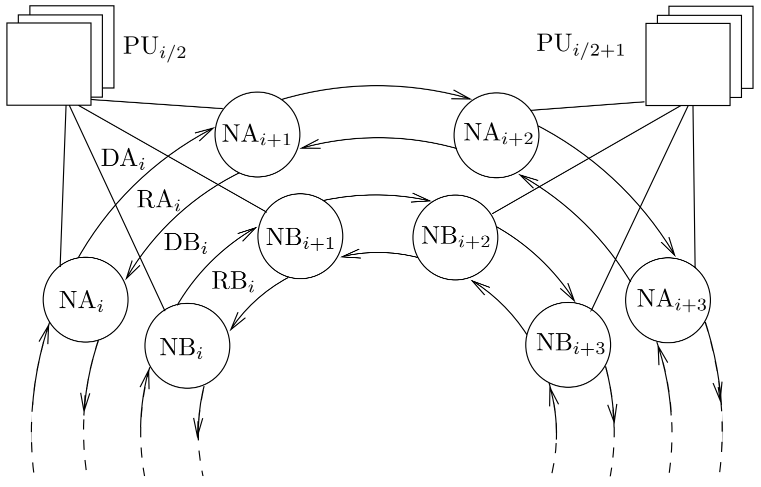

Figure 7: Architecture of example 2.

processing clusters with one unfailed processing unit, and the components of network A or B, with priority given to network A, required to build one of the operational configurations of the networks, described next in the order they are tried:
· A direct ring including all nodes and direct links.
· A reverse ring including all nodes and reverse links.
· A configuration, which is used when parallel direct and reverse links i are failed, including all nodes and links except the two failed links.
· A configuration, which is used when, for instance, node i fails, including all nodes except node i, and all links except those between node i and nodes $(i \pm 1) \bmod 8$.

When one of those network operational configurations is built, the corresponding network is said to be operational. The components included in the system operational configuration are called active. Active processing units, active nodes and active links fail with rates λ_{PU}, λ_N and λ_L, respectively. Inactive processing units fail with rate $\nu \times \lambda_{\mathrm{PU}}$, where ν, $0 < \nu < 1$ is a dormancy factor. Inactive nodes and links of network A fail, respectively, with rate $\nu \times \lambda_N$ and $\nu \times \lambda_L$ when the network is operational and do not fail otherwise. Inactive nodes and links of network B fail, respectively, with rate $\nu \times \lambda_N$ and $\nu \times \lambda_L$. Coverage is perfect for link faults. Faults in active processing units and nodes are covered with probabilities C_{PU} and C_N, respectively. Coverage faults are modeled by adding three fictitious components RSYS as explained in Section 2 and propagating to all of them any uncovered fault.

The system is operational if each processing cluster has at least one unfailed processing unit, one of the previously described operational network configurations for either network A or network B can be built, and at least one RSYS is unfailed.

The system fault tree has 53 inputs, all of which are modules, 40 gates, 4 of which are modules, and 764 edges. The fault tree is described by:

$$
\begin{aligned}
DRA &= \bigvee_{i=0}^{7} \big(S_i \vee T_i\big)\,, \\
DRB &= \bigvee_{i=0}^{7} \big(V_i \vee W_i\big)\,, \\
RRA &= \bigvee_{i=0}^{7} \big(S_i \vee U_i\big)\,, \\
RRB &= \bigvee_{i=0}^{7} \big(V_i \vee X_i\big)\,, \\
FRA_i &= \bigvee_{j=0}^{7} S_j \vee \bigvee_{\substack{j=0\\ j\neq i}}^{7} \big(T_j \vee U_j\big)\,, \\
FRB_i &= \bigvee_{j=0}^{7} V_j \vee \bigvee_{\substack{j=0\\ j\neq i}}^{7} \big(W_j \vee X_j\big)\,, \\
FRA_i^* &= \bigvee_{\substack{j=0\\ j\neq i}}^{7} S_j \vee \bigvee_{\substack{j=0\\ j\neq i,(i-1) \bmod 8}}^{7} \big(T_j \vee U_j\big), \\
FRB_i^* &= \bigvee_{\substack{j=0\\ j\neq i}}^{7} V_j \vee \bigvee_{\substack{j=0\\ j\neq i,(i-1) \bmod 8}}^{7} \big(W_j \vee X_j\big), \\
NETA &= DRA \wedge RRA \wedge \bigwedge_{i=0}^{7} FRA_i \wedge \bigwedge_{i=0}^{7} FRA_i^*\,, \\
NETB &= DRB \wedge RRB \wedge \bigwedge_{i=0}^{7} FRB_i \wedge \bigwedge_{i=0}^{7} FRB_i^*\,, \\
NET &= NETA \wedge NETB, \\
g_r &= NET \vee Z \vee \bigvee_{i=0}^{3} Y_i\,,
\end{aligned}
$$

The bags associated with the inputs of the fault tree are: b(S_i) = NA$_i$[1], b(T_i) = DA$_i$[1], b(U_i) = RA$_i$[1], b(V_i) = NB$_i$[1], b(W_i) = DB$_i$[1], b(X_i) = RB$_i$[1], b(Y_i) = PU$_i$[3], and b(Z) = RSYS[3]. Also, $L = 3$, $\widetilde{L} = 2$ and $|MC| = 32{,}405$.

The numerical results have been obtained for the parameter values: $\lambda_{\text{PU}} = 10^{-6}\,\text{h}^{-1}$, $\lambda_N = 5 \times 10^{-7}\,\text{h}^{-1}$, $\lambda_L = 3 \times 10^{-7}\,\text{h}^{-1}$, $\nu = 0.2$, $C_{\text{PU}} = 0.99$ and $C_N = 0.99$.

5.3 Results and Discussion

We use $K = 2, 3, 4, 5$ for example 1 and $K = 2, 3, 4$ for example 2.

Figures 8 and 9 show the unreliability bounds obtained using SC-BM for examples 1

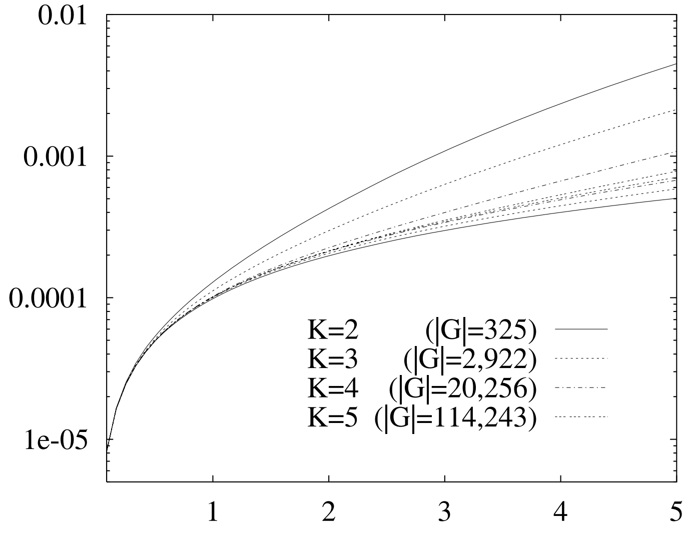

Figure 8: Unreliability bounds for example 1 as a function of time (years) and K.

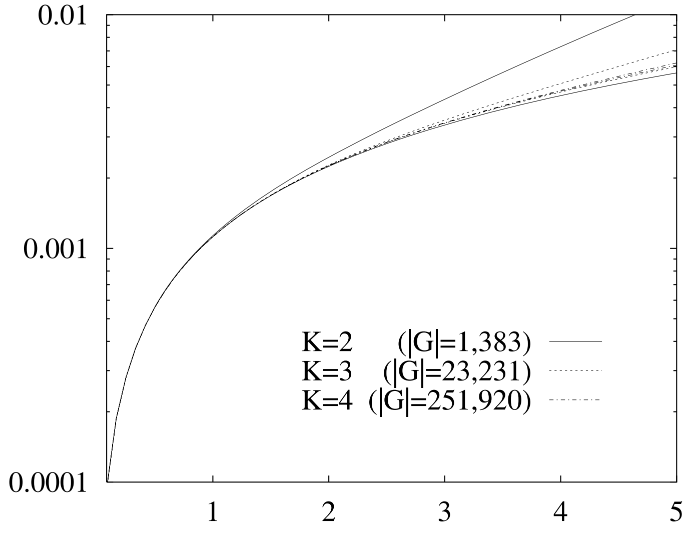

Figure 9: Unreliability bounds for example 2 as a function of time (years) and K.

& 2, respectively, as a function of time (in years).[2] The bounds degrade as time increases. In both examples, however, SC-BM achieves tight bounds for mission times up to 5 years using affordable numbers of states.

Tables 3 and 4 compare, for, respectively, examples 1 & 2, and several mission times, the relative unreliability band obtained with SC-BM,

$$\mathrm{urb}(t) = \frac{[\mathrm{ur}(t)]_{\mathrm{ub}} - [\mathrm{ur}(t)]_{\mathrm{lb}}}{[\mathrm{ur}(t)]_{\mathrm{lb}}}$$

against that obtained with BM-1,

$$\mathrm{urb}'(t) = \frac{[\mathrm{ur}(t)]'_{\mathrm{ub}} - [\mathrm{ur}(t)]_{\mathrm{lb}}}{[\mathrm{ur}(t)]_{\mathrm{lb}}}$$

[2]1 month = 730 hours;1 year = 8,760 hours.

Table 3: Example 1: relative unreliability band obtained with SC-BM, urb(t) (top), and T-SC-BM, urb$''(t)$ (bottom).

time	K ($\lvert G\rvert$)			
	2 (325)	3 (2,922)	4 (20,256)	5 (114,243)
1 month	5.6074×10^{-3}	9.0751×10^{-4}	2.4718×10^{-5}	2.4558×10^{-7}
	1.9121×10^{1}	5.3599×10^{-1}	5.9234×10^{-3}	5.1234×10^{-5}
2 months	1.4916×10^{-2}	3.4877×10^{-3}	1.1790×10^{-4}	1.7226×10^{-6}
	1.9310×10^{1}	7.5534×10^{-1}	1.4218×10^{-2}	1.9082×10^{-4}
6 months	8.8675×10^{-2}	3.0034×10^{-2}	1.7085×10^{-3}	5.1856×10^{-5}
	2.0721×10^{1}	1.6262	7.0069×10^{-2}	2.0870×10^{-3}
1 year	3.0844×10^{-1}	1.1625×10^{-1}	1.0286×10^{-2}	5.2415×10^{-4}
	2.4590×10^{1}	2.9683	2.1744×10^{-1}	1.1154×10^{-2}
2 years	1.1610	4.4492×10^{-1}	6.3144×10^{-2}	5.5772×10^{-3}
	3.7307×10^{1}	6.0617	7.2137×10^{-1}	6.4111×10^{-2}
5 years	7.9242	2.6360	6.0118×10^{-1}	1.0547×10^{-1}
	9.5814×10^{1}	1.9943×10^{1}	3.6513	6.0539×10^{-1}

and that obtained with T-SC-BM,

$$\text{urb}''(t) = \frac{[\text{ur}(t)]''_{\text{ub}} - [\text{ur}(t)]_{\text{lb}}}{[\text{ur}(t)]_{\text{lb}}} .$$

The relative band urb$'(t)$ is not shown in Table 3 because, for example 1, $[\text{ur}(t)]'_{\text{ub}} = [\text{ur}(t)]_{\text{ub}}$ and, therefore, urb$'(t) = $ urb(t).

SC-BM outperforms significantly T-SC-BM in terms of bounds tightness. Thus, for mission times up to 1 year, the ratio urb$''(t)/$ urb(t) is greater than or equal to 21 for example 1 and 30 for example 2. In addition, SC-BM can compute bounds that are almost as tight or even tighter than those given by T-SC-BM using appreciably fewer states. Thus, for example 1 and $t = 2$ years, the relative band obtained by SC-BM with $K = 4$ ($|G| = 20{,}256$) is better than that obtained by T-SC-BM with $K = 5$ ($|G| = 114{,}243$). For example 2 and $t = 2$ years, the relative band obtained by SC-BM with $K = 3$ ($|G| = 23{,}231$) is only slightly worse than that obtained by T-SC-BM with $K = 4$ ($|G| = 251{,}920$).

For example 1, SC-BM and BM-1 give exactly the same bounds. For example 2, the bounds obtained by SC-BM are only slightly worse than the bounds obtained by BM-1.

Table 5 gives the CPU times in seconds for $t = 5$ years for examples 1 & 2 and all three methods. The times were measured on a 167 MHz, 128 MB SPARC Ultra 1 workstation. As discussed in Section 4, with respect to T-SC-BM, SC-BM can introduce a significant CPU time overhead due to computing lower bounds for failure distances from states only when $\widetilde{L} > 1$. That is the case for examples 1 & 2 and the results given in Table 5 indicate that the overhead is reasonable, ranging from 20% for example 2 and $K = 4$ to 40% for example 1 and $K = 3$. In addition, SC-BM is always significantly faster than BM-1.

Table 4: Example 2: relative unreliability band obtained with SC-BM, urb(t) (top), BM-1, urb$'$(t) (middle), and T-SC-BM, urb$''$(t) (bottom).

time	K (|G|)		
	2 (1,383)	3 (23,231)	4 (251,920)
1 month	8.0995×10^{-6} 4.2519×10^{-6} 3.4966×10^{-3}	2.7765×10^{-8} 1.9822×10^{-8} 1.0013×10^{-5}	5.9186×10^{-11} 5.1065×10^{-11} 2.1062×10^{-8}
2 months	6.4332×10^{-5} 3.4014×10^{-5} 1.3816×10^{-2}	4.3975×10^{-7} 3.1450×10^{-7} 7.9310×10^{-5}	1.8731×10^{-9} 1.6169×10^{-9} 3.3356×10^{-7}
6 months	1.6870×10^{-3} 9.1670×10^{-4} 1.1952×10^{-1}	3.4190×10^{-5} 2.4625×10^{-5} 2.0576×10^{-3}	4.3531×10^{-7} 3.7652×10^{-7} 2.5930×10^{-5}
1 year	1.2903×10^{-2} 7.2810×10^{-3} 4.5153×10^{-1}	5.1391×10^{-4} 3.7395×10^{-4} 1.5491×10^{-2}	1.3015×10^{-5} 1.1291×10^{-5} 3.8967×10^{-4}
2 years	9.4070×10^{-2} 5.6666×10^{-2} 1.6114	7.2234×10^{-3} 5.3589×10^{-3} 1.0930×10^{-1}	3.6184×10^{-4} 3.1572×10^{-4} 5.4737×10^{-3}
5 years	1.0991 7.6243×10^{-1} 7.1970	1.8392×10^{-1} 1.4348×10^{-1} 1.1300	2.1999×10^{-2} 1.9505×10^{-2} 1.3744×10^{-1}

Table 5: CPU time in seconds to generate the CTMC and compute the unreliability bounds for $t = 5$ years for both examples using SC-BM (top), BM-1 (middle) and T-SC-BM (bottom).

example	K			
	2	3	4	5
first	0.271 1.72 0.197	2.12 4.38 1.52	14.8 29.3 11.0	90.4 268. 73.2
second	3.21 16.4 2.42	44.8 68.8 36.4	510. 998. 427.	— — —

To conclude, SC-BM seems to give significantly tighter bounds than T-SC-BM and seems to be only slightly slower than T-SC-BM, when it is not as fast or faster ($\widetilde{L} = 1$). Compared with BM-1, when the condition of Theorem 3 is satisfied or $L = 1$, SC-BM is guaranteed to give exactly the same bounds as BM-1, and, in those cases, SC-BM is definitely better, since it does not require the computation of MC, does not incur the memory overhead associated with the holding of MC and related data structures for failure distance computation [1], and seems to be faster. When the condition of Theorem 3 is not satisfied and $L > 1$, SC-BM will give, in general, less tighter bounds than BM-1, but, since SC-BM does not require the computing of MC, which is an NP-hard problem, there are cases in which SC-BM applies while BM-1 does not. In addition, even when MC can be computed, the memory overhead in BM-1 associated with MC is large if $|MC|$ is large, and, then, SC-BM might be better than BM-1 when considering the tradeoff between memory consumption and bounds tightness.

Acknowledgments

This work was partially supported by the "Comisión Interministerial de Ciencia y Tecnología" (CICYT) of the Ministry of Science and Technology of Spain under the research grant TIC95-0707-C02-02.

References

[1] V. Suñé and J. A. Carrasco, "A method for the computation of reliability bounds for non-repairable fault-tolerant systems," in *Proc. 5th IEEE Int. Symp. on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS'97)*, (Haifa, Israel), pp. 221–228, January 1997.

[2] M. A. Boyd, M. Veeraraghavan, J. B. Dugan, and K. Trivedi, "An approach to solving large reliability models," in *Proc. of the AIAA/IEEE 8th Conf. on Embedded Digital Avionics*, pp. 243–250, 1988.

[3] J. L. Peterson, "Computation sequence sets," *Journal of Computer and System Sciences*, vol. 13, pp. 223–252, August 1976.

[4] S. M. Ross, *Stochastic Processes.* John Wiley & Sons, 1983.

[5] R. A. Sahner and K. S. Trivedi, "Reliability modeling using SHARPE," *IEEE Transactions on Reliability*, vol. R-36, pp. 186–193, June 1987.

[6] D. Lee, J. Abraham, D. Rennels, and G. Gilley, "A numerical technique for the hierarchical evaluation of large, closed fault-tolerant systems," in *Proc. 3rd Working Conf. on Dependable Computing for Critical Applications*, pp. 95–114, Springer-Verlag, 1992.

[7] J. B. Dugan, "Fault trees and imperfect coverage," *IEEE Transactions on Reliability*, vol. 38, pp. 177–185, June 1989.

[8] S. A. Doyle and J. B. Dugan, "Dependability assessment using binary decision diagrams (BDDs)," in *Proc. 25th IEEE Int. Symp. on Fault-Tolerant Computing FTCS-25*, pp. 249–258, 1995.

[9] J. Carrasco and V. Suñé, "An algorithm to find minimal cuts of coherent fault trees with event classes, using a decision tree," *IEEE Transactions on Reliability*, vol. 48, pp. 31–41, March 1999.

[10] A. Rosenthal, "A computer scientist looks at reliability computations," in *Reliability and Fault Tree Analysis* (R. Barlow, J. Fusell, and N. Singpurwalla, eds.), pp. 133–152, Philadelphia, USA: SIAM, 1975.

[11] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research*. McGraw-Hill, 1982.

[12] T. Kohda, E. J. Henley, and K. Inoue, "Finding modules in fault trees," *IEEE Transactions on Reliability*, vol. 38, pp. 165–176, June 1989.

[13] Y. Dutuit and A. Rauzy, "A linear-time algorithm to find modules of fault trees," *IEEE Transactions on Reliability*, vol. 45, pp. 422–425, September 1996.

[14] D. Gross and D. Miller, "The randomization technique as a modeling tool and solution procedure for transient Markov processes," *Operations Research*, vol. 32, pp. 343–361, March-April 1984.