Failure Distance Based Bounds of Dependability Measures

Víctor Manuel Suñé Socías

Departament d'Enginyeria Electrònica
Universitat Politécnica de Catalunya

Doctoral dissertation
submitted in fulfillment of the requirements for the
Doctor degree in Industrial Engineering

Barcelona, May 2000

Contents

Preface

A la meva família.

The topic of this dissertation is efficient bounding methods for Markovian dependability models of fault-tolerant systems. Ultimately, the complexity of the fault-tolerant systems for which reasonably tight bounds can be computed is limited by the memory consumption of the methods, which depends on their efficiency. The work described in this dissertation has as baseline the idea that by exploiting the failure distance concept more efficient bounding methods could be developed. The research described in this dissertation has confirmed that intuitive idea. Although a lot of work remains to be done, we think that the work described in this dissertation is a good starting point for what is left.

I am grateful to Prof. Juan A. Carrasco for having introduced me in the area of this dissertation (Markovian modeling of fault-tolerant systems), for having proposed me such an exciting research topic, and for his help along the work, especially in the more difficult spots.

Chapter 1

Introduction

Increasing complexity of information processing systems and increasing importance of the dependability of those systems has fostered the use of fault-tolerance. Fault-tolerance is a methodology to achieve high dependability using components with moderate reliability and, thus, moderate cost. Fault-tolerance is achieved through the use of redundancy, that is information, resources, or time beyond what is strictly required for operation in the absence of faults. Redundancy can take one of several forms [54]:

- **Hardware redundancy**: extra hardware, usually for the purpose of either detecting or tolerating faults.
- **Software redundancy**: extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults.
- **Information redundancy**: extra information beyond that required to implement a given function; for example, error detecting codes.
- **Time redundancy**: time to perform functions such that fault detection and often fault tolerance can be achieved.

Through the use of redundancy a fault-tolerant system is able to detect and tolerate faults, so that the system still continues in correct operation in the presence of faulty components. Redundancy implies extra cost over a non-fault-tolerant system. However, fault-tolerance is an attractive alternative to the use of highly reliable components with high cost, and often results in less costly systems with similar or higher dependability. It is for this reason that fault-tolerance has achieved increasing popularity and its use

is not restricted to applications requiring very high dependability. Unmanned space flights, satellites, avionics, nuclear power plants, distributed systems, communication networks, routers, multiprocessors, interconnection networks and high-density memories are examples of application areas where fault-tolerance has found wide acceptance.

1.1 Fault-Tolerant Systems and Modeling

A fault-tolerant system is designed to achieve a certain degree of dependability. To achieve such a degree of dependability, it is necessary to put redundancy in the system. The addition of redundancy to the system increases its cost. Under that perspective, the design and implementation of a fault-tolerant system can be seen as an optimization problem, with the goal of achieving the desired dependability level at the minimum cost. Fault-tolerance is an art, making the problem very difficult. In practice, the design and implementation of a fault-tolerant system is strongly guided by heuristics and previous experience. Nevertheless, usually several candidate architectures with different degrees of redundancy are considered and their cost and dependability are analyzed. The one that better solves the optimization problem is the one selected at the considered design detail. The process continues till the design of the system is completed. After the system is implemented, it is necessary to certificate that the system achieves the required dependability level.

Dependability is a generic concept encompassing many measures, such as the steady-state availability, the mean time to failure, and the reliability. Depending on the application, one or several of such measures will be the more appropriate ones and will be selected to quantify system dependability. Often, the system has graceful degradation. This means that, as faults occur, the performance of the system degrades before failing. For that class of systems, a combined evaluation of the system's performance and dependability is appropriate. Such a combined measure is generically called *performability* [69]. Performability is itself a generic concept (as dependability is) and can be particularized into many measures. Many of those measures can be defined assuming a reward structure over a stochastic process [87]. In summary, the system ability to offer correct service is quantified using some or several dependability/performability measures.

Experimental quantification of the selected dependability/performability measure is difficult or impossible. It is clearly impossible in a design stage, since the real system is not available for experimentation. Even when the real system or a prototype is available, direct experimental evaluation of system's dependability/performability is often extremely

difficult. The reason is that, if the fault-tolerant system is well designed, system failures will be rare events and the time needed for an experimental evaluation of system's dependability/performability will be very long. Several failures of the system will have to be seen to have enough confidence on the estimation of the measure, making the required experimentation time even longer. From a practical point of view, direct experimental quantification of a fault-tolerant system's dependability/performability is only feasible "a posteriori", when a sufficient number of copies of the system have been deployed and are in operation. For these reasons, evaluation of a fault-tolerant system's dependability/performability is usually accomplished using models. Since component faults, the efficiency of fault-tolerance mechanisms incorporated in the system, and maintenance actions have stochastic nature, stochastic models are used in the evaluation.

Stochastic models are not the only methodology required for the evaluation of dependability/performability of fault-tolerant systems. The efficiency of fault-tolerance mechanisms is characterized using the so-called *coverage* parameter [14], defined as the probability that the system recovers given that a fault occurs. Coverage is sensitive to low-level details of the system design and, although stochastic models help [38], in general, coverage cannot be evaluated using only stochastic models and fault-injection experiments are used. Fault injection can be done on the real system or a prototype of it or in a more or less (typically very) detailed model of the system. Several tools for performing fault-injection experiments have been developed [7, 9, 26, 29, 50, 51, 53, 55, 56]. The values of the coverage parameters obtained using fault-injection, perhaps in combination with stochastic models are then used as parameters of a global dependability/performability model of the fault-tolerant system that captures faults, fault recovery (through the use of the coverage parameters), maintenance, and, if a performability measure is of interest, perhaps performance activities. Coverage parameters are incorporated into that global model of the system as switching probabilities, using the so-called *behavioral decomposition* [97]. That approach is justified by the fact that fault-handling activities are much faster than the remaining activities of the system.

The importance of stochastic modeling in the evaluation of fault-tolerant systems has fostered the development of software tools with facilities for the specification and solution of the stochastic models: SURF [58], ARIES [68], HARP [37], SAVE [46], METFAC [16], SHARPE [86], SPNP [30], UltraSAN [34], SURF-2 [12], METFAC-2 [21], and Galileo [96], among others. Homogeneous continuous-time Markov chains (CTMC) are the most widely used type of stochastic process. They are the type of stochastic processes that result naturally in dependability models when fault production times and maintenance actions have exponential distributions. For performability models including performance

actions, CTMC arise naturally when performance actions, in addition to fault occurrence and maintenance actions, have exponential distributions. CTMC models allow also to capture phase type distributions [73], although the cost is a considerable increase of the number of states. Since many distribution functions can be approximated with arbitrary accuracy using phase type distributions, the theoretical power of CTMC is very high. Most of the stochastic modeling tools accept only CTMC models. Others accept also non-homogeneous continuous-time Markov chains, which are useful to model non-repairable fault-tolerant systems with components having non-exponential time to failure. Finally, a few allow general distributions for the modeled actions, but then the only available solution method is simulation. In this dissertation we will restrict our attention to CTMC dependability models having a predefined structure. That class of models will be detailed later on.

1.2 CTMC and the Largeness Problem

CTMC models allow to capture in a faithful way important details that fault-tolerant systems have: component failure rates depending on the state of the system, failure propagation, coverage, limited repairmen, repair priorities, etc. However, they suffer from the well-known *largeness problem* (also known as state space explosion problem). In brief, the problem is that the number of states of the CTMC grows fast with the complexity of the fault-tolerant system and is far beyond current and foreseeable computation capabilities for systems of some complexity. To illustrate the point, assume an availability model of a fault-tolerant system having N distinguishable components, which can be unfailed or failed. The up/down state of the system depends on the subset of components which are failed. A CTMC modeling that system will have 2^N states. For $N = 20$ the number of states is 1,048,576, which is about what can be held in memory in a current high-end workstation; for $N = 30$, the number of states is 1,073,741,824, which is far beyond current computation capabilities, but which, maybe, will be affordable in 20 years from now; for $N = 40$, the number of states is about 1.0995×10^{12}, which, we think, is far beyond what will be affordable in the future. As the examples used in the dissertation will show, there exist many fault-tolerant systems yielding CTMC models of that size, and for those systems an exact numerical solution of the corresponding CTMC is simply infeasible.

The largeness problem is well-known (it also arises in performance CTMC models) and researchers have worked against it. The most radical approach is the use of

combinatorial solution methods. Generally speaking, that approach is possible for some dependability measures when components have independent behavior. It is certainly possible for computing the steady-state availability and the reliability of non-repairable systems. Many combinatorial techniques (see, for instance, [1, 3]) exist when the up/down state of the system is determined solely by a structure function, usually specified by a fault tree. However, those techniques do not allow to consider coverage failures, which in fault-tolerant systems are important. Combinatorial methods allowing imperfect coverage have been developed recently [39, 35, 6]. Those methods are extremely efficient and should be used when they are applicable. Another approach is the use of hierarchical model solution techniques. Those techniques are possible when the fault-tolerant system can be decomposed in subsystems, each of them behaving independently of the others. As combinatorial techniques, hierarchical model solution is only possible for some dependability measures. The software tool SHARPE has the flexibility required to accommodate those solution techniques and the examples presented in [86] illustrate them very well.

A general approach to tackle the largeness problem is the state aggregation. That approach is possible when the system exhibits symmetries (i.e. components or subsystems with identical behavior). A posteriori state aggregation performed at the state level is computationally expensive and does not solve the largeness problem. However, techniques have been developed to generate directly the aggregated CTMC from a high-level system description that makes the symmetries explicit. A simple example of that approach is the use of the concept of component type in the SAVE package [46], which allows to capture symmetries at the component level. A more general approach, allowing to capture hierarchically symmetries at the subsystem as well as the component level is offered by the UltraSAN software tool in the context of Stochastic Activity Networks [88]. A more general approach has been also proposed in the context of Stochastic Petri Nets [28].

Simulation is an approach that by nature does not suffer from the largeness problem, since only the current state or, at most, the path to it from the initial state has to be kept in memory. Standard simulation methods work very poorly for dependability models due to the rarity of the system failure event, which makes enormous the number of model events that have to be sampled to achieve confidence intervals with reasonable accuracy. However, fast simulation methods specifically targeted at dependability models have been developed recently. Most of these methods use importance sampling to drive the simulation effort towards the more important paths (i.e. those paths that see system failure and have higher probabilities) [17, 18, 25, 32, 47, 48, 65, 72, 74, 75, 76, 92, 91, 99].

Fast balance likelihood ratio simulation methods have been proposed more recently [4, 5], which seem to perform better than importance sampling methods for systems with high degree of accuracy. However, although fast simulation is a feasible alternative (and maybe the only alternative in some cases), the results they give are subject to a fundamental criticism: there is no absolute guarantee that the estimator is close enough to the real value of the estimated dependability measure. There is, certainly, a statistical assessment of the error, but that assessment is itself based on an estimation of the variance of the estimator and the shape of the distribution of the estimator is not known. Other problem is the possibility of having estimators with infinite variance. In that case, the estimated variance of the estimator increases as the simulation progresses and the user may be deceived by the method, thinking that it has a good estimate of the dependability measure when, in fact, it has not.

Finally, another approach (and the one we pursue in this dissertation) is the use of bounding methods. Bounding methods are based on the fact that the probability mass of the model is often concentrated in a small subset of states (for repairable systems and non-repairable systems with not too long mission time, in the states with a few failed components) and that detailed knowledge of the CTMC is required only in this subset. Bounding methods generate the CTMC in a subset G of states and bound using high-level knowledge about the model the behavior of the CTMC outside G, yielding *bounds* for the dependability/performability measure. Note that in contrast with fast simulation methods, in bounding methods there is a strict guarantee for the error[1]. Bounding methods have experimented a great development in the last years and will be briefly reviewed in the next section.

1.3 Bounding Methods

As it has been said in the previous section, in bounding methods a subset G of the state space of the CTMC is generated and the behavior in the remaining states is bounded somehow, yielding bounds for the desired dependability/performability measure. The bounds are tight if the global probability of the non-generated portion of the state space is small enough. Bounding transient dependability/performability measures is relatively simple. Consider, for instance, bounding the unreliability at time t, $ur(t)$. The exact measure can be computed by generating a CTMC $X = \{X(t); t \geq 0\}$ with state space

[1] Of course, the "bounding" model is solved using numerical methods that are subject to roundoff errors; however, typically, the numerical methods are stable and the resulting relative error in the computed bounds can be neglected, if double precision is used in the computations.

$\Omega \cup \{f\}$, where Ω is the set of states in which the system is operational and entry into the absorbing state f means system failure. Then, $ur(t) = P[X(t) = f]$. For simplicity assume $P[X(0) \in \Omega] = 1$. Then bounds for $ur(t)$ can be obtained by using a CTMC $X' = \{X'(t); t \geq 0\}$ with state space $G \cup \{f, a\}$, where G is a subset of Ω and f and a are absorbing states, having within G the same initial probability distribution as X, with $P[X'(0) = a] = P[X(0) \in \Omega - G]$, having within G and from G to f the same transition rates as X, and having transition rates from states in G to a equal to the transitions rates of the corresponding states in G to $\Omega - G$ in X. Then, it is easy to prove that $P[X'(t) = f]$ is a lower bound for $ur(t)$ and $P[X'(t) \in \{f, a\}]$ is an upper bound for $ur(t)$.

Bounding steady-state measures is much more complex and several methods have been developed in the last few years [19, 20, 22, 24, 63, 64, 66, 67, 70, 89, 95]. In the first of such methods [70], bounds for the steady-state availability were obtained by partitioning the non-generated portion of the state space, U, according to the number of failed components and bounding the behavior of the chain in U using upper bounds for failure transition rates and lower bounds for repair transition rates. The method was, however, computationally very costly because a linear system of size $\approx |G|$ had to be solved for each return state, i.e. each state through which G can be entered from U. In the same paper, a state cloning technique is proposed that reduces the number of linear systems that have to be solved but introduces some looseness in the bounds. The method proposed in [70] is not restricted to steady-state availability models and can encompass any finite CTMC having the same structure. Of course, the bounds will be tight only if the "repair" transition rates are much larger than the "failure" transition rates. In [63] a refinement of the method is proposed for the particular case in which all states but the one without failed components are cloned. The technique avoids a complete reapplication of the algorithm each time G is enlarged in the search for the desired accuracy but looses up further the bounds. This additional looseness has been reduced in another paper from the same authors [64]. In the method proposed in [19], the bounds of [70] are computed without cloning states solving only four linear systems of size $|G|$. In addition, for steady-state availability models, iterative methods are very fast for the solution of those four linear systems. In [89] another bounding method is developed in which the bounds are iteratively refined using detailed knowledge about the model in U in the neighborhood of G. No comparison was done with the method proposed in [70]. An important disadvantage of the method is the need for detailed knowledge of the model in U in the neighborhood of G, since that detailed knowledge can only be obtained by generating states in U, increasing significantly the memory requirements of

the method. In [22] a bounding method based on the failure distance concept is proposed that gives bounds that are never worse and are typically better than those given by the method proposed in [70]. The method uses the cloning technique of [70] but adapts one of the algorithms developed in [19] so that only five linear systems of size $|G|$ have to be solved to compute the bounds. The method proposed in [22] is specifically targeted at steady-state availability models. All previously reviewed methods assume that the state space of the CTMC is finite and that there is a repair transition involving a single component in all non-generated states of the CTMC. Both restrictions have been removed in the generalization of [70] proposed in [66, 67], which allows to obtain bounds for infinite CTMC and CTMC models in which no every state in U has "repair" transition. However, "repair" transitions have to involve a single component[2]. The method has been used to obtain bounds for the steady-state solution of queueing models. Another generalization of [70] for finite models has been recently proposed in [24]. In that method, group repair (i.e. the simultaneous repair of several components) and phase type repair distributions are allowed. In the methods reviewed so far G includes all states of the CTMC having up to K failed components. The issue of how to generate G so that it includes as few states as possible to achieve the required accuracy has also been investigated. In [95], state space exploration techniques were developed for the bounding method proposed in [70] with the cloning technique. However, these state space exploration techniques are expensive since they require the solution of a linear system of size $|G|$ after the expansion of every state. More efficient state space exploration techniques based on the concept of wave expansion and specifically targeted at the method developed in [22] have been developed in [20].

1.4 A Brief Review of Bag Theory

Throughout the dissertation we will use bag theory. In this section we will briefly review that theory. The review closely follows that provided in [78].

Bag theory is a natural extension of set theory. A bag, like a set, is a collection of elements over some domain $\mathcal{D}$. However, unlike a set, bags allow multiple occurrences (instances) of elements. Consider, for instance, the domain $\mathcal{D} = \{a, b, c, d\}$. Examples of bags over $\mathcal{D}$ are $b_1 = \{a, b, c\}$, $b_2 = \{a, b, c, c\}$, and $b_3 = \{b, c, b, c\}$. We will use the notation $b = c_1[n_1]c_2[n_2]\cdots c_k[n_k]$ for the bag containing exactly n_i instances of

[2]The subset U is conceptually partitioned into subsets U_k, $k \geq F+1$, where for steady-state availability models U_k would include the states in which there are k failed components and the transitions from U_k to the left, i.e. to $G \cup \cup_{l<k} U_l$, have to go to U_{k-1} if $k > F+1$ or to G if $k = F+1$.

element c_i, $1 \leq i \leq k$. With that notation, the previous example bags are described as $b_1 = a[1]b[1]c[1]$, $b_2 = a[1]b[1]c[2]$, and $b_3 = b[2]c[2]$.

In set theory, the basic concept is membership. In bag theory, that concept is replaced by the *number of instances* function. For an element $x \in \mathcal{D}$ and a bag b, we denote the number of instances of x in b by $\#(x, b)$. The *cardinality* $|b|$ of a bag b is simply the number of instances of elements in the bag, i.e.

$$|b| = \sum_{x \in \mathcal{D}} \#(x, b).$$

An element x is a member of a bag b if $\#(x, b) > 0$. This is denoted as $x \in b$. If $\#(x, b) = 0$, then $x \notin b$. The empty bag, $\emptyset$, is the bag with no members, i.e. $\#(x, \emptyset) = 0$, $x \in \mathcal{D}$. A bag a is a *subbag* of a bag b (denoted $a \subset b$) if every element of a is also an element of b at least as many times, i.e. if and only if $\#(x, a) \leq \#(x, b)$, $x \in \mathcal{D}$. Two bags a and b are equal (denoted $a = b$) if and only if $\#(x, a) = \#(x, b)$, $x \in \mathcal{D}$.

The following four operations are defined on bags:

bag union $a \cup b$ is the bag defined as

$$\#(x, a \cup b) = \max\{\#(x, a), \#(x, b)\}, \quad x \in \mathcal{D}.$$

bag intersection $a \cap b$ is the bag defined as

$$\#(x, a \cap b) = \min\{\#(x, a), \#(x, b)\}, \quad x \in \mathcal{D}.$$

bag sum $a + b$ is the bag defined as

$$\#(x, a + b) = \#(x, a) + \#(x, b), \quad x \in \mathcal{D}.$$

bag difference $a - b$ is the bag defined as

$$\#(x, a - b) = \#(x, a) - \#(x, a \cap b) = \max\{\#(x, a) - \#(x, b), 0\}, \quad x \in \mathcal{D}.$$

These operators have most of the properties that would be expected. Union, intersection, and sum are commutative and associative. In addition, the expected inclusions hold:

$$a \cap b \subset a \subset a \cup b,$$

$$a - b \subset a \subset a + b.$$

The distinction between union and sum is clearly stated by

$$|a \cup b| \leq |a| + |b|,$$

$$|a + b| = |a| + |b|.$$

No such simple statement distinguishes $a \cap b$ from $a - b$.

1.5 Class of Models Considered in the Dissertation

The topic of this dissertation is bounding methods for dependability measures. The bounding methods we will develop apply to a certain, wide class of dependability models. That class of models is described formally in this section. We will also discuss how coverage failures, which are important in the modeling of fault-tolerant systems, can be accommodated within the assumed modeling framework. Also, as it has been commented in Section 1.3, bounding methods require some high-level knowledge about the behavior of the CTMC model out of the generated state space G. The high-level knowledge required by the bounding methods developed in the dissertation will also be detailed in this section.

We consider dependability models that result from conceptualizing a fault-tolerant system as made up of components (hardware or software). In order to capture elementary symmetries, we allow component classes, a particular component being an instance of some component class. Formally, the system includes a bag C of component classes over a certain domain $\mathcal{D}$. We assume that the up/down state of the system is determined from the unfailed/failed state of the components of the system through a coherent structure function (see, for instance, [8]) specified by a fault tree with basic event classes made up of OR and AND gates and with inputs atoms of the form $c[n]$, $c \in C$, $n \geq 1$, which evaluate to true if and only if at least n instances of component class c are failed. We allow arbitrary connections between the gates of the fault tree (as far as the fault tree has only one output and it has not feedback). As an example, Figure 1.1 gives the fault tree corresponding to a fault-tolerant system that is failed if and only if one instance of component class a and two instances of component class b or two instances of component class a and one instance of component class b are failed.

The CTMC models we will consider have "failure" and, for repairable systems, "repair" transitions. Each state s of the CTMC (except the absorbing state indicating failure of the system when the measure under consideration is the unreliability at time t, $ur(t)$) has associated with it a bag of failed component classes $F(s)$. There is a single

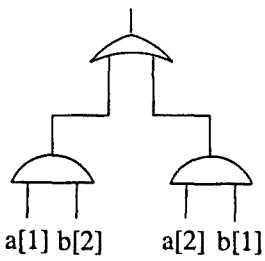

Figure 1.1: Example fault tree.

state o without failed components ($F(o) = \emptyset$). Each failure transition of the CTMC has associated with it a bag of component classes (the components that fail when the transition is followed). Similarly, in the case of models of repairable systems, each repair transition has associated with it a bag of component classes (the components that are repaired when the transition is followed). In addition, for models of repairable systems, we assume that every state $s \neq o$ has some outgoing repair transition. Finally, we assume that the CTMC is finite.

The considered class of models is quite wide and encompasses, for instance, all CTMC models which can be specified in the SAVE modeling language [46], a well-known tool. For repairable systems we do not allow deferred repair, i.e. the deferring of repair till some condition such as having a number of failed components $\geq K$ is achieved. Note that the CTMC may have several states with the same bag of failed components, allowing the consideration of several failure modes for the same component class. Also, in the case of repairable systems, very complex repair policies are supported: limited repairmen, priorities, repair preemption, etc. Group repair, i.e. the simultaneous repair of several components is allowed.

One might think the class of models just described does not encompass coverage failures, which are important when modeling fault-tolerant systems. Such failures can be however captured making a "trick". The trick is to introduce a class of "recovery" components with n instances, at least one of which has to be unfailed for the system to be up, which do not fail on their own, and to which uncovered failures of other components are propagated, causing a system failure. Although $n = 1$ would suffice, that choice may reduce the failure distances from the operational states to a value smaller than the value they would have there were not coverage failures and result in a degradation of the bounds given by the methods. The advisable choice is to take for n a value equal to the *redundancy level* of the system, i.e. the minimum number of components that have to be failed for the system to be failed. In the case of repairable systems, coverage failures are typically recovered by a restart or similar action, which returns the system to a correct

state. Such actions can be modeled by the repair of all the recovery components. Then, the importance of allowing group repair is clear: without it, n should be chosen equal to 1 and the bounds would be worse. Also, the introduction of recovery components allows to model more complex situations such as the coverage failures that take a subsystem down.

Given a fault-tolerant system, we define a *failure bag* as any bag of component classes over C that is associated with some failure transition of the model of the system. Similarly, for repairable systems, a *repair bag* will be any bag of component classes over C that is associated with some repair transition of the model of the system. We detail next the high-level information that is required by the bounding methods developed in this dissertation. The required information is the following:

- $N = |C|$, i.e. the number of components of the system,
- set E of failure bags of the model,
- for each failure bag $e \in E$, an upper bound, $\lambda_{\mathrm{ub}}(e)$, for the sum of the failure transition rates from any state of the model that have associated with them the failure bag e,
- for $1 \leq k \leq N$, a lower bound, $g(k)$, for the total repair rate from any state with k failed components.

That information could be extracted from an appropriate high-level specification of the model, such as the model specification language available in the SAVE tool [46], perhaps with some user intervention. In our experiments we have used METFAC-2 [21] and have integrated the bounding methods with that tool. The model specification language of that tool is based on production rules that may have annotations. Such annotations have been used to specify the "failure" or "repair" nature of the transitions and the failure or repair bags associated with them. The tool generates the CTMC from a "start" state, which, when the bounding methods are used, is always the state o. In that context, it is easy to keep track of the bag of failed components associated with the states and to identify E. N and the upper bounds $\lambda_{\mathrm{ub}}(e)$, $e \in E$ are provided directly by the user. The lower bounds $g(k)$, $1 \leq k \leq N$ are obtained calling a function provided by the user that is compiled and linked to the tool.

1.6 Outline of the Dissertation

The subject of the dissertation are failure distance based bounding methods for CTMC dependability models. The failure distance from a state of the fault-tolerant system is defined as the *minimum number of components that have to fail in addition to those already failed to take the system down*. That concept was introduced in [18] to devise efficient importance sampling based simulation methods for CTMC dependability models and has been used in [22] to devise a bounding method for the steady-state availability. Both the simulation and the bounding methods significantly improved the performance of previous non-failure distance based simulation and bounding methods. Those encouraging results were the motivation for the work included in this dissertation. As we will show in Chapter 2, computing failure distances is an NP-hard problem. The currently proposed method to perform that computation requires the knowledge of the minimal cuts [3] of the fault tree of the system. However, we will prove in Chapter 2 that there is no polynomial algorithm that solves the problem of computing the minimal cuts of a fault tree with the assumed structure. Thus, for some fault-tolerant systems computation of the failure distances may be very costly or infeasible. Also, the number of minimal cuts can be extremely large, introducing a significant memory overhead. The recognition of those facts led us to the development of failure distance-based bounding methods not requiring exact failure distances (and, thus, knowledge of the minimal cuts), but easily computable lower bounds for them.

Two dependability measures/scenarios are considered in the dissertation:

1. The unreliability at time t, $ur(t)$, for non-repairable systems.

2. The steady-state unavailability, UA, for repairable systems.

The unreliability at time t, $ur(t)$ is defined as the probability that the system will have failed by time t. It is an important dependability measure for many systems and, in particular, for non-repairable systems. The measure can be computed by using a CTMC $X = \{X(t); t \geq 0\}$ with state space $\Omega \cup \{f\}$, where Ω includes all operational states of the system and entry in the absorbing state f means failure of the system. Then, clearly

$$ur(t) = P[X(t) = f].$$

[3] We use the term minimal cut instead of the more common minimal cutset because our minimal cuts are bags.

The steady-state unavailability, UA, is an important dependability measure for repairable fault-tolerant systems. It is defined as the steady-state probability that the system is down (failed). It can be computed by using an irreducible CTMC $X = \{X(t); t \geq 0\}$ with state space $\Omega = U \cup D$, where U is the subset of up states and D is the subset of down states as

$$UA = \lim_{t \to \infty} P[X(t) \in D].$$

As it has been explained in Section 1.2, the state spaces of the CTMC X can be far beyond available computational resources, mainly due to memory limitations. The goal of bounding methods is to obtain accurate enough estimations for the dependability measure using substantially smaller CTMC models, which, hopefully, can be held in memory. Bounds tightness increases as more states are generated and the efficiency of the bounding method (i.e. the extent to which it provides tighter bounds with the same number of states or, in other words, the extent to which fewer states have to be generated to obtain the same bounds tightness) ultimately determines the complexity of the fault-tolerant systems that can be evaluated with reasonable accuracy.

The outline of the dissertation is as follows. Chapter 2 analyzes and solves the problem of computing failure distances. First, we prove that the problem is NP-hard. Then, we describe typically efficient algorithms to compute the failure distances assuming the knowledge of the minimal cuts of the structure function of the system. The efficiency of that algorithm translates the "complexity" of the problem of computing failure distances to the problem of obtaining the minimal cuts of the fault tree of the system, a problem for which there is no polynomial algorithm. Currently proposed algorithms for computing minimal cuts assume standard fault trees with simple basic events, while our bounding methods use fault trees with basic event classes (a basic event class is the failure of a component of a component class). In Chapter 2 we develop an algorithm to compute the minimal cuts for fault trees with basic event classes. The algorithm seems to be efficient and has minimum memory requirements. Chapters 3 and 4 presents and analyzes bounding methods for the unreliability of non-repairable fault-tolerant systems. The method described in Chapter 3 requires the computation of exact failure distances. The method described in Chapter 4 uses easily computable lower bounds for failure distances. Both methods have the interesting property that the bounds are obtained from the transient solution of "bounding" CTMC, and, thus, the methods can be used with relative ease in any general-purpose Markovian modeling tool. Chapters 5 and 6 describe and analyze bounding methods for the steady-state unavailability of repairable fault-tolerant systems. The work described in those chapters has [22] and [20] as starting points. Chapter 5 generalizes the bounding method described in [22] by allowing group

repair, which, as commented in Section 1.5, is an important generalization. In addition, the efficiency of the method is increased by improving the failure rate bounding structures used in the method proposed in [22]. Chapter 6 describes a bounding method with the same generality as the bounding method described in the previous chapter that uses the easily computable lower bounds for failure distances derived in Chapter 4. Finally, Chapter 7 contains the conclusions of the dissertation and highlights related future work.

Chapter 2

Failure Distances and Minimal Cuts

In this chapter we build the basis for the bounding methods which will be developed in Chapters 3 and 5. We start by stating formally the problem of computing the failure distance from a state. Next, we prove that the problem is NP-hard. Then, we describe typically efficient algorithms to compute the failure distances assuming the knowledge of the minimal cuts of the structure function of the system. The efficiency of that algorithm translates the "complexity" of the problem of computing failure distances to the problem of obtaining the minimal cuts of the structure function of the system, a problem for which there is no polynomial algorithm. Currently proposed algorithms for computing minimal cuts assume standard fault trees with simple basic events, while our bounding methods use fault trees with basic event classes (a basic event class is the failure of a component of a component class). Therefore, we develop an algorithm to compute the minimal cuts for fault trees with basic event classes. The algorithm seems to be efficient and has minimum memory requirements.

2.1 The Problem of Computing the Failure Distance from a State

The bounding methods developed in the dissertation use the failure distance concept. Two of them require the computation of failure distances from states of the CTMC. As it has been mentioned in Section 1.6, the failure distance, $d(s)$, from a state s is the

minimum number of components that have to fail in addition to those already failed ($F(s)$) to take the system down. Since the up/down state of the system is determined by a fault tree $\mathcal{F}$, with the structure described in Section 1.5, we can paraphrase the problem as

Problem FD Given a set of component classes D, a fault tree $\mathcal{F}$ including AND and OR gates and having as inputs atoms of the form $c[n]$, $c \in D$, $n \geq 1$ that evaluate to true if and only if at least n instances of component class c are failed, and a bag over D, b, determine the minimum number of components that have to be failed in addition to those in b for the fault tree to evaluate to true.

That problem seems to be hard, in general. We will solve it assuming that the minimal cuts of the fault tree $\mathcal{F}$ are known. A cut is any bag over D such that the failure of the components in the bag makes $\mathcal{F}$ evaluate to true (i.e. takes the system down). A cut b is minimal if and only if no bag strictly contained in b is a cut. In Section 2.3 we will develop a typically efficient algorithm to compute the minimal cuts of the structure function represented by a fault tree $\mathcal{F}$. It is known that, for ordinary fault trees, having as inputs simple events, the problem of computing the minimal cuts is NP-hard [81]. In fact, we will prove that for our fault trees the problem is not in P (there is no polynomial algorithm that solves it). Then, a question arises: is our procedure to solve Problem FD based on the knowledge of the minimal cuts of $\mathcal{F}$ more inefficient than it should be? We will give next a partial answer to that question, proving that Problem FD is NP-hard. Our algorithms to solve Problem FD have linear complexity on the number of minimal cuts of $\mathcal{F}$ and, thus, are not polynomial, but, since Problem FD is NP-hard, there is no polynomial algorithm to solve it unless NP = P, which would mean that for all NP-complete problems there would exist a polynomial algorithm (see [43] for background).

Let MC be the set of minimal cuts of $\mathcal{F}$. Then, since a state is down if and only if the failed components in the state make $\mathcal{F}$ evaluate to true and a bag b of failed components imply $\mathcal{F}$ to true if and only if b includes some minimal cut, it is clear that the failure distance $d(s)$ from a state s can be computed as

$$d(s) = \min_{m \in MC} |m - F(s)|. \tag{2.1}$$

Use of (2.1) is expensive if the number of minimal cuts is large, as it can be. In Section 2.2 we will describe typically much more efficient algorithms to compute failure distances.

Our algorithms to compute failure distances require the knowledge of all minimal cuts of the fault tree. We have the following result.

Theorem 2.1 *Computing all minimal cuts of a fault tree with the assumed structure is a problem outside P, i.e. there is no polynomial algorithm to solve it.*

Proof It is enough to show the existence of a fault tree with a number of minimal cuts exponential on the size of the fault tree. To find such a fault tree consider a system with $n = 2^k$ components of different classes that is failed if and only if $n/2$ or more components are failed. The minimal cuts of such system are all collections with $n/2$ components, whose number is $\binom{n}{n/2} \geq 2^{n/2}$. In addition, a fault tree for the system of size polynomial on n can be built by considering the set of components partitioned into two subsets of cardinality $n/2$ and building networks with outputs that evaluate to true if and only if k or more components of the subset are failed, $k = 1, 2, \ldots, n/2$. Those outputs can be combined using AND and OR gates to obtain outputs which evaluate to true if and only if k or more components are failed, $k = 1, 2, \ldots, n$. The reduction procedure can be applied recursively. The resulting fault tree has a number of "stages" logarithmic on n, and each stage has size quadratic on n. Then, it follows that for such fault tree the number of minimal cuts is exponential on the size of the fault tree. □

In the remaining of this section we will prove that the FD problem is NP-hard. We will consider the following closely related decision problem:

Problem FDD Given a set of component classes D, a fault tree $\mathcal{F}$ including AND and OR gates and having as inputs atoms of the form $c[n]$, $c \in D$, $n \geq 1$ that evaluate to true if and only if at least n instances of component class c are failed, a bag over D, b, and an integer $K \geq 0$, is there any bag b' over D, $|b'| \leq K$ such that the failure of the components in $b + b'$ implies $\mathcal{F}$ to true?

We will also consider the well-known Vertex Cover problem, which is known to be NP-complete [43]:

Problem VC Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$, is there a vertex cover of size K or less for G, that is, a subset $V' \subset V$ such that $|V'| \leq K$ and, for each edge $\{u, v\} \in E$, at least one of u and v belongs to V'?

Regarding the complexity of Problem FDD we have the following result.

Proposition 2.1 *Problem FDD is NP-complete.*

Proof We will use the more common approach to prove NP-completeness results [43]. Specifically, we will show that Problem FDD is NP. Then, we will construct a polynomial transformation of Problem VC, which is NP-complete, into Problem FDD.

To show that Problem FDD is NP it is enough to prove the existence of a polynomial nondeterministic algorithm to solve it. Such an algorithm can be easily built as follows. Let $D_{\mathcal{F}}$ be the subset of D including the component classes c having some input $c[n]$ in $\mathcal{F}$ and, for each $c \in D_{\mathcal{F}}$, let $n_{\mathcal{F}}(c,b)$ be the set of integers $n \geq 1$ such that $\mathcal{F}$ has some input $c[n + \#(c,b)]$. Consider all bags b' over $D'_{\mathcal{F}} = \{c \in D_{\mathcal{F}} : n_{\mathcal{F}}(c,b) \neq \emptyset\}$ with $\#(c,b') \in n_{\mathcal{F}}(c,b)$, $c \in D'_{\mathcal{F}}$ and $|b'| \leq K$. Then, the algorithm proceeds by considering all such bags b' and, for each b', evaluate $\mathcal{F}$ with the components in $b + b'$ failed. Note that such evaluation is linear on the size of $\mathcal{F}$. If the fault tree evaluates to true for some bag b' the answer to the FDD problem is yes.

It remains to find a polynomial transformation from Problem VC to Problem FDD. To find the transformation, let $D = V$ and consider the fault tree $\mathcal{F}$ built as follows. $\mathcal{F}$ has as inputs atoms $u[1]$ for each $u \in V$. For each edge $\{u,v\} \in E$, $\mathcal{F}$ has an OR gate fed by two atoms: $u[1]$ and $v[1]$. All OR gates feed an AND gate, whose output is the output of $\mathcal{F}$. Then, Problem FDD is invoked with set of component classes D, fault tree $\mathcal{F}$, $b = \emptyset$, and integer K (the integer K of Problem VC). By construction, it is clear that the answer to Problem VC is yes if and only if the answer to the invoked FDD problem is yes. But $|D|$ and the size of $\mathcal{F}$ are linear on the size of G and, therefore, the transformation is polynomial. □

Using Proposition 2.1 we have the desired result.

Theorem 2.2 *Problem FD is NP-hard.*

Proof Problem FDD can obviously be polynomically transformed into Problem FD, which by Proposition 2.1 is NP-complete. □

2.2 Algorithms to Compute Failure Distances

Let G be the set of generated states. The implementation of the bounding methods developed in this dissertation requires the computation of the failure distance from the states that are successors (are reached through a single transition) of a generated state $s \in G$. A trivial computation of these failure distances based on (2.1) can be time consuming if the number of minimal cuts is large. In this section we review, for the sake of completeness, significantly more efficient algorithms described in [20].

We start with the observation that most of the transitions leading to the state whose failure distance has to be computed will be typically of the failure type. To compute more

efficiently the failure distances associated with these states the concept of *after minimal cut* is introduced. The after minimal cut associated with a minimal cut m and a failure bag $e \in E$ is $m' = m - e$. Let AMC_e be the set of after minimal cuts associated with failure bag e, i.e.

$$AMC_e = \{m' | m' = m - e, m \in MC, m \cap e \neq \phi\}.$$

Then, the failure distance from any state reached from s through a failure transition associated with failure bag e, $ad(s, e)$, can be obtained as:

$$ad(s, e) = \min\{d(s), \min_{m \in AMC_e} |m - F(s)|\}. \tag{2.2}$$

Assuming that $d(s)$ is known, the use of (2.2) instead of (2.1) reduces the number of distances to minimal cuts $|m - F(s)|$ that have to be computed to determine $ad(s, e)$, $e \in E$ from $|E||MC|$ to the typically much smaller $\sum_{e \in E} |AMC_e|$. Further reduction in the number of minimal cut touches and the associated overhead can be obtained with the two algorithms we describe next. The algorithms assume known the redundancy level of the system $L = d(o)$ and $ad(o, e)$, $e \in E$. Those quantities can be computed once before the generation of the bounding model starts using (2.1) and (2.2).

Assume that an upper bound ub for $d(s)$ is known (for instance, $ub = L$). Since at most $|F(s)|$ components can be failed in any minimal cut we only need to consider the minimal cuts m with $|m| - |F(s)| < ub$. Assume also that we can access the minimal cuts indexed by order (cardinality) and selectors (bags included in the minimal cut) of order $\leq R$. For $|m - F(s)| < ub$, m must contain a selector p with all components failed and $|m| - |p| < ub$, i.e. $|p| \geq |m| - ub + 1$. Thus, for each possible minimal cut order c we can restrict our attention to the minimal cuts of order c containing selectors p with all components failed and $|p| = \min\{R, c - ub + 1\} = r$. Possible selectors can be examined by generating all bags of order r included in $F(s)$. Actual selectors can be identified easily if all selectors are kept in a hash table. The discussion justifies the algorithm *compute_d* given in Figure 2.2. The algorithm takes as input a bag of component classes b and gives as output the failure distance d from b. To compute $d(s)$, the algorithm should be invoked with $b = F(s)$.

A similar scheme can be used to compute $ad(s, e)$, $e \in E$, assuming knowledge of $d(s)$. To reduce the overhead associated with the control of the algorithm we use one bound and index the selectors for all the failure bags of the model together. The bound is initialized using $ad(o, e)$. The algorithm, called *compute_all_ad*, is given in Figure 2.2. The algorithm takes as inputs a bag of component classes b and the failure distance d

Algorithm *compute_d*(b, d)

$d = L$;
for (increasing minimal cut order c while $c < d + |b|$){
    $r = \min\{R, c - d + 1\}$;
    Let P be the set of bags of order r included in b;
    for (each $p \in P$){
        for (each minimal cut m with $|m| = c$ and $p \subset m$){
            $d = \min\{d, |m - b|\}$;
        }
    }
}

Figure 2.1: Algorithm to compute the failure distance d from a bag of component classes b.

Algorithm *compute_all_ad*$(d, b, d^*(e), e \in E)$

for (each $e \in E$) $d^*(e) = \min\{d, ad(o, e)\}$;
$adub = \max_{e \in E}\{d^*(e)\}$;
for (increasing after minimal cut order c while $c < adub + |b|$){
    $r = \min\{R, c - adub + 1\}$;
    Let P be the set of bags of order r included in b;
    for (each $p \in P$){
        for (each after minimal cut m' with $|m'| = c$ and $p \subset m'$){
            Let e be the failure bag associated with m';
            $d^*(e) = \min\{d^*(e), |m' - b|\}$;
        }
    }
}

Figure 2.2: Algorithm to compute the failure distance $d^*(e)$ from bags of component classes of the form $b + e$, $e \in E$ given the failure distance d from bag b.

from b and gives as output the failure distances $d^*(e)$ from $b + e$, $e \in E$. To compute $ad(s, e)$, $e \in E$ the algorithm should be invoked with $d = d(s)$ and $b = F(s)$.

These algorithms are used as follows. The failure distances from the generated states are kept in the state descriptions. When a state is expanded, failure distances from the new states reached through failure transitions are computed using algorithm *compute_all_ad*( ); the failure distances from the new states reached through repair transitions are computed using algorithm *compute_d*( ). Since typically most of the new states are reached through failure transitions, algorithm *compute_all_ad*( ) is invoked much more often than algorithm *compute_d*( ). The algorithms seem to be extremely efficient with moderate values of R (it is convenient not to take to a high value for R to keep small the memory overhead associated with the storage of the selectors) even when the number of minimal cuts is huge. Thus, for instance, for the second (largest) example used in Chapter 6, which has 87,031 minimal cuts, when the method developed in Chapter 5 was invoked with a target relative band of 0.01, which required the generation of 26,317 states, for $R = 2$ the number of minimal cut touches was 35,018,784 for a computation of 1,926,512 failure distances, i.e. about 18.2 minimal cut touches for computed failure distance. Using the trivial algorithm for computing the failure distances, the number of minimal cut touches would be 167,666,265,872 (about 4,788 times more touches). We have found $R = 2$ to be a good choice in all examples we have tried. With that selection, the time overhead associated with failure distances computation is very small.

2.3 An Algorithm to Compute Minimal Cuts

In this section we describe an algorithm to compute the minimal cuts of a fault tree with basic event classes such as the fault tree $\mathcal{F}$ of the class of models considered in the dissertation. The algorithm uses a decision tree. The search implemented by the decision tree is guided by heuristics that try to make the overall algorithm as efficient as possible. In addition, an irrelevance test on the inputs of the fault tree is used to prune the search. The performance of the new algorithm is illustrated and compared with the basic top-down and bottom-up algorithms using a set of fault trees, some of which are very hard. The new algorithm performs reasonably well even in the hard examples. Also, the memory requirements of the algorithm are small.

Fault trees are a very popular tool in reliability engineering. The knowledge of the minimal cuts allows the designer to analyze the criticality of the basic events and improve the reliability of the modeled system. It is well-known that computation of all minimal

cuts of an arbitrary fault tree is NP-hard [81]. In spite of this theoretical difficulty, there exist currently algorithms that will perform reasonably well in many practical cases. Early algorithms can be classified in two categories: *top-down* algorithms and *bottom-up* algorithms. Computation of all minimal cuts of a fanout free fault tree (i.e. a fault tree without repeated basic events or gates branching out to more than one gate input) can be done very easily by traversing the fault tree in a top-down fashion. In the basic top-down algorithm [42], a set of cuts, often called the superset, is obtained as if the fault tree were fanout free. The set of minimal cuts is then obtained by using in each cut the reduction rule $xx \rightarrow x$ and keeping those cuts which are not properly contained in any other cut. The algorithm involves $n(n-1)$ inclusion tests, where n is the cardinality of the superset. These inclusion tests can be performed very efficiently by associating different prime numbers to the basic events and representing the cuts by the product of the constituent basic events [90]. However, even using these techniques, reduction of the superset is an expensive task if n is large. Also, some fault trees with manageable number of minimal cuts have n so large that it is impossible to keep the superset in memory[1]. Some improvements to the basic top-down algorithm have been proposed. In [10] the size of the superset and the number of required inclusion tests is reduced by eliminating repeated events that only fanout to OR gates. The algorithm described in [79] stops the top-down expansion process at OR gates with basic event inputs, substitutes OR gates with repeated basic event inputs by one of those repeated basic events and performs reduction after each substitution step. The algorithm was to some extent faster than the basic top-down algorithm in almost all cases. In [59] it is proved that cuts of the superset without repeated basic events are all minimal and that the test for inclusion can be performed within the remaining cuts.

Bottom-up algorithms try to avoid the potentially large superset of the top-down algorithms. In the basic bottom-up algorithm [11] the fault tree is traversed from the inputs to the top event, obtaining at each step the set of minimal cuts associated with a given gate of the fault tree from the set of minimal cuts associated with the gates in its fanin. In general, each step of the bottom-up algorithm requires the reduction of the superset associated with the processed gate. The reduction is usually not very expensive for OR gates. For a two-input AND gate, the trivial procedure involves $n_1 n_2(n_1 n_2 - 1)$ tests, where n_1 and n_2 are the number of minimal cuts at the inputs. However, a more sophisticated algorithm is given in [71] that reduces the number of tests to $n_1 n_2(n_1 + n_2 - 1)$ in the worst case. A recent [52] elaboration of the bottom-up algorithm includes

[1]Thus, for instance, the example EDF of Section 2.4 has $n \approx 8.75983 \times 10^{24}$ but only 2,463 minimal cuts.

a preprocessing step that yields reduced level fault trees that are processed in descending level order. The new algorithm can speed up significantly the basic bottom-up algorithm when the fault tree has gates with fanout.

The concept of module [27] can be exploited to reduce significantly for some fault trees the computational cost of finding the minimal cuts. A module is a portion of the fault tree having basic events as inputs and a single gate (the output of the module) fanning out of the module. Efficient algorithms have been proposed to find modules [40], [57], [82], [98]. The analysis of a fault tree with modules can be reduced to the analysis of each module and the fault tree obtained by substituting each module by an independent input.

More recently, algorithms based on BDD representations [15] of the logic function implemented by the fault tree have been proposed. In [80], [93] the fault tree is assumed s-coherent and a BDD is constructed for it. That BDD is then transformed to obtain another BDD such that each path from the root to the leave 1 represents a minimal cut. The MetaPrime tool [33] uses a similar approach that can deal with non s-coherent fault trees, the BDD encoding the minimal cuts being called metaproduct. More recently [77], another algorithm has also been developed for non s-coherent fault trees that in many cases gives more compact BDD representations than the metaproducts obtained by MetaPrime. All these algorithms have been shown to be typically much faster than the early top-down and bottom-up algorithms.

The algorithm developed in this section considers generalized fault trees with basic event classes, being the basic events of each class indistinguishable. With classes, the fault tree does not longer represent a logic function with binary arguments. Also, cuts and minimal cuts are not longer sets but bags. The algorithm uses a decision tree to generate cuts of the fault tree. Cuts thus obtained are not guaranteed to be minimal and a test of minimality has to be carried out. However, instead of inclusion tests, we make an independent minimality test for each generated cut. Since cuts are generated one by one, this allows to write sequentially in secondary storage minimal cuts, making the memory requirements of the algorithm small and independent of the number of minimal cuts. These small memory requirements are an unique feature of our algorithm (BDD representations of fault trees are in the worst case of size exponential with the number of basic events).

In this section we will use the following notation and definitions.

Notation and Definitions

C set of basic event classes of the fault tree

$c_1[n_1]c_2[n_2]\ldots c_k[n_k]$ bag with $n_i > 0$ instances of basic event class c_i, $1 \leq i \leq k$; it will be said that $c_i[n_i]$ is part of the bag

I set of inputs of the fault tree; each input is a bag $c[n]$, $c \in C$, $n \geq 1$, meaning the realization of n basic events of class c

G set of gates of the fault tree

g_r root (top) gate

type(g) type of gate g; it may be AND or OR

val(.) value of an implied input or gate; it may be 0 or 1

node an input or a gate

fo(x) fanout of node x: set of gates fed by x

fi(g) fanin of gate g: set of gates or inputs that feed g

irrelevant a node x is irrelevant if all edges branching out of x are irrelevant; an edge e is irrelevant if either the gate g to which e goes is irrelevant or has been previously implied by a node connected to g by another edge

dfo(x) dynamic fanout of an unimplied node x: set of relevant edges fed by x

f_x fanout excess of an unimplied and relevant node x: $|\text{dfo}(x)| - 1$

f fanout excess of the fault tree: $\sum_{\substack{x \in I \cup G \\ x \text{ unimplied and relevant}}} f_x$

δ_x for an unimplied and relevant gate, $\delta_x = f_x$; for an unimplied and relevant input $c[n]$, $\delta_x = \sum_{\substack{c[n'] \in I,\, n' \leq n \\ c[n'] \text{ unimplied and relevant}}} f_{c[n']}$

input pattern any compatible combination of assignments of 0, 1 values to inputs of the fault tree (by compatible we mean that the existence of the assignment $(c[n], 1)$ implies the existence of the assignments $(c[n'], 1)$, $n' < n$, $c[n'] \in I$, and that the existence of the assignment $(c[n], 0)$ implies the existence of the assignments $(c[n'], 0)$, $n' > n$, $c[n'] \in I$)

reduction of l being $l = \{c_1[n_1], c_2[n_2], \ldots, c_k[n_k]\}$ a set of inputs implied at 1, generation of a bag b by traversing l and putting into b each $c[n] \in l$ such that no $c[n']$, $n' > n$ is in l

cont(x) controllability of an unimplied node x: if $x \in I$, $\text{cont}(x) = 1$; if $x \in G$ and $\text{type}(x) = \text{OR}$, $\text{cont}(x) = \min_{\substack{x' \in \text{fi}(x) \\ x' \text{ unimplied}}} \{\text{cont}(x')\}$; if $x \in G$ and $\text{type}(x) = \text{AND}$, $\text{cont}(x) = \sum_{\substack{x' \in \text{fi}(x) \\ x' \text{ unimplied}}} \text{cont}(x')$

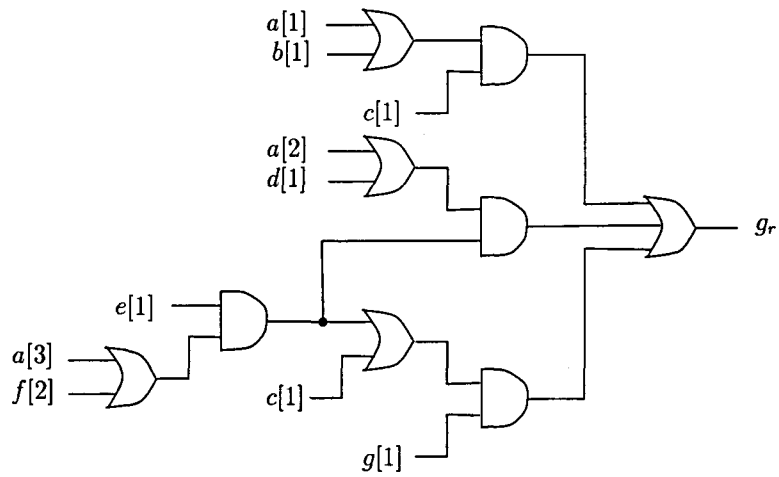

Figure 2.3: Example fault tree.

We consider s-coherent fault trees involving OR and AND gates. Figure 2.3 gives a small example that will be used for illustration purposes.

We begin by clarifying several concepts. Generalization to basic event classes introduces dependences among the inputs related to the same event class. An input $c_i[n_i]$ is implied at 1 if n_i events of class c_i are realized. An input $c_i[n_i]$ is implied at 0 if n_i events of class c_i are not realized. Then, $\text{val}(c_i[n_i]) = 1$ implies $\text{val}(c_i[j]) = 1$ for all $j < n_i$ because the realization of n_i basic events of class c_i obviously implies the realization of any number $j < n_i$ of basic events of that class. Similarly, $\text{val}(c_i[n_i]) = 0$ implies $\text{val}(c_i[j]) = 0$ for all $j > n_i$. Implications in the fault tree are performed from inputs to g_r in the usual way: an input of an OR gate at 1 implies the output of the gate at 1; an input of an AND gate at 0 implies the output of the gate at 0; the output of an OR gate is implied at 0 when all inputs are implied at 0, and the output of an AND gate is implied at 1 when all inputs are implied at 1. Since the fault tree is s-coherent and all gates are either OR or AND such a procedure is enough to know the implication state of g_r. To see that it suffices to note that 0's imply 0's and 1's imply 1's and, therefore, an unimplied output g_r can be implied to 0 or 1 by simply implying the unimplied inputs of the fault tree to 0 or 1, respectively. Therefore, when the procedure does not imply g_r, the root gate g_r is really unimplied.

For the example fault tree of Figure 2.3, $I = \{a[1], a[2], a[3], b[1], c[1], d[1], e[1], f[2], g[1]\}$. Two possible input patterns might be $l_1 = \{(a[1], 1), (a[2], 1), (a[3], 0), (b[1], 0), (c[1], 0), (d[1], 0), (e[1], 1), (f[2], 1), (g[1], 0)\}$ and $l_2 = \{(a[1], 0), (a[2], 0), (a[3], 0), (b[1], 0), (c[1], 0), (d[1], 0), (e[1], 1), (f[2], 1), (g[1], 0)\}$. For l_1, g_r is implied at 1. Then, that input pattern "contains" the cut $m = a[2]e[1]f[2]$ obtained by reducing the set of inputs

implied at 1 $\{a[1], a[2], e[1], f[2]\}$.

The problem to be solved is to find all minimal cuts of a given s-coherent fault tree. Such a problem may be viewed as a search in a finite space. The search space is given by the next Theorem:

Theorem 2.3 *All minimal cuts are of the form* $c_1[n_1]c_2[n_2]\ldots c_k[n_k]$, *where each* $c_i[n_i] \in I$.

Proof By contradiction. Let $m = c_1[n_1]c_2[n_2]\ldots c_k[n_k]$ be a minimal cut not satisfying the condition. Then, there exists $c_i[n_i]$ part of m such that $c_i[n_i] \notin I$. Assume that there exists $c_i[j] \in I$ with $j < n_i$. Let J_i be the greatest of such integers and consider the bag m' obtained from m by substituting $c_i[n_i]$ by $c_i[J_i]$. If there not exists $c_i[j] \in I$ with $j < n_i$, let m' be bag obtained by eliminating $c_i[n_i]$ from m. Clearly, m' performs the same implications as m does. Therefore, m' is a cut, but being $m' \subset m$, m is not minimal. □

From the irrelevance definitions, the value of an irrelevant node does not affect $\text{val}(g_r)$. The following theorem relates irrelevance and minimality and will be exploited in the algorithm.

Theorem 2.4 *Let S be the set of implied fault tree inputs. If there exists* $c_i[n_i] \in S$ *that is implied at 1 and is irrelevant, and there not exists any* $c_i[j] \in I$ *with* $j > n_i$, *no input pattern obtained by implying more inputs will contain a minimal cut.*

Proof Let m be any cut obtained by implying more inputs and reducing the set of inputs implied at 1. Because no $c_i[j] \in I$, $j > n_i$ exists, $c_i[n_i] \in m$. Assume that there exists at least one $c_i[j] \in I$, $j < n_i$ and let J_i be the greatest integer $j < n_i$ with $c_i[j] \in I$. Consider the bag m' obtained from m by substituting $c_i[n_i]$ by $c_i[J_i]$. If there not exists any $c_i[j] \in I$, $j < n_i$, let m' be the bag obtained from m by eliminating $c_i[n_i]$. Because $c_i[n_i]$ is irrelevant, the value of g_r is not affected by the value of $c_i[n_i]$ and m' is also a cut. Furthermore, $m' \subset m$, implying that m is not minimal. □

To avoid performing inclusion tests, which are time and memory consuming, a criterion to determine when a cut is minimal or not without knowing any other cut would be useful. The next theorem gives such a criterion.

Theorem 2.5 *A cut* $m = c_1[n_1]c_2[n_2]\ldots c_k[n_k]$ *is minimal if and only if, after implying at 1 each* $c_i[n_i]$, *each unimplication of* $c_i[n_i]$, $1 \leq i \leq k$, *followed by, if such* J_i *exists,*

implication of $c_i[J_i]$, where J_i is the greatest integer $< n_i$ with $c_i[J_i] \in I$, leaves g_r unimplied.

Proof We will show necessity and sufficiency.

Necessity: If there exists any $c_i[n_i] \in m$ such that its unimplication followed, if existent, by the implication of $c_i[J_i]$, where J_i is the greatest integer $< n_i$ with $c_i[J_i] \in I$, leaves g_r implied at 1, the bag m' obtained from m by either deleting $c_i[n_i]$ or replacing $c_i[n_i]$ by $c_i[J_i]$ is also a cut, and being $m' \subset m$, m is not minimal.

Sufficiency: Assume that g_r is unimplied for each unimplication, followed, if existent, by implication of $c_i[J_i]$, and assume that there exists a bag $m' \subset m$ that is a cut. Being $m' \subset m$, there exists m'', $m' \subseteq m'' \subset m$ such that m'' is obtained from m by either deleting $c_i[n_i]$ for some $1 \leq i \leq k$ or replacing for some $1 \leq i \leq k$ $c_i[n_i]$ by $c_i[n_i']$ with $n_i' < n_i$. In both cases g_r is not implied at 1 by m'' and m'' is not a cut, implying that neither is m' a cut. Then, there not exists any cut $m' \subset m$ and m is minimal. □

From Theorem 2.3, the search space is the space of input patterns of the fault tree. All minimal cuts can be found by exhaustively searching that space and, for each input pattern for which $\text{val}(g_r) = 1$, obtaining its associated cut m by reduction of the set of inputs implied at 1, and testing m as Theorem 2.5 indicates. However, the algorithm does not explicitly generate all input patterns, since in some circumstances it detects that no input pattern containing the current set of implications would yield a minimal cut. The algorithm also detects when the fault tree becomes fanout free (f becomes 0) and uses the top-down algorithm to obtain all potential minimal cuts that can be generated by performing more implications at the inputs of the fault tree.

The algorithm we propose traverses a decision tree such as the one shown in Figure 2.5. Initially, all nodes of the fault tree are unimplied and the current node is the root of the DT. A backtrace procedure selects an input $c_i[n_i]$. The selected input is implied at $v = 1$, a successor of the current node is constructed with the pair $(c_i[n_i], v)$ assigned to it, and the successor is visited. The process continues in a similar way from that node. After each implication, the relevance status of edges and nodes, f_x, f, δ_x, and $\text{cont}(x)$ are updated. In some cases, the search can be pruned. When the search is pruned, the DT is traversed up to the root till a node y is found that has only a successor with $v = 1$. If no such a node y is found, the algorithm finishes. Otherwise, the search is backtracked up to node y. Backtracking involves the deletion of all the implications done as a direct consequence of the input assignments associated with the nodes in the path from the current node to y. It also involves the restoring of the old values of the relevance status

of the edges and nodes, f_x, f, δ_x, and cont(x). Deletion of implications and restoring of old values of the relevance status of edges and nodes, f_x, f, δ_x and cont(x) is made easily because the algorithm stores for each node of the DT the implications performed and the old values of the relevance status of edges and nodes, f_x, f, δ_x and cont(x) which change as a result of the input implication performed at the node. After backtracking, a successor of y associated with the input assignment $(c_i[n_i], 0)$, where $c_i[n_i]$ is the input of the successor of y, is created, the assignment implied, and the process continues from that successor. If the implied input value v is 1 the search can be pruned in the following cases:

1. val(g_r) = 1. Extra input implications at 1 will give cuts which are guaranteed to be non-minimal.
2. $f = 0$. The fault tree has been reduced to a fault tree that is fanout free[2] and all minimal cuts beyond that point of the search have to be included in the cuts that are obtained by adding to the current set of inputs implied at 1 the inputs that are found using the basic top-down algorithm on the reduced fanout free fault tree and reducing those sets of inputs.
3. An input $c[n]$ implied at 1 is irrelevant and there not exists $c[n'] \in I$, $n' > n$. According to Theorem 2.4, no minimal cut will be found by implying more inputs.

For $v = 0$, case 1 is impossible. Besides cases 2 and 3, there is another situation in which no more implications are necessary:

4. val(g_r) = 0. Further input assignments will not change the value of g_r and no minimal cut exists from that point.

In case 1, a potential minimal cut is obtained by reducing the set I of inputs implied at 1. In case 2, the top-down algorithm is used to find potential minimal cuts. The cuts thus obtained are checked for minimality using the procedure of Theorem 2.5 and recorded if they are minimal.

Our algorithm has similarities with some ATPG algorithms [2], [41], [44]. Figure 2.4 shows a recursive high-level description of the algorithm that uses a stack to store the path in the DT to the currently processed node. The algorithm is invoked with an empty stack. In the worst case the stack will have $|I|$ cells. This together with the fact that

[2]The reduced fault tree is defined by the relevant and unimplied nodes of the original fault tree.

Algorithm *compute_cuts (INPUTS_ASSIGNMENTS_STACK s)*

```
backtrace(g_r, c_i[n_i]);
v = 1;
end = NO;
while (!end) {
    imply c_i[n_i] at v;
    push((c_i[n_i], v), s);
    if (val(g_r) == 1 || f == 0) {
        if (val(g_r) == 1) {
            get cut m;
            if (test_minimality(m)) store m;      /* m is minimal */
        }
        else {      /* f == 0 */
            collect potential minimal cuts by adding to the current set of
            inputs implied at 1 the inputs found by the top-down algorithm
            and reducing those sets, for each cut perform the minimality test
            and store the cut if it is minimal;
        }
    }
    else if (!(val(g_r) == 0 or an input c[n] implied at 1 has become irrelevant
            and does not exist c[n'] ∈ I, n' > n)) compute_cuts(s);
    unimply all c_i[j] with either j ≤ n_i (case v = 1) or j ≥ n_i (case v = 0)
    that were implied as a direct consequence of setting c_i[n_i] at v;
    pop((c_i[n_i], v), s);
    if (v == 1) v = 0 else end = YES;
}
```

Figure 2.4: High-level description of the algorithm.

only a minimal cut has to be stored at a given time (possible because the minimality check does not involve inclusion tests) makes the memory requirements of the algorithm small. It has been assumed that the fault tree to be solved is not fanout free. To take into account the special case in which the fault tree is fanout free, it suffices to check the value of f before calling *compute_cuts* and invoke the top-down algorithm if $f = 0$.

The backtrace procedure that selects input assignments is crucial for the performance of the algorithm and is inspired in ATPG algorithms. The procedure starts at the output g_r of the fault tree and follows a path to a fault tree input by selecting at each gate one of its unimplied inputs. Gate inputs are selected using the following criteria:

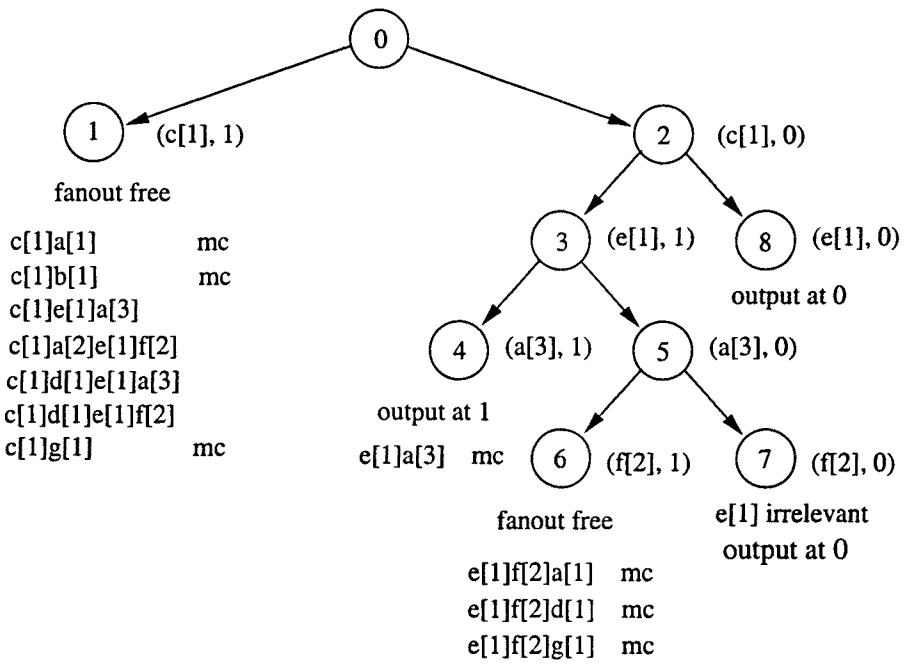

Figure 2.5: Decision tree for the example of Figure 2.3.

1. Choose the input x with highest $\delta_x/\text{cont}(x)$.

2. Among the inputs with same $\delta_x/\text{cont}(x)$ choose the input connected to the node with lowest $\text{cont}(x)$.

If several inputs are identical according to both criteria, then the gate input is chosen following a predefined ordering (for the fault tree of Figure 2.3 the ordering is from top to bottom of the figure).

The heuristics used in the backtrace procedure try to reach as soon as possible nodes with backtracking, either because the fault tree becomes fanout free or because g_r is implied at 1. The δ_x is a local measure of how much f will be decreased if x is implied at 1. The $\text{cont}(x)$ is a measure of the ease with which the considered node will be implied at 1 (the higher $\text{cont}(x)$ the more difficult) and is taken directly from heuristic measures used to guide ATPG algorithms [45], being the measure in that context called 1-controllability. We tried other combinations, such as selecting first according to $\text{cont}(x)$ and then according to δ_x and found that the chosen heuristics gave better performance.

In order to illustrate the algorithm we give in Figure 2.5 the DT corresponding to the fault tree of Figure 2.3. Nodes are numbered following the creation order. For each node except the root we give the corresponding input assignment. Also, for nodes with

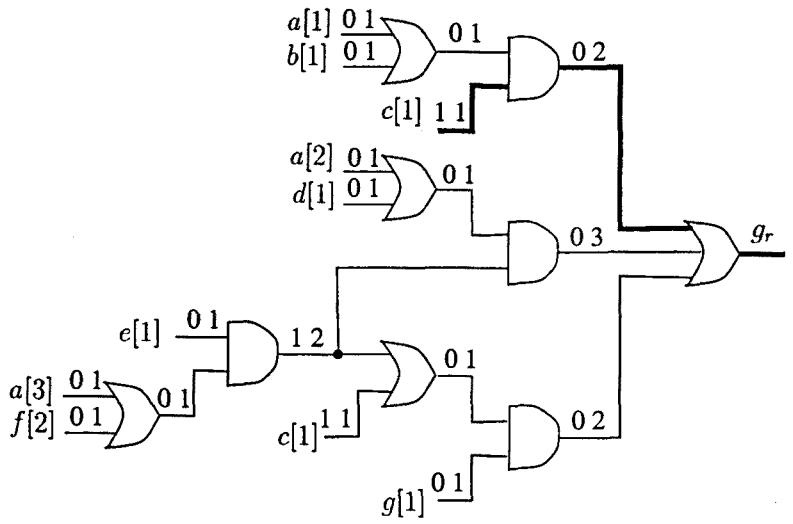

Figure 2.6: Illustration of the backtrace procedure in the case in which there is no input assignment (node 0 of the DT of Figure 2.5). The δ_x and cont(x) are written next to each node from left to right, respectively.

backtracking we give the reason for backtracking and, for the nodes in which the fault tree became fanout free or g_r was implied at 1, we give the cuts found, indicating by "mc" those that are minimal. The fault tree has only 7 minimal cuts.

Figure 2.6 illustrates the backtrace procedure in the case in which there is no input assignment (node 0 of the DT). The input selected is c[1]. Figure 2.7 gives the implication status of the fault tree when c[1] is implied at 1. That implication makes irrelevant the edge marked as i, making the fault tree fanout free ($f = 0$). The corresponding reduced fault tree is given in Figure 2.8. Use of the top-down algorithm on the reduced fault tree gives the set of inputs $\{a[1]\}$, $\{b[1]\}$, $\{a[2], e[1], a[3]\}$, $\{a[2], e[1], f[2]\}$, $\{d[1], e[1], a[3]\}$, $\{d[1], e[1], f[2]\}$ and $\{g[1]\}$. The input currently implied at 1 ($c[1]$) is added to each sets and the resulting sets are reduced, yielding the cuts shown in Figure 2.5. To illustrate the minimality test, Figure 2.9 gives the implication status for the cut $c[1]e[1]a[3]$, showing crossed the unimplications which result when $a[3]$ is unimplied, keeping implied $a[2]$ and $a[1]$. Since g_r remains implied at 1, according to Theorem 2.5, the cut is not minimal. Finally, in order to illustrate backtracking by detection of an irrelevant input implied at 1, we give in Figure 2.10 the implication status and irrelevant nodes and edges corresponding to node 7 of the DT. Input $e[1]$ is implied at 1, is irrelevant and there not exists any other input $e[j]$ with $j > 1$. Then, according to Theorem 2.4, no minimal cut can be found from that point and the search can be backtracked. At that node of the DT the search could also be backtracked for g_r being implied at 0.

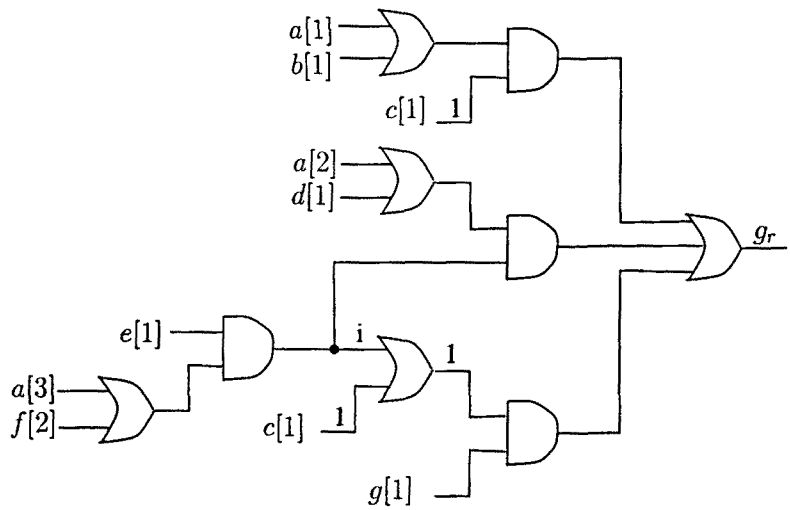

Figure 2.7: Implication status at node 1 of the DT of Figure 2.5 with indication of irrelevant edges.

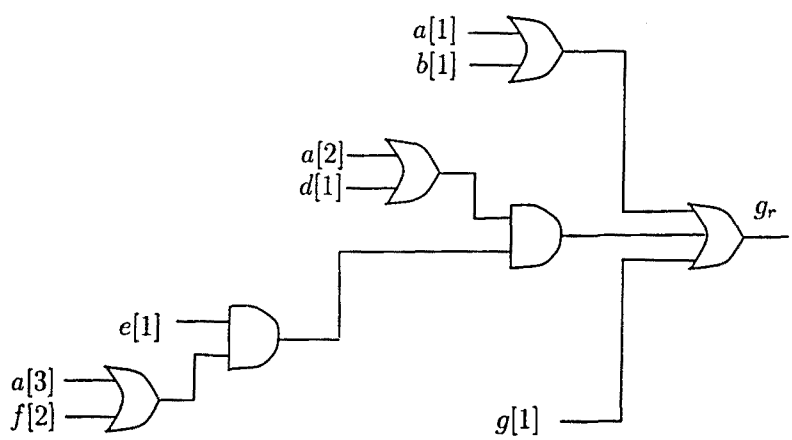

Figure 2.8: Reduced fault tree corresponding to node 1 of the DT of Figure 2.5.

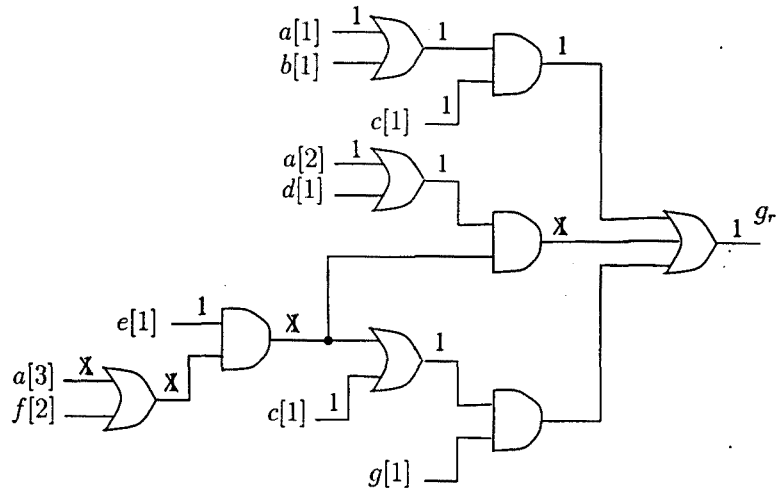

Figure 2.9: Implication status corresponding to cut $c[1]e[1]a[3]$ and unimplication of $a[3]$, leaving $a[2]$ and $a[1]$ implied for the fault tree of Figure 2.3.

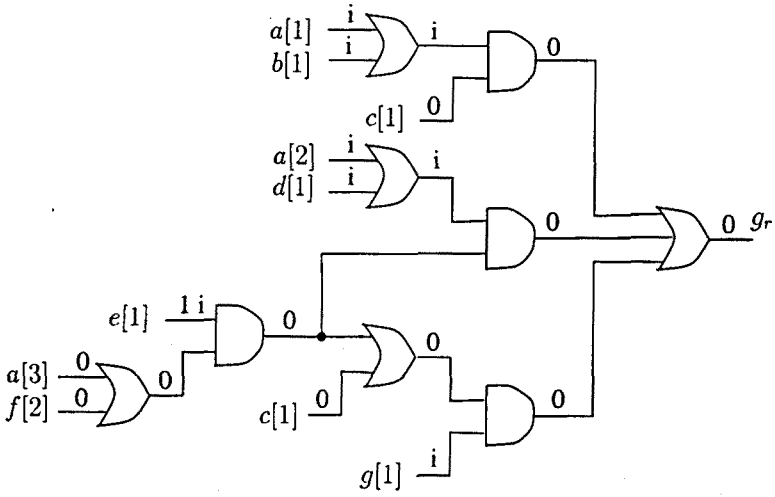

Figure 2.10: Implication status and irrelevance status of the nodes for node 7 of the DT of Figure 2.5.

Table 2.1: Fault tree characteristics.

tree	$\|I\|$	$\|G\|$	fanout excess	depth	minimal cuts
MS5	38	103	92	7	511
MS10	68	203	192	7	1,911
BR40	120	42	3,080	2	3,160
BR80	240	82	12,560	2	12,720
DR35	112	46	152	10	3,698
DR70	217	46	257	10	14,157
EDF	39	43	9	34	2,463
ELF1	61	122	12	147	46,188
ELF2	32	65	57	12	4,805
ELF3	80	107	87	17	24,386

2.4 Algorithm Analysis

In this section we analyze the algorithm described in Section 2.3 by means of several examples. Table 2.1 summarizes the characteristics of the fault trees that have been used to test the algorithm. In all cases the number of basic event classes is equal to the number of inputs. For each fault tree we give the number of inputs, number of gates, fanout excess (f) in the unimplied fault tree, depth (maximum number of gates from a fault tree input to g_r) and number of minimal cuts.

Fault trees MS5 and MS10 correspond to the master-slave system depicted in Figure 2.11 with $n = 5$ and $n = 10$, respectively. That system is made up of a cluster of redundant master processing units MPU_1 and MPU_2 that are communicated with n clusters of redundant slave processing units $\mathrm{SPU}_{i,1}$ and $\mathrm{SPU}_{i,2}$, $1 \leq i \leq n$. Communication is done through two redundant buses BA and BB to which the master and slave units are connected through dedicated interfaces. The system is operational if some fault-free master processing unit can communicate directly (i.e. through one fault-free bus and two fault-free interfaces) with at least one fault-free slave processing unit of each slave cluster. Denoting by "·" and by "+" the logical operators AND and OR, respectively, and naming each event class as the component type whose failure models, the expression

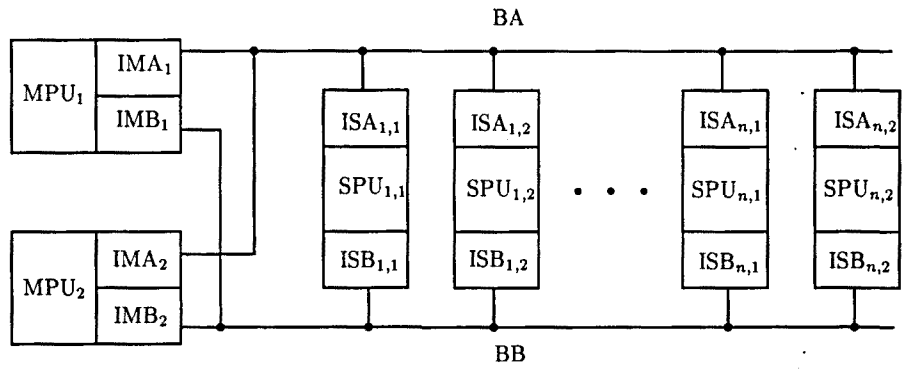

Figure 2.11: Master-slave system with n clusters of redundant slave processing units.

of the fault tree of that system is:

$$g_r = \Bigg[\sum_{i=1}^{n} \Big(\mathrm{MPU}_1[1] + \big(\mathrm{IMA}_1[1] + \mathrm{BA}[1] + \big(\mathrm{ISA}_{i,1}[1] + \mathrm{SPU}_{i,1}[1]\big)\cdot \big(\mathrm{ISA}_{i,2}[1] + \mathrm{SPU}_{i,2}[1]\big)\big) \cdot \big(\mathrm{IMB}_1[1] + \mathrm{BB}[1] + \big(\mathrm{ISB}_{i,1}[1] + \mathrm{SPU}_{i,1}[1]\big) \cdot \big(\mathrm{ISB}_{i,2}[1] + \mathrm{SPU}_{i,2}[1]\big)\big)\Big)\Bigg] \cdot \Bigg[\sum_{i=1}^{n} \Big(\mathrm{MPU}_2[1] + \big(\mathrm{IMA}_2[1] + \mathrm{BA}[1] + \big(\mathrm{ISA}_{i,1}[1] + \mathrm{SPU}_{i,1}[1]\big)\cdot \big(\mathrm{ISA}_{i,2}[1] + \mathrm{SPU}_{i,2}[1]\big)\big) \cdot \big(\mathrm{IMB}_2[1] + \mathrm{BB}[1] + \big(\mathrm{ISB}_{i,1}[1] + \mathrm{SPU}_{i,1}[1]\big) \cdot \big(\mathrm{ISB}_{i,2}[1] + \mathrm{SPU}_{i,2}[1]\big)\big)\Big)\Bigg]$$

Fault trees BR40 and BR80 model the failure of the braided ring system of Figure 2.12 with $n = 40$ and $n = 80$, respectively. The braided ring is composed of stations S_i, $0 \leq i \leq n-1$. There are links D_i between S_i and $\mathrm{S}_{(i+1) \bmod n}$ and links $\mathrm{T}_{(i+1) \bmod n}$ between S_i and $\mathrm{S}_{(i+2) \bmod n}$. All these links are directed. The system is up if it is possible to build a ring connecting at least $n-1$ fault-free stations S_i. The expression for the fault tree is:

$$g_r = \sum_{i=0}^{n-1} (\mathrm{S}_i[1] + \mathrm{D}_i[1]) \prod_{i=0}^{n-1} \left(\sum_{\substack{j=0 \\ j \neq i}}^{n-1} \mathrm{S}_j[1] + \sum_{\substack{j=0 \\ j \neq i, (i-1) \bmod n}}^{n-1} \mathrm{D}_j[1] + \mathrm{T}_i[1] \right).$$

Fault trees DR35 and DR70 correspond to the system of Figure 2.13 with $n = 35$ and $n = 70$, respectively. Two redundant servers S_1, S_2 are communicated with gateways G_1, G_2 through a double ring network composed of nodes N_i, $0 \leq i \leq n-1$. There are

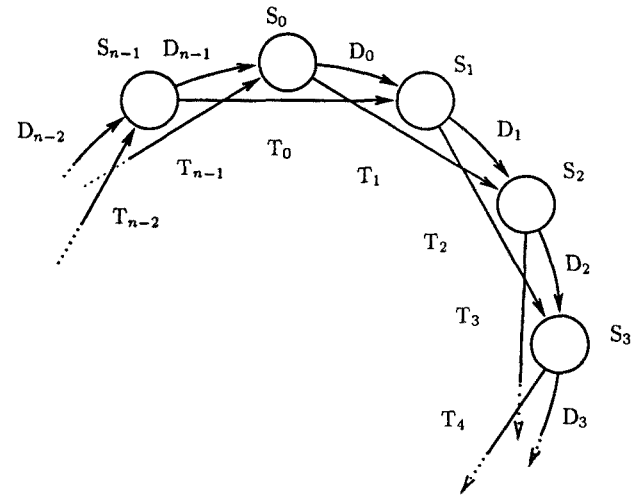

Figure 2.12: Braided ring system with n stations.

clockwise links I_i from $N_{(i+1) \bmod n}$ to N_i and counter-clockwise links D_i from N_i to $N_{(i+1) \bmod n}$. Each node has a spare module SN_i that may bypass N_i if it has failed. However, those spare components are not connected to servers or gateways. The servers are connected to nodes $N_{\lfloor n/2 \rfloor}$ and $N_{\lfloor n/2 \rfloor+1}$; gateways G_1 and G_2 are connected to nodes N_0 and N_1, respectively. The system is operational if there exists communication in both directions between at least one fault-free server and one fault-free gateway. S_1 and S_2 are indistinguishable; N_i and SN_i, $2 \leq i \leq \lfloor n/2 \rfloor - 1$, $\lfloor n/2 \rfloor + 2 \leq i \leq n-1$ are also indistinguishable. Denoting by S the event class that models the failure of the components S_1, S_2, by M_i the event class that models the failure of the components N_i, SN_i, $2 \leq i \leq \lfloor n/2 \rfloor - 1$, $\lfloor n/2 \rfloor + 2 \leq i \leq n-1$, and denoting the event classes that model the failure of the other components by the components' names, the fault tree of such a system is given by the following expressions (there is an expression for g_r and each gate with fanout):

$$
\begin{aligned}
g_r \;=\; & \mathrm{S}[2] + \Big(\mathrm{G}_1[1] + \mathrm{N}_0[1] + (\mathrm{N}_{\lfloor n/2 \rfloor}[1] + \mathrm{C}_{0,\lfloor n/2 \rfloor}) \cdot (\mathrm{N}_{\lfloor n/2 \rfloor+1}[1] + \mathrm{C}_{0,\lfloor n/2 \rfloor+1})\Big) \cdot \\
& \Big(\mathrm{G}_2[1] + \mathrm{N}_1[1] + (\mathrm{N}_{\lfloor n/2 \rfloor}[1] + \mathrm{C}_{1,\lfloor n/2 \rfloor}) \cdot (\mathrm{N}_{\lfloor n/2 \rfloor+1}[1] + \mathrm{C}_{1,\lfloor n/2 \rfloor+1})\Big), \\
\mathrm{C}_{0,\lfloor n/2 \rfloor} = {} & (\mathrm{N}_1[1] \cdot \mathrm{SN}_1[1] + \mathrm{N}_{\lfloor n/2 \rfloor+1}[1] \cdot \mathrm{SN}_{\lfloor n/2 \rfloor+1}[1] + \mathrm{DR}) \cdot (\mathrm{N}_1[1] \cdot \mathrm{SN}_1[1] + \mathrm{I}_0[1] + \\
& \mathrm{D}_0[1] + \mathrm{RFR}) \cdot (\mathrm{N}_{\lfloor n/2 \rfloor+1}[1] \cdot \mathrm{SN}_{\lfloor n/2 \rfloor+1}[1] + \mathrm{I}_{\lfloor n/2 \rfloor}[1] + \mathrm{D}_{\lfloor n/2 \rfloor}[1] + \mathrm{LFR}), \\
\mathrm{C}_{0,\lfloor n/2 \rfloor+1} = {} & (\mathrm{N}_1[1] \cdot \mathrm{SN}_1[1] + \mathrm{N}_{\lfloor n/2 \rfloor}[1] \cdot \mathrm{SN}_{\lfloor n/2 \rfloor}[1] + \mathrm{DR}) \cdot (\mathrm{N}_1[1] \cdot \mathrm{SN}_1[1] + \\
& \mathrm{N}_{\lfloor n/2 \rfloor}[1] \cdot \mathrm{SN}_{\lfloor n/2 \rfloor}[1] + \mathrm{I}_0[1] + \mathrm{D}_0[1] + \mathrm{I}_{\lfloor n/2 \rfloor}[1] + \mathrm{D}_{\lfloor n/2 \rfloor}[1] + \mathrm{RFR}) \cdot \mathrm{LFR},
\end{aligned}
$$

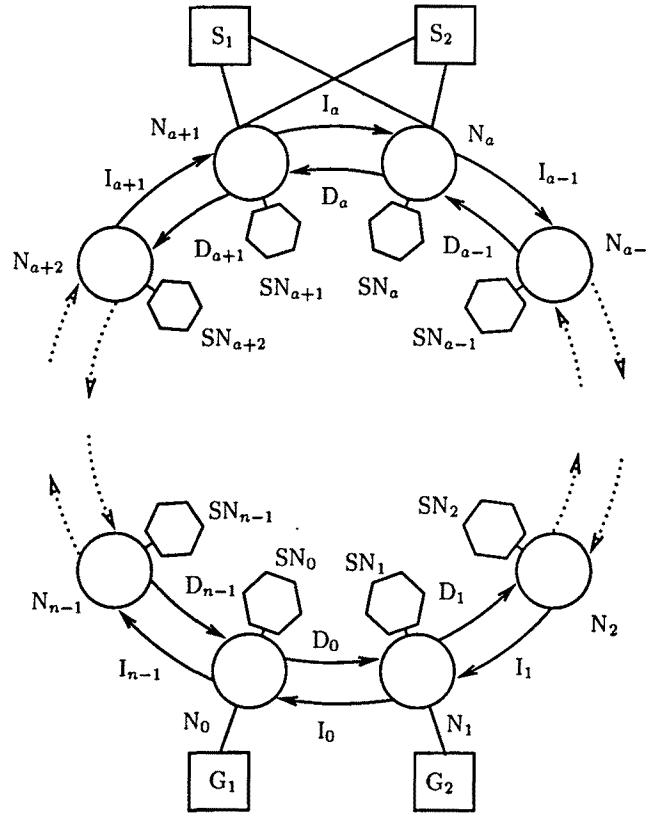

Figure 2.13: Double ring network ($a = \lfloor n/2 \rfloor$).

$$
\begin{aligned}
C_{1,\lfloor n/2 \rfloor} &= (N_0[1] \cdot SN_0[1] + N_{\lfloor n/2 \rfloor+1}[1] \cdot SN_{\lfloor n/2 \rfloor+1}[1] + DR) \cdot RFR \cdot (N_0[1] \cdot SN_0[1] + \\
&\quad N_{\lfloor n/2 \rfloor+1}[1] \cdot SN_{\lfloor n/2 \rfloor+1}[1] + I_0[1] + D_0[1] + I_{\lfloor n/2 \rfloor}[1] + D_{\lfloor n/2 \rfloor}[1] + LFR), \\
C_{1,\lfloor n/2 \rfloor+1} &= (N_0[1] \cdot SN_0[1] + N_{\lfloor n/2 \rfloor}[1] \cdot SN_{\lfloor n/2 \rfloor}[1] + DR) \cdot (N_{\lfloor n/2 \rfloor}[1] \cdot SN_{\lfloor n/2 \rfloor}[1] + \\
&\quad I_{\lfloor n/2 \rfloor}[1] + D_{\lfloor n/2 \rfloor}[1] + RFR) \cdot (N_0[1] \cdot SN_0[1] + I_0[1] + D_0[1] + LFR), \\
DR &= \sum_{i=2}^{\lfloor n/2 \rfloor-1} M_i[2] + \sum_{i=\lfloor n/2 \rfloor+2}^{n-1} M_i[2] + \left(\sum_{i=0}^{n-1} I_i[1]\right) \cdot \left(\sum_{i=0}^{n-1} D_i[1]\right), \\
RFR &= \sum_{i=2}^{\lfloor n/2 \rfloor-1} M_i[2] + \sum_{i=1}^{\lfloor n/2 \rfloor-1} (I_i[1] + D_i[1]), \\
LFR &= \sum_{i=\lfloor n/2 \rfloor+2}^{n-1} M_i[2] + \sum_{i=\lfloor n/2 \rfloor+1}^{n-1} (I_i[1] + D_i[1]).
\end{aligned}
$$

Fault tree EDF models the failure of the communication network with 14 nodes and 25 directed links depicted in Figure 2.14. Such a network is up if the sender node S and the receiver node R are both fault-free and there is a path of fault-free components from S to R.

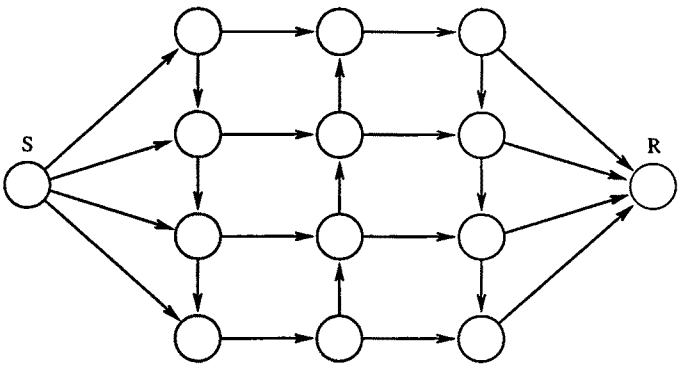

Figure 2.14: Communication network for fault tree EDF.

Fault trees ELF1, ELF2 and ELF3 are approximately the fault trees called with these names in [33].

Table 2.2 shows the results obtained for each fault tree. CPU times have been measured in a Sparc10 workstation. The table also shows the number of cuts which would be processed by the basic top-down and bottom-up algorithms. Notice that for all fault trees except MS5 and MS10 the use of either top-down or bottom-up algorithm is impractical. For all fault trees the algorithm outperforms the classical algorithms, with a ratio of number of processed cuts between 5.6 and 9.8×10^{173}.

In order to illustrate the impact of: 1) the heuristics we have used in the backtrace procedure, and 2) the irrelevance test, we show in Table 2.3 the number of backtracks that are performed when either the inputs are chosen at random or the test of irrelevance is disabled. The CPU time was limited to 5 hours and results are not given for the cases in which more than 5 hours were necessary to compute all minimal cuts. In all cases both the heuristics for input selection and the irrelevance test reduce significantly the number of backtracks.

2.5 Conclusions

In this chapter we have stated formally the problem of computing failure distances from a state (problem FD) and have shown that it is NP-hard. We have reviewed two efficient algorithms to compute such failure distances using the set of minimal cuts of the fault tree of the system. The algorithms are not polynomial, but we have proved that there could exist polynomial algorithms for the problem only if NP = P. Finally, an algorithm to compute the set of minimal cuts has been described. The algorithm performs reasonably

Table 2.2: Algorithm performance: number of minimal cuts, number of processed cuts, number of backtracks, CPU time in seconds, and number of cuts that would be processed by the basic top-down and bottom-up algorithms.

tree	minimal cuts	processed cuts	backtracks	CPU time (s)	cuts top-down	cuts bottom-up
MS5	511	1,473	1,936	1.74	34,225	8,285
MS10	1,911	5,208	23,135	13.6	136,900	29,345
BR40	3,160	3,161	821	11.4	3.86255×10^{77}	3.86255×10^{77}
BR80	12,720	12,721	3,241	175	1.24936×10^{178}	1.24936×10^{178}
DR35	3,698	118,444	60,099	51.1	1.22672×10^{26}	2.90796×10^{7}
DR70	14,157	808,953	395,907	536	8.32281×10^{30}	4.75491×10^{8}
EDF	2,463	3,435	1,683	4.26	8.75983×10^{24}	3.81949×10^{10}
ELF1	46,188	112,606	169,278	235	1.26358×10^{20}	2.86939×10^{8}
ELF2	4,805	13,754	16,842	23.6	4.17538×10^{17}	5.48907×10^{8}
ELF3	24,386	69,488	150,601	150	1.45039×10^{16}	7.66257×10^{7}

Table 2.3: Algorithm performance: number of backtracks in the algorithm, and when either fault tree inputs are selected at random or the irrelevance test is disabled.

tree	algorithm	w/ random selection	w/o relevance test
MS5	1,936	67,619	309,245
MS10	23,135	—	—
BR40	821	1,849	3,943
BR80	3,241	10,985	15,883
DR35	60,099	233,663	248,414
DR70	395,907	1,798,840	1,791,888
EDF	1,683	266,611	22,159
ELF1	169,278	9,115,802	—
ELF2	16,842	226,704	101,994
ELF3	150,601	—	—

well even in difficult examples. The number of processed cuts is usually not much larger than the number of minimal cuts of the fault tree, and, in many cases, it is much smaller than the number of cuts processed by the basic top-down and bottom-up algorithms. Compared with recent algorithms based on BDD representations of the fault tree [80, 33, 77], the algorithm seems to be somehow slower. However, the algorithm has very modest memory requirements whereas in the worst-case the algorithms based on BDD representations require memory that is exponential in the number of basic events.

Chapter 3

Reliability Bounds of Non-repairable Systems using FD

In this chapter we will develop the first failure distance based bounding method of the dissertation. The method obtains bounds for the unreliability at time t, $ur(t)$, of a non-repairable fault-tolerant system and requires the computation of failure distances. It has the interesting property that the bounds are obtained from the transient solution of a "bounding" CTMC and, thus, the bounding method can be accommodated in any general-purpose Markovian modeling tool. We will start by describing and justifying theoretically the bounding method. Then, we will analyze its performance and will compare it with the performance of the trivial bounding method described in Section 1.3, showing that the proposed method can outperform significantly the trivial method.

3.1 Method Description and Justification

The measure $ur(t)$ can be computed exactly using a CTMC $X = \{X(t); t \geq 0\}$ that is acyclic, has a finite state space $\Omega \cup \{f\}$, where f is an absorbing state that represents the failure of the system and Ω is the set of states in which the system is operational, and has transition rates with failure bags $e \in E$ associated with them. We have $ur(t) = P[X(t) = f]$. Given $i \in \Omega$, $j \in \Omega \cup \{f\}$ we will denote by $\lambda_{i,j}$ the transition rate of X from state i to state j and by $\lambda_i = \sum_{l \in \Omega \cup \{f\}} \lambda_{i,l}$ the output rate of X from state i. Being B a subset of states, we will denote by $\lambda_{i,B} = \sum_{l \in B} \lambda_{i,l}$ the transition rate from state i to B.

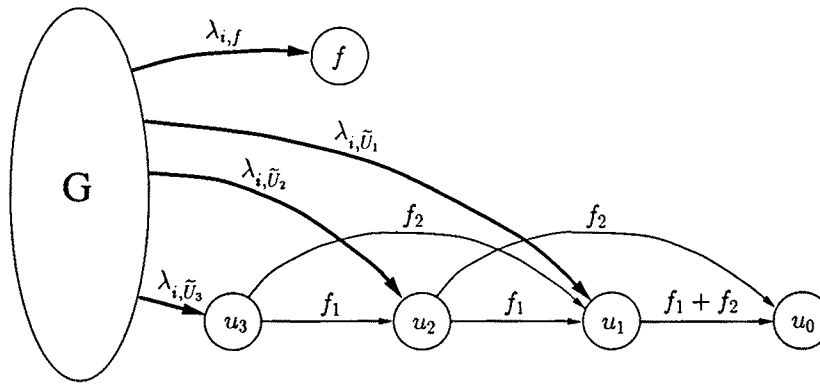

Figure 3.1: State transition diagram of X' for $L = 3$ and $FC = \{1, 2\}$.

The bounds for $ur(t)$ will be obtained using a CTMC $X' = \{X'(t); t \geq 0\}$ with state space $G \cup \{f\} \cup \{u_0, \cdots, u_L\}$, $L = \min_{m \in MC} |m|$. G is the subset of Ω that is generated and the method assumes $P[X(0) \in G] = 1$; f represents the failure of the system from a state belonging to G; the states u_d, $0 \leq d \leq L$ "bound" the behavior of X after it exits G through a non-failed state. Let $U = \Omega - G$. The formal proof of the method assumes that there are not transitions from U to G. For a given partition $\Omega = G \cup U$, depending on G it could well be that X had transitions from U to G, violating the assumption. However, the assumption does not in fact impose any real restriction to the selection of G, since it is enough to redefine X so that U includes copies of the states of G reachable from U to satisfy the assumption. Thus, the only limitation imposed to G is $P[X(0) \in G] = 1$. The transition rates among the states of G are as in X; the transition rates from states $a \in G$ to u_d, $1 \leq d \leq L$ have values λ_{a,U_d}, being U_d the subset of U including the states with failure distance d; finally, denoting by FC the set of different cardinalities of the failure events of the model and by E_i the subset of failure events with cardinality i, and defining $f_i = \sum_{e \in E_i} \lambda_{\text{ub}}(e)$, for each $1 \leq d \leq L$, $i \in FC$ there is a transition rate f_i from u_d to $u_{\max\{0, d-i\}}$. Figure 3.1 illustrates the structure of X'. The initial probability distribution of X' is $P[X'(0) = i] = P[X(0) = i]$, $i \in G$; $P[X'(0) = f] = 0$; $P[X'(0) = u_d] = 0$, $0 \leq d \leq L$.

The bounds are:

$$[ur(t)]_{\text{lb}} = P[X'(t) = f], \tag{3.1}$$

$$[ur(t)]_{\text{ub}} = P[X'(t) \in \{u_0, f\}]. \tag{3.2}$$

The correctness of the lower bound (3.1) is trivial, since X' enters f when X enters f from G; the correctness of the upper bound (3.2) will be shown next.

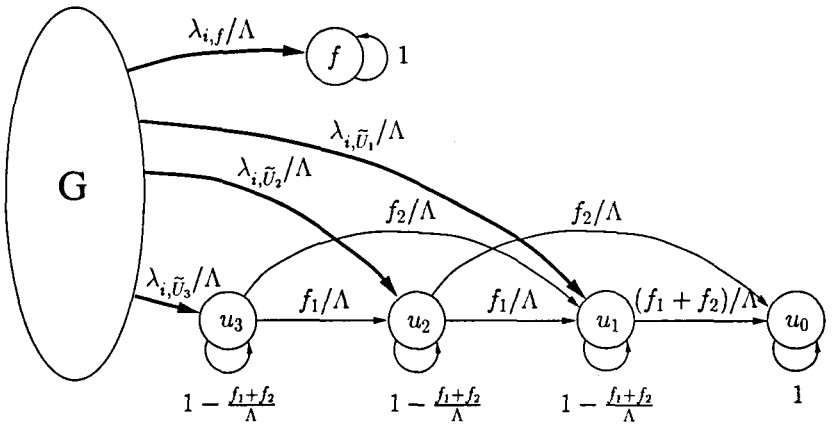

Figure 3.2: State transition diagram of DTMC Y' for $L = 3$ and $FC = \{1, 2\}$.

The proof of the upper bound will be done through a lemma, two propositions and a theorem and will make reference to the discrete-time Markov chains (DTMC) $Y = \{Y_n; n = 0, 1, \ldots\}$ and $Y' = \{Y'_n; n = 0, 1, \ldots\}$ obtained by randomizing [49], respectively, X and X' with a rate Λ, greater than or equal to the maximum output rate of X (for instance, $\Lambda = \sum_{e \in E} \lambda_{\text{ub}}(e)$). The DTMC Y has the same state space and initial probability distribution as X and transition probabilities $q_{a,b} = \lambda_{a,b}/\Lambda$, $a \neq b$, $q_{a,a} = 1 - \lambda_a/\Lambda$. The DTMC Y' has the same state space and initial probability distribution as X' and transition probabilities $q'_{a,b} = \lambda'_{a,b}/\Lambda$, $a \neq b$, $q'_{a,a} = 1 - \lambda'_a/\Lambda$, where $\lambda'_{a,b}$ and λ'_a are, respectively, the transition and output rates of X'. Figure 3.2 illustrates the state transition diagram of Y'. It is well-known (see, for instance, [83]) that $X(t) = Y_{N(t)}$ and $X'(t) = Y'_{N(t)}$, where $N = \{N(t); t \geq 0\}$ is a Poisson process with rate Λ. These results allow to express the transient solution of X (X') in terms of the transient solution of Y (Y'):

$$P[X(t) = a] = \sum_{n=0}^{\infty} \frac{(\Lambda t)^n}{n!} e^{-\Lambda t} P[Y_n = a], \tag{3.3}$$

$$P[X'(t) = a] = \sum_{n=0}^{\infty} \frac{(\Lambda t)^n}{n!} e^{-\Lambda t} P[Y'_n = a]. \tag{3.4}$$

Intuitively, it is clear that the probability that Y' will reach the absorbing state u_0 from u_d, $d \geq 0$ in m steps decreases with d. The result is established in the following lemma. Let

$$R'_m(d) = P[Y'_m = u_0 | Y'_0 = u_d].$$

Lemma 3.1 *$R'_m(d), m > 0, d \geq 0$ is decreasing on d.*

Proof From the structure of Y' we can write

$$R'_m(0) = 1,\ m > 0\,, \tag{3.5}$$

$$R'_1(d) = \sum_{\substack{i \in FC \\ i \geq d}} \frac{f_i}{\Lambda},\ d > 0\,, \tag{3.6}$$

and for $m > 1\,, d > 0$,

$$R'_m(d) = \Big(1 - \frac{1}{\Lambda} \sum_{i \in FC} f_i\Big) R'_{m-1}(d) + \sum_{i \in FC} \frac{f_i}{\Lambda} R'_{m-1}(\max\{0, d-i\})\,. \tag{3.7}$$

The proof is by induction on m.

Base case ($m = 1$): We show $R'_1(d) \leq R'_1(d-1)\,,\ d > 0$. For $d = 1$, using (3.6) and (3.5) we have:

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

For $d > 1$, using (3.6),

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

Induction step: Let $m > 0$; we will assume that $R'_m(d)\,,\ d \geq 0$ is decreasing on d and will show $R'_{m+1}(d) \leq R'_{m+1}(d-1)\,,\ d > 0$. For $d = 1$, using (3.7), $R'_m(1) \leq 1$ and (3.5) we have:

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

For $d > 1$, using (3.7) and the induction hypothesis,

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

Let us define $R_m(a) = P[Y_m = f | Y_0 = a]$. We have the following result.

Proposition 3.1 $R_m(a) \leq R'_m(d), a \in U_d, m > 0, d > 0$.

Proof Let $\lambda^i_{a,f}$ be the contribution to $\lambda_{a,f}$ associated with failure bags $e \in E_i$. We have $\lambda^i_{a,f} \leq f_i$. Since a failure bag $e \in E_i$ reduces the failure distance at most by i, $\lambda_{a,f}$ will not have contributions $\lambda^i_{a,f}$ for $i < d$, and

$$R_1(a) = \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f}}{\Lambda}. \tag{3.8}$$

Let us denote by $U_{k,d}$ the subset of U including the states with k failed components and failure distance d. For $m > 1$, taking into account that f is absorbing,

$$\begin{aligned} R_m(a) &= \left(1 - \frac{\lambda_a}{\Lambda}\right) R_{m-1}(a) + \sum_{\substack{i \in FC \\ i \geq d}} \left[\frac{\lambda^i_{a,f}}{\Lambda} + \sum_{d'=1}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R_{m-1}(b)\right] \\ &+ \sum_{\substack{i \in FC \\ i < d}} \sum_{d'=d-i}^{d} \sum_{b \in U_{k+i,d'}} \frac{\lambda_{a,b}}{\Lambda} R_{m-1}(b). \end{aligned} \tag{3.9}$$

The proof is by induction on m.

Base case ($m = 1$): We will show $R_1(a) \leq R'_1(d)$, $a \in U_d, d > 0$. Using (3.8), $\lambda^i_{a,f} \leq f_i$, and (3.6),

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

Induction step: Let $m > 0$; we will assume $R_m(a) \leq R'_m(d), a \in U_d, d > 0$ and show $R_{m+1}(a) \leq R'_{m+1}(d), a \in U_d, d > 0$. Using (3.9) and the induction hypothesis,

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

Taking into account that $R'_m(d') \leq 1$, $\sum_{b \in U_{k+i,d'}} \lambda_{a,b} = \lambda_{a,U_{k+i,d'}}$, and using Lemma 3.1,

$$R_{m+1}(a) \leq \left(1 - \frac{\lambda_a}{\Lambda}\right) R'_m(d) + \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f} + \sum_{d'=1}^{d} \lambda_{a,U_{k+i,d'}}}{\Lambda} + \sum_{\substack{i \in FC \\ i < d}} R'_m(d-i) \sum_{d'=d-i}^{d} \frac{\lambda_{a,U_{k+i,d'}}}{\Lambda}. \tag{3.10}$$

Let U^k be the subset of U including the states with k failed components. The inequality (3.10) can be written as

$$R_{m+1}(a) \leq \left(1 - \frac{\lambda_a}{\Lambda}\right) R'_m(d) + \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda} + \sum_{\substack{i \in FC \\ i < d}} R'_m(d-i) \frac{\lambda_{a,U^{k+i}}}{\Lambda}. \tag{3.11}$$

Taking into account that $\lambda_a = \lambda_{a,f} + \sum_{i \in FC} \lambda_{a,U^{k+i}}$,

$$\frac{\lambda_a}{\Lambda} = \sum_{\substack{i \in FC \\ i \geq d}} \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda} + \sum_{\substack{i \in FC \\ i < d}} \frac{\lambda_{a,U^{k+i}}}{\Lambda}. \tag{3.12}$$

Combining (3.11) and (3.12),

$$R_{m+1}(a) \leq R'_m(d) + \sum_{\substack{i \in FC \\ i \geq d}} \left[1 - R'_m(d)\right] \frac{\lambda^i_{a,f} + \lambda_{a,U^{k+i}}}{\Lambda} + \sum_{\substack{i \in FC \\ i < d}} \left[R'_m(d-i) - R'_m(d)\right] \frac{\lambda_{a,U^{k+i}}}{\Lambda}.$$

Finally, noting that, for $i \geq d$, $\lambda^i_{a,f} + \lambda_{a,U^{k+i}} \leq f_i$ and that, for $i < d$, $\lambda_{a,U^{k+i}} \leq f_i$, and using (3.5) and (3.7),

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

Using Proposition 3.1 it is possible to show the following result.

Proposition 3.2 $P[Y_n = f] \leq P[Y'_n \in \{u_0, f\}], n > 0$.

Proof Y can enter f through U or directly from G. Taking into account that f is absorbing and conditioning the entry of Y in f through U to the step in which Y leaves G and the entry state in U, we have:

$$\begin{aligned} P[Y_n = f] &= \sum_{m=1}^{n-1}\sum_{d=1}^{L}\sum_{a\in U_d} P[Y_{m-1} \in G \wedge Y_m = a]P[Y_n = f|Y_m = a] \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f] \\ &= \sum_{m=1}^{n-1}\sum_{d=1}^{L}\sum_{a\in U_d} P[Y_{m-1} \in G \wedge Y_m = a]R_{n-m}(a) \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f]. \end{aligned}$$

Invoking Proposition 3.1 and using the relationships between Y and Y',

$$\begin{aligned} P[Y_n = f] &\leq \sum_{m=1}^{n-1}\sum_{d=1}^{L}\sum_{a\in U_d} P[Y_{m-1} \in G \wedge Y_m = a]R'_{n-m}(d) \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f] \\ &= \sum_{m=1}^{n-1}\sum_{d=1}^{L} P[Y'_{m-1} \in G \wedge Y'_m = u_d]R'_{n-m}(d) \\ &\quad + \sum_{m=1}^{n} P[Y'_{m-1} \in G \wedge Y'_m = f] \\ &= \sum_{m=1}^{n-1}\sum_{d=1}^{L} P[Y'_{m-1} \in G \wedge Y'_m = u_d]P[Y'_n = u_o|Y'_m = u_d] \\ &\quad + \sum_{m=1}^{n} P[Y'_{m-1} \in G \wedge Y'_m = f] \\ &= P[Y'_n = u_0] + P[Y'_n = f] = P[Y'_n \in \{u_0, f\}]. \square \end{aligned}$$

The last proposition allows us to prove the following, desired result.

Theorem 3.1 $\mathrm{ur}(t) \leq [\mathrm{ur}(t)]_{\mathrm{ub}}$.

Proof Since Y is the result of randomizing X, using (3.3), taking into account that

$P[Y_0 = f] = P[X(0) = f] = 0,$

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

Invoking Proposition 3.2 and using (3.4) with $P[Y'_0 \in \{u_0, f\}] = P[X'(0) \in \{u_0, f\}] = 0$, and (3.2):

$$ur(t) \leq \sum_{n=1}^{\infty} \frac{(\Lambda t)^n}{n!} e^{-\Lambda t} P[Y'_n \in \{u_0, f\}] = P[X'(t) \in \{u_0, f\}] = [ur(t)]_{\text{ub}}. \square$$

3.2 Analysis and Comparison

In this section we analyze the performance of the proposed bounding method using a complex example with dependencies that prevent from using combinatoric and hierarchical solution methods, and compare the quality of the bounds obtained with the proposed method with the bounds obtained using the trivial method in which the upper bound assumes that the system fails when the model exits G. That bound can be expressed in terms of the transient regime of X' as:

$$[ur(t)]'_{\text{ub}} = P[X'(t) \in \{u_0, \cdots, u_L, f\}].$$

The transient regime of X' has been solved using the randomization method [49].

The example is a system made up of 38 components. The architecture of the system is shown in Figure 3.3. The system is made up of a cluster of redundant master processing units MPU_1 and MPU_2 that are communicated with $n = 5$ clusters of redundant slave processing units $\text{SPU}_{i,1}$ and $\text{SPU}_{i,2}$, $1 \leq i \leq n$. Communication is done through two redundant buses BA and BB to which the master and slave units are connected through dedicated interfaces. The system is operational if some fault-free master processing unit can communicate directly (i.e. through one fault-free bus and two fault-free interfaces) with at least one fault-free slave processing unit of each slave cluster. The active configuration of the system includes a master processing unit, with priority given to MPU_1, all fault-free slave processing units which can communicate with the active master processing unit, and the busses and interfaces among these units and the active processing master unit. Master processing unit MPU_2 is activated only if MPU_1 is faulty or it is impossible to build up an operational configuration with MPU_1 (for instance, because both interfaces associated with MPU_1 are failed).

Active master processing units, slave processing units, interfaces and busses fail with rates λ_M, λ_S, λ_I, and λ_B, respectively. Passive components fail with rates $\delta_M \lambda_M$,

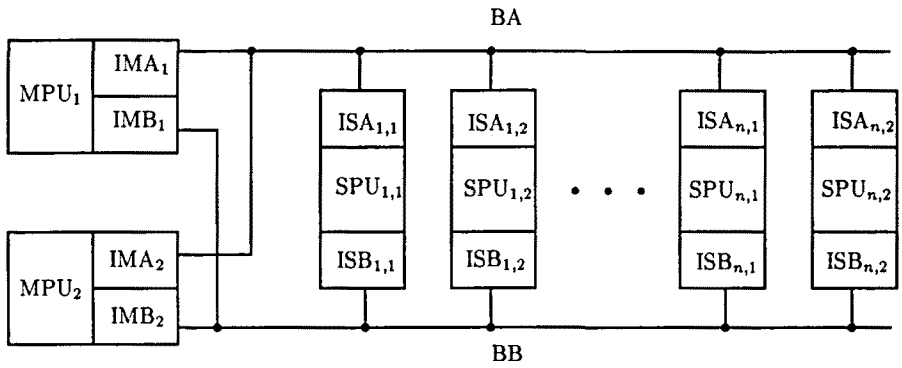

Figure 3.3: Architecture of the example.

$\delta_S\lambda_S$, $\delta_I\lambda_I$, and $\delta_B\lambda_B$, respectively, being δ_M, δ_S, δ_I, and δ_B dormancy coefficients < 1. The fault of an active or passive interface is propagated to the bus to which the interface is connected with probability ν. Coverage failures are modeled by propagating the component fault to two "recovery" components, one of which has to be unfailed for the system to be up. The coverage model of the example includes the parameters C_M, coverage to the failure of MPU_1, C_S^H, C_B^H, C_I^H, and C_{IB}^H, coverages to the failures of, respectively, a slave processing unit, a bus, an interface whose failure is not propagated to the bus, and an interface whose failure is propagated to the bus, when the reconfiguration of the system does not involve the activation of MPU_2, and C_S^L, C_B^L, C_I^L, and C_{IB}^L, homologous coverages when the reconfiguration involves the activation of MPU_2.

For the example, $FC = \{1, 2, 3, 4\}$ and for the upper bounds f_i we can take:

$$
\begin{aligned}
f_1 &= \max\{\lambda_M C_M, \lambda_M \delta_M\} + \lambda_M + 10\max\{\lambda_S C_S^H, \lambda_S C_S^L, \lambda_S \delta_S\} \\
&\quad + 2\max\{\lambda_B C_B^H, \lambda_B C_B^L, \lambda_B \delta_B\} + 24(1-\nu)\max\{\lambda_I C_I^H, \lambda_I C_I^L, \lambda_I \delta_I\}, \\
f_2 &= 24\nu \max\{\lambda_I C_{IB}^H, \lambda_I C_{IB}^L\}, \\
f_3 &= \lambda_M(1 - C_M) + 10\max\{\lambda_S(1 - C_S^H), \lambda_S(1 - C_S^L)\} \\
&\quad + 2\max\{\lambda_B(1 - C_B^H), \lambda_B(1 - C_B^L)\} \\
&\quad + 24(1-\nu)\max\{\lambda_I(1 - C_I^H), \lambda_I(1 - C_I^L)\}, \\
f_4 &= 24\nu \max\{\lambda_I(1 - C_{IB}^H), \lambda_I(1 - C_{IB}^L)\}.
\end{aligned}
$$

The numerical results have been obtained for $\lambda_M = 1.2\times10^{-6}\,\mathrm{h}^{-1}$, $\lambda_S = 6\times10^{-7}\,\mathrm{h}^{-1}$, $\lambda_B = 6 \times 10^{-8}\,\mathrm{h}^{-1}$, $\lambda_I = 1.2 \times 10^{-7}\,\mathrm{h}^{-1}$, $\delta_M = \delta_S = \delta_B = \delta_I = 0.2$, $\nu = 0.1$, $C_M = 0.95$, $C_S^H = C_B^H = C_I^H = 0.99$, $C_S^L = C_B^L = C_I^L = 0.95$, $C_{IB}^H = 0.97$ and $C_{IB}^L = 0.93$. The corresponding values of the bounds f_i are $f_1 = 1.096488 \times 10^{-5}\,\mathrm{h}^{-1}$, $f_2 = 2.7936 \times 10^{-7}\,\mathrm{h}^{-1}$, $f_3 = 4.956 \times 10^{-7}\,\mathrm{h}^{-1}$ and $f_4 = 2.016 \times 10^{-8}\,\mathrm{h}^{-1}$. As initial state we have

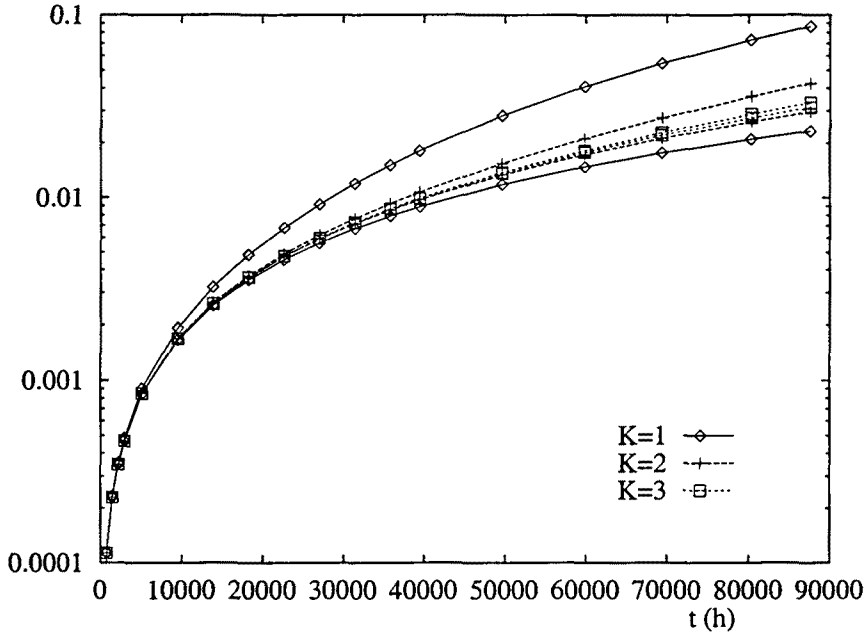

Figure 3.4: Unreliability bounds obtained with the proposed bounding method as a function of the time in hours for $K = 1, 2, 3$.

assumed the state in which no component is failed. As subset of generated states G we have taken the set including all operational states with up to K failed components, with $K = 1, 2, 3$. The cardinality $|G|$ of the generated state space is 39 for $K = 1$, 735 for $K = 2$, and 8,871 for $K = 3$. The fault tree of the system has 512 minimal cuts: 8 of cardinality 2, 48 of cardinality 3, 96 of cardinality 4 and 360 of cardinality 6. Failure distances have been computed with the control parameter R set to 2. The time overhead associated with failure distances computation is about 10 %. However, the part of that overhead which depends on the number of minimal cut touches is only 0.3 % and, thus, we feel that systems with of the order of tens of thousands of minimal cuts can be dealt with without significant time overhead due to failure distances computations. However, for systems having that number of minimal cuts the memory overhead associated with them and the structures required for the efficient computation of failure distances would be significant. For the example, whose fault tree has only 512 minimal cuts, the memory overhead for $K = 3$ was insignificant.

Figure 3.4 shows the unreliability bounds as a function of the time t for the three considered values of K. It can be shown that very tight bounds are obtained with a reasonable number of states (8,871 for $K = 3$) even for large times. The tightness of the bounds increases for decreasing times. Table 3.1 shows, for several mission times, the relative band obtained by the proposed method, $([ur(t)]_{ub} - [ur(t)]_{lb})/[ur(t)]_{lb}$, and

Table 3.1: Relative bands for several mission times obtained with the proposed method (top) and the trivial method (down).

time	K (states)		
	1 (39)	2 (735)	3 (8,871)
1 month	0.007761	3.213×10^{-5}	6.665×10^{-8}
	1.887	1134×10^{-5}	3118×10^{-8}
2 months	0.01637	1.295×10^{-4}	5.300×10^{-7}
	2.063	230.7×10^{-4}	1245×10^{-7}
6 months	0.05846	1.197×10^{-3}	1.400×10^{-5}
	2.711	73.45×10^{-3}	111.1×10^{-5}
1 year	0.1408	4.930×10^{-3}	1.084×10^{-4}
	3.548	157.2×10^{-3}	43.80×10^{-4}
2 years	0.3545	0.02032	8.114×10^{-4}
	4.860	0.3424	169.2×10^{-4}
5 years	1.187	0.1249	0.01043
	7.202	0.9228	0.09300
10 years	2.724	0.4339	0.06110
	8.987	1.740	0.2926

the relative band obtained with the trivial method, $([ur(t)]'_{ub} - [ur(t)]_{lb})/[ur(t)]_{lb}$. The proposed bounding method outperforms significantly the trivial method, especially for short and medium mission times.

3.3 Conclusions

The failure distance-based bounding method for $ur(t)$ developed in this chapter seems to outperform significantly the trivial method. Its performance degrades as t increases. Nevertheless, using the method, it is possible to obtain tight bounds with relatively few states even for significantly large t. The time overhead due to failure distances computation is small and will remain small even if the fault tree has many minimal cuts (of the order of tens of thousands). The memory overhead due to holding the minimal cuts and the structures required for the efficient computation of failure distances can be significant if the number of minimal cuts is very large. In the next chapter, we will develop another method to obtain bounds for $ur(t)$ based on lower bounds for failure

distances that does not have that memory overhead.

Chapter 4

Reliability Bounds of Non-repairable Systems using FD Bounds

In this chapter we will develop and describe a method to compute bounds for the unreliability at time t, $ur(t)$, based on lower bounds for failure distances which are computed on the fault tree and, thus, does not require the knowledge of the minimal cuts of the fault tree. The motivation is, first, to avoid having to solve a hard problem (the determination of the minimal cuts), for which algorithms could fail to provide a solution in reasonable memory and time. Secondly, the number of minimal cuts could be very large, making significant the memory overhead of the bounding method described in Chapter 3 associated with the holding of the minimal cuts and structures used by the algorithms for failure distances computation. Thus, the bounding method developed in this chapter could be more efficient in terms of memory required to achieve a given bounds tightness than the method proposed in Chapter 3. We start by stating the properties that the lower bounds for the failure distances will fulfill and showing how $ur(t)$ can be bounded using those failure distances bounds. Then we show how lower bounds for failure distances fulfilling those properties can be computed on the fault tree. We end the chapter by analyzing the performance of the proposed bounding method and comparing it in terms of bounds tightness and CPU time with the performances of both the trivial method described in Section 1.3 and the method proposed in Chapter 3. Throughout the chapter we will use the following specific notation. Also, to avoid trivialities, we will assume that no fault tree inputs x, y with associated bags $c[n]$, $c[n']$, $n \neq n'$ feed the same gate. This is not a

real restriction since for $n' > n$ and an OR gate, x, y can be substituted by x and for an AND gate by y.

Notation and Definitions

$[ur(t)]_{\text{lb}}$ lower bound for $ur(t)$ obtained with the method proposed in the chapter, the trivial method and the method proposed in Chapter 3

$[ur(t)]_{\text{ub}}$ upper bound for $ur(t)$ obtained with the method proposed in this chapter

$[ur(t)]'_{\text{ub}}$ upper bound for $ur(t)$ obtained with the method proposed in Chapter 3

$[ur(t)]''_{\text{ub}}$ upper bound for $ur(t)$ obtained with the trivial method

$X = \{X(t); t \geq 0\}$ acyclic CTMC modeling the system

$X' = \{X'(t); t \geq 0\}$ acyclic CTMC used for computing $[ur(t)]_{\text{lb}}$ and $[ur(t)]_{\text{ub}}$

λ_a, $\lambda_{a,b}$, $\lambda_{a,B}$ respectively, output rate of state a, transition rate from state a to state b and $\sum_{b \in B} \lambda_{a,b}$ for CTMC X

$Y = \{Y_n; n = 0, 1, \ldots\}$, $Y' = \{Y'_n; n = 0, 1, \ldots\}$ respectively, DTMC obtained by randomizing [83] X, X' with rate Λ greater than or equal to the maximum output rate of both X and X'

O set of states of X in which the system is operational (up states)

f absorbing state that represents the failure of the system (down state)

G subset of O that is generated

U $O - G$

L $d(o)$

U_d $\{a \in U | \, d(a) = d\}$

U^k $\{a \in U |$ the number of failed components in a is $k\}$

$U_{k,d}$ $U^k \cap U_d$

$\widetilde{d}(a)$ lower bound for $d(a)$; it verifies $\widetilde{d}(a) = 0$ if and only if $d(a) = 0$, and $\widetilde{d}(a) \leq \widetilde{d}(o)$

$\widetilde{L}$ $\widetilde{d}(o)$

$\widetilde{U}_{d,i}$ $\{a \in U_d | \, \widetilde{d}(a) = i\}$

$\widetilde{U}_d$ $\{a \in U | \, \widetilde{d}(a) = d\}$

E_i $\{e \in E | \, |e| = i\}$

FC set of different cardinalities of failure bags

$\lambda(e)$ rate with which failure bag $e \in E$ is realized; it may be state dependent

$f_i \quad \sum_{e \in E_i} \lambda_{\mathrm{ub}}(e)$

4.1 Bounding Method

We give in Figure 4.1 the architecture of an example system, adapted from [61], which will be used for illustration purposes. The system consists of two memory modules MM_1 and MM_2, three identical CPU chips $CPUC$ and two identical port chips PTC. One CPU chip and one port chip are spares. Each memory module MM_j is made up of ten memory chips MC_j, two of which are spares, and one interface chip IC_j. Active memory chips MC_j and interface chips IC_j fail, respectively, with rate λ_{MC_j} and λ_{IC_j}. Active port chips PTC and CPU chips $CPUC$ fail, respectively, with rates λ_{PTC} and λ_{CPUC}. Spare chips fail with rates $\nu \times \lambda_{MC_j}$, $\nu \times \lambda_{IC_j}$, $\nu \times \lambda_{PTC}$, and $\nu \times \lambda_{CPUC}$, being ν, $0 < \nu < 1$ a dormancy factor. Recovery is hierarchical. A fault in a memory chip is covered with probability C_{MC}. A faulty memory module, CPU chip and port chip are successfully covered with probabilities C_{MM}, C_{CPUC} and C_{PTC}, respectively. To model imperfect coverage, an uncovered fault in a memory chip of memory module MM_j is propagated to a fictitious component RMM_j, and an uncovered failure of a memory module MM_j, a CPU chip or a port chip is propagated to two fictitious components RCM. Memory module MM_j is operational if at least eight memory chips MC_j, the interface chip IC_j and the fictitious component RMM_j are unfailed. The system is operational if at least one memory module MM_j is operational, and at least two CPU chips $CPUC$, one port chip PTC and one fictitious component RCM are unfailed.

Table 4.1 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 dormancy factor. Thus, for instance, failure bag e_1 stands for the fault of a memory chip of the first memory module that is covered at memory module level, e_2 stands for the fault of that chip that is uncovered at memory module level and covered at system level, and e_3 stands for the uncovered fault of the chip. Note that $FC = \{1, 2, 3, 4\}$ and that $f_1 = \lambda_{\mathrm{ub}}(e_1) + \lambda_{\mathrm{ub}}(e_4) + \lambda_{\mathrm{ub}}(e_6) + \lambda_{\mathrm{ub}}(e_9) + \lambda_{\mathrm{ub}}(e_{11}) + \lambda_{\mathrm{ub}}(e_{13})$, $f_2 = \lambda_{\mathrm{ub}}(e_2) + \lambda_{\mathrm{ub}}(e_7)$, $f_3 = \lambda_{\mathrm{ub}}(e_5) + \lambda_{\mathrm{ub}}(e_{10}) + \lambda_{\mathrm{ub}}(e_{12}) + \lambda_{\mathrm{ub}}(e_{14})$, and $f_4 = \lambda_{\mathrm{ub}}(e_3) + \lambda_{\mathrm{ub}}(e_8)$.

The CTMC X' used to compute $[ur(t)]_{\mathrm{lb}}$ and $[ur(t)]_{\mathrm{ub}}$ has state space $G \cup \{f\} \cup \{u_0, \ldots, u_{\widetilde{L}}\}$. Although other selections for G would be possible, we assume that G includes all the states of the model with up to K failed components and that $P[X(0) \in G] = 1$. The states u_d, $0 \leq d \leq \widetilde{L}$ pessimistically approximate the behavior of X in U. The transition rates in X' from a to b, $a, b \in G$ and from a to f, $a \in G$ are as in

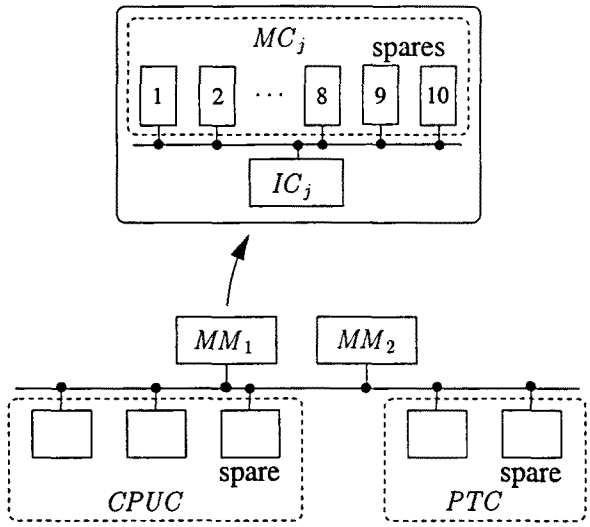

Figure 4.1: Architecture of the example system.

Table 4.1: Failure bags of the example system and, for each failure bag e, a suitable upper bound $\lambda_{\text{ub}}(e)$.

	description	$\lambda_{\text{ub}}(e)$
e_1	$MC_1[1]$	$(8+2\nu)\lambda_{MC_1}C_{MC}$
e_2	$MC_1[1]\ RMM_1[1]$	$(8+2\nu)\lambda_{MC_1}(1-C_{MC})C_{MM}$
e_3	$MC_1[1]\ RMM_1[1]\ RCM[2]$	$(8+2\nu)\lambda_{MC_1}(1-C_{MC})(1-C_{MM})$
e_4	$IC_1[1]$	$\lambda_{IC_1}C_{MM}$
e_5	$IC_1[1]\ RCM[2]$	$\lambda_{IC_1}(1-C_{MM})$
e_6	$MC_2[1]$	$(8+2\nu)\lambda_{MC_2}C_{MC}$
e_7	$MC_2[1]\ RMM_2[1]$	$(8+2\nu)\lambda_{MC_2}(1-C_{MC})C_{MM}$
e_8	$MC_2[1]\ RMM_2[1]\ RCM[2]$	$(8+2\nu)\lambda_{MC_2}(1-C_{MC})(1-C_{MM})$
e_9	$IC_2[1]$	$\lambda_{IC_2}C_{MM}$
e_{10}	$IC_2[1]\ RCM[2]$	$\lambda_{IC_2}(1-C_{MM})$
e_{11}	$CPUC[1]$	$(2+\nu)\lambda_{CPUC}C_{CPUC}$
e_{12}	$CPUC[1]\ RCM[2]$	$(2+\nu)\lambda_{CPUC}(1-C_{CPUC})$
e_{13}	$PTC[1]$	$(1+\nu)\lambda_{PTC}C_{PTC}$
e_{14}	$PTC[1]\ RCM[2]$	$(1+\nu)\lambda_{PTC}(1-C_{PTC})$

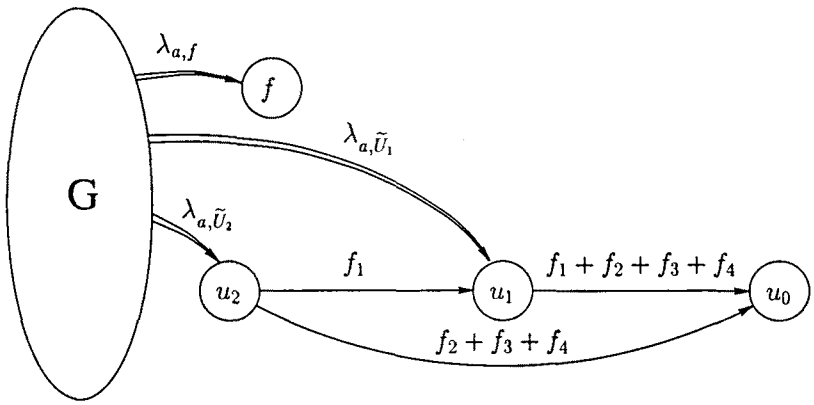

Figure 4.2: State transition diagram of X' for the example system ($\widetilde{L} = 2$, $FC = \{1,2,3,4\}$).

X. The transition rates from states $a \in G$ to u_d, $1 \leq d \leq \widetilde{L}$ have values $\lambda_{a,\widetilde{U}_d}$, and for each $1 \leq d \leq \widetilde{L}$, $i \in FC$ 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. As it will be shown in Section 4.2, $\widetilde{L} = 2$ for the example system. Figure 4.2 shows the structure of X' for the example system.

The bounds are

$$\begin{aligned} [ur(t)]_{\text{lb}} &= P[X'(t) = f]\,, \\ [ur(t)]_{\text{ub}} &= P[X'(t) \in \{u_0, f\}]\,. \end{aligned} \tag{4.1}$$

The correctness of $[ur(t)]_{\text{lb}}$ is trivial. The correctness of $[ur(t)]_{\text{ub}}$ is proved next under the conditions:

1. $1 \leq \widetilde{d}(a) \leq d(a), a \in U$,
2. $\widetilde{d}(a) \leq \widetilde{L}$.

We will construct the proof with the aid of the DTMC Y and Y'. Since [83] $X(t) = Y_{N(t)}$ and $X'(t) = Y'_{N(t)}$, where $N = \{N(t); t \geq 0\}$ is a Poisson process with arrival rate Λ, we have

$$P[X(t) = a] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} P[Y_n = a]\,, \tag{4.2}$$

$$P[X'(t) = a] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} P[Y'_n = a]\,. \tag{4.3}$$

Let

$$R'_m(d) = P[Y'_m = u_0 \mid Y'_0 = u_d], \tag{4.4}$$

$$R_m(a) = P[Y_m = f \mid Y_0 = a]. \tag{4.5}$$

The following two results related to the "bounding" states u_i are taken from Chapter 3.

Lemma 4.1 *$R'_m(d)$, $m > 0$, $d > 0$ is decreasing on d.*

Proposition 4.1 *$R_m(a) \leq R'_m(d)$, $a \in U_d$, $m > 0$, $d > 0$.*

In Chapter 3 it has also been shown that if X' is built using transition rates λ_{a,U_d} from states $a \in G$ to u_d, $1 \leq d \leq L$, then $P[Y_n = f] \leq P[Y'_n \in \{u_0, f\}]$. Since $R'_m(d)$ decreases on d, it seems intuitively clear that after substituting rates λ_{a,U_d} by rates $\lambda_{a,\tilde{U}_d}$ (i.e. leading to u_i, $i < d$ part of the transitions which went from $a \in G$ to u_d), $P[Y'_n \in \{u_0, f\}]$ will also upper bound $P[Y_n = f]$. This result is formally proved next.

Proposition 4.2 *Assume $1 \leq \tilde{d}(a) \leq d(a)$, $a \in U$, $\tilde{d}(a) \leq \tilde{L}$ and $P[X(0) \in G] = 1$. Then, $P[Y_n = f] \leq P[Y'_n \in \{u_0, f\}]$, $n > 0$.*

Proof Y can enter f through U or directly from G. Taking into account that 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 (4.5),

$$\begin{aligned} P[Y_n = f] &= \sum_{m=1}^{n-1} \sum_{a \in U} P[Y_{m-1} \in G \wedge Y_m = a] P[Y_n = f \mid Y_m = a] \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f] \\ &= \sum_{m=1}^{n-1} \sum_{a \in U} P[Y_{m-1} \in G \wedge Y_m = a] R_{n-m}(a) + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f]. \end{aligned}$$

Since $1 \leq \tilde{d}(a) \leq d(a)$, $\tilde{d}(a) \leq \tilde{L}$, $a \in U$, U_d can be partitioned as

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

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 the above partition of U and Proposition 4.1,

$$\begin{aligned} P[Y_n = f] &= \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \sum_{d=i}^{L} \sum_{a \in \widetilde{U}_{d,i}} P[Y_{m-1} \in G \wedge Y_m = a] R_{n-m}(a) \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f] \\ &\leq \sum_{m=1}^{n-1} \sum_{i=1}^{\widetilde{L}} \sum_{d=i}^{L} \sum_{a \in \widetilde{U}_{d,i}} P[Y_{m-1} \in G \wedge Y_m = a] R'_{n-m}(d) \\ &\quad + \sum_{m=1}^{n} P[Y_{m-1} \in G \wedge Y_m = f]\,. \end{aligned}$$

Then, using Lemma 4.1, $\widetilde{U}_i = \bigcup_{d=i}^{L} \widetilde{U}_{d,i}$, the relations between Y and Y' and (4.4),

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

Finally,

Theorem 4.1 *Assume* $1 \leq \widetilde{d}(a) \leq d(a)$, $a \in U$, $\widetilde{d}(a) \leq \widetilde{L}$ *and* $P[X(0) \in G] = 1$. *Then,* $ur(t) \leq [ur(t)]_{\mathrm{ub}}$.

Proof Using (4.2) and $P[Y_0 = f] = P[X(0) = f] = 0$,

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

Using Proposition 4.2, $P[Y_0' \in \{u_0, f\}] = P[X'(0) \in \{u_0, f\}] = 0$, (4.3), and (4.1),

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

4.2 Lower Bounds for Failure Distances

In this section we obtain lower bounds $\widetilde{d}(a) = \widetilde{d}_{\mathrm{b}}(F(a), g_r)$ for failure distances from states a. We prove that the bounds fulfill the necessary requirements: $\widetilde{d}(a) = 0$ if and only if $d(a) = 0$ (to know from $\widetilde{d}(a)$ if a is operational or down) and $1 \leq \widetilde{d}(a) \leq d(a)$, $a \in U$, $\widetilde{d}(a) \leq \widetilde{L}$ (Theorem 4.1). We give a sufficient condition for $d(a) = \widetilde{d}(a)$ and a lower and an upper bound for $\widetilde{d}_{\mathrm{b}}(F, x)$, $F = F' + F''$ in terms of $\widetilde{d}_{\mathrm{b}}(F', x)$. Finally, an efficient algorithm to compute the lower bounds for failure distances on the fault tree is derived and illustrated. Throughout the section we will use the following specific notation and definitions.

Notation and Definitions

C	set of component classes
I	set of inputs (basic events) of the fault tree; each input x has associated a different bag, $\mathrm{b}(x)$, of the form $c[n]$, $c \in C$, $n \geq 1$, meaning the failure of n components of class c
P	set of gates (complex events) of the fault tree
g_r	root gate (top event) of the fault tree
node or event	gate or input of the fault tree
type(x)	type of node x: AND, OR if $x \in G$; input if $x \in I$
related	two inputs x, y are related if $\mathrm{b}(x) = c[n]$ and $\mathrm{b}(y) = c[n']$, $n \neq n'$
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 of an input or a gate, which may be 1 or 0. For inputs $x \in I$, $\mathrm{b}(x) = c[n]$, $\mathrm{val}(x) = 1$ if and only if n or more components of class c are failed; for gates $x \in P$, its value is determined as usual from the values of $y \in \mathrm{fi}(x)$

realized an event x is said to be realized if $\mathrm{val}(x) = 1$

$\mathrm{lev}(\cdot)$ level of an input or a gate. For inputs $x \in I$, $\mathrm{lev}(x) = 0$; for gates $y \in P$, $\mathrm{lev}(y) = 1 + \max_{z \in \mathrm{fi}(y)}\{\mathrm{lev}(z)\}$

path a sequence of nodes $x_1 \ldots x_k$ 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

$\mathrm{Reach}(x)$ set of nodes reachable from node x plus x itself

$\mathrm{Support}(x)$ $I \cap \mathrm{Reach}(x)$, $x \in I \cup P$

independent two nodes $x, y \in I \cup P$ are said to be independent if $\mathrm{Support}(x) \cap \mathrm{Support}(y) = \emptyset$ and for each $z \in \mathrm{Support}(x)$, $\mathrm{b}(z) = c[n]$, no $t \in \mathrm{Support}(y)$, $\mathrm{b}(t) = c[n']$ exists

$S(F, x)$ $\left|F \cap \left(\sum_{y \in \mathrm{Support}(x)} \mathrm{b}(y)\right)\right|$, being F a bag of failed component classes and x a node

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

$d_{\mathrm{b}}(F, x)$ being F a bag of component classes and x an event, minimum number of components which have to fail in addition to those in F to realize x; it is called distance from bag F to 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 failure distance from a failure bag $e \in E$ to g_r

4.2.1 Recursive Definition of Lower Bounds for Failure Distances

Note that from the above definitions, $d(a) = d_{\mathrm{b}}(F(a), g_r)$. The computed lower bounds are $\widetilde{d}(a) = \widetilde{d}_{\mathrm{b}}(F(a), g_r)$. The lower bounds $\widetilde{d}_{\mathrm{b}}(F(a), g_r)$ are computed on the fault tree of the system using the concept of *module*, which generalizes to component classes the definition given in [57, 40], 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, we use the algorithm LTA/DR of [40] with a small modification to take into account component classes: during the first depth-first left-most traversal of the fault tree (step no. 2 of the algorithm), visit to $x \in I$ implies simultaneous visit (i.e. with the same "time stamp" as for x) to all inputs related to it.

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

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

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \begin{cases} n & \text{if no } c[n'] \text{ is part of } F \\ \max\{0, n-n'\} & \text{if } c[n'] \text{ is part of } F \end{cases} . \tag{4.6}$$

$\underline{x \in P;\ \mathrm{type}(x) = \mathrm{OR}}$

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \min_{y \in \mathrm{fi}(x)} \left\{ \widetilde{d}_{\mathrm{b}}(F,y) \right\} . \tag{4.7}$$

$\underline{x \in P;\ \mathrm{type}(x) = \mathrm{AND}}$

$$\widetilde{d}_{\mathrm{b}}(F,x) \equiv \sum_{y \in A(x)} \widetilde{d}_{\mathrm{b}}(F,y) + \max\left\{ \sum_{y \in B(x)} \widetilde{d}_{\mathrm{b}}(F,y) \,,\, \max_{y \in C(x)} \left\{ 0, \widetilde{d}_{\mathrm{b}}(F,y) \right\} \right\} , \tag{4.8}$$

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

Expressions (4.6), (4.7) and (4.8) allow to compute $\widetilde{d}_{\mathrm{b}}(F, g_r)$ by traversing the fault tree depth-first left-most starting at g_r. We depict in Figure 4.3 the fault tree of the example system and show in Table 4.2 how $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$ would be computed for that fault tree. Note that all gates and inputs of the fault tree are modules and, therefore, (4.8) reduces to $\widetilde{d}_{\mathrm{b}}(F,x) = \sum_{y \in \mathrm{fi}(x)} \widetilde{d}_{\mathrm{b}}(F,y)$.

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

First, we prove that given a bag of component classes F and $x \in I \cup P$, $0 \leq \widetilde{d}_{\mathrm{b}}(F,x) \leq d_{\mathrm{b}}(F,x)$ and $\widetilde{d}_{\mathrm{b}}(F,x) = 0$ if and only if $d_{\mathrm{b}}(F,x) = 0$. The proof consists of a sequence of a lemma, two propositions and a theorem.

Lemma 4.2 *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 $y \in \mathrm{Reach}(x)$.*

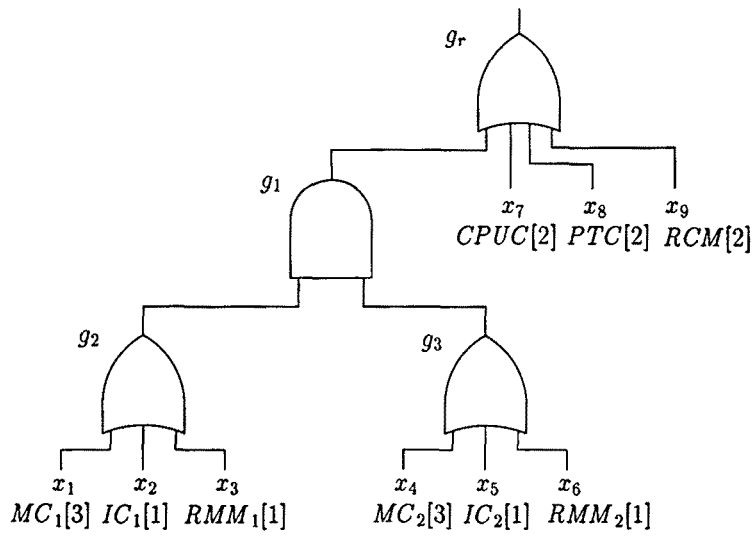

Figure 4.3: Fault tree of the example system. Bags associated with inputs are given below to them.

Table 4.2: Computation of $\widetilde{L} = \widetilde{d}_{\mathrm{b}}(\emptyset, g_r)$ traversing depth-first left-most the fault tree of the example system and using expressions (4.6), (4.7) and (4.8).

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	

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

Proposition 4.3 *Let $x, y \in I \cup P$, $z \in P$, $x, y \in \text{fi}(z)$, 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 $|\text{fo}(x)| = 1$.

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

Condition a: If $x \in I$, $\text{Reach}(x) \cap \text{Reach}(y) = \emptyset$ by Lemma 4.2. If $x \in P$, $\text{Reach}(x) \cap \text{Reach}(y) = \text{Reach}(x) \cap \{y\} \neq \emptyset$ if and only if $y \in \text{Reach}(x)$ by Lemma 4.2 ($x \in \text{Reach}(y)$ is not possible). But $y \in \text{fi}(z)$ implies the existence of the path zy not containing x, and, then, $y \in \text{Reach}(x)$ would contradict the fact that x is a module.

Condition b: Assume that $\text{Reach}(x) \cap \text{Reach}(y) \neq \emptyset$. Using Lemma 4.2, either $x \in \text{Reach}(y)$ or $y \in \text{Reach}(x)$. $x \in \text{Reach}(y)$ and the existence of the path zx contradicts the assumption that y is a module. If $x \in I$, $y \in \text{Reach}(x)$ is not possible. If $x \in P$, $y \in \text{Reach}(x)$ and the existence of the path zy contradicts the assumption that x is a module.

Condition c: Assume as before that $\text{Reach}(x) \cap \text{Reach}(y) \neq \emptyset$. From Lemma 4.2, either $y \in \text{Reach}(x)$ or $x \in \text{Reach}(y)$. If $x \in I$, $y \in \text{Reach}(x)$ is not possible. If $x \in P$, $y \in \text{Reach}(x)$ and the existence of the path zy contradicts the assumption that x is

a module. $x \in \text{Reach}(y)$ implies $|\text{fo}(x)| > 1$ since $x \in \text{fi}(z)$, in contradiction with $|\text{fo}(x)| = 1$.□

Proposition 4.4 *Let $x \in P$, $\text{type}(x) = \text{AND}$. Assume that $\text{fi}(x)$ does not contain related inputs. Let the partition $\text{fi}(x) = A(x) \cup B(x) \cup C(x)$, where $A(x) \equiv \{y \in \text{fi}(x) | y$ is a module $\wedge\ |\text{fo}(y)| = 1\}$, $B(x) \equiv \{y \in \text{fi}(x) | y$ is a module $\wedge\ |\text{fo}(y)| > 1 \vee y$ is not a module $\wedge\ y \in I\}$ and $C(x) \equiv \{y \in \text{fi}(x) | y$ 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 We begin with part a. Consider $y, y' \in A(x)$. y is a module. If $y' \in I$, condition a of Proposition 4.3 is satisfied. If $y' \in P$, since y' is also a module, condition b of Proposition 4.3 is satisfied. To show part b, consider first $y \in A(x)$, $y' \in B(x)$. y is a module and y' is an input or a gate. If $y' \in I$, condition a of Proposition 4.3 is satisfied; if $y' \in P$, y' is a module and condition b of Proposition 4.3 is satisfied. Now we deal with the case $y \in A(x)$, $y' \in C(x)$. y is a module, $|\text{fo}(y)| = 1$ and $y' \in P$. Therefore, condition c of Proposition 4.3 is fulfilled. Regarding part c, let $y, y' \in B(x)$. We have to consider four cases: 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, by assumption, y and y' are not related. In case 2, y' must be a module and y, y' satisfy condition a of Proposition 4.3. Case 3 is symmetric to case 2. Finally, in case 4 both y and y' are modules and condition b of Proposition 4.3 is satisfied. □

Theorem 4.2 *Let F be a bag of component classes and $x \in I \cup P$. Assume that for every $z \in P$, $\text{fi}(z)$ does not contain related inputs. Then, the $\tilde{d}_b(F, x)$ defined recursively by (4.6), (4.7) and (4.8) verify*

a) $0 \leq \tilde{d}_b(F, x) \leq d_b(F, x)$,

b) $\tilde{d}_b(F, x) = 0$ if and only if $d_b(F, x) = 0$.

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

Base case ($\mathrm{lev}(x) = 0$): In this case, $x \in I$, $\mathrm{b}(x) = c[n]$. From (4.6) and the definition of $d_{\mathrm{b}}(F,x)$, $0 \leq \widetilde{d}_{\mathrm{b}}(F,x) = d_{\mathrm{b}}(F,x)$, showing both a and b.

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

We begin with part a. 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 for $y \in \mathrm{fi}(x)$ since $\mathrm{lev}(y) \leq l$, the monotonicity of $\min(\cdot)$, and (4.7),

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

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 4.4 and let $t = \bigwedge_{y \in B(x)} y$ and $u = \bigwedge_{y \in C(x)} y$ (if the subsets $B(x)$ or $C(x)$ are empty, the corresponding logical variable is equal to the logical constant 1 and $\widetilde{d}_{\mathrm{b}}(F,1) = 0$). Using the fact that x is realized if and only if all $y \in \mathrm{fi}(x)$ are realized, the definition of $d_{\mathrm{b}}(F,x)$ and parts a and b of Proposition 4.4,

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

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 4.4, the induction hypothesis, the monotonicity of $\max(\cdot)$, and (4.8),

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

where $\max_{y \in C(x)} \{0, \widetilde{d}_{\mathrm{b}}(F,y)\}$ allows to deal correctly with the case $C(x) = \emptyset$. Regarding part b of the theorem, the if implication follows from $0 \leq \widetilde{d}_{\mathrm{b}}(F,x) \leq d_{\mathrm{b}}(F,x)$. The only if implication is proved as follows. If $\mathrm{type}(x) = \mathrm{OR}$, $\widetilde{d}_{\mathrm{b}}(F,x) = 0$ implies (4.7) the existence of a $y \in \mathrm{fi}(x)$ with $\widetilde{d}_{\mathrm{b}}(F,y) = 0$. From the induction hypothesis, this implies $d_{\mathrm{b}}(F,y) = 0$, which leads to $d_{\mathrm{b}}(F,x) = 0$ 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 (4.8), $\widetilde{d}_{\mathrm{b}}(F,x) = 0$ requires $\widetilde{d}_{\mathrm{b}}(F,y) = 0$ for all $y \in \mathrm{fi}(x)$. As before, the induction hypothesis implies $d_{\mathrm{b}}(F,y) = 0$ for all $y \in \mathrm{fi}(x)$, and, hence,

$d_b(F,x) = 0$ by the definition of $d_b(F,x)$ and the fact that x is realized if and only if all $y \in \mathrm{fi}(x)$ are realized. □

Next, we give a sufficient condition for $\tilde{d}_b(F,x) = d_b(F,x)$.

Theorem 4.3 *Let $x \in I \cup P$ and F be a bag of component classes. Then, $\tilde{d}_b(F,x) = d_b(F,x)$ if for every $z \in P$ with $\mathrm{type}(z) = \mathrm{AND}$, $\mathrm{fi}(z)$ does not contain related inputs and one of the following 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 $d_b(F,x)$ and (4.6), $\tilde{d}_b(F,x) = d_b(F,x)$.

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

We begin by analyzing the case $\mathrm{type}(x) = \mathrm{OR}$. Using the definition of $d_b(F,x)$, the fact that x is realized if and only if some $y \in \mathrm{fi}(x)$ is realized, the induction hypothesis, and (4.7),

$$d_b(F,x) = \min_{y \in \mathrm{fi}(x)} \left\{ d_b(F,y) \right\} = \min_{y \in \mathrm{fi}(x)} \left\{ \tilde{d}_b(F,y) \right\} = \tilde{d}_b(F,x) .$$

Now we deal with the case $\mathrm{type}(x) = \mathrm{AND}$. Let the partition $\mathrm{fi}(x) = A(x) \cup B(x) \cup C(x)$ defined in Proposition 4.4. 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 $d_b(F,x)$, the fact that $C(x) = \emptyset$, and that, according to Proposition 4.4, all $y \in \mathrm{fi}(x)$ are mutually independent,

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

Using the induction hypothesis, the fact that $C(x) = \emptyset$ and (4.8),

$$\begin{aligned} d_b(F,x) &= \sum_{y \in A(x) \cup B(x)} \tilde{d}_b(F,x) \\ &= \sum_{y \in A(x)} \tilde{d}_b(F,y) + \max\Big\{ \sum_{y \in B(x)} \tilde{d}_b(F,y), \max_{y \in C(x)} \{0, \tilde{d}_b(F,y)\} \Big\} \\ &= \tilde{d}_b(F,x) . \end{aligned}$$

Assume now that condition b of the theorem holds. We have $B(x) = \emptyset$ 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 4.4, all $y \in \mathrm{fi}(x)$ are mutually independent,

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

Finally, using the induction hypothesis, the fact that $B(x) = \emptyset$ and $C(x) = \{u\}$, and (4.8):

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

Finally, we give a lower and an upper bound for $\tilde{d}_{\mathrm{b}}(F, x)$, $F = F' + F''$ in terms of $\tilde{d}_{\mathrm{b}}(F', x)$.

Theorem 4.4 *Let F, F' and F'' be bags of component classes with $F = F' + F''$ and let $x \in I \cup P$. Assume that for every $z \in P$, $\mathrm{fi}(z)$ does not contain related inputs. Then,*

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

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

Base case ($\mathrm{lev}(x) = 0$): Since $\mathrm{lev}(x) = 0$, $x \in I$, $\mathrm{b}(x) = c[n]$. The following three cases cover all possibilities: a) $\tilde{d}_{\mathrm{b}}(F', x) = 0$, b) $\tilde{d}_{\mathrm{b}}(F', x) > 0$ and no $c[n']$ is part of F'', and c) $\tilde{d}_{\mathrm{b}}(F', x) > 0$ and there exists $c[n']$ part of F''. In case a, there exists (4.6) $c[n']$ part of F' with $n' \geq n$. Since $F = F' + F''$, $c[n'']$, $n'' \geq n'$ part of F exists. Then, $\tilde{d}_{\mathrm{b}}(F, x) = 0 = \tilde{d}_{\mathrm{b}}(F', x)$ showing both inequalities since $S(F'', x) \geq 0$. In case b, clearly $\tilde{d}_{\mathrm{b}}(F, x) = \tilde{d}_{\mathrm{b}}(F', x)$ and both inequalities are also shown. In case c, let $\tilde{d}_{\mathrm{b}}(F', x) = n''$, $0 < n'' \leq n$. Since $\mathrm{Support}(x) = \{x\}$, $\mathrm{b}(x) = c[n]$, it follows that $S(F'', x) = n'$. From (4.6), $c[n''']$, $n''' = n - n''$ is part of F', and since $F = F' + F''$, $c[n''' + n']$ is part of F. Then, using (4.6)

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

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

First of all, since $\mathrm{Support}(x) = \bigcup_{y \in \mathrm{fi}(x)} \mathrm{Support}(y)$, it is immediate to see that for any $x \in P$,

$$S(F'', x) \geq S(F'', y),\ y \in \mathrm{fi}(x)\,, \tag{4.9}$$

$$S(F'', x) \geq \max_{y \in \mathrm{fi}(x)} \{S(F'', y)\}\,. \tag{4.10}$$

Consider first the case $\mathrm{type}(x) = \mathrm{OR}$. Using (4.7), the induction hypothesis, the monotonicity of $\min(\cdot)$, and (4.9),

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

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 4.4 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 $\delta = \max_{y \in C(x)} \{S(F'', y)\}$. We have $\mathrm{Support}(x) = \mathrm{Support}(t) \cup \mathrm{Support}(u \wedge v)$ and, from part b of Proposition 4.4, $\mathrm{Support}(t) \cap \mathrm{Support}(u \wedge v) = \emptyset$. Then, using the definition of $S(F'', \cdot)$, (4.9) and (4.10),

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

From parts a and c of Proposition 4.4, $S(F'', t) = \beta$ and $S(F'', u) = \gamma$. Then, using the definition of α the last inequality becomes

$$\alpha \geq \beta + \max\{\gamma, \delta\}\,. \tag{4.11}$$

Using (4.8), the induction hypothesis, the definition of α, β, γ and δ, the monotonicity of $\max(\cdot)$, and (4.11),

$$\tilde{d}_{\mathrm{b}}(F', x) =$$

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

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

Let a be a state and let $x = g_r$. With $F = F(a)$, part b of Theorem 4.2 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 $F' = \emptyset$, $F'' = F(a)$ and $F = F' + F'' = F(a)$, the left inequality of Theorem 4.4 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 $\widetilde{d}(a)$ satisfy the requirements of Theorem 4.1. Finally, taking $F = F(a)$, Theorem 4.3 gives a sufficient condition for $\widetilde{d}(a) = d(a)$.

4.2.3 Algorithms for the Computation of the Lower Bounds for Failure Distances

Let a be a state in G and let b be a successor of a reached in a single transition associated with failure bag e. The generation of the CTMC X' requires to know $\widetilde{d}(b)$ (recall that,

as consequence of Theorem 4.2, b is a down state if and only if $\widetilde{d}(b) = 0$).

We have that $F(b) = F(a)+e$ and, therefore, $\widetilde{d}(b) = \widetilde{d}_{\mathrm{b}}\big(F(b), g_r\big)$ could be computed traversing the fault tree depth-first left-most starting at g_r and using (4.6), (4.7) and (4.8). However, this procedure could be expensive if the fault tree is large. Next, we describe an algorithm and, based on it, two procedures to compute $\widetilde{d}(b)$ more efficiently.

We first describe the algorithm. The algorithm takes as inputs a bag of component classes F and a positive integer ub. Each node x of the fault tree holds a "distance variable" $dv(x)$ properly initialized. The fault tree is processed from inputs to g_r as follows. For each $c[n]$ that is part of F, we make $dv(x) = \max\{0, dv(x) - n\}$ for each input x, $\mathrm{b}(x) = c[n']$. Each change of $dv(x)$ for an input x that results in $dv(x) < ub$ is propagated up the fault tree while $dv(z)$ changes to a value $< ub$ for the visited gate z. Distance variables of gates are not updated unless the new value is distinct from the previous one and $< ub$. $dv(z)$, $z \in P$ is computed from (4.7) for OR gates and (4.8) for AND gates using $dv(y)$, $y \in \mathrm{fi}(z)$ instead of $\widetilde{d}_{\mathrm{b}}(F, y)$, $y \in \mathrm{fi}(z)$.

The first procedure is called *comp_d*(F, lb, ub, CS), where F is a bag of component classes, lb, ub, $lb \leq ub$ are non-negative integers and CS is a stack. If $lb = ub$, the procedure returns lb without doing anything else. If $lb < ub$, the fault tree is processed using the above algorithm. During the traversal of the fault tree, the nodes x whose distance variable changes as well as the corresponding old value $dv(x)$ are bookkept in CS. At the end, the procedure returns $\min\{dv(g_r), ub\}$. The second procedure is called *restore_d*(CS), where CS is a stack. The procedure simply restores the distance variable of the nodes kept in CS to its old value.

We prove next two results. The first one shows that if invoked with appropriate arguments, the procedure *comp_d* returns $\widetilde{d}_{\mathrm{b}}(F, g_r)$. The second result shows how $\widetilde{d}_{\mathrm{b}}(F, g_r)$ can be computed calling the procedure consecutively twice. The usefulness of the latter result will become apparent when describing the way the CTMC X' is generated. The proof consists of a sequence of three propositions and two theorems.

Proposition 4.5 *Assume that the distance variable of each node x has been initialized to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$. After running the algorithm with inputs a bag of component classes F and a positive integer ub, $\widetilde{d}_{\mathrm{b}}(F, x) = dv(x)$ if $dv(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$ if $dv(x) \geq ub$, $x \in I \cup P$.*

Proof By complete induction on $\mathrm{lev}(x)$, $x \in I \cup P$.

Base case ($\mathrm{lev}(x) = 0$). We have $x \in I$, $\mathrm{b}(x) = c[n]$. After running the algorithm, $dv(x) = n$ if no $c[n']$ is part of F and $dv(x) = \max\{0, n - n'\}$ if $c[n']$ is part of F. Then (4.6), $dv(x) = \widetilde{d}_{\mathrm{b}}(F, x)$, showing both $\widetilde{d}_{\mathrm{b}}(F, x) = dv(x)$ if $dv(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$ if $dv(x) \geq ub$.

Induction step. We will assume that the result holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$ and show that it also holds for all $x \in P$, $\mathrm{lev}(x) = l + 1$ ($x \notin I$ because $l + 1 > 0$). Let $x \in P$, $\mathrm{lev}(x) = l + 1$ and consider the partition $\mathrm{fi}(x) = \alpha \cup \beta$ with $\alpha = \{y \in \mathrm{fi}(x) \mid dv(y) < ub$ after running the algorithm$\}$ and $\beta = \mathrm{fi}(x) - \alpha = \{y \in \mathrm{fi}(x) \mid dv(y) \geq ub$ after running the algorithm$\}$. We will show the result first for $\mathrm{type}(x) = \mathrm{OR}$ and next for $\mathrm{type}(x) = \mathrm{AND}$.

Let $\mathrm{type}(x) = \mathrm{OR}$. Assume $\alpha \neq \emptyset$. Since, trivially, the values held by the distance variables cannot increase, $dv(y) \geq ub$, $y \in \beta$ throughout the execution of the algorithm and, hence, any update of $dv(y)$, $y \in \beta$ (only possible if y is an input) cannot lead to updating $dv(x)$. Therefore, only updates of $dv(y)$, $y \in \alpha$ to values $< ub$ may result in updating $dv(x)$. Note that, using the induction hypothesis, at the end of the algorithm, $\widetilde{d}_{\mathrm{b}}(F, y) = dv(y) < ub$, $y \in \alpha$ and $\widetilde{d}_{\mathrm{b}}(F, y) \geq ub$, $y \in \beta$. Then (4.7), $\widetilde{d}_{\mathrm{b}}(F, x) = \min_{y \in \alpha}\{dv(x)\} < ub$ after running the algorithm. If no $dv(y)$, $y \in \alpha$ has been updated, since $dv(x) = \min_{y \in \mathrm{fi}(x)}\{dv(y)\}$ at the beginning of the algorithm, we have that, at the end of the algorithm, $dv(x) = \min_{y \in \alpha}\{dv(y)\} = \widetilde{d}_{\mathrm{b}}(F, x) < ub$. If some $dv(y)$, $y \in \alpha$ has been updated, consider the last time a $dv(y)$, $y \in \alpha$ is updated to a value $< ub$ (since at the end of the algorithm $dv(y) < ub$ for all $y \in \alpha$, at least one such update must have happened). All $dv(y)$, $y \in \alpha$ will hold their final values, which are $< ub$, and $dv(x)$ will be set to $dv' = \min_{y \in \alpha}\{dv(y)\}$ unless $dv(x)$ were already equal to dv'. Then, at the end of the algorithm, $dv(x) = \min_{y \in \alpha}\{dv(y)\} = \widetilde{d}_{\mathrm{b}}(F, x) < ub$. Assume now $\alpha = \emptyset$ and, therefore, $\beta \neq \emptyset$. As it has been discussed before, any update of $dv(y)$, $y \in \beta$ cannot have led to updating $dv(x)$. Therefore, $dv(x)$ has not been updated and after running the algorithm its value is equal to its initial value, which, given the way the distance variables are initialized and that the values they hold cannot increase, is $\geq ub$. Since $\mathrm{fi}(x) = \beta$ and, by the induction hypothesis, $\widetilde{d}_{\mathrm{b}}(F, y) \geq ub$, $y \in \beta$, it follows (4.7) that $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$.

Let $\mathrm{type}(x) = \mathrm{AND}$. Assume $\beta \neq \emptyset$. As it has been discussed for the case $\mathrm{type}(x) = \mathrm{OR}$, $dv(y) \geq ub$, $y \in \beta$ throughout the execution of the algorithm. The computed values for $dv(x)$ will be

$$\sum_{y \in A(x)} dv(y) + \max\Big\{ \sum_{y \in B(x)} dv(y), \max_{y \in C(x)}\{0, dv(y)\} \Big\},$$

which are not smaller than $dv(y)$, $y \in \beta$. Then, the computed values for $dv(x)$ will be $\geq ub$ and $dv(x)$ will never be updated. Given the way the distance variables are initialized and that the values they hold cannot increase, the final value of $dv(x)$ will be $\geq ub$. Moreover, from (4.8) and the fact that, by the induction hypothesis, $\widetilde{d}_{\mathrm{b}}(F, y) \geq ub$, $y \in \beta$, it follows that $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$. Assume now $\beta = \emptyset$. We have $\mathrm{fi}(x) = \alpha$. If no $dv(y)$, $y \in \alpha$ has been updated, after running the algorithm $dv(x)$ and $dv(y)$, $y \in \alpha$ hold their initial values. By the induction hypothesis and the fact that $\mathrm{fi}(x) = \alpha$, $dv(y) = \widetilde{d}_{\mathrm{b}}(F, y)$, $y \in \mathrm{fi}(x)$. Then, from (4.8) and the way the distance variables have been initialized, $dv(x) = \widetilde{d}_{\mathrm{b}}(F, x)$ after running the algorithm, which shows both $\widetilde{d}_{\mathrm{b}}(F, x) = dv(x)$ if $dv(x) < ub$ and $\widetilde{d}_{\mathrm{b}}(F, x) \geq ub$ if $dv(x) \geq ub$. If some $dv(y)$, $y \in \alpha$ has been updated, consider the last time a $dv(y)$, $y \in \alpha$ is updated to a value $< ub$ (since at the end of the algorithm $dv(y) < ub$ for all $y \in \alpha$, at least one such update must have happened). Since $\mathrm{fi}(x) = \alpha$, all $dv(y)$, $y \in \mathrm{fi}(x)$ will hold their final values and the value dv' to which $dv(x)$ might be set will be computed as

$$dv' = \sum_{y \in A(x)} dv(y) + \max\{ \sum_{y \in B(x)} dv(y), \max_{y \in C(x)} \{0, dv(y)\} \}.$$

From (4.8), the fact that $\mathrm{fi}(x) = \alpha$ and that, by the induction hypothesis, $dv(y) = \widetilde{d}_{\mathrm{b}}(F, y)$, $y \in \alpha$, it follows that $\widetilde{d}_{\mathrm{b}}(F, x) = dv'$. Also, since the values held by the distance variables cannot increase, the current $dv(x)$ will be $\geq dv'$. If $dv' \geq ub$, $dv(x)$ will not be set to dv' and, hence, $\widetilde{d}_{\mathrm{b}}(F, x) = dv' \geq ub$ and $dv(x) \geq dv' \geq ub$. If $dv' < ub$, $dv(x)$ will be set to dv' unless it were already equal to dv' and, thus, $\widetilde{d}_{\mathrm{b}}(F, x) = dv' = dv(x) < ub$. □

Proposition 4.6 *Let F, F' and F'' be bags of component classes with $F = F' + F''$ and let ub, ub' with $ub' \leq ub$ be positive integers. Let $dv_1(x)$, $x \in I \cup P$ be the values of the distance variables that will result after initializing them to $\widetilde{d}_{\mathrm{b}}(\emptyset, x)$ and next running the algorithm with inputs F and ub', and let $dv_2(x)$, $x \in I \cup P$ be the corresponding values that will result if after performing the same initialization the algorithm is run consecutively twice, first with inputs F' and ub and next with inputs F'' and ub'. Then, $dv_1(x) = dv_2(x)$ or $dv_1(x) \geq ub'$ and $dv_2(x) \geq ub'$, $x \in I \cup P$*

Proof For the sake of conciseness, let case 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 algorithm has been run with inputs F and ub'" and let case 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 algorithm has been run consecutively twice, first with inputs F' and ub and next with inputs F'' and ub'". Note that if $ub' = ub$ the

fault tree is dealt with the same in both cases and $dv_1(x) = dv_2(x)$, $x \in I \cup P$. Next, we show the result for the case $ub' < ub$ using complete induction on $\mathrm{lev}(x)$, $x \in I \cup P$.

Base case ($\mathrm{lev}(x) = 0$). We have $x \in I$. Since the initial value of $dv(x)$ is the same in both cases, $dv(x)$ is updated depending only on $\mathrm{b}(x)$ and the bag of failed component classes, and $F = F' + F''$, it follows that $dv_1(x) = dv_2(x)$.

Induction step. We will assume that the result holds for all $x \in I \cup P$, $\mathrm{lev}(x) \leq l$, $l \geq 0$ and show that it 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 $\alpha = \{y \in \mathrm{fi}(x) \mid dv_1(y) = dv_2(y)\}$ and $\beta = \{y \in \mathrm{fi}(x) \mid dv_1(y) \neq dv_2(y)\}$. Note that, using the induction hypothesis, $dv_1(y) \geq ub'$ and $dv_2(y) \geq ub'$, $y \in \beta$ and since the values held by the distance variables cannot increase, $dv(y)$, $y \in \beta$ will always be $\geq ub'$ in both cases. We will show the result first for $\mathrm{type}(x) = \mathrm{OR}$ and next for $\mathrm{type}(x) = \mathrm{AND}$.

Let $\mathrm{type}(x) = \mathrm{OR}$. Assume $\alpha = \emptyset$. We have $\mathrm{fi}(x) = \beta$. Since $dv(y) \geq ub'$, $y \in \mathrm{fi}(x)$, $dv(x)$ has not been updated in case 1 and, thereby, it holds the initial value, which, since $dv(y)$, $y \in \mathrm{fi}(x)$ cannot increase, is $\geq ub'$. Therefore, we have $dv_1(x) \geq ub'$. Regarding case 2, either $dv(x)$ has not been updated and, hence, $dv_2(x) = dv_1(x)$, or it has been updated (recall that $ub' < ub$) and, since $dv(y) \geq ub'$, $y \in \mathrm{fi}(x)$, $dv_2(x) \geq ub'$. Assume now $\alpha \neq \emptyset$. If there exist $y \in \alpha$ such that $dv_1(y) = dv_2(y) < ub'$, since $ub' < ub$, we have

$$dv_1(x) = \min_{\substack{y \in \alpha \\ dv_1(y) < ub'}} \{dv_1(y)\} = \min_{\substack{y \in \alpha \\ dv_2(y) < ub'}} \{dv_2(y)\} = dv_2(x)\,.$$

If no such $y \in \alpha$ exists, $dv_1(x)$ holds the initial value, which, since $dv(y)$, $y \in \mathrm{fi}(x)$ cannot increase, is $\geq ub'$, and either $dv_2(x)$ also holds the initial value and, hence, $dv_2(x) = dv_1(x)$, or $dv(x)$ has been updated in case 2 and, since $dv(y) \geq ub'$, $y \in \mathrm{fi}(x)$, $dv_2(x) \geq ub'$.

Let $\mathrm{type}(x) = \mathrm{AND}$. If $\beta \neq \emptyset$, in both cases the computed values for $dv(x)$ will be

$$\sum_{y \in A(x)} dv(y) + \max\{\sum_{y \in B(x)} dv(y), \max_{y \in C(x)} \{0, dv(y)\}\} \geq ub'\,.$$

Then, $dv(x)$ will not be updated in case 1. Therefore, $dv_1(x)$ will hold the initial value, which, since $dv(y)$, $y \in \mathrm{fi}(x)$ cannot increase, is $\geq ub'$. If $dv(x)$ is not updated in case 2, $dv_1(x) = dv_2(x)$; if $dv(x)$ is updated in case 2, since $dv(y) \geq ub'$, $y \in \beta$, $dv_2(x) \geq ub'$. Assume now $\beta = \emptyset$ and, therefore, $\mathrm{fi}(x) = \alpha$. If $dv(x)$ is updated in case 1,

$$\begin{aligned} dv_1(x) &= \sum_{y \in A(x)} dv_1(y) + \max\{\sum_{y \in B(x)} dv_1(y), \max_{y \in C(x)} \{0, dv_1(y)\}\} \\ &= \sum_{y \in A(x)} dv_2(y) + \max\{\sum_{y \in B(x)} dv_2(y), \max_{y \in C(x)} \{0, dv_2(y)\}\} < ub'\,. \end{aligned}$$

Therefore, since $ub' < ub$, $dv(x)$ is also updated in case 2 and $dv_2(x) = dv_1(x)$. If $dv(x)$ is not updated in case 1, it may or may not be updated in case 2. If $dv(x)$ is updated in case 2,

$$\begin{aligned} dv_2(x) &= \sum_{y \in A(x)} dv_2(y) + \max\{ \sum_{y \in B(x)} dv_2(y), \max_{y \in C(x)} \{0, dv_2(y)\}\} \\ &= \sum_{y \in A(x)} dv_1(y) + \max\{ \sum_{y \in B(x)} dv_1(y), \max_{y \in C(x)} \{0, dv_1(y)\}\}, \end{aligned}$$

with either $ub' \leq dv_2(x) < ub$ or $dv_2(x) < ub'$. Note, however, that if $dv_2(x)$ were $< ub'$, $dv(x)$ would have been updated to $dv_2(x)$ by the end of the algorithm in case 1. Therefore, $dv_2(x) \geq ub'$ and since the values held by the distance variables cannot increase and they are initialized the same in both cases, $dv_1(x) > dv_2(x) \geq ub'$. Finally, if $dv(x)$ is not updated in either case, $dv_1(x) = dv_2(x)$. □

Proposition 4.7 *Let F, F' and F'' be bags of component classes with $F = F' + F''$ and let ub, ub' with $ub' \leq ub$ be positive integers. Assume that the distance variable of each node x of the fault tree has been initialized to $\widetilde{d}_b(\emptyset, x)$. After running the algorithm consecutively twice, first with inputs F' and ub and next with inputs F'' and ub', we have that $\widetilde{d}_b(F, x) = dv(x)$ if $dv(x) < ub'$ and $\widetilde{d}_b(F, x) \geq ub'$ if $dv(x) \geq ub'$.*

Proof Let $x \in I \cup P$ and let $dv_1(x)$ and $dv_2(x)$ as in Proposition 4.6. In this regard, we have to show that $\widetilde{d}_b(F, x) = dv_2(x)$ if $dv_2(x) < ub'$ and $\widetilde{d}_b(F, x) \geq ub'$ if $dv_2(x) \geq ub'$. Using Proposition 4.6, $dv_2(x) = dv_1(x)$ or $dv_2(x) \geq ub'$ and $dv_1(x) \geq ub'$, and the results follows immediately from Proposition 4.5. □

Theorem 4.5 *Let F be a bag of component classes, lb and ub non-negative integers with $lb \leq \widetilde{d}_b(F, g_r) \leq ub$, and CS a stack. Assume that the distance variable of each node x of the fault tree has been initialized to $\widetilde{d}_b(\emptyset, x)$. Then, the call comp_d($F, lb, ub, CS$) returns $\widetilde{d}_b(F, g_r)$.*

Proof If $lb = ub$, the call *comp_d*(F, lb, ub, CS) returns lb and since, by assumption, $lb \leq \widetilde{d}_b(F, g_r) \leq ub$, $\widetilde{d}_b(F, g_r) = lb$. If $lb < ub$, the call returns $\min\{dv(g_r), ub\}$.. Two cases are possible: a) $dv(g_r) < ub$ and b) $dv(g_r) \geq ub$. Since $lb < ub$ and, by assumption, $lb \geq 0$, we have that ub is > 0 and, therefore, Proposition 4.5 with $x = g_r$ can be invoked in both cases. In case a, $\min\{dv(g_r), ub\} = dv(g_r) = \widetilde{d}_b(F, g_r)$. In case b, $\min\{dv(g_r), ub\} = ub$ and $\widetilde{d}_b(F, g_r) \geq ub$, and since, by assumption, $\widetilde{d}_b(F, g_r) \leq ub$, $\widetilde{d}_b(F, g_r) = ub$. □

Theorem 4.6 *Let F, F' and F'' be bags of component classes with $F = F' + F''$, lb and ub' non-negative integers and ub a positive integer with $lb \leq \widetilde{d}_b(F, g_r) \leq ub' \leq ub$, and let CS, CS' be stacks. Assume that the dv variable of each node x of the fault tree has been initialized to $\widetilde{d}_b(\emptyset, x)$. Then, after invoking $comp\_d(F', 0, ub, CS)$, the call $comp\_d(F'', lb, ub', CS')$ returns $\widetilde{d}_b(F, g_r)$.*

Proof If $lb = ub'$, the call $comp\_d(F'', lb, ub', CS')$ returns lb and, since by assumption, $lb \leq \widetilde{d}_b(F, g_r) \leq ub'$, $\widetilde{d}_b(F, g_r) = lb$. If $lb < ub'$, the call returns $\min\{dv(g_r), ub'\}$. Two cases are possible: a) $dv(g_r) < ub'$ and b) $dv(g_r) \geq ub'$. We have, by assumption, that $ub > 0$ and $lb \geq 0$. The fact that $ub > 0$ implies that the call $comp\_d(F', 0, ub, CS)$ results in processing the fault tree with the described algorithm with inputs F' and ub; the fact that $lb \geq 0$ together with $lb < ub'$ implies that $ub' > 0$. Therefore, Proposition 4.7 with $x = g_r$ can be invoked in both cases. In case a, $\min\{dv(g_r), ub'\} = dv(g_r) = \widetilde{d}_b(F, g_r)$. In case b, $\min\{dv(g_r), ub'\} = ub'$ and $\widetilde{d}_b(F, g_r) \geq ub'$, and since, by assumption, $\widetilde{d}_b(F, g_r) \leq ub'$, $\widetilde{d}_b(F, g_r) = ub'$. □

Computation of $\widetilde{d}(b) = \widetilde{d}_b(F(b), g_r)$ using the procedure *comp_d* requires to know a lower and an upper bound bound for $\widetilde{d}(b)$. From what was shown at the end of Section 4.2.2, $0 \leq \widetilde{d}(b) \leq \widetilde{L}$. If we recall that b is a successor of a reached in a single transition associated with failure bag e, i.e. $F(b) = F(a) + e$, these bounds can be tightened up with the aid of Theorem 4.4 (the tighter the bounds, the faster *comp_d*). Let $x = g_r$. Taking $F' = F(a)$ and $F'' = e$, the theorem yields $\widetilde{d}(a) \geq \widetilde{d}(b) \geq \widetilde{d}(a) - S(e, g_r)$; taking $F' = e$ and $F'' = F(a)$, $\widetilde{\eta}(e) \geq \widetilde{d}(b) \geq \widetilde{\eta}(e) - S(F(a), g_r)$. Then, since $S(e, g_r) \leq |e|$, $S(F(a), g_r) \leq |F(a)|$ and $\widetilde{d}(a) \leq \widetilde{L}$, we have $\max\{0, \widetilde{d}(a) - |e|, \widetilde{\eta}(e) - |F(a)|\} \leq \widetilde{d}(b) \leq \min\{\widetilde{d}(a), \widetilde{\eta}(e)\}$.

The $\widetilde{L}$ and $\widetilde{\eta}(e)$, $e \in E$ are computed prior to the generation of the CTMC X'. The distance variable of each node x of the fault tree is initialized to $\widetilde{d}_b(\emptyset, x)$ by traversing the fault tree depth-first left-most starting at g_r and using (4.6), (4.7) and (4.8). After the initialization, $\widetilde{L} = dv(g_r)$. Computation of $\widetilde{\eta}(e) = \widetilde{d}_b(e, g_r)$, $e \in E$ using *comp_d* requires to know a lower and an upper bound for $\widetilde{\eta}(e)$. From part a of Theorem 4.2 with $x = g_r$ and $F = e$, $0 \leq \widetilde{d}_b(e, g_r) = \widetilde{\eta}(e)$; from the left inequality of Theorem 4.4 with $x = g_r$, $F' = \emptyset$ and $F'' = e$, $\widetilde{L} = \widetilde{d}_b(\emptyset, g_r) \geq \widetilde{d}_b(e, g_r) = \widetilde{\eta}(e)$. Therefore, based on Theorem 4.5 with $F = e$, $lb = 0$ and $ub = \widetilde{L}$, $\widetilde{\eta}(e)$, $e \in E$ are computed calling, for each e, $comp\_d(e, 0, \widetilde{L}, CS)$ followed by a call to $restore\_d(CS)$. Note that after computing all $\widetilde{\eta}(e)$, $e \in E$, the distance variable of each node x still holds its initial value $\widetilde{d}_b(\emptyset, x)$.

We sketch next how the CTMC X' is generated breadth-first using $\widetilde{\eta}(e)$, $e \in E$ and

the procedures *comp_d* and *restore_d*. The state o in which all components are unfailed is put in G and, with $\widetilde{d}(o) = \widetilde{L}$, in an empty first-in first-out (FIFO) queue. That queue contains the states to be processed and each state a in the queue has associated with it its lower bound for the failure distance $\widetilde{d}(a)$. From that point, the generation process continues as follows. The first state a in the queue is pulled out, $comp\_d(F(a), 0, \widetilde{d}(a), CS)$ is called and the list of failure bags which are active in a is obtained. Each such active $e \in E$ has associated with it one or more transitions to states b satisfying $F(b) = F(a)+e$. For each active e, $lb = \max\{0, \widetilde{d}(a) - |e|, \widetilde{\eta}(e) - |F(a)|\}$ and $ub' = \min\{\widetilde{d}(a), \widetilde{\eta}(e)\}$ are computed and, based on Theorem 4.6 with $F' = F(a)$, $F'' = e$, $F = F' + F'' = F(b)$, and $ub = \widetilde{d}(a)$ (it is trivial to check that $ub' = \min\{\widetilde{d}(a), \widetilde{\eta}(e)\} \leq \widetilde{d}(a) = ub$, $ub > 0$ and $ub' \geq 0$), $\widetilde{d}(b)$ for the successors b of a associated with e is computed by calling $comp\_d(e, lb, ub', CS')$. If $\widetilde{d}(b) = 0$, states b are down and the corresponding transition rates $\lambda_{a,b}$ are directed to f; if $\widetilde{d}(b) > 0$ and $|F(a) + e| > K$, states b belong to U and the corresponding transition rates $\lambda_{a,b}$ are directed to $u_{\widetilde{d}(b)}$; if $\widetilde{d}(b) > 0$ and $|F(a) + e| \leq K$, states b are put in G and, with the corresponding $\widetilde{d}(b)$, in the FIFO queue if they were not in G yet and the corresponding transition rates $\lambda_{a,b}$ are directed to b. Once all the successors b of a associated with e have been processed, $restore\_d(CS')$ is invoked to allow another failure bag active in a to be dealt with. When all active failure bags in a have been processed, $restore\_d(CS)$ is invoked before processing another state of the FIFO queue. The generation of X' finishes when the FIFO queue becomes empty.

To illustrate how the procedures *comp_d* and *restore_d* work, consider again the example system described in Section 4.1 and its fault tree depicted in Figure 4.3. Let $a \in G$ with $F(a) = MC_1[1]$ and let us discuss the processing of two successors, b and c, where transition from a to b is associated with failure bag $e_4 = IC_1[1]$ (see Table 4.1) and transition from a to c is associated with failure bag $e_7 = MC_2[1]\,RMM_2[1]$. For the example, $\widetilde{L} = 2$, $\widetilde{\eta}(e_4) = 1$, $\widetilde{\eta}(e_7) = 1$, and $\widetilde{d}(a) = 2$. Let CS and CS' be empty stacks. As it has been explained before, computation of $\widetilde{d}(b)$ and $\widetilde{d}(c)$ requires to invoke $comp\_d(F(a), 0, \widetilde{d}(a) = 2, CS)$ first. Figure 4.4 illustrates the processing of the fault tree during that invocation. Recall that $dv(x)$ is initialized to $\widetilde{d}_b(\emptyset, x)$ for all nodes x. The procedure starts by making (4.6) $dv(x_1) = \max\{0, 3 - 1\} = 2$. Since this value is not smaller than $\widetilde{d}(a) = 2$, the change is not propagated up the fault tree and the procedure finishes. The only contents of the stack CS is the pair $(x_1, 3)$. The $\widetilde{d}(b)$ is the value returned by the call $comp\_d(e_4, lb, ub, CS')$ with $lb = \max\{0, \widetilde{d}(a) - |e_4|, \widetilde{\eta}(e_4) - |F(a)|\} = \max\{0, 2 - 1, 1 - 1\} = 1$ and $ub = \min\{\widetilde{d}(a), \widetilde{\eta}(e_4)\} = \min\{2, 1\} = 1$. Since $lb = ub$, the invocation returns $lb = 1$ without traversing the fault tree. Finally, $\widetilde{d}(c)$ is the value returned by the call $comp\_d(e_7, lb, ub, CS')$ with $lb = \max\{0, \widetilde{d}(a) - |e_7|, \widetilde{\eta}(e_7) - |F(a)|\} =$

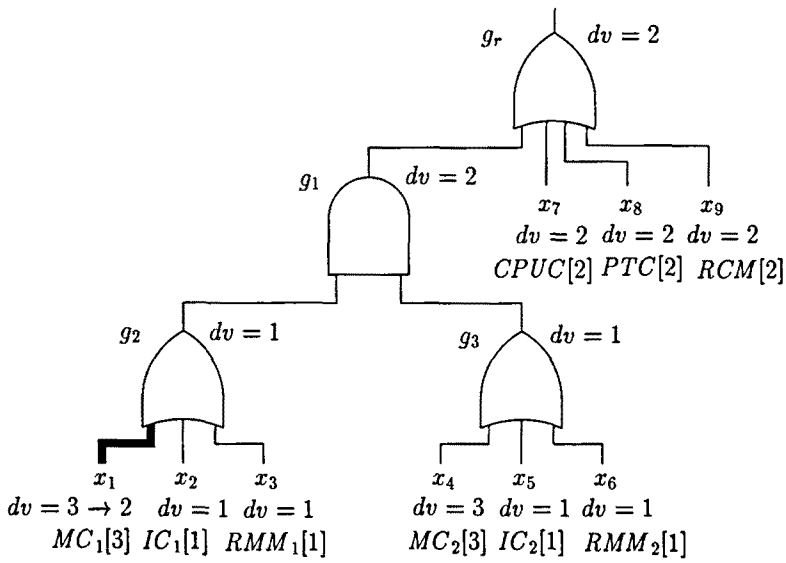

Figure 4.4: Up traversal (thick line) of the fault tree of the example system during the call $comp\_d(F(a), 0, 2, CS)$, $F(a) = MC_1[1]$. The values of the distance variables are given next to each node. A rightward arrow signals the values of the distance variables that change during the traversal.

$\max\{0, 2-2, 1-1\} = 0$ and $ub = \min\{\widetilde{d}(a), \widetilde{\eta}(e_7)\} = \min\{2, 1\} = 1$. Figure 4.5 illustrates the processing of the fault tree during the invocation $comp\_d(e_7, 0, 1, CS')$. The procedure starts with the part $MC_2[1]$ of e_7. Using (4.6) and the dv value of input x_4, the procedure makes $dv(x_4) = 2$. This change is bookkept in the stack CS' but it is not propagated up the fault tree since this new dv value is not smaller than $ub = 1$. Next, the procedure deals with the remaining part $RMM_2[1]$. Using (4.6) and the value of $dv(x_6)$, the procedure makes $dv(x_6) = 0 < 1$ and propagates the change up to g_3. Using (4.7), $dv(g_3)$ changes from 1 to 0, which is again < 1. Since this new value of $dv(g_3)$ would result (4.8) in $dv(g_1) = 1$, which is not smaller than $ub = 1$, $dv(g_1)$ is not updated and the procedure finishes returning $\min\{dv(g_r), ub\} = \min\{2, 1\} = 1$. The contents of the stack CS' are, from top to bottom, the pairs $(g_3, 1)$, $(x_6, 1)$ and $(x_4, 3)$. The fault tree is restored to its original state by calling, in this order, $restore\_d(CS')$ and $restore\_d(CS)$.

The trivial bounding method requires the knowledge of the operational/down state of the single-transition successors b of an operational state a. We describe next the procedures to determine that which have been used in our implementation of the trivial bounding method. The procedures are analogous to the procedures used in the proposed bounding method to compute lower bounds for failure distances and allow to make a fair comparison of both methods regarding CPU time consumption. The procedures are

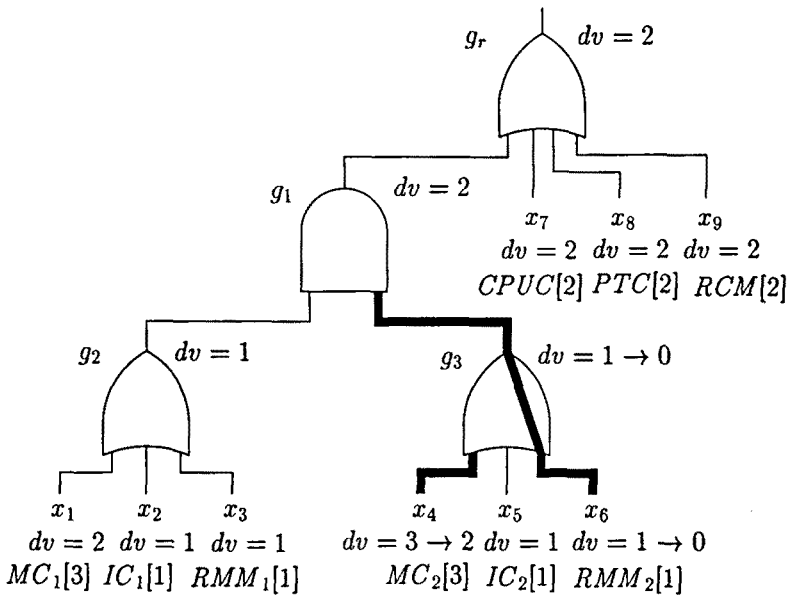

Figure 4.5: Up traversal (thick lines) of the fault tree of the example system during the call $comp\_d(e_7, 0, 1, CS')$, $e_7 = MC_2[1]\,RMM_2[1]$. The values of the distance variables are given next to each node. A rightward arrow signals the values of the distance variables that change during the traversal.

eval_ft and *restore_v*. The procedure *eval_ft* returns $\mathrm{val}(g_r)$ for a given bag of failed component classes. Using both procedures as described next, we imply to 1 the inputs x, $\mathrm{b}(x) = c[n]$ such that $c[n']$, $n' \geq n$ is part of the bag of failed component classes under consideration. Implications in the fault tree are done from inputs to g_r in the usual way: if $\mathrm{type}(x) = \mathrm{OR}$, $\mathrm{val}(y) = 1$ for some $y \in \mathrm{fi}(x)$ implies $\mathrm{val}(x) = 1$; if $\mathrm{type}(x) = \mathrm{AND}$, $\mathrm{val}(y) = 1$ for all $y \in \mathrm{fi}(x)$ implies $\mathrm{val}(x) = 1$. Since all gates are either OR or AND, that procedure is enough to know $\mathrm{val}(g_r)$ (if g_r is implied to 1, $\mathrm{val}(g_r) = 1$; if g_r is not implied, $\mathrm{val}(g_r) = 0$ since implication to 0 of the unimplied inputs of the fault tree would imply g_r to 0). Each input x, $\mathrm{b}(x) = c[n]$ of the fault tree holds a "value" variable $vv(x) = \max\{0, n - n'\}$, where n' is the number of components of class c that are failed. Initially, $vv(x) = n$ and all gates of the fault tree are set to the unimplied state. The procedure *eval_ft*(F, CS), where F is a bag of failed component classes and CS is a stack, works as follows. For each input x, $\mathrm{b}(x) = c[n]$ for which $c[n']$ is part of F, we make $vv(x) = \max\{0, vv(x) - n'\}$ and imply x to 1 if $vv(x)$ becomes 0. Each implication of an input x is propagated up the fault tree while the visited gate z becomes implied. Inputs x whose value variable changes as well as gates z that become implied are bookkept in CS. A call to *restore_v*(CS) undoes the changes done within the previous call to *eval_ft*(F, CS). Recall that in the trivial method the CTMC is modified so that exits of

X from G not going to f are directed to an absorbing state u_0. Such a modified CTMC is generated breadth-first in a similar way as X' is generated in the proposed bounding method. Starting from the point in which a state a has been pulled out of the FIFO queue and the list of failure bags that are active in a has been obtained, the generation procedure proceeds as follows. The fault tree is set up by calling *eval_ft*$(F(a), CS)$. For each active failure bag e, the operational/down state of all successors b of a associated with e is obtained by calling *eval_ft*(e, CS'). If the call results in $\mathrm{val}(g_r) = 1$, successors b are down and the corresponding transition rates $\lambda_{a,b}$ are directed to f. If $\mathrm{val}(g_r) = 0$, successors b are up. In that case, the corresponding transition rates $\lambda_{a,b}$ are directed to u_0 if $|F(b)| = |F(a)+e| > K$; otherwise, successors b are put in G and in the FIFO queue if they were not in G yet and transitions $\lambda_{a,b}$ are directed to b. Each call *eval_ft*(e, CS') is followed by a call *restore_v*(CS'), and once all successors of a have been processed, *restore_v*(CS) is invoked. The generation of the modified CTMC finishes when the FIFO queue becomes empty.

We next compare for the particular case $\widetilde{L} = 1$ the effort in the proposed bounding method associated with the computation of the lower bound for the failure distances from states with the effort in our implementation of the trivial method associated with the evaluation of the fault tree. Note that $\widetilde{d}(a) = 1$ for up states a and that $0 \leq \widetilde{\eta}(e) \leq 1$. Beginning at the point in which a state a has been pulled out of the FIFO queue and the list of failure bags that are active in a has been obtained, the proposed bounding method continues by calling *comp_d*$(F(a), 0, 1, CS)$. From the description of *comp_d*, this call results in traversing up the fault tree following the nodes z such that $dv(z)$ changes to 0, i.e. the nodes that would become implied, exactly as *eval_ft*$(F(a), CS)$ does in our implementation of the trivial bounding method. Next, in the proposed bounding method $lb = \max\{0, \widetilde{d}(a) - |e|, \widetilde{\eta}(e) - |F(a)|\}$ and $ub = \min\{\widetilde{d}(a), \widetilde{\eta}(e)\}$ are computed for each active e, and $\widetilde{d}(b)$ for the successors b of a associated with e is obtained by calling *comp_d*(e, lb, ub, CS') once. Since a is an up state, $\widetilde{d}(a) = 1$. Furthermore, $0 \leq \widetilde{\eta}(e) \leq 1$ and $|e|, |F(a)| \geq 1$. Then, $lb = 0$ and $0 \leq ub \leq 1$, with $ub = 0$ only if $\widetilde{\eta}(e) = 0$. Therefore, if $\widetilde{\eta}(e) > 0$, the call *comp_d*$(e, 0, 1, CS')$ again involves traversing up the fault tree following the nodes which would become implied, exactly as *eval_ft*(e, CS') does in our implementation of the trivial bounding method; if $ub = 0$, *comp_d* returns 0 without traversing the fault tree. Therefore, for the particular case $\widetilde{L} = 1$ the effort in the proposed bounding method associated with the computation of the lower bounds for the failure distances from states is at most equal to the effort in our implementation of the trivial method associated with the evaluation of the fault tree.

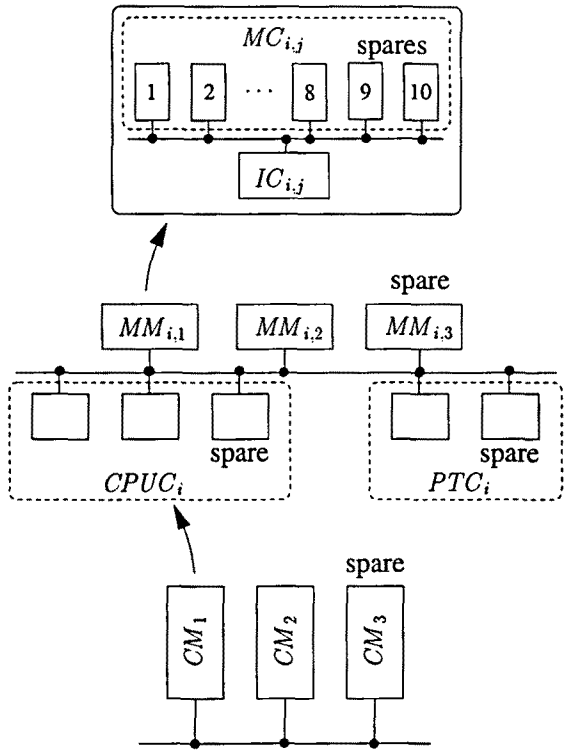

Figure 4.6: Architecture of the first example.

4.3 Analysis and Comparison

In this section we analyze the proposed bounding method by means of two large examples representing the following two scenarios:

1. the fault tree satisfies the conditions of Theorem 4.3,
2. the fault tree does not satisfy the conditions of Theorem 4.3 and $\widetilde{L} > 1$.

In both examples the state o without failed components is the initial state and the CTMC X' is solved using the randomization technique [49].

The bounds obtained with the proposed bounding method are compared with those obtained with the trivial method and the method described in Chapter 3. The lower bound is the same for the three methods.

The first example, adapted from [61] and corresponding to scenario 1, is a system including 114 components whose architecture is shown in Figure 4.6. The system is made up of three computing modules CM_i, $1 \leq i \leq 3$, one of which is spare. Each computing

module CM_i includes three memory modules $MM_{i,j}$, $1 \leq j \leq 3$, three identical CPU chips $CPUC_i$ and two identical port chips PTC_i. One memory module, one CPU chip and one port chip are spares. Each memory module $MM_{i,j}$ is made up of ten identical memory chips $MC_{i,j}$, two of which are spares, and one interface chip $IC_{i,j}$. All memory chips within the same memory module are identical. The memory chips and interface chips included in each memory module, and the CPU and port chips included in each computer module are different and have different failure rates. Active memory chips $MC_{i,j}$ and active interface chips $IC_{i,j}$ fail, respectively, with rates λ_{MC_j} and λ_{IC_j}; active port chips PTC_i and active CPU chips $CPUC_i$ fail, respectively, with rates λ_{PTC_i} and λ_{CPUC_i}. Spare chips fail with rates $\nu \times \lambda_{MC_j}$, $\nu \times \lambda_{IC_j}$, $\nu \times \lambda_{PTC_i}$, and $\nu \times \lambda_{CPUC_i}$, being ν, $0 < \nu < 1$ a dormancy factor. Recovery is hierarchical. A fault in a memory chip is covered with probability C_{MC}. A failure of a memory module, CPU chip and port chip is successfully covered with probabilities C_{MM}, C_{CPUC} and C_{PTC}, respectively. The failure of a computing module is covered with probability C_{CM}.

Coverage faults are modeled by introducing "recovery" components. In this example, an uncovered fault in a memory chip of memory module $MM_{i,j}$ is propagated to a recovery component $RMM_{i,j}$; an uncovered failure of a memory module $MM_{i,j}$, a CPU chip $CPUC_i$ or a port chip PTC_i is propagated to two recovery components RCM_i; an uncovered failure in a computing module CM_i is propagated to four recovery components $RSYS$. Memory module $MM_{i,j}$ is operational if at least eight memory chips $MC_{i,j}$, the interface chip $IC_{i,j}$ and the recovery component $RMM_{i,j}$ are unfailed. Computing module CM_i is operational if at least two memory modules $MM_{i,j}$, two CPU chips $CPUC_i$, one port chip PTC_i, and one recovery component RCM_i are unfailed. Finally, the system is operational if at least two computing modules CM_i an one recovery component $RSYS$ are unfailed. Components of non-operational memory modules and non-operational computing modules do not fail.

The fault tree of the first example has 37 inputs, all of which are modules, 25 gates, 13 of which are modules, and 73 edges. Note that the generalization to component classes allows to exploit the symmetries of the system thus reducing the number of inputs of the fault tree from 133 (114 components plus 19 fictitious components) to 37. The fault tree is defined by the following logical expressions:

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

$$g_r = Z \vee FCC_{1,2} \vee FCC_{1,3} \vee FCC_{2,3},$$

the bags associated with the inputs of the fault tree being $b(T_{i,j}) = MC_{i,j}[3]$, $b(U_{i,j}) = IC_{i,j}[1]$, $b(V_{i,j}) = RMM_{i,j}[1]$, $b(W_i) = CPUC_i[2]$, $b(X_i) = PTC_i[2]$, $b(Y_i) = RCM_i[2]$, and $b(Z) = RSYS[4]$.

The example has 2,701 minimal cuts and $L = \widetilde{L} = 4$. The numerical results have been obtained for the following parameter values: $\lambda_{MC_1} = 10^{-7}\,h^{-1}$, $\lambda_{MC_2} = 2 \times 10^{-7}\,h^{-1}$, $\lambda_{MC_3} = 3 \times 10^{-7}\,h^{-1}$, $\lambda_{IC_1} = 2 \times 10^{-7}\,h^{-1}$, $\lambda_{IC_2} = 4 \times 10^{-7}\,h^{-1}$, $\lambda_{IC_3} = 6 \times 10^{-7}\,h^{-1}$, $\lambda_{CPUC_1} = 6 \times 10^{-7}\,h^{-1}$, $\lambda_{CPUC_2} = 1.2 \times 10^{-6}\,h^{-1}$, $\lambda_{CPUC_3} = 1.8 \times 10^{-6}\,h^{-1}$, $\lambda_{PTC_1} = 6 \times 10^{-7}\,h^{-1}$, $\lambda_{PTC_2} = 1.2 \times 10^{-6}\,h^{-1}$, $\lambda_{PTC_3} = 1.8 \times 10^{-6}\,h^{-1}$, $\nu = 0.2$, $C_{MC} = 0.99$, $C_{MM} = 0.95$, $C_{CPUC} = 0.99$, $C_{PTC} = 0.97$, and $C_{CM} = 0.95$.

The second example, corresponding to scenario 2, is the system made up of 60 components whose architecture is sketched in Figure 4.7. The system includes four processing clusters that communicate through two independent double-ring networks A and B. Processing cluster i, $0 \leq i \leq 3$ includes three identical processing units PU_i. Network A includes eight nodes NA_i, $0 \leq i \leq 7$ and direct (clockwise) links DA_i and reverse (counter-clockwise) links RA_i, linking nodes NA_i and $NA_{(i+1) \bmod 8}$. The structure of network B is the same as that of network A and its nodes, direct links and reverse links are called, respectively, NB_i, DB_i and RB_i. Processing clusters can communicate using one of the configurations for network A or B described in the following. 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 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. The network configuration that is tried first is a direct ring including all nodes and direct links. The second configuration that is tried is a reverse ring including all nodes and reverse links. The third configuration is used when parallel direct and inverse links i are failed and includes all nodes and links except the two failed links. The last configuration, which is used when, for instance, node i fails, includes all nodes except node i and all links except those between node i and nodes $(i \pm 1) \bmod 8$. The components included in the operational configuration of the system are called active. Active processing units, active nodes and active links fail with rates λ_{PU}, λ_N and λ_L, respectively. Inactive components fail with rates $\nu \times \lambda_{PU}$, $\nu \times \lambda_N$ and $\nu \times \lambda_L$, where ν, $0 < \nu < 1$ is a dormancy factor. Components of non-operational network A do not fail. Coverage is assumed perfect for link faults. Faults in processing units and nodes are covered with probabilities C_{PU} and C_N, respectively. Coverage faults are modeled by adding three recovery components $RSYS$ and propagating to them all

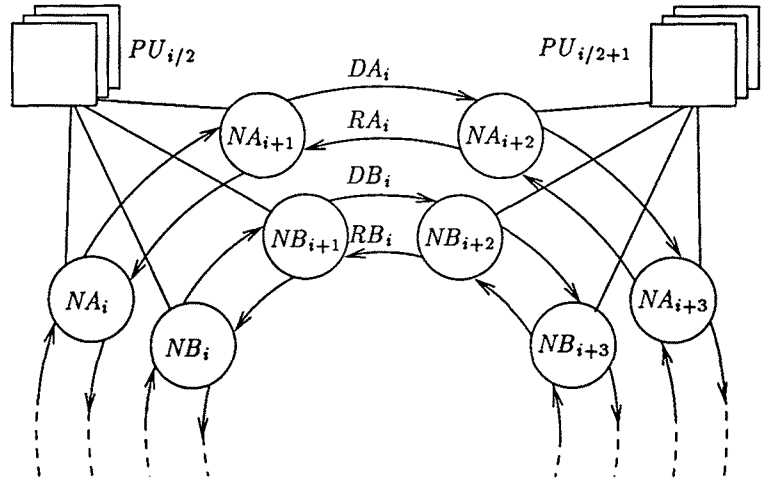

Figure 4.7: Architecture of the second example.

any uncovered fault. The system is operational if each processing cluster has at least one unfailed processing unit, all processing clusters can communicate using one of the given configurations of network A or B, and at least one recovery component $RSYS$ is unfailed.

The fault tree of the system has 53 inputs, all of which are modules, 40 gates, 4 of which are modules, and 764 edges. The fault tree can be described by the following expressions:

$$DRA = \bigvee_{i=0}^{7} (S_i \vee T_i) ,$$

$$DRB = \bigvee_{i=0}^{7} (V_i \vee W_i) ,$$

$$RRA = \bigvee_{i=0}^{7} (S_i \vee U_i) ,$$

$$RRB = \bigvee_{i=0}^{7} (V_i \vee X_i) ,$$

$$FRA_i = \bigvee_{j=0}^{7} S_j \vee \bigvee_{\substack{j=0 \\ j \neq i}}^{7} (T_j \vee U_j) ,$$

$$FRB_i = \bigvee_{j=0}^{7} V_j \vee \bigvee_{\substack{j=0 \\ j \neq i}}^{7} (W_j \vee X_j) ,$$

$$FRA_i^* = \bigvee_{\substack{j=0 \\ j\neq i}}^{7} S_j \vee \bigvee_{\substack{j=0 \\ j\neq i,(i-1)\bmod 8}}^{7} (T_j \vee U_j),$$

$$FRB_i^* = \bigvee_{\substack{j=0 \\ j\neq i}}^{7} V_j \vee \bigvee_{\substack{j=0 \\ j\neq i,(i-1)\bmod 8}}^{7} (W_j \vee X_j),$$

$$NETA = DRA \wedge RRA \wedge \bigwedge_{i=0}^{7} FRA_i \wedge \bigwedge_{i=0}^{7} FRA_i^*,$$

$$NETB = DRB \wedge RRB \wedge \bigwedge_{i=0}^{7} FRB_i \wedge \bigwedge_{i=0}^{7} FRB_i^*,$$

$$NET = NETA \wedge NETB,$$

$$g_r = NET \vee Z \vee \bigvee_{i=0}^{3} Y_i,$$

the bags associated with the inputs of the fault tree being $\mathrm{b}(S_i) = NA_i[1]$, $\mathrm{b}(T_i) = DA_i[1]$, $\mathrm{b}(U_i) = RA_i[1]$, $\mathrm{b}(V_i) = NB_i[1]$, $\mathrm{b}(W_i) = DB_i[1]$, $\mathrm{b}(X_i) = RB_i[1]$, $\mathrm{b}(Y_i) = PU_i[3]$, and $\mathrm{b}(Z) = RSYS[3]$.

The second example has 32,405 minimal cuts and $L = 3$, $\widetilde{L} = 2$. The numerical results have been obtained for the following parameter values: $\lambda_{PU} = 10^{-6}\,\mathrm{h}^{-1}$, $\lambda_N = 5 \times 10^{-7}\,\mathrm{h}^{-1}$, $\lambda_L = 3 \times 10^{-7}\,\mathrm{h}^{-1}$, $\nu = 0.2$, $C_{PU} = 0.99$ and $C_N = 0.99$.

We have considered $K = 2$, 3, 4, and 5 for the first example and $K = 2$, 3 and 4 for the second one. Figures 4.8 and 4.9 show the unreliability bounds obtained using the proposed bounding method for the first and second examples, respectively, as a function of time in years[1]. The bounds degrade as time increases. In both examples, however, the proposed bounding method achieves tight bounds for mission times up to 5 years using affordable numbers of states (respectively, 114,243 and 251,920).

In Tables 4.3 and 4.4 we compare, for the two examples and several mission times, the relative unreliability band obtained with the proposed bounding method, $urb(t) = ([ur(t)]_{\mathrm{ub}} - [ur(t)]_{\mathrm{lb}})/[ur(t)]_{\mathrm{lb}}$ against that obtained with the method proposed in Chapter 3, $urb'(t) = ([ur(t)]'_{\mathrm{ub}} - [ur(t)]_{\mathrm{lb}})/[ur(t)]_{\mathrm{lb}}$ and that obtained with the trivial method, $urb''(t) = ([ur(t)]''_{\mathrm{ub}} - [ur(t)]_{\mathrm{lb}})/[ur(t)]_{\mathrm{lb}}$. The band $urb'(t)$ is not shown in Table 4.3 since for the first example $[ur(t)]'_{\mathrm{ub}} = [ur(t)]_{\mathrm{ub}}$ and, therefore, $urb'(t) = urb(t)$. The proposed bounding method clearly outperforms the trivial method. Thus, for mission times up to 1 year, the ratio $urb''(t)/urb(t)$ is greater than or equal to 21 for the first example and 30

[1] 1 month = 730 h and 1 year = 8,760 h.

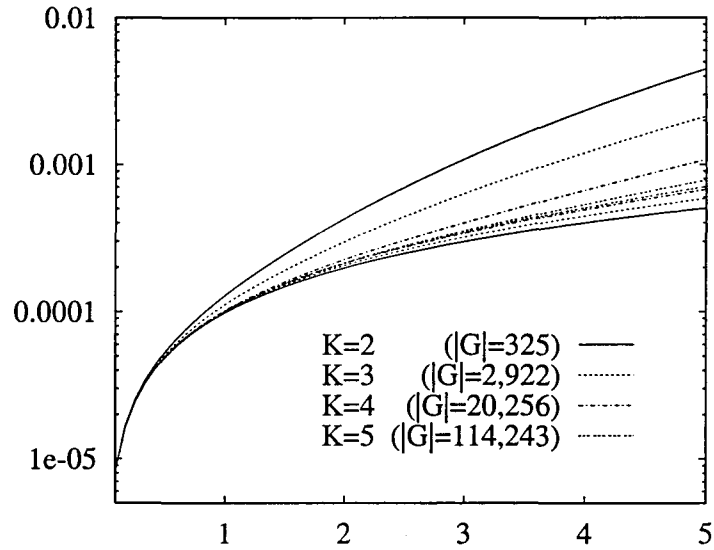

Figure 4.8: Unreliability bounds for the first example as a function of time in years and the value of K.

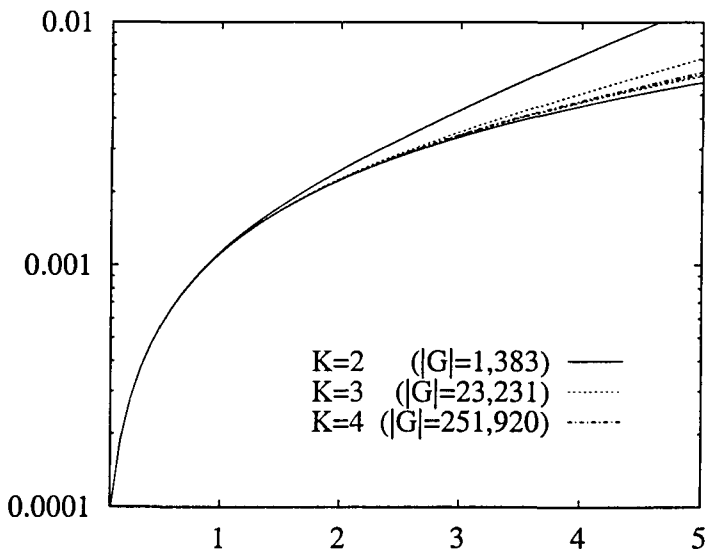

Figure 4.9: Unreliability bounds for the second example as a function of time in years and the value of K.

Table 4.3: Relative unreliability band obtained with the proposed bounding method, $urb(t)$ (top) and with the trivial method, $urb''(t)$ (bottom) for the first example.

time	K (states)			
	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}

for the second one. In addition, the proposed method allows to compute bounds that are almost as tight or even tighter than those given by the trivial method using significantly fewer states. Thus, for the first example and $t = 2$ years, the relative band obtained by the proposed method with $K = 4$ ($|G| = 20{,}256$) is better than that obtained by the trivial method with $K = 5$ ($|G| = 114{,}243$). For the second example and $t = 2$ years, the relative band obtained by the proposed method with $K = 3$ ($|G| = 23{,}231$) is only slightly worse than that obtained by the trivial method with $K = 4$ ($|G| = 251{,}920$). The bounds looseness introduced by the proposed method with regard to the method described in Chapter 3 is reasonable. For the first example, $urb(t) = urb'(t)$ because the fault tree satisfies the conditions of Theorem 4.3; for the second example, the use of minimal cuts to compute exact failure distances less than halves the relative unreliability band.

In Table 4.5 we give the CPU time in seconds to generate the CTMC and compute the unreliability bounds for $t = 5$ years for both examples and the three methods. The CPU time has been measured in a 167 MHz, 128 MB SPARC Ultra 1 workstation. With respect to the trivial method, our method introduces a CPU time overhead due to the computation of lower bounds for failure distances from states only when $\widetilde{L} > 1$. This

Table 4.4: Relative unreliability band obtained with the proposed bounding method, $urb(t)$ (top), with the method described in Chapter 3, $urb'(t)$ (middle) and with the trivial method, $urb''(t)$ (bottom) for the second example.

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

Table 4.5: CPU time in seconds to generate the CTMC and compute the unreliability bounds for $t = 5$ years for both examples using the proposed bounding method (top), the method described in Chapter 3 (middle) and the trivial method (bottom).

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

overhead is reasonable, ranging from 20% in the second example with $K = 4$ to 40% in the first example with $K = 3$. Regarding the method proposed in Chapter 3, the method proposed in this chapter is always faster.

For the first example, the proposed bounding method should be the method of choice since it is faster than the method described in Chapter 3 and gives the same unreliability bounds. For the second example, at the price of introducing some looseness in the bounds, the proposed bounding method is faster than that described in Chapter 3. In addition, the method described in Chapter 3 has a significant memory overhead due to storage of minimal cuts, so the proposed method is also the method of choice.

4.4 Conclusions

In this chapter, we have developed a bounding method for the unreliability at time t, $ur(t)$, which exploits the concept of failure distance, but does not require the knowledge of the minimal cuts of the fault tree of the system. The method is based on lower bounds for failure distances that can be computed inexpensively on the fault tree. The method can be preferable to both the trivial method and the method proposed in Chapter 3 with the same generated subset of states in several cases. Thus, the proposed method gives tighter bounds than the trivial method with a moderate CPU time overhead if $\widetilde{L} > 1$ and without any overhead if $\widetilde{L} = 1$. Regarding the method described in Chapter 3, the proposed method seems to be faster and in some cases it gives exactly the same bounds.

In many other cases, though, the proposed method gives bounds that are looser than the bounds given by the method proposed in Chapter 3. However, that method requires the knowledge of the minimal cuts of the system, whose computation can be infeasible in a reasonable amount of time. Also, the number of minimal cuts can be very large and, in those cases, the method proposed in this chapter may be more efficient from a memory usage point of view.

Chapter 5

Availability Bounds of Repairable Systems using FD

Chapters 3 and 4 have been devoted to develop bounding methods for the unreliability of non-repairable systems. In this chapter we develop a method to upper bound the steady-state unavailability, UA, for repairable systems. The method requires the computation of failure distances. The method developed here takes a particular case of the method described in [22] as starting point and uses one of the state exploration algorithms developed in [20]. The method proposed in [22] was developed for the same class of models considered in this dissertation except that it assumed that repairs involved just one component. We begin by extending that method and showing that it allows group repair, which, as it has been commented in Section 1.5, is an important generalization. Secondly, we improve that method by deriving failure rate bounding structures that are typically better and never worse than the ones used in [22]. Next, we review the state space exploration algorithm developed in [20] we use in our method. Finally, we analyze the performance of the proposed bounding method and compare it with the performances of the methods described in [22] and [70].

5.1 Extension to Group Repair

Let Ω be the state space of the CTMC modeling the system. The method proposed in [22] obtains bounds for the steady-state unavailability generating a subset, G, of Ω. That method also uses a cloning technique that consists in the modification of Ω by adding to $U = \Omega - G$ clones of states of G with more than F failed components. Note that the

cloning technique does not modify UA. In this section we review that method for the particular case $F = 0$, i.e. when all states of G but o (the only state of Ω without failed components) are cloned, and show that it allows group repair.

We start by introducing some notation. Let $X = \{X(t); t \geq 0\}$ be the CTMC modeling the system with the state cloning technique applied and let Ω be the state space of X. Throughout the section we will denote by λ_{ij}, $i, j \in \Omega$ the transition rate from state i to state j, by $\lambda_i = \sum_{j \in \Omega} \lambda_{ij}$, $i \in \Omega$ the output rate of state i, and by $\lambda_{i,B} = \sum_{j \in B} \lambda_{ij}$, $i \in \Omega$, $B \subset \Omega$ the transition rate from state i to subset B, all referred to X unless stated otherwise. We will also consider a number of transient CTMCs Y. Each such CTMC Y has a state space of the form $B \cup \{a\}$, where all states in B are transient and a is an absorbing state, and has a well-defined initial probability distribution with $P[Y(0) \in B] = 1$. We will denote by $\tau(i, Y)$, $i \in B$ the mean time spent by Y in i before absorption ($\tau(i, Y) = \int_0^\infty P[Y(t) = i]dt$). We will also use the notation $\tau(B', Y) = \sum_{i \in B'} \tau(i, Y)$, $B' \subset B$. It is well known (see, for instance, [13]) that the mean times to absorption vector $\tau = (\tau(i, Y))_{i \in B}$ is the solution of the linear system $\tau^T \mathrm{A} = -\mathrm{q}^T$, where A is the restriction of the transition rate matrix of Y to B and $\mathrm{q} = (P[Y(0) = i])_{i \in B}$. The expected number of times that a transition (i, j) with rate λ_{ij} is followed is $\mu_{ij} = \tau(i, Y)\lambda_{ij}$. The result follows easily: $\mu_{ij} = \int_0^\infty P[Y(t) = i]\lambda_{ij}dt = \lambda_{ij} \int_0^\infty P[Y(t) = i]dt = \lambda_{ij}\tau(i, Y)$.

Consider the regenerative behavior of X defined by the times at which X hits state o from U. Let T_G and T_U be, respectively, the contributions of the subsets G and U to the mean time between regenerations. Let C_G and C_U be, respectively, the contributions of the subsets G and U to the mean down time between regenerations. Let $[T_U]_{\mathrm{ub}}$ be an upper bound for T_U and let $[C_U]_{\mathrm{ub}}$ be an upper bound for C_U. Then, we have [22] the following lower and upper bounds for the steady-state unavailability UA:

$$[UA]_{\mathrm{lb}} = \frac{C_G}{T_G + [T_U]_{\mathrm{ub}}}, \tag{5.1}$$

$$[UA]_{\mathrm{ub}} = \frac{C_G + [C_U]_{\mathrm{ub}}}{T_G + [C_U]_{\mathrm{ub}}}. \tag{5.2}$$

Let D be the subset of down states of X and let Y_G be the transient CTMC with state space $G \cup \{a\}$ and initial state o, built from X by directing to a transitions from states in G to states in U. The quantities T_G and C_G can be computed from the mean

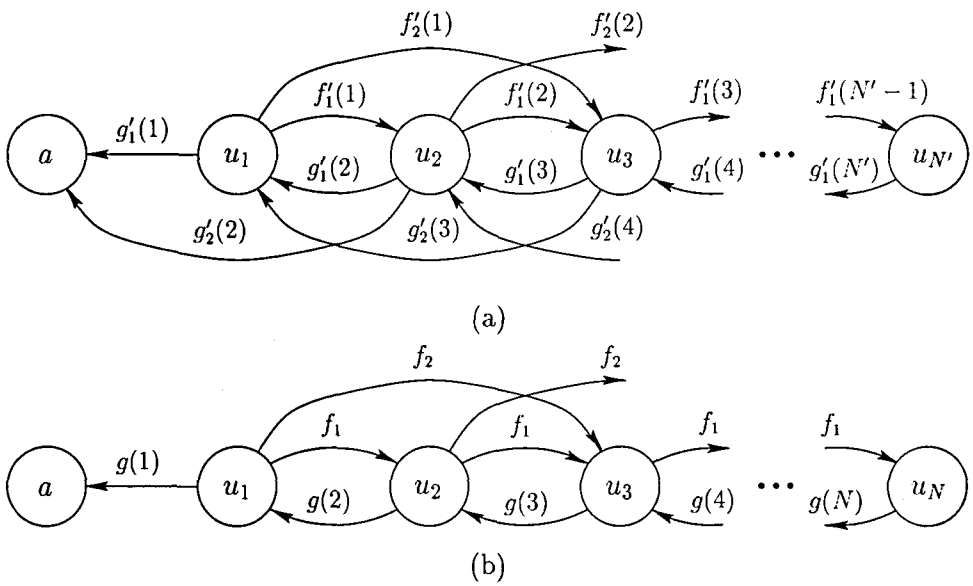

Figure 5.1: State transition diagrams of the transient CTMCs Y' (top) and Y (bottom) of Lemma 5.1.

time to absorption vector of Y_G, $(\tau(i, Y_G))_{i\in G}$, as

$$T_G = \sum_{i\in G} \tau(i, Y_G)\,, \tag{5.3}$$

$$C_G = \sum_{i\in G\cap D} \tau(i, Y_G)\,. \tag{5.4}$$

In the following we will denote by C, N and MC the bag of component classes, the number of components and the set of minimal cuts of the system, respectively. We will also denote by FC the set of distinct cardinalities of the failure bags of the system and by E_i, $i \in FC$ the set of failure bags e, $e \in E$ with cardinality i. Let U_k be the subset of U including the states with k failed components. Let Y^{u_k} be a transient CTMC with state space $\{u_1, \ldots, u_N\} \cup \{a\}$, initial state u_k and state transition diagram like the one shown in Figure 5.1, b: there is a transition with rate $g(1)$ from u_1 to a, a transition with rate $g(k)$, $2 \le k \le N$ from u_k to u_{k-1}, and, for each $i \in FC$, $i \le N-k$, a transition with rate $f_i = \sum_{e\in E_i} \lambda_{\mathrm{ub}}(e)$ from u_k to u_{k+i}. The upper bound $[T_U]_{\mathrm{ub}}$ is

$$[T_U]_{\mathrm{ub}} = \sum_{k=1}^{N} \pi_k\, T(k)\,, \tag{5.5}$$

where

$$\pi_k = \sum_{i\in G} \tau(i, Y_G)\, \lambda_{i,U_k} \tag{5.6}$$

is the probability that X with initial state o will enter U through subset U_k and $T(k)$ is the mean time to absorption of Y^{u_k}. An efficient procedure for computing $T(k)$, $k = 1, \ldots, N$ is described in [22].

Next, we prove the correctness of $[T_U]_{\text{ub}}$ when repairs may involve an arbitrary number of components. Let $N' \leq N$ and let $Y' = \{Y'(t); t \geq 0\}$ be a transient CTMC with state space $\{u_1, \ldots, u_{N'}\} \cup \{a\}$, initial probability distribution $\mathrm{P}[Y'(0) = u_i] = \pi_i$, $1 \leq i \leq N'$, $\sum_{i=1}^{N'} \pi_i = 1$, and the state transition diagram shown in Figure 5.1, a: there is a transition with rate $g'_i(k)$, $1 < k \leq N'$, $1 \leq i < k$ from u_k to u_{k-i}, a transition with rate $g'_k(k)$, $1 \leq k \leq N'$ from u_k to a, and a transition with rate $f'_i(k)$, $i \leq N' - k$, $1 \leq k < N'$ from u_k to u_{k+i}. Let $Y = \{Y(t); t \geq 0\}$ be a transient CTMC with state space $\{u_1, \ldots, u_N\} \cup \{a\}$, initial probability distribution $\mathrm{P}[Y(0) = u_i] = \pi_i$, $1 \leq i \leq N'$, $\mathrm{P}[Y(0) = u_i] = 0$, $N' < i \leq N$, and same state transition diagram as Y^{u_k} (the one shown in Figure 5.1, b). We have the following result.

Lemma 5.1 *Assume* $f_j \geq 0$, $f_j \geq f'_j(i)$, $1 \leq i \leq N'$ *and* $0 < g(i) \leq \sum_{j=1}^{i} g'_j(i)$, $1 \leq i \leq N'$. *Then,* $\tau(u_i, Y) \geq \tau(u_i, Y')$, $1 \leq i \leq N'$.

Proof The proof is by induction on k. We will use the balance equation applied to a subset of states of a transient CTMC, which states that the initial probability of the subset plus the expected number of entries in it is equal to the final probability of the subset plus the expected number of exits of it. Note that the states u_i, $1 \leq i \leq N$ of Y and the states u_i, $1 \leq i \leq N'$ of Y' are transient and, therefore, have final probabilities equal to zero.

Let $\tau_i = \tau(u_i, Y)$ and $\tau'_i = \tau(u_i, Y')$. The balance equation applied to the subset of states $\cup_{i=1}^{N'} u_i$ of Y' yields

$$1 = \sum_{i=1}^{N'} \tau'_i g'_i(i), \tag{5.7}$$

$$\tau'_1 = \frac{1 - \sum_{i=2}^{N'} \tau'_i g'_i(i)}{g'_1(1)}. \tag{5.8}$$

The balance equation applied to the subset of states $\cup_{i=1}^{N} u_i$ of Y yields

$$1 = \tau_1 g(1), \tag{5.9}$$

$$\tau_1 = \frac{1}{g(1)}. \tag{5.10}$$

Using (5.8), $\tau'_i \geq 0$, $1 \leq i \leq N'$, $g(1) \leq g'_1(1)$ and (5.10),

$$\tau'_1 \leq \frac{1}{g'_1(1)} \leq \frac{1}{g(1)} = \tau_1,$$

which proves the result for $k = 1$. Next, we assume $\tau'_i \leq \tau_i$, $1 \leq i \leq k-1$, $k \geq 2$ and show $\tau'_k \leq \tau_k$. The balance equation applied to the subset of states $\cup_{i=1}^{k-1} u_i$ of Y' yields

$$\sum_{i=1}^{k-1} \pi_i + \sum_{i=k}^{N'} \tau'_i \sum_{j=i-k+1}^{i-1} g'_j(i) = \sum_{i=1}^{k-1} \tau'_i g'_i(i) + \sum_{i=1}^{k-1} \tau'_i \sum_{j=k-i}^{N'-i} f'_j(i) .$$

Using $1 = \sum_{i=1}^{k-1} \pi_i + \sum_{i=k}^{N'} \pi_i$ and (5.7),

$$1 - \sum_{i=k}^{N'} \pi_i + \tau'_k \sum_{j=1}^{k-1} g'_j(k) + \sum_{i=k+1}^{N'} \tau'_i \sum_{j=i-k+1}^{i-1} g'_j(i) = 1 - \sum_{i=k}^{N'} \tau'_i g'_i(i) + \sum_{i=1}^{k-1} \tau'_i \sum_{j=k-i}^{N'-i} f'_j(i) ,$$

$$\tau'_k = \frac{\sum_{i=k}^{N'} \pi_i - \sum_{i=k+1}^{N'} \tau'_i \sum_{j=i-k+1}^{i} g'_j(i) + \sum_{i=1}^{k-1} \tau'_i \sum_{j=k-i}^{N'-i} f'_j(i)}{\sum_{j=1}^{k} g'_j(k)} . \tag{5.11}$$

Similarly, applying the balance equation to the subset of states $\cup_{i=1}^{k-1} u_i$ of Y and using $1 = \sum_{i=1}^{k-1} \pi_i + \sum_{i=k}^{N'} \pi_i$ and (5.9) gives

$$\tau_k = \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau_i \sum_{j=k-i}^{N-i} f_j}{g(k)} . \tag{5.12}$$

Finally, using (5.11), $\tau'_i \geq 0$, $1 \leq i \leq N'$, the induction hypothesis, $f_j \geq 0$, $f_j \geq f'_j(i)$, $1 \leq i \leq N'$, $N' \leq N$, $g(i) \leq \sum_{j=1}^{i} g'_j(i)$, $1 \leq i \leq N'$, and (5.12),

$$\begin{aligned}
\tau'_k &\leq \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau'_i \sum_{j=k-i}^{N'-i} f'_j(i)}{\sum_{j=1}^{k} g'_j(k)} \leq \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau_i \sum_{j=k-i}^{N'-i} f'_j(i)}{\sum_{j=1}^{k} g'_j(k)} \\
&\leq \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau_i \sum_{j=k-i}^{N'-i} f_j}{\sum_{j=1}^{k} g'_j(k)} \leq \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau_i \sum_{j=k-i}^{N-i} f_j}{\sum_{j=1}^{k} g'_j(k)} \\
&\leq \frac{\sum_{i=k}^{N'} \pi_i + \sum_{i=1}^{k-1} \tau_i \sum_{j=k-i}^{N-i} f_j}{g(k)} = \tau_k . \square
\end{aligned}$$

The remaining of the proof is based on the concept of exact aggregation for transient CTMCs. The following result, which defines the exact aggregation of a transient CTMC, is proved in [22, Theorem 3].

Theorem 5.1 (Exact aggregation for transient CTMCs) *Let $Y = \{Y(t); t \geq 0\}$ be a transient CTMC with state space $B \cup \{a\}$, where all states in B are transient and a is an absorbing state, transition rates $\lambda_{i,j}$, $i \in B$, $j \in B \cup \{a\}$, $i \neq j$, and initial probability distribution $\mathrm{P}[Y(0) = i] = \pi_i$, $i \in B$, $\sum_{i \in B} \pi_i = 1$. Assume $\tau(i, Y) > 0$ for all $i \in B$. Let $B_1 \cup B_2 \cup \cdots \cup B_n$ be a partition of B. Then, there exists a transient CTMC $Y' = \{Y'(t); t \geq 0\}$ (the exact aggregation of Y) with state space $\{b_1, b_2, \ldots, b_n\} \cup \{a\}$,*

transition rates $\lambda'_{b_k,b_l} = \sum_{i\in B_k} \omega_i^k \lambda_{i,B_l}$, $1 \le k,l \le n$, $k \ne l$, *and* $\lambda'_{b_k,a} = \sum_{i\in B_k} \omega_i^k \lambda_{i,a}$, $1 \le k \le n$, *with* $\omega_i^k > 0$, $\sum_{i\in B_k} \omega_i^k = 1$, *and initial probability distribution* $\mathrm{P}[Y'(0) = b_k] = \pi'_k = \sum_{i\in B_k} \pi_i$, *such that* $\tau(b_k, Y') = \tau(B_k, Y)$.

Let Y_U^s, $s \in U$ be the transient CTMC with state space $U^s \cup \{a\}$, U^s including the states reachable from s before exit from U, and initial state s, built from X by directing to a the transitions from states in U^s to o. Let T_U^s be the mean time to absorption of Y_U^s. Noting that $\sum_{i\in G} \tau(i, Y_G)\lambda_{i,j}$ is the probability that X with initial state o will enter U through state j and grouping the contributions of the states $j \in U$ according to the subsets U_k they belong to, we can write

$$\begin{aligned} T_U &= \sum_{j\in U}\sum_{i\in G} \tau(i,Y_G)\lambda_{i,j}T_U^j = \sum_{i\in G}\sum_{j\in U} \tau(i,Y_G)\lambda_{i,j}T_U^j \\ &= \sum_{i\in G}\sum_{k=1}^{N}\sum_{s\in U_k} \tau(i,Y_G)\lambda_{i,s}T_U^s\,. \end{aligned} \tag{5.13}$$

Theorem 5.2 $T_U \le [T_U]_{\mathrm{ub}}$, *where* $[T_U]_{\mathrm{ub}}$ *is given by* (5.5) *and* (5.6).

Proof Let $s \in U_k$ and consider the exact aggregation $Y_U^{s'}$ of Y_U^s under the partition $\cup_{l=1}^{N'_s} U_l^s$, where U_l^s is the subset of U_l including the states reachable from s before exit from U and $1 \le N'_s \le N$. The state transition diagram of $Y_U^{s'}$ looks like the one depicted in part a of Figure 5.1, with N' replaced by N'_s. Using the notation of the figure we have, by Theorem 5.1,

$$\begin{aligned} f'_j(l) &= \sum_{i\in U_l^s} \omega_i^{l,s}\,\lambda_{i,U_{l+j}}\,, \quad 1 \le l < N'_s,\ j \le N'_s - l\,, \\ g'_j(l) &= \sum_{i\in U_l^s} \omega_i^{l,s}\,\lambda_{i,U_{l-j}}\,, \quad 1 < l \le N'_s,\ j < l\,, \\ g'_l(l) &= \sum_{i\in U_l^s} \omega_i^{l,s}\,\lambda_{i,o}\,, \qquad 1 \le l \le N'_s\,, \end{aligned}$$

and

$$\tau(U_i^s, Y_U^s) = \tau(u_i, Y_U^{s'})\,. \tag{5.14}$$

The transition rates $\lambda_{i,U_{l+j}}$, $i \in U_l^s$ are associated with failure bags involving j components and, therefore, are upper bounded by f_j. The repair rate of $i \in U_l^s$, $\lambda_{i,o} + \sum_{j=1}^{l-1} \lambda_{i,U_{l-j}}$, is lower bounded by $g(l)$. Then, using $\omega_i^{l,s} > 0$, $i \in U_l^s$ and $\sum_{i\in U_l^s} \omega_i^{l,s} = 1$,

we have

$$f'_j(l) = \sum_{i \in U_l^s} \omega_i^{l,s} \lambda_{i,U_{l+j}} \le \sum_{i \in U_l^s} \omega_i^{l,s} f_j = f_j \,,$$

$$\sum_{j=1}^{l} g'_j(l) = g'_l(l) + \sum_{j=1}^{l-1} g'_j(l) = \sum_{i \in U_l^s} \omega_i^{l,s} \lambda_{i,o} + \sum_{j=1}^{l-1} \sum_{i \in U_l^s} \omega_i^{l,s} \lambda_{i,U_{l-j}}$$

$$= \sum_{i \in U_l^s} \omega_i^{l,s} \Big(\lambda_{i,o} + \sum_{j=1}^{l-1} \lambda_{i,U_{l-j}} \Big) \ge \sum_{i \in U_l^s} \omega_i^{l,s} g(l) = g(l) \,.$$

Since $\mathrm{P}[Y_U^{s'}(0) = u_k] = 1$ and $N'_s \le N$, $Y_U^{s'}$ and Y^{u_k} fulfill the requirements of Lemma 5.1. Then, using (5.14), $N'_s \le N$ and the lemma,

$$T_U^s = \sum_{i=1}^{N'_s} \tau(U_i^s, Y_U^s) = \sum_{i=1}^{N'_s} \tau(u_i, Y_U^{s'}) \le \sum_{i=1}^{N'_s} \tau(u_i, Y^{u_k}) \le \sum_{i=1}^{N} \tau(u_i, Y^{u_k}) = T(k) \,.$$

Finally, using (5.5), (5.6) and (5.13),

$$T_U \le \sum_{i \in G} \sum_{k=1}^{N} \sum_{s \in U_k} \tau(i, Y_G) \lambda_{i,s} T(k) = \sum_{i \in G} \sum_{k=1}^{N} \tau(i, Y_G) \lambda_{i,U_k} T(k)$$

$$= \sum_{k=1}^{N} \sum_{i \in G} \tau(i, Y_G) \lambda_{i,U_k} T(k) = \sum_{k=1}^{N} \pi_k T(k) = [T_U]_{\mathrm{ub}} \,. \square$$

Let $U_{k,d}$ be the subset of U including the states with k failed components and failure distance d. Let $L = \min_{m \in MC} |m|$ be the redundancy level of the system. The domain $\mathcal{R}$ of pairs (k, d) for which $U_{k,d}$ may be non-empty is defined by [22]

$$\mathcal{R} = \{(k, d) : 1 \le k \le N, \max\{0, L - k\} \le d \le \min\{L, N - k\}\}. \tag{5.15}$$

Let C_U^s be the mean down time to absorption of Y_U^s. The upper bound $[C_U]_{\mathrm{ub}}$ is

$$[C_U]_{\mathrm{ub}} = \sum_{(k,d) \in \mathcal{R}} \pi_{k,d} C(k, d) \,, \tag{5.16}$$

where

$$\pi_{k,d} = \sum_{i \in G} \tau(i, Y_G) \lambda_{i,U_{k,d}} \tag{5.17}$$

is the probability that X with initial state o will enter U through subset $U_{k,d}$, and $C(k, d)$ are upper bounds for C_U^s, $s \in U_{k,d}$.

The upper bounds $C(k, d)$, $(k, d) \in \mathcal{R}$ are computed using an iterative procedure. The procedure starts with

$$C(k, d) = C(k) = \sum_{i=L}^{N} \tau(u_i, Y^{u_k}) \,, \tag{5.18}$$

```
for (all (k, d) ∈ R) C(k, d) = C(k);
do {
   ε' = 0;
   for (k = 1; k ≤ N; k++)
      for (d = max{0, L − k}; d ≤ min{L, N − k}; d++) {
         Compute C'(k, d) using (5.22);
         if (C'(k, d) < C(k, d)) {
            ε' = max{ε', (C(k, d) − C'(k, d))/C'(k, d)};
            C(k, d) = C'(k, d);
         }
      }
} while (ε' ≥ ε);
```

Figure 5.2: Algorithm to compute the $C(k,d)$ bounds.

and improves the bounds using potentially better bounds $C'(k,d)$ until no significant improvement is achieved. Let

$$\mathcal{R}' = \big\{(k,d,i,r) : (k,d) \in \mathcal{R}, i \in FC, i \le N-k, \\ \max\{0, d-i\} \le r \le \min\{d, N-k-i\}\big\}. \tag{5.19}$$

Let $F_{i,r}(k,d)$, $(k,d,i,r) \in \mathcal{R}'$ be upper bounds for $\lambda_{l,\cup_{0\le d'\le r} U_{k+i,d'}}$, $l \in U_{k,d}$, i.e. for the total failure rate involving i components from any state in U with k failed components and failure distance d to states with failure distance $\le r$. Let $J_m(k,d,i) = \max\{0, k+d+i-N\}$ and $J_M(d,i) = \min\{d,i\}$ and let

$$f_{ij}(k,d) = F_{i,d-j}(k,d) - F_{i,d-j-1}(k,d)\,, \quad J_m(k,d,i) \le j < J_M(d,i)\,, \tag{5.20}$$

$$f_{i,J_M(d,i)}(k,d) = F_{i,d-J_M(d,i)}(k,d)\,. \tag{5.21}$$

The upper bounds $C'(k,d)$, $(k,d) \in \mathcal{R}$ are computed using

$$C'(k,d) = \frac{I_{d=0}}{g(k)} + I_{k>1}\left[I_{d>L-k}C(k-1,d) + I_{d\le L-k}C(k-1,d+1)\right] \\ + \frac{1}{g(k)} \sum_{\substack{i\in FC \\ i\le N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} f_{ij}(k,d)C(k+i,d-j)\,, \tag{5.22}$$

where I_c denotes the indicator function returning the value 1 if condition c is satisfied and the value 0 otherwise. The algorithm to compute the $C(k,d)$ bounds is given in Figure 5.2. The parameter ϵ is a tolerance factor that determines when the improvement is small enough for the algorithm to stop.

In [22, Theorem 5] it is proved that $C_U \leq [C_U]_{\text{ub}}$ provided that $C_U^s \leq C(k,d)$, $s \in U_{k,d}$. In the following we prove that if $F_{i,r}(k,d)$, $(k,d,i,r) \in \mathcal{R}'$, and $F_{i,d}(k,d)$, $(k,d,i,d) \in \mathcal{R}'$ are decreasing on d, then the bounds $C(k,d)$ computed by the algorithm of Figure 5.2 upper bound C_U^s, $s \in U_{k,d}$ when repairs may involve an arbitrary number of components. The proof consists of a sequence of two propositions, a lemma and a theorem. In the following, we will denote by RC the set of distinct cardinalities of the repair bags of the system.

Proposition 5.1 *Assume $C_U^s \leq C(k,d)$, $s \in U_{k,d}$ and that $C(k,d)$, $(k,d) \in \mathcal{R}$ are increasing on k and decreasing on d. Then, $C_U^s \leq C'(k,d)$, $s \in U_{k,d}$.*

Proof Let $s \in U_{k,d}$. C_U^s is equal to the mean down time in s, if $d = 0$, plus the mean down time from the next visited state m, if $m \in U$. Let us discuss now to which subsets $U_{k',d'}$ the state m may belong to. Consider first repair transitions involving i, $i \in RC$, $i \leq k$ components. These transitions lead to states with $k' = k - i$ failed components whose failure distance is neither smaller than d nor larger than $\min\{L, d+i\}$. Also, if $i = k$, the reached state is $o \notin U$. Therefore, only repair transitions involving $i \in RC$, $i \leq k-1$ components have to be considered and they may lead to $m \in U_{k-i,d+j}$, $0 \leq j \leq i$. By imposing $(k-i, d+j) \in \mathcal{R}$ we get $\max\{0, L-k+i-d\} \leq j \leq \min\{i, L-d\}$. Transitions associated with failure bags $e \in E_i$, $i \in FC$, $i \leq N-k$ can be analyzed in a similar way by noting that they lead to states with $k+i$ failed components whose failure distance is neither larger than d nor smaller than $\max\{0, d-i\}$. Imposing $(k+i, d-j) \in \mathcal{R}$ yields $J_m(k,d,i) \leq j \leq J_M(d,i)$. Based on the previous discussion and denoting $J'_m(k,d,i) = \max\{0, L-k+i-d\}$ and $J'_M(d,i) = \min\{i, L-d\}$, we can write

$$C_U^s = T_1(d) + T_2(k,d,i) + T_3(k,d,i)\,, \tag{5.23}$$

$$T_1(d) = \frac{I_{d=0}}{\lambda_s}\,,$$

$$T_2(k,d,i) = \sum_{\substack{i \in RC \\ i \leq k-1}} \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m \in U_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} C_U^s\,,$$

$$T_3(k,d,i) = \sum_{\substack{i \in FC \\ i \leq N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} \sum_{m \in U_{k+i,d-j}} \frac{\lambda_{s,m}}{\lambda_s} C_U^s\,.$$

Since $\lambda_s \geq g(k)$,

$$T_1(d) \leq \frac{I_{d=0}}{g(k)}\,. \tag{5.24}$$

If $k = 1$, $T_2(k,d,i) = 0$. Assume $k > 1$. Using $C_U^m \le C(k',d')$, $m \in U_{k',d'}$, the fact that $C(k',d')$ are increasing on k' and decreasing on d', and noting that $\lambda_s \ge \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m\in U_{k-i,d+j}} \lambda_{s,m}$,

$$
\begin{aligned}
T_2(k,d,i) &\le \sum_{\substack{i\in RC\\ i\le k-1}} \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} C(k-i,d+j) \sum_{m\in U_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} \\
&\le \sum_{\substack{i\in RC\\ i\le k-1}} C(k-i,d+J'_m(k,d,i)) \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m\in U_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} \\
&= \sum_{\substack{i\in RC\\ i\le k-1}} C(k-i,d+\max\{0,L-k+i-d\}) \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m\in U_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} \\
&\le C(k-1,d+\max\{0,L-k+1-d\}) \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m\in U_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} \\
&\le C(k-1,d+\max\{0,L-k+1-d\}) .
\end{aligned}
$$

If $d > L-k$, $\max\{0,L-k+1-d\} = 0$ and $T_2(k,d,i) \le C(k-1,d)$; if $d \le L-k$, since $(k,d) \in \mathcal{R}$ implies $d = L-k$, we have $\max\{0,L-k+1-d\} = 1$ and, therefore, $T_2(k,d,i) \le C(k-1,d+1)$. To summarize,

$$
T_2(k,d,i) \le I_{k>1}(I_{d>L-k}C(k-1,d) + I_{d\le L-k}C(k-1,d+1)) . \tag{5.25}
$$

To bound $T_3(k,d,i)$, let us denote $f_{i,j}(s) = \lambda_{s,U_{k+i,d-j}}$. Recalling that $C_U^m \le C(k+i,d-j)$, $m \in U_{k+i,d-j}$,

$$
\begin{aligned}
T_3(k,d,i) &\le \sum_{\substack{i\in FC\\ i\le N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} \sum_{m\in U_{k+i,d-j}} \frac{\lambda_{s,m}}{\lambda_s} C(k+i,d-j) \\
&= \sum_{\substack{i\in FC\\ i\le N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} \frac{f_{i,j}(s)}{\lambda_s} C(k+i,d-j) .
\end{aligned}
$$

Let $F_{i,r}(s)$ be the sum of failure transition rates from s involving i components and leading to states with failure distance $\le r$, i.e. $F_{i,r}(s) = \sum_{j=d-r}^{J_M(d,i)} f_{i,j}(s)$, $d - J_M(d,i) \le r \le d - J_m(k,d,i)$. Clearly, $F_{i,r}(s) \le F_{i,r}(k,d)$. Note also that $f_{i,J_M(d,i)}(s) = F_{i,d-J_M(d,i)}(s)$ and that $f_{i,j}(s) = F_{i,d-j}(s) - F_{i,d-j-1}(s)$, $J_m(k,d,i) \le j < J_M(d,i)$. Then, using

$\lambda_s \geq g(k)$, (5.20), (5.21) and the fact that the bounds $C(k', d')$ are decreasing on d',

$$
\begin{aligned}
T_3(k,d,i) \leq & \sum_{\substack{i \in FC \\ i \leq N-k}} \left[\sum_{j=J_m(k,d,i)}^{J_M(d,i)-1} \frac{F_{i,d-j}(s) - F_{i,d-j-1}(s)}{\lambda_s} C(k+i, d-j) \right. \\
& \left. + \frac{F_{i,d-J_M(d,i)}(s)}{\lambda_s} C(k+i, d-J_M(d,i)) \right] \\
= & \sum_{\substack{i \in FC \\ i \leq N-k}} \left[\frac{F_{i,d-J_m(k,d,i)}(s)}{\lambda_s} C(k+i, d-J_m(k,d,i)) \right. \\
& \left. + \sum_{j=J_m(k,d,i)+1}^{J_M(d,i)} \frac{F_{i,d-j}(s)}{\lambda_s} \left(C(k+i, d-j) - C(k+i, d-j+1) \right) \right] \\
\leq & \sum_{\substack{i \in FC \\ i \leq N-k}} \left[\frac{F_{i,d-J_m(k,d,i)}(k,d)}{g(k)} C(k+i, d-J_m(k,d,i)) \right. \\
& \left. + \sum_{j=J_m(k,d,i)+1}^{J_M(d,i)} \frac{F_{i,d-j}(k,d)}{g(k)} \left(C(k+i, d-j) - C(k+i, d-j+1) \right) \right] \\
= & \sum_{\substack{i \in FC \\ i \leq N-k}} \left[\sum_{j=J_m(k,d,i)}^{J_M(d,i)-1} \frac{F_{i,d-j}(k,d) - F_{i,d-j-1}(k,d)}{g(k)} C(k+i, d-j) \right. \\
& \left. + \frac{F_{i,d-J_M(d,i)}(k,d)}{g(k)} C(k+i, d-J_M(d,i)) \right] \\
= & \sum_{\substack{i \in FC \\ i \leq N-k}} \left[\sum_{j=J_m(k,d,i)}^{J_M(d,i)-1} \frac{f_{i,j}(k,d)}{g(k)} C(k+i, d-j) \right. \\
& \left. + \frac{f_{i,J_M(d,i)}(k,d)}{g(k)} C(k+i, d-J_M(d,i)) \right] \\
= & \frac{1}{g(k)} \sum_{\substack{i \in FC \\ i \leq N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} f_{i,j}(k,d) C(k+i, d-j) \,. \qquad (5.26)
\end{aligned}
$$

Finally, the result follows from (5.24), (5.25), (5.26), (5.23), and (5.22). □

Lemma 5.2 *The bounds $C(k)$, $1 \leq k \leq N$ defined by* (5.18) *are increasing on k.*

Proof The proof is by induction on k. Let $\lambda(k) = g(k) + \sum_{\substack{i \in FC \\ k+i \leq N}} f_i$. The bound $C(k)$ is equal to the mean time in u_k, if $k \geq L$, plus the mean down time from the next visited

state. Thus, in view of Figure 5.1, b we can write

$$C(N) = \frac{1}{g(N)} + C(N-1)\,, \tag{5.27}$$

$$C(k) = \frac{I_{k\geq L}}{\lambda(k)} + \frac{g(k)}{\lambda(k)}C(k-1) + \sum_{\substack{i\in FC\\ k+i\leq N}} \frac{f_i}{\lambda(k)}C(k+i)\,, \quad 1 < k < N\,. \tag{5.28}$$

The case $k = N$ is trivial since, from (5.27), $C(N) \geq C(N-1)$. Assume that $C(k')$, $k < k' \leq N$, $1 < k < N$ are increasing on k. Using (5.28), the definition of $\lambda(k)$ and the induction hypothesis,

$$\begin{aligned} C(k-1) &= \frac{1}{g(k)}\Big[\lambda(k)C(k) - I_{d\geq L} - \sum_{\substack{i\in FC\\ k+i\leq N}} f_i C(k+i)\Big] \\ &= \frac{1}{g(k)}\Big[\Big(g(k) + \sum_{\substack{i\in FC\\ k+i\leq N}} f_i\Big)C(k) - I_{k\geq L} - \sum_{\substack{i\in FC\\ k+i\leq N}} f_i C(k+i)\Big] \\ &= C(k) - \frac{I_{k\geq L}}{g(k)} - \frac{1}{g(k)} \sum_{\substack{i\in FC\\ k+i\leq N}} f_i\Big(C(k+i) - C(k)\Big) \qquad (5.29) \\ &\leq C(k)\,.\square \end{aligned}$$

Using (5.27) and (5.29) it is possible to define an efficient procedure to compute $C(k)$, $1 \leq k \leq N$. Let $\mathbf{A}_U$ be the restriction of the transition rate matrix of the CTMC Y^{u_k} to its transient states $\cup_{i=1}^{N}\{u_i\}$ and let $\mathbf{q}_N$ be a column vector with component N set to one and all its remaining components set to zero. The mean times to absorption vector of Y^{u_k} with initial state u_N, $\tau_N = (\tau(u_i, Y^{u_k}))_{1\leq i\leq N}$, is the solution of the linear system

$$\tau_N^{\mathrm{T}}\mathrm{A}_U = -\mathbf{q}_N^{\mathrm{T}}\,. \tag{5.30}$$

Then, we solve (5.30), compute $C(N)$ using (5.18), compute (5.27) $C(N-1) = C(N) - 1/g(N)$ and, finally, compute $C(k)$, $1 \leq k \leq N-2$ using (5.29).

Proposition 5.2 *Assume that $F_{i,r}(k,d)$, $(k,d,i,r) \in \mathcal{R}'$ and $F_{i,d}(k,d)$, $(k,d,i,d) \in \mathcal{R}'$ are decreasing on d. Then, the bounds $C(k,d)$, $(k,d) \in \mathcal{R}$ are increasing on k and decreasing on d.*

Proof Consider the algorithm that improves the bounds $C(k,d)$ split into phases, where each phase includes the operations performed within the k-loop, and let $C^{(m)}(k,d)$, $m \geq 0$

be the bounds $C(k,d)$ available after phase m. We start by proving that the bounds are decreasing on d. The proof is by induction on m. For $m=0$, $C^{(0)}(k,d)=C(k)$, which, trivially, are (non-strictly) decreasing on d. Assume that $C^{(m')}(k,d)$, $0\le m'\le m$, $m\ge 0$ are decreasing on d and let k' be the value of k for which the bounds are updated in phase $m+1$. We have (5.22) that $C'(k',d)$ only depend on $C^{(m')}(k,d)$, $0\le m'\le m$, $k\neq k'$. Using the induction hypothesis, this implies [22, Proposition 3] that $C'(k',d)$ are decreasing on d. Therefore, since, by hypothesis, $C^{(m)}(k',d)$ are decreasing on d, $C^{(m+1)}(k',d)=\min\{C'(k',d),C^{(m)}(k',d)\}$ are decreasing on d.

Next, we prove by induction on m that the bounds $C(k,d)$ are increasing on k. Let $(k,d),(k-1,d)\in\mathcal{R}$. For $m=0$, using Lemma 5.2,

$$C^{(0)}(k,d)=C(k)\ge C(k-1)=C^{(0)}(k-1,d)\,.$$

Assume that $C^{(m')}(k,d)$, $0\le m'\le m$, $m\ge 0$ are increasing on k. Let k' be the value of k for which the bounds are updated in phase $m+1$. Let $(k'-1,d)\in\mathcal{R}$, which implies $k'>1$ and $d\ge\max\{0,L-k'+1\}>L-k'$. We have (5.22)

$$C'(k',d)=\frac{I_{d=0}}{g(k')}+C^{(m)}(k'-1,d)+\frac{1}{g(k')}\sum_{\substack{i\in FC\\ i\le N-k'}}A(k',d,i)\,,$$

$$A(k',d,i)=\sum_{j=J_m(k',d,i)}^{J_M(d,i)}f_{i,j}(k',d)C^{(m_i)}(k'+i,d-j)\,,\qquad m_i<m\,.$$

Using (5.20), (5.21) and recalling that $C^{(m)}(k,d)$ are decreasing on d,

$$\begin{aligned}A(k',d,i)&=\sum_{j=J_m(k',d,i)}^{J_M(d,i)-1}\left[F_{i,d-j}(k',d)-F_{i,d-j-1}(k',d)\right]C^{(m_i)}(k'+i,d-j)\\&\quad+F_{i,d-J_M(d,i)}(k',d)C^{(m_i)}(k'+i,d-J_M(d,i))\\&=\sum_{j=J_m(k',d,i)+1}^{J_M(d,i)}F_{i,d-j}(k',d)\left[C^{(m_i)}(k'+i,d-j)-C^{(m_i)}(k'+i,d-j+1)\right]\\&\quad+F_{i,d-J_m(k',d,i)}(k',d)C^{(m_i)}(k'+i,d-J_m(k',d,i))\ge 0\,.\end{aligned}$$

Therefore, $C'(k',d)\ge C^{(m)}(k'-1,d)$. Using the induction hypothesis, this implies

$$C^{(m+1)}(k',d)=\min\{C'(k',d),C^{(m)}(k',d)\}\ge C^{(m)}(k'-1,d)\,.\square$$

Theorem 5.3 *Assume that $F_{i,r}(k,d)$, $(k,d,i,r)\in\mathcal{R}'$ and $F_{i,d}(k,d)$, $(k,d,i,d)\in\mathcal{R}'$ are decreasing on d. Then, $C_U^s\le C(k,d)$, $s\in U_{k,d}$.*

Proof Consider the algorithm the algorithm that improves the bounds $C(k,d)$ split into phases as in the proof of Proposition 5.2. The proof is by induction on m. For $m = 0$, $C^{(0)}(k,d) = C(k)$, which [22, Theorem 6] upper bound C_U^s, $s \in U_{k,d}$. Assume that $C^{(m')}(k,d)$, $0 \leq m' \leq m$, $m \geq 0$ upper bound C_U^s, $s \in U_{k,d}$. Let k' be the value of k for which the bounds are updated in phase $m+1$. According to (5.22), $C(k',d)$ only depend on $C^{(m')}(k,d)$, $0 \leq m' \leq m$, $k \neq k'$. Proposition 5.2 guarantees that $C^{(m)}(k,d)$ are increasing on k and decreasing on d. Then, using the induction hypothesis and invoking Proposition 5.1, $C'(k',d)$ upper bound C_U^s, $s \in U_{k',d}$. Finally, recalling that, by hypothesis, $C^{(m)}(k',d)$ upper bound C_U^s, $s \in U_{k',d}$, we have that $C^{(m+1)}(k',d) = \min\{C'(k',d), C^{(m)}(k',d)\}$ upper bound C_U^s, $s \in U_{k',d}$. □

5.2 Improved Failure Rate Bounding Structures

The set of bounds $F_{i,r}(k,d)$, $(k,d,i,r) \in \mathcal{R}'$ used in [22] was

$$F_{i,\min\{d,N-k-i\}}(k,d) = f_i, \quad (k,d,i,\min\{d,N-k-i\}) \in \mathcal{R}', \tag{5.31}$$

$$F_{i,r}(k,d) = \sum_{\substack{e \in E_i \\ m \cap e \neq \emptyset \text{ for some } m \in MC \\ Imp(e) \leq k+r \\ Act(e) \geq d-r}} \lambda_{\text{ub}}(e), \quad (k,d,i,r) \in \mathcal{R}', r < \min\{d,N-k-i\}, \tag{5.32}$$

where

$$Imp(e) = \min_{\substack{m \in MC \\ m \cap e \neq \emptyset}} |m - e|,$$

$$Act(e) = \max_{m \in MC} |m \cap e|.$$

In this section we derive bounds $F_{i,r}(k,d)$, $(k,d,i,r) \in \mathcal{R}'$, $r < \min\{d,N-k-i\}$ that are potentially better than the bounds given by (5.32). In the following we will call $F'_{i,r}(k,d)$ the bounds given by (5.32) and $F''_{i,r}(k,d)$ the new bounds. The bounding method will use (5.31) and

$$F_{i,r}(k,d) = \min\{F'_{i,r}(k,d), F''_{i,r}(k,d)\}, \quad (k,d,i,r) \in \mathcal{R}', r < \min\{d,N-k-i\}. \tag{5.33}$$

The bound $[UA]_{\text{lb}}$ does not depend on $C(k,d)$ and, thus, that bound will be identical for the method described in [22] and the method proposed here. However, the bound

$[UA]_{\text{ub}}$ does depend on $C(k,d)$ and that bound may be different for the method described in [22] and the method proposed here. We start proving that smaller bounds $F_{i,r}(k,d)$ potentially give a smaller and thus tighter $[UA]_{\text{ub}}$.

Theorem 5.4 *Smaller $F_{i,r}(k,d)$ bounds give a potentially smaller* $[\text{UA}]_{\text{ub}}$.

Proof Combining (5.20), (5.21) and (5.22) we obtain

$$
\begin{aligned}
C'(k,d) = {} & \frac{I_{d=0}}{g(k)} + I_{k>1}\left[I_{d>L-k}C(k-1,d) + I_{d\leq L-k}C(k-1,d+1)\right] \\
& + \frac{1}{g(k)} \sum_{\substack{i\in FC \\ i\leq N-k}} \Bigg[\sum_{j=J_m(k,d,i)}^{J_M(d,i)-1} \left(F_{i,d-j}(k,d) - F_{i,d-j-1}(k,d)\right) C(k+i,d-j) \\
& \qquad + F_{i,d-J_M(d,i)}(k,d)\, C\left(k+i,d-J_M(d,i)\right)\Bigg] \\
= {} & \frac{I_{d=0}}{g(k)} + I_{k>1}\left[I_{d>L-k}C(k-1,d) + I_{d\leq L-k}C(k-1,d+1)\right] \\
& + \frac{1}{g(k)} \sum_{\substack{i\in FC \\ i\leq N-k}} \Bigg[F_{i,d-J_m(k,d,i)}(k,d)\, C\left(k+i,d-J_m(k,d,i)\right) \\
& \qquad + \sum_{j=J_m(k,d,i)+1}^{J_M(d,i)} F_{i,d-j}(k,d)\left(C(k+i,d-j) - C(k+i,d-j+1)\right)\Bigg].
\end{aligned}
$$

But, since the bounds $C(k,d)$ are decreasing on d by Proposition 5.2, it follows that smaller $F_{i,r}(k,d)$ bounds give smaller $C'(k,d)$ bounds, potentially smaller final $C(k,d)$ bounds, a potentially smaller (5.16) $[C_U]_{\text{ub}}$ and, since $[UA]_{\text{ub}}$ is increasing on $[C_U]_{\text{ub}}$ (5.2), a potentially smaller $[UA]_{\text{ub}}$. □

In the following we derive the new bounds $F''_{i,r}(k,d)$. The intuition on which the new bounds are based is the following. Consider a fault-tolerant system composed of 3 instances of component classes c_i, $1 \leq i \leq n$, with n large. The failure bags of the model are $c_i[1]$, $1 \leq i \leq n$. The system is failed if and only if all 3 instances of the same component class are failed. Thus, the minimal cuts are $c_i[3]$, $1 \leq i \leq n$. Assume that we want to upper bound the transition rate to states with 2 failed components and failure distance ≤ 1 from any state with 1 failed component and failure distance 2 and that the $\lambda_{\text{ub}}(e)$, $e \in E$ are approximately equal. Using (5.32), all failure bags will be included in the summatory. This is, really, a consequence of the fact that $F'_{i,r}(k,d)$ is obtained by adding up the $\lambda_{\text{ub}}(e)$ for all failure bags e for which there exists some state with

$k = 1$ and $d = 2$ for which the failure of the components in e make the failure distance ≤ 1. For the example and failure bag $c_i[1]$, such a state is the state x with $F(x) = c_i[1]$. However, this is very pessimistic, since it is clear that no state with $k = 1$ and $d = 2$ exists in which all failure bags make the failure distance ≤ 1. In fact, for the example, for a state x with $F(x) = c_i[1]$ only the failure bag $e = c_i[1]$ makes the failure distance ≤ 1, and a significantly tighter upper bound (since n is large and the $\lambda_{\text{ub}}(e)$, $e \in E$ are approximately equal) to the rate to states with 2 failed components and failure distance ≤ 1 from any state with 1 failed component and failure distance 2 is $\max_{e \in E} \lambda_{\text{ub}}(e)$. For other values of k and d, the situation is more complex, but the basic intuition still applies: we have to take into account the failure bags that may reduce *simultaneously* the failure distance.

Let S be a bag (set). A collection $\Gamma = \{S_1, S_2, \ldots, S_n\}$ of subbags (subsets) of S is a *Sperner collection* on S if no subbag (subset) of Γ contains another. Let Γ be a collection of bags (sets). We will denote by $N_i(\Gamma)$ the number of bags (sets) in Γ of cardinality i. Using Lubell's theorem [62], we will prove a sequence of two lemmas. The first lemma extends Lubell's theorem to bags; the second lemma is a direct consequence of the first one and will be used in future developments.

Lemma 5.3 *Let Γ be a Sperner collection on a bag S. Then*

$$\sum_{i=1}^{|S|} \frac{N_i(\Gamma)}{\binom{|S|}{i}} \leq 1.$$

Proof Let $\Gamma = \{S_1, S_2, \ldots, S_n\}$. Let D be the domain of S (set of different elements in S). Consider the set S' obtained from S by replacing for each $x \in D$ the $\#(x, S)$ occurrences of x in S by distinct elements $x_1, x_2, \ldots, x_{\#(x,S)}$. For instance, if $S = a[2]b[3]c[1]$, S' would be $\{a_1, a_2, b_1, b_2, b_3, c_1\}$. Consider the collection Γ' of subsets of S' obtained from Γ by replacing each subbag S_i of S by all different subsets $S'_{i,1}, S'_{i,2}, \ldots$ that can be obtained by replacing the $\#(x, S_i)$ occurrences of x in S_i by distinct elements from $x_1, x_2, \ldots, x_{\#(x,S)}$. For instance, for $S = a[2]b[3]c[1]$ and $S_i = a[1]b[2]$ we would have $S'_{i,1} = \{a_1, b_1, b_2\}$, $S'_{i,2} = \{a_1, b_1, b_3\}$, $S'_{i,3} = \{a_1, b_2, b_3\}$, $S'_{i,4} = \{a_2, b_1, b_2\}$, $S'_{i,5} = \{a_2, b_1, b_3\}$, and $S'_{i,6} = \{a_2, b_2, b_3\}$. Since Γ is a Sperner collection on S, Γ' is a Sperner collection on S'. In addition, $N_i(\Gamma) \leq N_i(\Gamma')$ and $|S| = |S'|$. Then, we have

$$\sum_{i=1}^{|S|} \frac{N_i(\Gamma)}{\binom{|S|}{i}} \leq \sum_{i=1}^{|S'|} \frac{N_i(\Gamma')}{\binom{|S'|}{i}}.$$

But, using Lubell's Theorem [62],

$$\sum_{i=1}^{|S'|} \frac{N_i(\Gamma')}{\binom{|S'|}{i}} \leq 1$$

and the result follows. □

Lemma 5.4 *Let Γ be a Sperner collection on a bag S and let $M_i \geq 0$, $i = 1, 2, \ldots, |S|$. Then*

$$\sum_{i=1}^{|S|} N_i(\Gamma) M_i \leq \max_{1 \leq i \leq |S|} \binom{|S|}{i} M_i.$$

Proof Let j be any index $1 \leq j \leq |S|$ for which $\binom{|S|}{j} M_j = \max_{1 \leq i \leq |S|} \binom{|S|}{i} M_i$. For $1 \leq i \leq |S|$ we have $\binom{|S|}{i} M_i \leq \binom{|S|}{j} M_j$, i.e.

$$M_i \leq \frac{\binom{|S|}{j}}{\binom{|S|}{i}} M_j.$$

Then, using Lemma 5.3

$$\begin{aligned} \sum_{i=1}^{|S|} N_i(\Gamma) M_i &\leq \sum_{i=1}^{|S|} N_i(\Gamma) \frac{\binom{|S|}{j}}{\binom{|S|}{i}} M_j = \left(\sum_{i=1}^{|S|} \frac{N_i(\Gamma)}{\binom{|S|}{i}} \right) \binom{|S|}{j} M_j \\ &\leq \binom{|S|}{j} M_j = \max_{1 \leq i \leq |S|} \binom{|S|}{i} M_i . \square \end{aligned}$$

Let $E_i(r)$ be the set of failure bags of cardinality i such that the failure of the components in the failure bag makes the failure distance $\leq r$, i.e. the set of failure bags $e \in E_i$ with $\min_{m \in MC} |m - e| \leq r$. Let F be a subbag of C, let l be an integer ≥ 0, and consider the collections of subbags of F, $A_j^l(F)$, $j = 1, 2, \ldots, |F|$ defined recursively as follows. $A_{|F|}^l(F)$ is $\{F\}$ if F is included in some minimal cut of cardinality $|F| + l$ and $\emptyset$ otherwise. For $1 \leq j < |F|$, $A_j^l(F)$ includes the subbags of F of cardinality j included in some minimal cut of cardinality $j + l$ and not included in any subbag of $\cup_{p=j+1}^{|F|} A_p^l(F)$. We have the following result.

Theorem 5.5 *Let $e \in E_i - E_i(r)$, $i \in FC$ and let F be a subbag of C with $\min_{m \in MC} |m - F| = d$. Then, for $0 \leq r < d$, $\min_{m \in MC} |m - F - e| \leq r$ if and only if $\min_{m \in MC} |m - s - e| \leq r$ for some $s \in \cup_{1 \leq j \leq |F|} \cup_{d \leq l \leq r+i} A_j^l(F)$.*

Proof The sufficiency is obvious since the bags s in $\cup_{1\le j\le|F|}\cup_{d\le l\le r+i} A_j^l(F)$ are subbags of F and, therefore, $\min_{m\in MC}|m-F-e| \le \min_{m\in MC}|m-s-e| \le r$. We prove next the necessity. If $\min_{m\in MC}|m-F-e| \le r$, there exists $m' \in MC$ with $|m'-F-e| \le r$. Let $n = |(m'-F)\cap e|$ and $j = |m'|-r-n$. Since $e \in E_i - E_i(r)$, $n = |(m'-F)\cap e| \le |e| = i$. Also, recall from Section 1.4 that given bags x, y,

$$|x-y| = |x| - |x\cap y|\,. \tag{5.34}$$

Then, using (5.34) with $x = m'-F$ and $y = e$, $\min_{m\in MC}|m-F| = d$ and $|m'-F-e| \le r$,

$$n = |(m'-F)\cap e| = |m'-F| - |m'-F-e| \ge d-r\,.$$

To summarize,

$$d-r \le n \le i\,. \tag{5.35}$$

Regarding j, we have

$$1 \le j \le |m'\cap F| \le |F|\,. \tag{5.36}$$

The left-hand side of the previous inequality follows using (5.34) with $x = m'$ and $y = e$, and noting that, since $e \in E_i - E_i(r)$, $\min_{m\in MC}|m-e| > r$:

$$\begin{aligned} j &= |m'|-r-n = |m'-e| + |m'\cap e| - r - n \ge |m'-e| + |(m'-F)\cap e| - r - n \\ &> r + |(m'-F)\cap e| - r - n = r+n-r-n = 0\,. \end{aligned}$$

The right-hand side of (5.36) can be shown using (5.34) twice, first with $x = m'-F$ and $y = e$, and next with $x = m'$ and $y = F$, and recalling that $\min_{m\in MC}|m'-F-e| \le r$:

$$\begin{aligned} j &= |m'|-r-n = |m'-F-e| + |m'\cap F| + |(m'-F)\cap e| - r - n \\ &\le r + |m'\cap F| + |(m'-F)\cap e| - r - n \\ &= r + |m'\cap F| + n - r - n = |m'\cap F| \le |F|\,. \end{aligned}$$

Consider any subbag b of $m'\cap F$ of cardinality j. Since (5.36) $j \ge 1$, such a subbag exists and, obviously, it is included in both m' and F. Then, either (1) $b \in A_j^{|m'|-j}(F)$ or (2) b is (strictly) contained in some bag, s, of $\cup_{p=j+1}^{|F|} A_p^{|m'|-j}(F)$. Note that (5.35), $d \le r+n \le r+i$. Then, since $|m'|-j = r+n$,

$$d \le |m'|-j \le r+i\,. \tag{5.37}$$

In case 1, using (5.34) twice, first with $x = m'-b$ and $y = e$ and next with $x = m'$ and $y = b$, and recalling that $b \subset m'$, $b \subset F$, and $|m'| = j+r+n$,

$$\begin{aligned} \min_{m\in MC}|m-b-e| &\le |m'-b-e| = |m'| - |m'\cap b| - |(m'-b)\cap e| \\ &= |m'| - |b| - |(m'-b)\cap e| \le |m'| - |b| - |(m'-F)\cap e| \\ &= j+r+n-j-n = r\,. \end{aligned} \tag{5.38}$$

Therefore (5.36), (5.37), there exists $s = b \in \cup_{1\le j\le|F|}\cup_{d\le l\le r+i} A_j^l(F)$ for which $\min_{m\in MC}|m-s-e| \le r$. In case 2, there exists p, $j+1 \le p \le |F|$ and (5.37), $s \in A_p^{|m'|-j}(F) = \cup_{d\le l\le r+i} A_p^l(F)$ that strictly includes b. But, since $b \subset s$ and (5.38), $\min_{m\in MC}|m-s-e| \le \min_{m\in MC}|m-b-e| \le r$. Therefore, (5.36), (5.37), there exists $s \in \cup_{j+1\le p\le|F|}\cup_{d\le l\le r+i} A_p^l(F) \in \cup_{1\le j\le|F|}\cup_{d\le l\le r+i} A_j^l(F)$ for which $\min_{m\in MC}|m-s-e| \le r$. □

Denote by MC_c the set of minimal cuts of cardinality c and let

$$\Lambda(c,n,i,r) = \begin{cases} \max\limits_{\substack{m\in MC_c \\ b\subset m,|b|=n}} \left\{ \sum\limits_{\substack{e\in E_i-E_i(r) \\ \min_{m'\in MC}|m'-b-e|\le r}} \lambda_{\mathrm{ub}}(e) \right\} & \text{if } MC_c \ne \emptyset \text{ and } 1 \le n \le c \\ 0 & \text{otherwise} \end{cases} \tag{5.39}$$

For $MC_c \ne \emptyset$ and $1 \le n \le c$, $\Lambda(c,n,i,r)$ is the maximum sum of $\lambda_{\mathrm{ub}}(e)$ corresponding to the failure bags $e \in E_i - E_i(r)$ that assuming n components included in some minimal cut of cardinality c failed make the failure distance after e smaller than or equal to r. Note that, for $r \ge L$, $E_i(r) = E_i$, and, therefore, only $\Lambda(c,n,i,r)$, $r < L$ have to be computed. We have the following result.

Theorem 5.6 *Let $E_i^s(r)$, $i \in FC$ be the set of failure bags $e \in E_i - E_i(r)$ for which $\min_{m\in MC}|m-s-e| \le r$. Let F be a subbag of C with $\min_{m\in MC}|m-F| = d$. Then,*

$$\sum_{e\in E_i^F(r)} \lambda_{\mathrm{ub}}(e) \le \sum_{l=d-r}^{i} \max_{1\le j\le|F|} \binom{|F|}{j} \Lambda(j+r+l,j,i,r).$$

Proof Using Theorem 5.5

$$E_i^F(r) = \bigcup_{1\le j\le|F|} \bigcup_{d-r\le l\le i} \bigcup_{s\in A_j^{r+l}(F)} E_i^s(r).$$

Therefore,

$$\sum_{e\in E_i^F(r)} \lambda_{\mathrm{ub}}(e) \le \sum_{j=1}^{|F|} \sum_{l=d-r}^{i} \sum_{s\in A_j^{r+l}(F)} \sum_{e\in E_i^s(r)} \lambda_{\mathrm{ub}}(e).$$

Assume $MC_{j+r+l} \ne \emptyset$. By definition, $\sum_{e\in E_i^s(r)} \lambda_{\mathrm{ub}}(e)$, $s \in A_j^{r+l}(F)$ is upper bounded by $\Lambda(j+r+l,j,i,r)$. In the case $MC_{j+r+l} = \emptyset$, $A_j^{r+l}(F) = \emptyset$, $\Lambda(j+r+l,j,i,r) = 0$ and we

can write $\sum_{s\in A_j^{r+l}(F)}\sum_{e\in E_i^s(r)}\lambda_{\text{ub}}=0\le\sum_{s\in A_j^{r+l}(F)}\Lambda(j+r+l,j,i,r)=0$. Then

$$\sum_{e\in E_i^F(r)}\lambda_{\text{ub}}(e)\le\sum_{j=1}^{|F|}\sum_{l=d-r}^{i}\sum_{s\in A_j^{r+l}(F)}\Lambda(j+r+l,j,i,r)$$
$$=\sum_{j=1}^{|F|}\sum_{l=d-r}^{i}|A_j^{r+l}(F)|\Lambda(j+r+l,j,i,r)=\sum_{l=d-r}^{i}\sum_{j=1}^{|F|}|A_j^{r+l}(F)|\Lambda(j+r+l,j,i,r).$$

But the collection $\{s:s\in A_j^{l+r}(F),1\le j\le|F|\}$ is a Sperner collection on F and, since the subbags $s\in A_j^{r+l}(F)$ have cardinality j, using Lemma 5.4,

$$\sum_{e\in E_i^F(r)}\lambda_{\text{ub}}(e)\le\sum_{l=d-r}^{i}\max_{1\le j\le|F|}\binom{|F|}{j}\Lambda(j+r+l,j,i,r)\,.\square$$

We are now in position to derive the new bounds $F''_{i,r}(k,d)$. Let $x\in U_{k,d}$, $(k,d)\in\mathcal{R}$ and let $(k,d,i,r)\in\mathcal{R}'$, $r<\min\{d,N-k-i\}$. Failure transitions from x associated with failure bags $e\in E_i(r)$ lead to states with failure distance $\min_{m\in MC}|m-F(x)-e|\le\min_{m\in MC}|m-e|\le r$. The failure transitions associated with failure bags in $E_i-E_i(r)$ that lead to states with failure distance $\le r$ are those associated with failure bags in $E_i^{F(x)}(r)$. Then, we have

$$\lambda_{x,\cup_{d'=0}^{r}U_{k+i,d'}}\le\sum_{e\in E_i(r)}\lambda_{\text{ub}}(e)+\sum_{e\in E_i^{F(x)}(r)}\lambda_{\text{ub}}(e),$$

and since $|F(x)|=k$, using Theorem 5.6,

$$\lambda_{x,\cup_{d'=0}^{r}U_{k+i,d'}}\le\sum_{e\in E_i(r)}\lambda_{\text{ub}}(e)+\sum_{l=d-r}^{i}\max_{1\le j\le k}\binom{k}{j}\Lambda(j+r+l,j,i,r)=F''_{i,r}(k,d)\,. \tag{5.40}$$

It is immediate to see that $F'_{i,r}(k,d)$ and $F''_{i,r}(k,d)$, $(k,d,i,r)\in\mathcal{R}'$ are (5.32), (5.40) decreasing on d and, thereby, $F_{i,r}(k,d)=\min\{F'_{i,r}(k,d),F''_{i,r}(k,d)\}$, $(k,d,i,r)\in\mathcal{R}'$ are decreasing on d. Also, for $(k,d,i,d)\in\mathcal{R}'$, we have (5.31) $F_{i,d}(k,d)=\sum_{e\in E_i}\lambda_{\text{ub}}(e)$, independent on d and, therefore, $F_{i,d}(k,d)$, $(k,d,i,d)\in\mathcal{R}'$ are (non-strictly) decreasing on d. Then, the new bounds $F_{i,r}(k,d)$ satisfy the conditions required by Theorem 5.3.

Computation of $\Lambda(c,n,i,r)$ using (5.39) may introduce an unaffordable overhead especially when the number of minimal cuts is large. First, all subbags of cardinality n of each minimal cut must be generated and, for each such subbag b and for each $e\in E_i-E_i(r)\neq\emptyset$, all minimal cuts have to be visited to know whether $\min_{m'\in MC}|m'-b-e|\le r$. Note that if we regard $\min_{m'\in MC}|m'-b|$ as the failure distance from a state s with

$F(s) = b$, then $\min_{m' \in MC} |m' - b - e|$ can be viewed as the failure distance from a state s' reached from s in a single transition associated with failure bag e. Therefore, computation of $\min_{m' \in MC} |m' - b - e|$ can be done efficiently using the algorithms *compute_d*( ) and *compute_all_ad*( ) reviewed in Section 2.2. Besides the previous comment, the main idea to improve the efficiency of the expensive trivial procedure to compute $\Lambda(c, n, i, r)$ is the observation that, typically, many minimal cuts will share the same subbag, especially when $|b|$ is small, and that computation of $\min_{m' \in MC} |m' - b - e| \leq r$ should not be repeated. Recall from Section 2.2 that the control parameter R of algorithm *compute_d*( ) stands for the maximum cardinality of selectors (bags included in some minimal cut) that are stored. Then, we proceed as follows. Let R', $\max_{m \in MC} |m| \geq R' \geq R$. First, we generate and store all distinct subbags of cardinality up to R' included in some minimal cut. Let $\mathcal{B}$ be the set of such subbags. For each $b \in \mathcal{B}$, we also generate and store the list, $l(b)$, of distinct cardinalities of minimal cuts including b. We initialize all $\Lambda(c, n, i, r)$ to 0. Then, for each $b \in \mathcal{B}$, we compute $\delta(b) = \min_{m' \in MC} |m' - b|$ invoking *compute_d*$(b, \delta(b))$, compute $\nu_b(e) = \min_{m' \in MC} |m' - b - e|$, $e \in E$ invoking *compute_all_ad*$(\delta(b), b, \nu_b(e))$, and for each $c \in l(b)$, set (5.39) $\Lambda(c, |b|, i, r) = \max\{\Lambda(c, |b|, i, r), \sum_{e \in A} \lambda_{ub}(e)\}$, $A = \{e \in E_i - E_i(r) : \nu_b(e) \leq r\}$. The remaining updates associated with subbags $b \notin \mathcal{R}$ are done generating and processing for each $m \in MC$ all subbags $b \in m$, $|b| > R'$. Subbags $b \in \mathcal{B}$ with $|b| > R$ are freed once the coefficients $\Lambda(c, n, i, r)$ are computed (*compute_d*( ) requires the knowledge of all subbags b, $|b| \leq R$). Selection of an appropriate value for R' involves a tradeoff between memory consumption and CPU time. As R' gets higher, the algorithm to compute the $\Lambda(c, n, i, r)$ coefficients becomes faster but the memory requirements increase. Memory requirements are only significant when $|MC|$ is large. We will illustrate the tradeoff using an example with 87,031 minimal cuts.

We finish this section by showing how to reduce the effort required to compute the set of bounds $F_{i,r}(k, d)$, $r < \min\{d, N - k - i\}$ once $F'_{i,r}(k, d)$ and $\Lambda(c, n, i, r)$ have been computed. To that end, we need the following result.

Proposition 5.3 *The set of bounds* $F''_{i,r}(k, d)$, $r < \min\{d, N - k - i\}$, $(k, d, i, d) \in \mathcal{R}'$, *are increasing on* k.

Proof Let (k, d, i, r'), $(k + 1, d, i, r') \in \mathcal{R}'$, $r' < \min\{d, N - (k + 1) - i\}$. Then, using

(5.40) and noting that $r' < \min\{d, N-k-i\}$,

$$F''_{i,r'}(k+1,d) = \sum_{e\in E_i(r')} \lambda_{\mathrm{ub}}(e) + \sum_{l=d-r'}^{i} \max\Big\{ \max_{1\le j\le k} \binom{k+1}{j} \Lambda(j+r'+l,j,i,r'),$$

$$\Lambda(k+1+r'+l,k+1,i,r')\Big\}$$

$$= \sum_{e\in E_i(r')} \lambda_{\mathrm{ub}}(e) + \sum_{l=d-r'}^{i} \max\Big\{ \max_{1\le j\le k} \frac{k+1}{k+1-j}\binom{k}{j} \Lambda(j+r'+l,j,i,r'),$$

$$\Lambda(k+1+r'+l,k+1,i,r')\Big\}$$

$$> \sum_{e\in E_i(r')} \lambda_{\mathrm{ub}}(e) + \sum_{l=d-r'}^{i} \max\Big\{ \max_{1\le j\le k} \binom{k}{j} \Lambda(j+r'+l,j,i,r'),$$

$$\Lambda(k+1+r'+l,k+1,i,r')\Big\}$$

$$\ge \sum_{e\in E_i(r')} \lambda_{\mathrm{ub}}(e) + \sum_{l=d-r'}^{i} \max_{1\le j\le k} \binom{k}{j} \Lambda(j+r'+l,j,i,r') = F''_{i,r'}(k,d)\,. \square$$

Let $k \ge \max_{e\in E} Imp(e)$ and assume $F''_{i,r}(k,d) \ge F'_{i,r}(k,d)$ for all (d,i,r). It is immediate to see (5.32) that $F'_{i,r}(k,d) = F'_{i,r}(k',d)$, $k' \ge k$. Then, for $k' \ge k$ we have, using Proposition 5.3,

$$F''_{i,r}(k',d) \ge F''_{i,r}(k,d) \ge F'_{i,r}(k,d) = F'_{i,r}(k',d)\,.$$

Therefore, for $k' \ge k$ we can stop computing $F''_{i,r}(k',d)$ and set $F_{i,r}(k',d) = F'_{i,r}(k',d)$.

5.3 State Space Exploration Algorithm

In the method proposed in this chapter the subset G is enlarged incrementally until the relative unavailability band, $rb = ([UA]_{\mathrm{ub}} - [UA]_{\mathrm{lb}})/[UA]_{\mathrm{lb}}$, is smaller than or equal to the desired one, rb_r, using the state space exploration algorithm *CONT_TG_W* proposed in [20]. In this section we review that algorithm.

The algorithm uses approximate estimates for the unavailability band, $[UA]_{\mathrm{ub}} - [UA]_{\mathrm{lb}}$, and performs the expansions by *waves*. A wave includes a set of consecutive additions of states into G without computing $(\tau(s,Y_G))_{s\in G}$. Let G^* and $\tau^* = (\tau(s,Y_{G^*}))_{s\in G^*}$

Compute $T(k)$, $1 \leq k \leq N$, and $C(k,d)$, $(k,d) \in \mathcal{R}$ using the methods described in [22], Section 5.1, (5.18), and the algorithm of Figure 5.2;
$G = \{o\}$, $T = \emptyset$, $\tau(o, Y_G) = 1/\lambda_o$, $T_G = \tau(o, Y_G)$, $C_G = 0$;
$G^* = G$, $\tau^* = (\tau(s, Y_G))_{s \in G}$, $T_{G^*} = T_G$, $C_{G^*} = C_G$;
Compute $rb = ([UA]_{\text{ub}} - [UA]_{\text{lb}})/[UA]_{\text{lb}}$ using (5.5), (5.6), (5.16), (5.17), (5.1), and (5.2);
while $(rb > rb_r)$ {
  Compute b_c using (5.41);
  $b_t = \max\{BR \times b_c, (rb_r/rb) \times b_c\}$;
  while $(b_c > b_t)$ {
    Select the tuple (s,k,d), $s \in G^*$ with largest $\tau(s, Y_{G^*})\beta_s(k,d)$;
    $b_c = b_c - \tau(s, Y_{G^*})\beta_s(k,d)$;
    Let $S(s)$ be the set of states in $U_{k,d}$ reached from s in a single transition;
    for (each $s' \in S(s)$) {
      Add to T all transitions from s' to G;
      Add to T all transitions from $s'' \in G$ to s' updating b_c if necessary;
      $G = G \cup \{s'\}$;
    }
  }
  Compute $(\tau(s, Y_G))_{s \in G}$;
  Compute C_G and T_G using (5.4) and (5.3);
  Compute $rb = ([UA]_{\text{ub}} - [UA]_{\text{lb}})/[UA]_{\text{lb}}$ using (5.5), (5.6), (5.16), (5.17), (5.1), and (5.2);
  $G^* = G$, $\tau^* = (\tau(s, Y_G))_{s \in G}$, $T_{G^*} = T_G$, $C_{G^*} = C_G$;
}

Figure 5.3: Algorithm *CONT_TG_W*.

be, respectively, the subset G and the corresponding mean times to absorption vector at the end of the last wave. The unavailability band is approximated in terms of a sum, b_c, of contributions associated with transition groups (s,k,d), $s \in G^*$, $(k,d) \in \mathcal{R}$. Each transition group accounts for the transitions from state s to states $s' \notin G$ with k failed components and failure distance d:

$$b_c = \sum_{\substack{s \in G^* \\ (k,d) \in \mathcal{R}}} \tau(s, Y_{G^*})\, \beta_s(k,d)\,, \tag{5.41}$$

with

$$\beta_s(k,d) = \lambda_{s,U_{k,d}} \Big[\frac{C_{G^*}}{T_{G^*}^2} T(k) + \frac{T_{G^*} - C_{G^*}}{T_{G^*}^2} C(k,d) \Big]. \tag{5.42}$$

Initially, $G = G^* = \{o\}$, $\tau^* = [1/\lambda_o]$, $T_{G^*} = 1/\lambda_o$ and $C_{G^*} = 0$. The next wave starts by choosing the tuple (s,k,d), $s \in G^*$ with largest $\tau(s, Y_{G^*})\beta_s(k,d)$. Next, we add into G the states in $U_{k,d}$ reached from s in a single transition and update $\lambda_{s,U_{k,d}}$, $s \in G$ and b_c accordingly. This procedure continues until b_c becomes small enough. In that point, we set $G^* = G$, compute τ^*, T_{G^*} and C_{G^*} and, if the new relative unavailability band is still larger than the desired one, continue with a new wave. We give in Figure 5.3 a description of the algorithm. The description is done in terms of the subset T of transitions included in the generated portion of X as well as the equations derived so far. The control parameter BR, $0 \leq BR < 1$ allows to tradeoff the number of times $(\tau(s, Y_G))_{s \in G}$ is computed against how accurately the state space is explored (the larger BR, the more accurate but the more costly the exploration). After performing some experiments we have chosen $BR = 0.1$.

5.4 Analysis and Comparison

In this section we analyze the performance of the bounding method using two examples and compare it with that of the bounding methods described in [22] and [70]. For the method proposed in [22] we use the state space exploration algorithm *CONT_TG_W*. The lower bound in the method described in [70] is the same as the one proposed in this chapter and the upper bound is

$$[UA]'_{\text{ub}} = \frac{C_G + [T_U]_{\text{ub}}}{T_G + [T_U]_{\text{ub}}}.$$

The state space exploration algorithm we use with that method is analogous to the algorithm *CONT_TG_W*. The unavailability band, $b' = [UA]'_{\text{ub}} - [UA]_{\text{lb}}$, is approximated

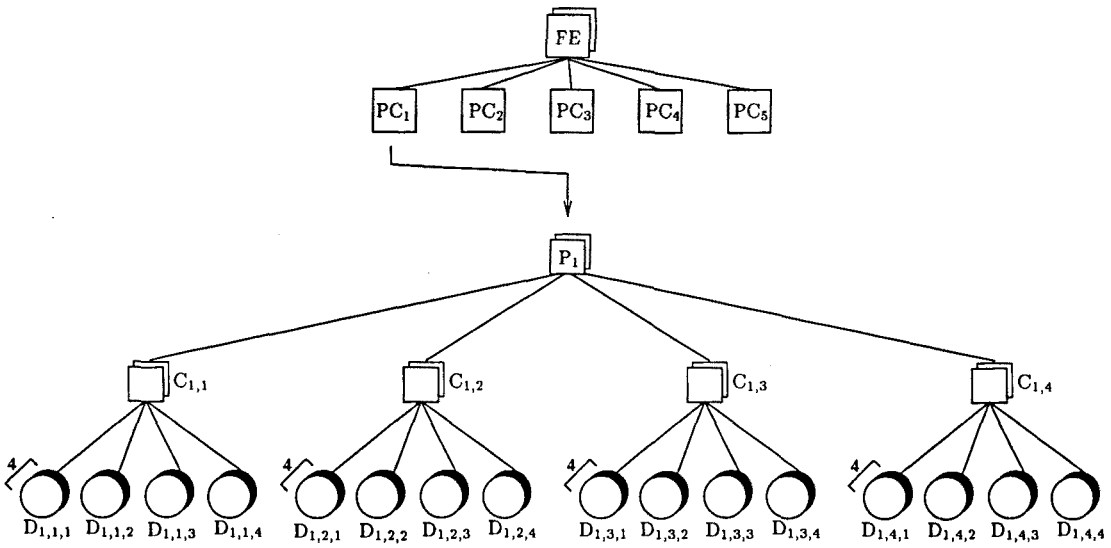

Figure 5.4: Block diagram of the fault-tolerant database example.

as a sum, b'_c, of contributions associated with transition groups (s,k), $s \in G^*$, $1 \leq k \leq N$. Each transition group accounts for the transitions from s to states $s' \notin G$ with k failed components:

$$b'_c = \sum_{\substack{s \in G^* \\ 1 \leq k \leq N}} \tau^*(s, Y_{G^*})\, \gamma_s(k)\,,$$

with

$$\gamma_s(k) = \lambda_{s,U_k} \frac{T(k)}{T_{G^*}}\,.$$

Within a wave we choose the pairs (s,k), $s \in G^*$ with largest $\tau^*(s, Y_{G^*})\gamma_s(k)$, and add to G all states in U_k reached from s in a single transition.

The results have been obtained using a 128 MB UltraSparc 1 workstation. For both examples the control parameter R to compute failure distances has been set to 2 and the total memory consumption has been limited to 100 MB.

The first example is the fault-tolerant database system whose block diagram is sketched in Figure 5.4. The system includes two front-ends FE and five processing clusters. Processing cluster PC_i, $1 \leq i \leq 5$ consists of two processing units P_i, four controller sets $\mathrm{C}_{i,j}$, $j \leq 4$ with two controllers per set and sixteen disk clusters $\mathrm{D}_{i,j,k}$, $1 \leq k \leq 4$ with four disks per cluster. Each controller set controls four disk clusters. The system has 372 components. The system is operational if at least one front-end is unfailed and all processing clusters are operational. A processing cluster is operational if at least one

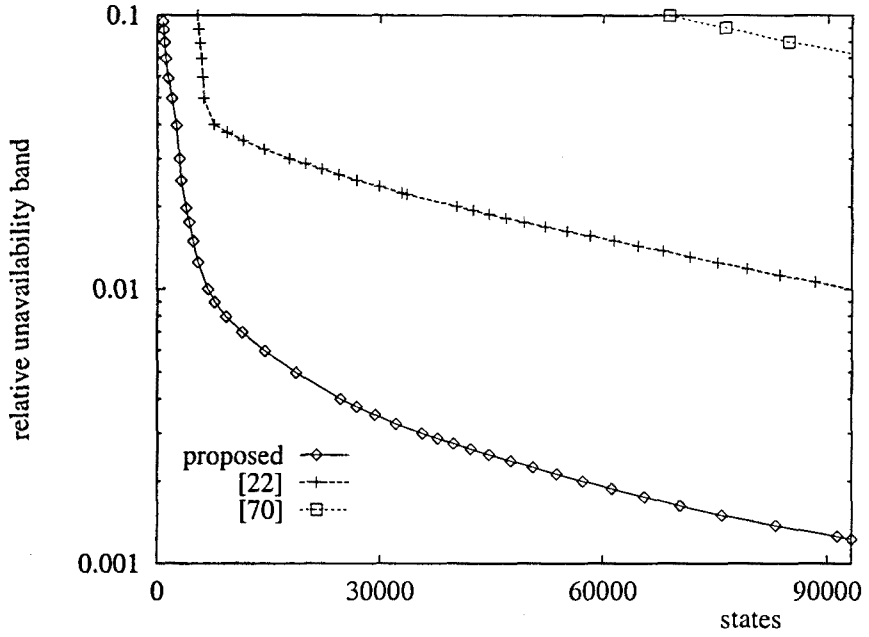

Figure 5.5: Relative unavailability band for the fault-tolerant database example as a function of the size of the generated subset.

processing unit is operational, at least one controller of each set is unfailed, and at least three disks of each cluster are unfailed. Front-ends fail with rate $1/20{,}000\,\mathrm{h}^{-1}$. Processing units P_i fail with rate $1/40{,}000 + 5 \times 10^{-6}(i-1)\ \mathrm{h}^{-1}$. Controllers of set $\mathrm{C}_{i,j}$ fail with rate $1/100{,}000 + 2 \times 10^{-6}(i-1) + 5 \times 10^{-7}(j-1)\ \mathrm{h}^{-1}$. Finally, disks of cluster $\mathrm{D}_{i,j,k}$ fail with rate $1/200{,}000 + 2 \times 10^{-6}(i-1) + 2.5 \times 10^{-7}(j-1) + 6.25 \times 10^{-8}(k-1)\ \mathrm{h}^{-1}$. When both front-ends are unfailed, the failure of one of them is propagated to the other with probability 0.01. Similarly, when both processing units of a processing cluster are operational, failure of a processing unit is propagated to the other unfailed processing unit with probability 0.02. Components continue to fail when the system has failed. Repair rates are $1/8\,\mathrm{h}^{-1}$ for front-ends and processing units and $1/12\,\mathrm{h}^{-1}$ for controllers and disks. There is a single repair person who gives priority first to front-ends, next to processing units, next to controllers, and next to disks. Failed components with the same priority are taken at random for repair. For this example $L = 2$ and the structure function has 106 minimal cuts, all of cardinality 2.

In Figure 5.5 we give the relative unavailability band as a function of the number of states in the generated subset G. It can be seen that the bounding method developed in this chapter outperforms significantly the methods described in [70] and [22]. Thus, the number of states required by the method described in [70] to achieve a given relative unavailability band ranges from 83 to 92 times the number of states required by the

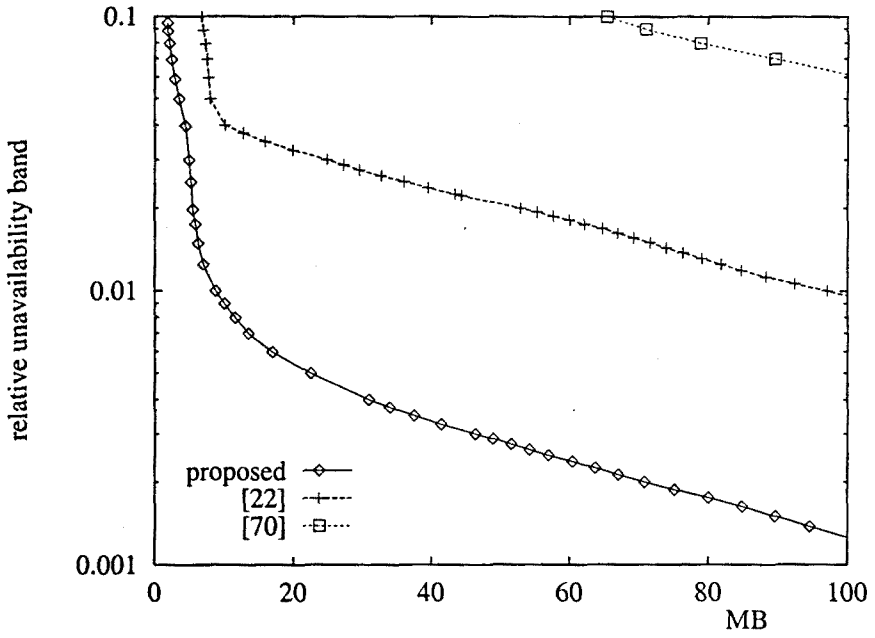

Figure 5.6: Relative unavailability band for the fault-tolerant database example as a function of the memory consumption in MB.

proposed bounding method. Regarding the bounding method described in [22], the proposed method requires a number of states between 2.6 and 13.7 times smaller.

Since the memory requirements for the same number of generated states of the three methods are different, we also compare them in terms of memory consumption. The comparison is done in Figure 5.6. Again, the proposed method is far better than the methods described in [70] and [22]. The amount of memory required by the method described in [70] to achieve a given relative unavailability band ranges from 35 to 37 times the amount of memory required by the proposed method. Note that the improvement in terms of memory consumption is smaller than it was in terms of size of G. This can be explained as follows. The proposed method has to hold, for each state in the frontier of G, lists of contributions to the unavailability band associated with the parameters k and d, while the lists of contributions that have to be held in the method described in [70] are associated only with the parameter k. With regard to the method described in [22], the improvement factor in terms of memory consumption is of the same order as it was in terms of size of the generated subset, ranging from 2.3 to 11.

According to Theorem 5.4, the improvement that the bounding method described in this chapter may achieve in relation to the one proposed in [22] is due to the reduction in the $F_{i,r}(k,d)$ bounds and, consequently, in the $C(k,d)$ bounds. It is difficult to predict

Table 5.1: First values of the bounds $C(k,d)$ for the fault-tolerant database example.

		$C(k,d)$	
k	d	proposed	[22]
1	2	7.382×10^{-3}	9.418×10^{-2}
1	1	2.299×10^{-2}	0.6174
2	2	2.413×10^{-2}	0.2812
2	1	0.1005	1.881
2	0	13.23	13.23

up to which extend the new bounds $F_{i,r}(k,d)$ will be smaller than the ones given by (5.32), since the reduction depends on the structure of the minimal cuts of the system. Apart from the $F_{i,r}(k,d)$ bounds being smaller, the proposed method requires the down states in the frontier of G to be sparse for the method to be significantly better than the one described in [22]. This is so because although not mentioned in Section 5.3, the bounds $C(k,0)$ are the same for both methods and, therefore (5.16), the upper bound $[C_U]_{\text{ub}}$ will be appreciably reduced only if the outgoing transitions of G to down states (states with $d = 0$) are relatively rare. In Table 5.1 we show the first values of the bounds $C(k,d)$ obtained with the proposed method and the method described in [22]. The bounds $C(k,0)$ are the same but the bounds $C(k,d)$, $d > 0$ are significantly smaller. Since in this example most of the exits from G are made through states with non-zero failure distance, the bounds $[C_U]_{\text{ub}}$ and (5.2) $[UA]_{\text{ub}}$ obtained with the proposed method are, for a given subset G, appreciably smaller than the corresponding bounds obtained with the method described in [22].

The second example, whose architecture is depicted in Figure 5.7, includes five processing clusters that communicate through two independent double-ring networks A and B. Processing cluster i, $0 \leq i \leq 4$, includes three identical processing units PU_i. Network A consists of ten nodes NA_i, $0 \leq i \leq 9$, and direct (clockwise) and reverse (counterclockwise) links, DA_i and RA_i, respectively, linking nodes NA_i and $\text{NA}_{i+1 \bmod 10}$. Network B has the same structure as network A and its direct and reverse links are called, respectively, DB_i and RB_i. The system has 78 components. The system is operational if each processing cluster has at least an unfailed processing unit and all processing clusters can communicate using one of the networks. The operational configuration of the system includes two processing units for the processing clusters with two or three unfailed pro-

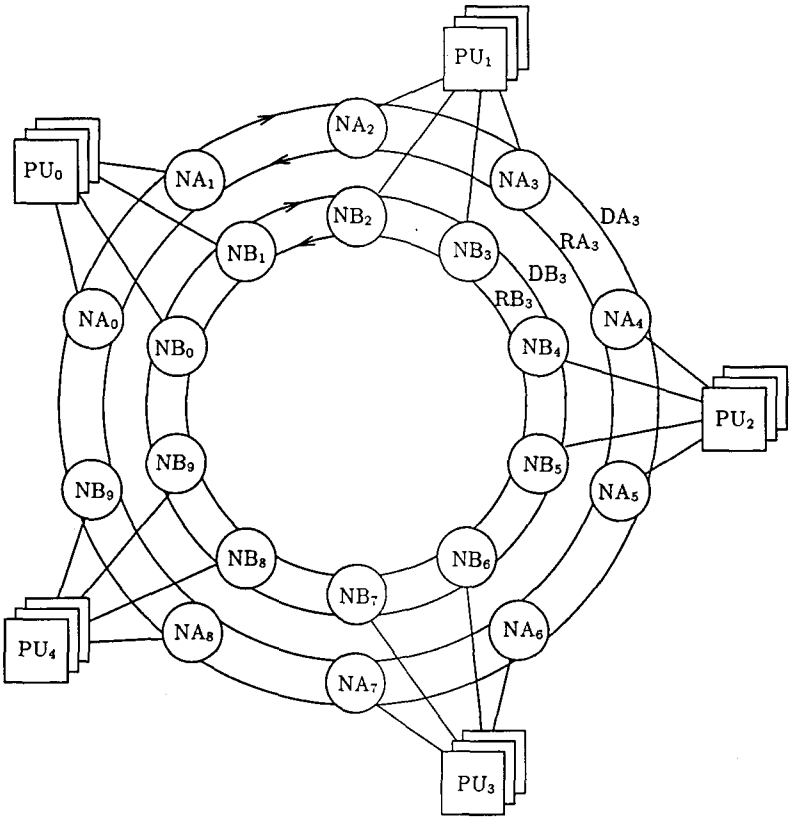

Figure 5.7: Block diagram of the second example.

cessing units, one processing unit for the processing clusters with one unfailed processing unit, and the components of either network A or B, with priority given to network A, required to build one of the operational configurations of the networks described next. The network configuration that is tried first is a direct ring including all nodes and direct links. The second configuration that is tried is a reverse ring including all nodes and reverse rings. The third configuration is used when parallel direct and inverse link i fail and it includes all nodes and links except links between nodes i and $i+1$ mod 10. The last configuration is used when node i fails and it includes all nodes except node i and all links except those between node i and nodes $i \pm 1$ mod 10. A fault in a processing unit of a cluster contaminates another unfailed unit in the same cluster with probability 0.05. The components included in the operational configuration of the system are called active. Active processing units, active nodes and active links fail with rates 4.6×10^{-4} h^{-1}, 2.3×10^{-4} h^{-1} and 1.1×10^{-4} h^{-1}. Inactive components fail with the same rates multiplied by a dormancy factor of 0.2. We assume that there is a single repair person who takes for repair failed components at random. Repair rates for processing units, nodes and links are, respectively, 0.5 h^{-1}, 0.7 h^{-1} and 1.0 h^{-1}. Components continue to fail

Table 5.2: Memory overhead in MB due to storing all distinct subbags of cardinality $> R$ and $\leq R'$ included in some minimal cut and CPU time in seconds required to compute the $\Lambda(c, n, i, r)$ coefficients for the second example as a function of R' .

R'	2	3	4	5	6
overhead (MB)	0	3.0	10	11	11
CPU time (s)	3,611	1,906	1,216	1,197	1,197

when the system has failed. For this example $L = 3$ and the structure function has 87,031 minimal cuts, 6 of cardinality 3, 75,625 of cardinality 4, 11,000 of cardinality 5, and 400 of cardinality 6.

In this example the number of minimal cuts, and, therefore, the number of subbags that would have to be processed to compute the $\Lambda(c, n, i, r)$ coefficients using the trivial procedure is quite large. We start by illustrating the tradeoff involved by the selection of an appropriate value for R'. In Table 5.2 we show, for $R' = 2$, 3, 4, 5 and 6 the memory overhead in MB due to storing all distinct subbags of cardinality $> R$ and $\leq R'$ included in some minimal cut, and the CPU time in seconds required to compute the $\Lambda(c, n, i, r)$ coefficients. Note that, as it was anticipated in Section 5.2, the larger R' the more faster and memory consuming the computation of the coefficients. Note also that beyond $R' = 4$ no significant improvement is achieved. This is mainly due to the fact that subbags with small cardinalities tend to be shared by several minimal cuts much more often than subbags with larger cardinalities do. Thus, for instance, among the 420,506 subbags of cardinality 3, only 18,946 are really distinct, while 12,600 subbags of cardinality 5 out of 13,400 are distinct.

In Figure 5.8 we give the relative unavailability band as a function of the number of states in the generated subset G. The results have been obtained with $R' = 6$. For this example, the reduction in the size of G achieved by the proposed method with regard to the method described in [70] is still appreciable. However, that reduction is more modest when the proposed method is compared with the method described in [22]. We have analyzed the bounds $F'_{i,r}(k, d)$ and $F''_{i,r}(k, d)$ and found that because of the particular structure of the minimal cuts of the example, the bounds $F''_{i,r}(k, d)$ are not much smaller than the bounds $F'_{i,r}(k, d)$ and, consequently, the reduction in the $C(k, d)$ bounds is more modest than it was for the previous example. The relative unavailability band as a function of the memory consumption in MB is given in Figure 5.9. Note, again, that the method described in this chapter and the one proposed in [22] compare the same in

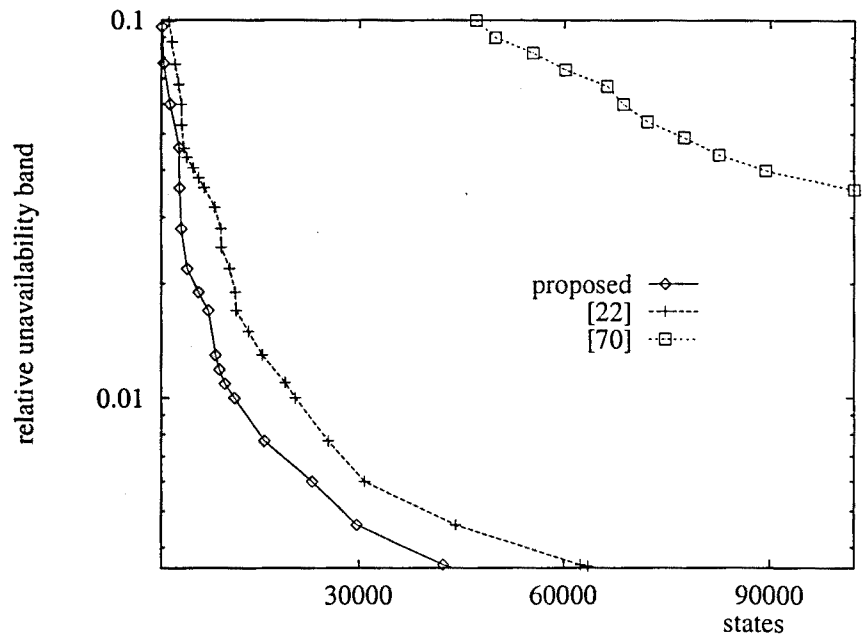

Figure 5.8: Relative unavailability band for the second example as a function of the size of the generated subset.

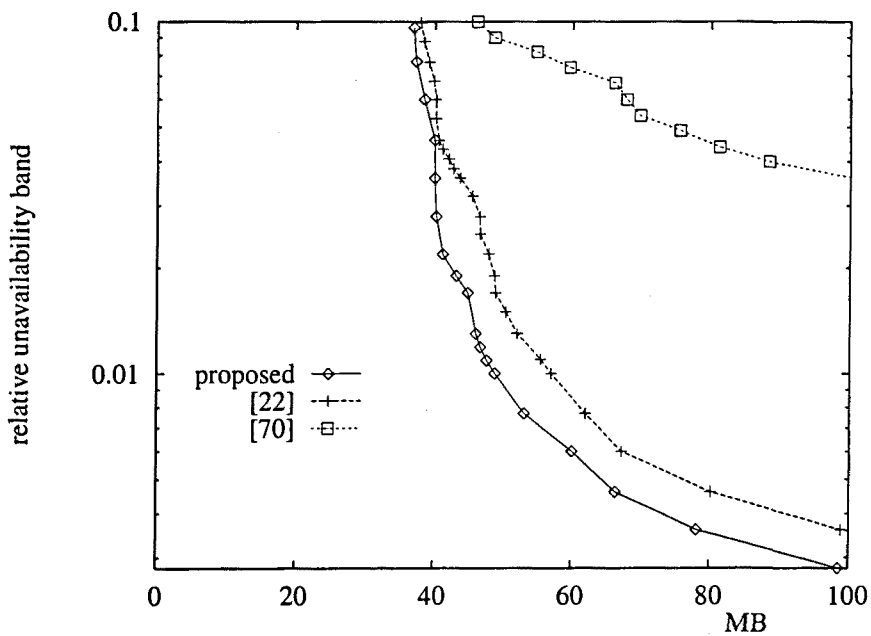

Figure 5.9: Relative unavailability band for the second example as a function of the memory consumption in MB.

terms of memory consumption as they did in terms of size of the generated subset, while the improvement in terms of memory consumption achieved by the proposed method with respect to the one of [70] is appreciably smaller than it was in terms of size of G, especially for medium and large values of the relative unavailability band. For this example the reduction in the improvement is due not only to the different size of the lists of contributions both methods have to bookkeep for each state in the frontier of G, but also to the amount of memory required to store the set of minimal cuts and related structures for the computation of the failure distances. For this example, $|MC| = 87{,}031$ and its storage takes about 35 MB.

5.5 Conclusions

In this chapter we have developed a bounding method for the steady-state unavailability that exploits the failure distance concept and requires the knowledge of the minimal cuts of the system. From a memory consumption point of view, the method seems to be better than the method described in [70], which is not based on the failure distance concept. The proposed method may also outperform significantly a previous method [22] based on the failure distance concept. It is difficult, however, to predict the improvement in advance since it depends on the sparseness of the down states in the frontier of G and the structure of the minimal cuts of the fault tree of the system. The method suffers from a memory overhead due to holding the minimal cuts and subbags required for the computation of failure distances and $\Lambda(c, n, i, r)$ coefficients, which, as it has been shown by means of the second example analyzed in this chapter, can be significant if the number of minimal cuts is large. At this point one might ask whether it would not be possible to devise another bounding method for the steady-state unavailability based on lower bounds for failure distances, similarly as we did for the first method presented in this dissertation. The next chapter is devoted to give an affirmative answer to that question.

Chapter 6

Availability Bounds of Repairable Systems using FD Bounds

In this chapter we will develop a method to compute bounds for the steady-state unavailability, UA, based on the lower bounds for failure distances developed in Section 4.2. The motivation is similar to the one that brought us to develop the method to compute bounds for the unreliability described in Chapter 4: we want to have a bounding method that does not rely on solving a hard problem (the determination of the minimal cuts) and, secondly, it is desirable to be able to bound UA without incurring the memory overhead due to storing minimal cuts and related structures, especially when the number of minimal cuts is large. The first section of this chapter will be devoted to describe the method and prove its correctness. Next, we will derive suitable failure rate bounding structures based on lower bounds for failure distances. Finally, we will analyze the method by means of several examples and compare it with the method described in the previous chapter and the methods described in [22, 70].

6.1 Bounding Method

Let $X = \{X(t); t \geq 0\}$ be the finite CTMC modeling the system and let Ω be its state space. The method computes bounds for UA using detailed knowledge of X in the generated subset, G, and bounding the behavior of X in $U = \Omega - G$. The method uses the state cloning technique proposed in [70]. The technique consists in modifying X by adding to U clones of the states in G with more than F failed components, accounting for the visits to the corresponding states of G after X exits G and before the number of

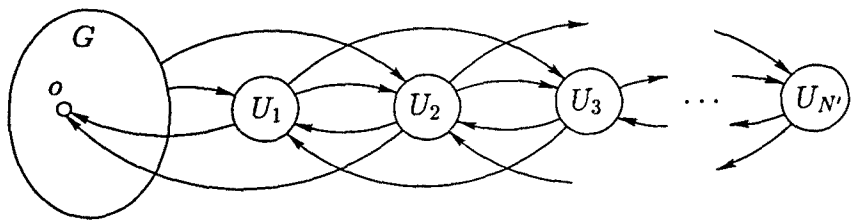

Figure 6.1: State transition diagram of the CTMC X after applying the state cloning technique with $F = 0$.

failed components has fallen below $F+1$. Note that the application of the state cloning technique does not modify the steady-state unavailability UA. We apply the technique with $F = 0$ in order to ease state space exploration. The modified X has the structure depicted in Figure 6.1, where U_k includes all states in U with exactly k failed components and $N' \leq N$, being N the number of components of the system. In the following, X will denote the modified X.

The notation we will use throughout this chapter is essentially the same we used in Chapter 5. Thus, we will denote by $\lambda_{s,s'}$, $s, s' \in \Omega$, the transition rate from state s to state s', by $\lambda_s = \sum_{\substack{s' \in \Omega \\ s \neq s'}} \lambda_{s,s'}$, $s \in \Omega$, the output rate of s, and by $\lambda_{s,C} = \sum_{s' \in C} \lambda_{s,s'}$, $s \in \Omega$, $C \subset \Omega$, the transition rate from s to the subset of states C, all referred to X unless otherwise stated. We will also consider several transient CTMC Y. Each such Y has state space $B \cup \{a\}$, where all states in B are transient and a is an absorbing state, and has defined initial probability distribution with $P[Y(0) \in B] = 1$. $\tau(s, Y)$, $s \in B$, will denote the mean time spent by Y in s before absorption, and $\tau(C, Y) = \sum_{s \in C} \tau(s, Y)$, $C \subset B$, will denote the mean time to absorption in subset C. Finally, recall that $\tau(s, Y)\lambda_{s,s'}$ is the expected number of times that a transition from s to s', $s \in B$, $s' \in B \cup \{a\}$, is followed.

Consider the regenerative behavior of X, taking as regeneration points the times at which X enters o from U. Let T_G and T_U be the contributions of G and U to the mean time between regenerations of X, and let C_G and C_U be the respective contributions to the mean down time. From regeneration process theory (see, for instance [31]), we have:

$$UA = \frac{C_G + C_U}{T_G + T_U}.$$

Assume that upper bounds $[T_U]_{\text{ub}}$ and $[C_U]_{\text{ub}}$ for, respectively, T_U and C_U are known.

Then [22, Theorem 2]

$$[UA]_{\text{lb}} = \frac{C_G}{T_G + [T_U]_{\text{ub}}}, \tag{6.1}$$

$$[UA]_{\text{ub}} = \frac{C_G + [C_U]_{\text{ub}}}{T_G + [C_U]_{\text{ub}}}, \tag{6.2}$$

are, respectively, a lower and an upper bound for UA.

The quantities T_G and C_G are computed as in Section 5.1 from the transient CTMC Y_G as

$$T_G = \sum_{s \in G} \tau(s, Y_G), \tag{6.3}$$

$$C_G = \sum_{s \in G \cap D} \tau(s, Y_G), \tag{6.4}$$

where D is the subset of down states of X.

In the following, we will denote by FC and RC the set of distinct cardinalities of, respectively, failure and repair bags of the system, and by E_i the set of failure bags with cardinality i.

The bound $[T_U]_{\text{ub}}$ and, therefore , the lower bound $[UA]_{\text{lb}}$ are the same as those of the method developed in Chapter 5. Since the transient CTMC Y^{u_k} introduced in Section 5.1 to compute $[UA]_{\text{lb}}$ will be referred to later on in this section, let us recall its definition. The chain has state space $\{u_1, \ldots, u_N\} \cup \{a\}$, initial state u_k and state transition diagram like the one shown in Figure 5.1, b: there is a transition with rate $g(1)$ from u_1 to a, a transition with rate $g(k)$, $2 \leq k \leq N$ from u_k to u_{k-1}, and, for each $i \in FC$, $i \leq N - k$, a transition with rate $f_i = \sum_{e \in E_i} \lambda_{\text{ub}}(e)$ from u_k to u_{k+i}.

The upper bound $[C_U]_{\text{ub}}$ is based on the lower bounds for failure distances derived in Section 4.2. Let $s \in \Omega$ and let $\widetilde{d}(s)$ be a lower bound for the failure distance from s, $d(s)$. Since theorems 4.2 and 4.4 we have:

$$0 \leq \widetilde{d}(s) \leq d(s), \tag{6.5}$$

$$\widetilde{d}(s) = 0 \text{ if and only if } s \in D, \tag{6.6}$$

$$\widetilde{d}(s) - |F(s') - F(s)| \leq \widetilde{d}(s') \leq \widetilde{d}(s), \quad F(s) \subset F(s'). \tag{6.7}$$

Let $s' \in \Omega$ be a state reached from s in a single-step transition associated with failure bag $e \in E$ and let $s'' \in \Omega$ be a state reached from s in a repair transition involving i components. Then, (6.7) implies

$$\widetilde{d}(s) - |e| \leq \widetilde{d}(s') \leq \widetilde{d}(s), \tag{6.8}$$

and

$$\widetilde{d}(s) \leq \widetilde{d}(s'') \leq \widetilde{d}(s) + i\,. \tag{6.9}$$

Let $\widetilde{U}_{k,d}$ be the subset of U including all states s with k failed components and $\widetilde{d}(s) = d$, and let $\widetilde{L} = \widetilde{d}(o)$. The set $\widetilde{\mathcal{R}}$ of (k,d) pairs for which $\widetilde{U}_{k,d}$ might be $\neq \emptyset$ is given by the constraints

$$\begin{aligned} &1 \leq k \leq N, \\ &\max\{0, \widetilde{L} - k\} \leq d \leq \min\{\widetilde{L}, N - k\}\,. \end{aligned}$$

The constraints on k are obvious. The constraints $\widetilde{L} - k \leq d$ and $d \leq \widetilde{L}$ follow from (6.7) and the definition of $\widetilde{L}$; $0 \leq d$ follows from (6.5). Finally, $d \leq N - k$ follows from (6.7) and the fact that the structure function of the system is coherent by taking s' the state with all components failed and noting that $\widetilde{d}(s') \leq d(s') = 0$ and, therefore, $\widetilde{d}(s') = 0$.

Let Y_U^s, $s \in U$ be the transient CTMC with state space $U^s \cup \{a\}$, U^s including the states reachable from s before exit from U, and initial state s, built from X by directing to a the transitions from states in U to o. Let C_U^s be the mean down time to absorption of Y_U^s. Recalling that $\sum_{s' \in G} \tau(s', Y_G)\lambda_{s',s}$, $s \in U$, is the probability that X enters U through s, we have

$$C_U = \sum_{s' \in G} \sum_{s \in U} \tau(s', Y_G)\lambda_{s',s}\, C_U^s = \sum_{s' \in G} \sum_{(k,d) \in \widetilde{\mathcal{R}}} \sum_{s \in \widetilde{U}_{k,d}} \tau(s', Y_G)\lambda_{s',s}\, C_U^s\,. \tag{6.10}$$

Let $\widetilde{C}(k,d)$ be upper bounds for C_U^s, $s \in \widetilde{U}_{k,d}$, and

$$\widetilde{\pi}_{k,d} = \sum_{s \in G} \tau(s, Y_G)\lambda_{s,\widetilde{U}_{k,d}}\,. \tag{6.11}$$

Let

$$[C_U]_{\text{ub}} = \sum_{(k,d) \in \widetilde{\mathcal{R}}} \widetilde{\pi}_{k,d}\, \widetilde{C}(k,d)\,. \tag{6.12}$$

We have the following result.

Theorem 6.1 *Assume* $C_U^s \leq \widetilde{C}(k,d)$, $s \in \widetilde{U}_{k,d}$, *Then,* $C_U \leq [C_U]_{\text{ub}}$.

Proof Using (6.10), the fact that $C_U^s \leq \widetilde{C}(k,d)$, $s \in \widetilde{U}_{k,d}$, (6.11), and (6.12):

$$\begin{aligned} C_U &= \sum_{s' \in G} \sum_{(k,d) \in \widetilde{\mathcal{R}}} \sum_{s \in \widetilde{U}_{k,d}} \tau(s', Y_G) \lambda_{s',s} C_U^s \leq \sum_{s' \in G} \sum_{(k,d) \in \widetilde{\mathcal{R}}} \sum_{s \in \widetilde{U}_{k,d}} \tau(s', Y_G) \lambda_{s',s} \widetilde{C}(k,d) \\ &= \sum_{s' \in G} \sum_{(k,d) \in \widetilde{\mathcal{R}}} \tau(s', Y_G) \lambda_{s', \widetilde{U}_{k,d}} \widetilde{C}(k,d) = \sum_{(k,d) \in \widetilde{\mathcal{R}}} \sum_{s' \in G} \tau(s', Y_G) \lambda_{s', \widetilde{U}_{k,d}} \widetilde{C}(k,d) \\ &= \sum_{(k,d) \in \widetilde{\mathcal{R}}} \widetilde{\pi}_{k,d} \widetilde{C}(k,d) = [C_U]_{\text{ub}} . \square \end{aligned}$$

Let L be the exact failure distance from state o, i.e. $L = d(o)$, and let

$$\widetilde{C}(k) = \sum_{i=\widetilde{L}}^{N} \tau(u_i, Y^{u_k}) . \tag{6.13}$$

Theorem 6.2 $C_U^s \leq \widetilde{C}(k)$, $s \in U_k$.

Proof Since (6.5), $\widetilde{L} \leq L$. Using that [22, Theorem 6] $C_U^s \leq \sum_{i=L}^{N} \tau(u_i, Y^{u_k})$ and (6.13):

$$C_U^s \leq \sum_{i=L}^{N} \tau(u_i, Y^{u_k}) \leq \sum_{i=\widetilde{L}}^{N} \tau(u_i, Y^{u_k}) = \widetilde{C}(k). \square$$

The bounds $\widetilde{C}(k,d)$ are computed using an iterative procedure that starts with $\widetilde{C}(k,d) = \widetilde{C}(k)$ and improves the bounds using potentially better bounds $\widetilde{C}'(k,d)$ until no significant improvement is achieved.

Let $s \in \widetilde{U}_{k,d}$ and consider a transition from s to $s' \in U$ associated with failure bag $e \in E_i$, $i \in FC$. Clearly, $s' \in \widetilde{U}_{k+i,d'}$ for suitable d' values. Imposing $(k+i, d') \in \widetilde{\mathcal{R}}$ yields $i \leq N - k$ and $d' \leq \min\{\widetilde{L}, N - k - i\}$. Moreover (6.5), (6.8), $\max\{0, d - i\} \leq d' \leq d$. Therefore, the only feasible destination subsets $\widetilde{U}_{k+i,d'}$, $i \in FC$, are those satisfying $i \leq N - k$ and (recall that $d \leq \widetilde{L}$) $\max\{0, d - i\} \leq d' \leq \min\{d, N - k - i\}$. Let $\widetilde{\mathcal{R}}' = \{(k,d,i,d') | \, (k,d) \in \widetilde{\mathcal{R}}, \, i \in FC, \, \max\{0, d-i\} \leq d' \leq \min\{d, N-k-i\}\}$. Assume that upper bounds $\widetilde{F}_{i,r}(k,d)$, $(k,d,i,r) \in \widetilde{\mathcal{R}}'$ for $\sum_{d'=0}^{r} \lambda_{s, \widetilde{U}_{k+i,d'}}$, $s \in \widetilde{U}_{k,d}$, are available and let

$$\widetilde{f}_{i,j}(k,d) = \begin{cases} \widetilde{F}_{i,d-j}(k,d) - \widetilde{F}_{i,d-j-1}(k,d) , & J_m(k,d,i) \leq j < J_M(d,i) \\ \widetilde{F}_{i,d-j}(k,d) , & j = J_M(d,i) , \end{cases} \tag{6.14}$$

where $J_m(k,d,i) = \max\{0, k+d+i-N\}$ and $J_M(d,i) = \min\{i,d\}$. The upper bounds $\widetilde{C}'(k,d)$ are computed using

$$\widetilde{C}'(k,d) = \frac{I_{d=0}}{g(k)} + I_{k>1}\left[I_{d>\widetilde{L}-k}\,\widetilde{C}(k-1,d) + I_{d\leq\widetilde{L}-k}\,\widetilde{C}(k-1,d+1)\right] + \frac{1}{g(k)}\sum_{\substack{i\in FC\\ i\leq N-k}}\sum_{j=J_m(k,d,i)}^{J_M(d,i)}\widetilde{f}_{i,j}(k,d)\,\widetilde{C}(k+i,d-j)\,, \tag{6.15}$$

where I_c is the indicator function returning 1 if c is true and 0 otherwise. The algorithm to compute the $\widetilde{C}(k,d)$ bounds is exactly the same as the one given in Figure 5.2 replacing L, $\mathcal{R}$, $C(k)$, $C(k,d)$, and $C'(k,d)$ by, respectively, $\widetilde{L}$, $\widetilde{\mathcal{R}}$, $\widetilde{C}(k)$, $\widetilde{C}(k,d)$, and $\widetilde{C}'(k,d)$.

Next, we prove $C_U^s \leq \widetilde{C}(k,d)$, $s \in \widetilde{U}_{k,d}$, $(k,d) \in \widetilde{\mathcal{R}}$ provided that $\widetilde{F}_{i,r}(k,d)$, $(k,d,i,r) \in \widetilde{\mathcal{R}}'$, and $\widetilde{F}_{i,}(k,d)$, $(k,d,i,d) \in \widetilde{\mathcal{R}}'$ are decreasing on d. The proof consists of a sequence of two lemmas, three propositions and a theorem.

We start with two technical results. The first one has been adapted from [22, Proposition 2] and is included here for the sake of completeness. The second result is very similar to Lemma 5.2.

Lemma 6.1 *Assume that* $\widetilde{C}(k,d)$, $(k,d) \in \widetilde{\mathcal{R}}$, $\widetilde{F}_{i,r}(k,d)$, $(k,d,i,r) \in \widetilde{\mathcal{R}}'$ *and* $\widetilde{F}_{i,d}(k,d)$, $(k,d,i,d) \in \widetilde{\mathcal{R}}'$ *are decreasing on* d. *Then,*

$$\widetilde{A}(k,d,i) = \sum_{j=J_m(k,d,i)}^{J_M(d,i)}\widetilde{f}_{i,j}(k,d)\,\widetilde{C}(k+i,d-j)\,, \quad i \in FC\,, i \leq N-k\,,$$

is decreasing on d.

Proof Let (k,d), $(k,d+1) \in \widetilde{\mathcal{R}}$. Using (6.14) and making the index change $r = d-j$,

$$\begin{aligned}
\widetilde{A}(k,d,i) &= \sum_{j=J_m(k,d,i)}^{J_M(d,i)-1}\left[\widetilde{F}_{i,d-j}(k,d) - \widetilde{F}_{i,d-j-1}(k,d)\right]\widetilde{C}(k+i,d-j) \\
&\quad + \widetilde{F}_{i,d-J_M(d,i)}(k,d)\widetilde{C}(k+i,d-J_M(d,i)) \\
&= \sum_{j=J_m(k,d,i)+1}^{J_M(d,i)}\widetilde{F}_{i,d-j}(k,d)\left[\widetilde{C}(k+i,d-j) - \widetilde{C}(k+i,d-j+1)\right] \\
&\quad + \widetilde{F}_{i,d-J_m(k,d,i)}(k,d)\widetilde{C}(k+i,d-J_m(k,d,i)) \\
&= \sum_{r=d-J_M(d,i)}^{d-J_m(k,d,i)-1}\widetilde{F}_{i,r}(k,d)\left[\widetilde{C}(k+i,r) - \widetilde{C}(k+i,r+1)\right] \\
&\quad + \widetilde{F}_{i,d-J_m(k,d,i)}(k,d)\widetilde{C}(k+i,d-J_m(k,d,i))\,.
\end{aligned}$$

Similarly, for $\widetilde{A}(k,d+1,i)$,

$$\widetilde{A}(k,d+1,i) = \sum_{r=d+1-J_M(d+1,i)}^{d-J_m(k,d+1,i)} \widetilde{F}_{i,r}(k,d+1)\left[\widetilde{C}(k+i,r) - \widetilde{C}(k+i,r+1)\right] + \widetilde{F}_{i,d+1-J_m(k,d+1,i)}(k,d+1)\widetilde{C}(k+i,d+1-J_m(k,d+1,i))\,.$$

Therefore,

$$\begin{aligned}
&\widetilde{A}(k,d,i) - \widetilde{A}(k,d+1,i) = \\
&\sum_{r=d+1-J_M(d+1,i)}^{d-J_m(k,d,i)-1} \left[\widetilde{F}_{i,r}(k,d) - \widetilde{F}_{i,r}(k,d+1)\right]\left[\widetilde{C}(k+i,r) - \widetilde{C}(k+i,r+1)\right] \\
&+ \sum_{r=d-J_M(d,i)}^{d-J_M(d+1,i)} \widetilde{F}_{i,r}(k,d)\left[\widetilde{C}(k+i,r) - \widetilde{C}(k+i,r+1)\right] \\
&+ \widetilde{F}_{i,d-J_m(k,d,i)}(k,d)\widetilde{C}(k+i,d-J_m(k,d,i)) \\
&- \sum_{r=d-J_m(k,d,i)}^{d-J_m(k,d+1,i)} \widetilde{F}_{i,r}(k,d+1)\left[\widetilde{C}(k+i,r) - \widetilde{C}(k+i,r+1)\right] \\
&- \widetilde{F}_{i,d+1-J_m(k,d+1,i)}(k,d+1)\widetilde{C}(k+i,d+1-J_m(k,d+1,i))\,.
\end{aligned}$$

The assumed monotonic properties for $\widetilde{F}_{i,r}(k,d)$ and $\widetilde{C}(k,d)$ ensure that the first two terms above are non-negative. Let $\widetilde{B}(d,d,i)$ be the sum of the remaining three terms. Note that, trivially,

$$d - J_m(k,d,i) = \begin{cases} N-k-i & \text{if } k+d+i > N-1 \\ d & \text{otherwise} \end{cases},$$

and

$$d - J_m(k,d+1,i) = \begin{cases} N-k-i-1 & \text{if } k+d+i > N-1 \\ d & \text{otherwise} \end{cases}.$$

Then, if $k+d+i > N-1$,

$$\begin{aligned}
B(k,d,i) &= \widetilde{F}_{i,N-k-i}(k,d)\widetilde{C}(k+i,N-k-i) - \widetilde{F}_{i,N-k-i}(k,d+1)\widetilde{C}(k+i,N-k-i) \\
&= \left[\widetilde{F}_{i,N-k-i}(k,d) - \widetilde{F}_{i,N-k-i}(k,d+1)\right]\widetilde{C}(k+i,N-k-i) \geq 0\,,
\end{aligned}$$

and, if $k+d+i \leq N-1$,

$$\begin{aligned}
B(k,d,i) &= \widetilde{F}_{i,d}(k,d)\widetilde{C}(k+i,d) - \widetilde{F}_{i,d}(k,d+1)\left[\widetilde{C}(k+i,d) - \widetilde{C}(k+i,d+1)\right] \\
&\quad - \widetilde{F}_{i,d+1}(k,d+1)\widetilde{C}(k+i,d+1) \\
&= \left[\widetilde{F}_{i,d}(k,d) - \widetilde{F}_{i,d}(k,d+1)\right]\left[\widetilde{C}(k+i,d) - \widetilde{C}(k+i,d+1)\right] \\
&\quad + \left[\widetilde{F}_{i,d}(k,d) - \widetilde{F}_{i,d+1}(k,d+1)\right]\widetilde{C}(k+i,d+1) \geq 0\,.\square
\end{aligned}$$

Lemma 6.2 *The bounds* $\widetilde{C}(k)$, $1 \leq k \leq N$ *defined by* (6.13) *are increasing on* k.

Proof The proof is exactly as the one of Lemma 5.2 replacing L and $C(k)$ by $\widetilde{L}$ and $\widetilde{C}(k)$ respectively.

Proposition 6.1 *Let* $(k,d) \in \widetilde{\mathcal{R}}$. *Assume* $C_U^s \leq \widetilde{C}(k,d)$, $s \in \widetilde{U}_{k,d}$, *and that* $\widetilde{C}(k,d)$ *are increasing on* k *and decreasing on* d. *Then,* $C_U^s \leq \widetilde{C}'(k,d)$, $s \in \widetilde{U}_{k,d}$.

Proof Let $s \in \widetilde{U}_{k,d}$. Since (6.6), C_U^s is equal to the mean time in s, if $d = 0$, plus the mean down time from the next visited state m, if $m \in U$. Let us analyze the subsets $\widetilde{U}_{k',d'}$ to which m may belong. From (6.9), a repair transition involving i, $i \in RC$, $i \leq k$ components can only lead to states in $\widetilde{U}_{k-i,d'}$, $k > i$ (if $k = i$ the reached state would be $o \notin U$), $d \leq d' \leq d+i$. Making the change $d' = d+j$ and imposing $(k-i, d+j) \in \widetilde{\mathcal{R}}$ yields $\max\{0, \widetilde{L} - k + i - d\} \leq j \leq \min\{\widetilde{L} - d, i\}$. Consider now the states that can be reached from s following a transition associated with failure bag $e \in E_i$, $i \in FC$. From (6.8), $m \in \widetilde{U}_{k+i,d-j}$, $0 \leq j \leq i$. Imposing $(k+i, d-j) \in \widetilde{\mathcal{R}}$ we get $J_m(k,d,i) \leq j \leq J_M(d,i)$. Based on the previous discussion and denoting $J'_m(k,d,i) = \max\{0, \widetilde{L} - k + i + d\}$ and $J'_M(d,i) = \min \widetilde{L} - d, i$ we have

$$
\begin{aligned}
C_U^s &= \widetilde{T}_1(d) + \widetilde{T}_2(k,d,i) + \widetilde{T}_3(k,d,i)\,, \\
\widetilde{T}_1(d) &= \frac{I_{d=0}}{\lambda_s}\,, \\
\widetilde{T}_2(k,d,i) &= \sum_{\substack{i \in RC \\ i \leq k-1}} \sum_{j=J'_m(k,d,i)}^{J'_M(d,i)} \sum_{m \in \widetilde{U}_{k-i,d+j}} \frac{\lambda_{s,m}}{\lambda_s} C_U^s\,, \\
\widetilde{T}_3(k,d,i) &= \sum_{\substack{i \in FC \\ i \leq N-k}} \sum_{j=J_m(k,d,i)}^{J_M(d,i)} \sum_{m \in \widetilde{U}_{k+i,d-j}} \frac{\lambda_{s,m}}{\lambda_s} C_U^s\,.
\end{aligned}
$$

From this point the proof continues exactly as the proof of Proposition 5.1 replacing $T_1(d)$, $T_2(k,d,i)$, $T_3(k,d,i)$, and $U_{k,d}$ by, respectively, $\widetilde{T}_1(d)$, $\widetilde{T}_2(k,d,i)$, $\widetilde{T}_3(k,d,i)$, and $\widetilde{U}_{k,d}$, and using $\widetilde{C}(k,d)$, $\widetilde{L}$, $\widetilde{\mathcal{R}}$, $\widetilde{F}_{i,r}(k,d)$, and $\widetilde{f}_{i,j}(k,d)$ instead of $C(k,d)$, L, $\mathcal{R}$, $F_{i,r}(k,d)$, and $f_{i,j}(k,d)$.□

Proposition 6.2 *Assume that* $\widetilde{C}(k,d)$, $(k,d) \in \widetilde{\mathcal{R}}$, $\widetilde{F}(k,d,\, i,r)$, $(k,d,\, i,r) \in \widetilde{\mathcal{R}}'$, *and* $\widetilde{F}(k,d,\, i,d)$, $(k,d,i,d) \in \widetilde{\mathcal{R}}'$, *are decreasing on* d. *Then, the bounds* $\widetilde{C}'(k,d)$, $(k,d) \in \widetilde{\mathcal{R}}$, *are decreasing on* d.

Proof Let (k,d), $(k,d+1) \in \widetilde{\mathcal{R}}$. Using (6.15),

$$\widetilde{C}'(k,d) - \widetilde{C}'(k,d+1) = T_1 + T_2 + \sum_{\substack{i \in FC \\ i \le N-k}} T_3(i)\,,$$

with

$$T_1 = \frac{I_{d=0} - I_{d+1=0}}{g(k)}\,,$$

$$T_2 = I_{k>1}\Big[I_{d>\widetilde{L}-k}\,\widetilde{C}(k-1,d) + I_{d\le\widetilde{L}-k}\,\widetilde{C}(k-1,d+1) - I_{d+1>\widetilde{L}-k}\,\widetilde{C}(k-1,d+1) - I_{d+1\le\widetilde{L}-k}\,\widetilde{C}(k-1,d+2)\Big]\,,$$

$$T_3(i) = \frac{\widetilde{A}(k,d,i) - \widetilde{A}(k,d+1,i)}{g(k)}\,,$$

where $\widetilde{A}(k,d,i)$ is defined as in Lemma 6.1. We will show that T_1, T_2 and $T_3(i)$ are all ≥ 0. Since $(k,d) \in \widetilde{\mathcal{R}}$, $d \ge 0$ and $d+1 > 0$. Therefore, $T_1 = I_{d=0}/g(k) \ge 0$. Regarding T_2, three cases must be considered: a) $k = 1$, b) $k > 1$, $d > \widetilde{L} - k$, and c) $k > 1$, $d \le \widetilde{L} - k$. In case a, $T_2 = 0$; in case b, $T_2 = \widetilde{C}(k-1,d) - \widetilde{C}(k-1,d+1) \ge 0$ because $\widetilde{C}(k',d')$, $(k',d') \in \widetilde{\mathcal{R}}$, is assumed decreasing on d; in case c, $d+1 > \widetilde{L} - k$ because (k,d), $(k,d+1) \in \widetilde{\mathcal{R}}$, and, thereby, $T_2(i) = \widetilde{C}(k-1,d+1) - \widetilde{C}(k-1,d+1) = 0$. Finally, $T_3(i) \ge 0$ by Lemma 6.1.□

Proposition 6.3 *Assume that* $\widetilde{C}(k,d)$, $(k,d) \in \widetilde{\mathcal{R}}$, $\widetilde{F}(k,d,i,r)$, $(k,d,i,r) \in \widetilde{\mathcal{R}}'$, *and* $\widetilde{F}(k,d,i,d)$, $(k,d,i,d) \in \widetilde{\mathcal{R}}'$, *are decreasing on* d. *Then, the bounds* $\widetilde{C}(k,d)$, $(k,d) \in \widetilde{\mathcal{R}}$, *are increasing on* k.

Proof Consider the algorithm which improves the bounds $\widetilde{C}(k,d)$ split into phases, where each phase includes the operations performed within the k-loop, and let $\widetilde{C}^{(m)}(k,d)$, $m \ge 0$ be the bounds $\widetilde{C}(k,d)$ available after phase m. The proof is by induction on m. Let (k,d), $(k-1,d) \in \widetilde{\mathcal{R}}$. For $m = 0$, using Lemma 6.2,

$$\widetilde{C}^{(0)}(k,d) = \widetilde{C}(k) \ge \widetilde{C}(k-1) = \widetilde{C}^{(0)}(k-1,d)\,.$$

Assume that $\widetilde{C}^{(m')}(k,d)$, $0 \le m' \le m$, $m \ge 0$ are increasing on k and let k' be the value of k for which the bounds are updated in phase $m+1$. Let $(k'-1,d) \in \widetilde{\mathcal{R}}$, which implies $k' > 1$ and $d \ge \max\{0, \widetilde{L} - k' + 1\} > \widetilde{L} - k'$. We have (6.15)

$$\widetilde{C}'(k',d) = \frac{I_{d=0}}{g(k')} + \widetilde{C}^{(m)}(k'-1,d) + \frac{1}{g(k')} \sum_{\substack{i \in FC \\ i \le N-k'}} \widetilde{A}(k',d,i)\,,$$

$$\widetilde{A}(k',d,i) = \sum_{j=J_m(k',d,i)}^{J_M(d,i)} \widetilde{f}_{i,j}(k',d)\widetilde{C}^{(m_i)}(k'+i,d-j)\,, \qquad m_i < m\,.$$

Using (6.14) and Proposition 6.2,

$$\begin{aligned}\widetilde{A}(k',d,i) &= \sum_{j=J_m(k',d,i)}^{J_M(d,i)-1} [\widetilde{F}_{i,d-j}(k',d) - \widetilde{F}_{i,d-j-1}(k',d)]\widetilde{C}^{(m_i)}(k'+i,d-j) \\ &\quad + \widetilde{F}_{i,d-J_M(d,i)}(k',d)\widetilde{C}^{(m_i)}(k'+i,d-J_M(d,i)) \\ &= \sum_{j=J_m(k',d,i)+1}^{J_M(d,i)} \widetilde{F}_{i,d-j}(k',d)\left[\widetilde{C}^{(m_i)}(k'+i,d-j) - \widetilde{C}^{(m_i)}(k'+i,d-j+1)\right] \\ &\quad + \widetilde{F}_{i,d-J_m(k',d,i)}(k',d)\widetilde{C}^{(m_i)}(k'+i,d-J_m(k',d,i)) \geq 0\,.\end{aligned}$$

Therefore, $\widetilde{C}'(k',d) \geq \widetilde{C}^{(m)}(k'-1,d)$. Using the induction hypothesis, this implies

$$\widetilde{C}^{(m+1)}(k',d) = \min\{\widetilde{C}'(k',d), \widetilde{C}^{(m)}(k',d)\} \geq \widetilde{C}^{(m)}(k'-1,d)\,.\square$$

Theorem 6.3 *Assume that $\widetilde{F}_{i,r}(k,d)$, $(k,d,i,r) \in \widetilde{\mathcal{R}'}$ and $\widetilde{F}_{i,d}(k,d)$, $(k,d,i,d) \in \widetilde{\mathcal{R}'}$ are decreasing on d. Then, $C_U^s \leq \widetilde{C}(k,d)$, $s \in \widetilde{U}_{k,d}$ and the bounds $\widetilde{C}(k,d)$ are decreasing on d.*

Proof Consider the algorithm that improves the bounds $\widetilde{C}(k,d)$ split into phases as in the proof of Proposition 6.3. The proof is by induction on m. For $m = 0$, Theorem 6.2 ensures that $\widetilde{C}^{(0)}(k,d) = \widetilde{C}(k)$, which are (non-strictly) decreasing on d, upper bound C_U^s, $s \in \widetilde{U}_{k,d}$. Assume that $\widetilde{C}^{(m')}(k,d)$, $0 \leq m' \leq m$, $m \geq 0$ upper bound C_U^s, $s \in \widetilde{U}_{k,d}$ and are decreasing on d. Let k' be the value of k for which the bounds are updated in phase $m+1$. According to (6.15), $\widetilde{C}'(k',d)$ only depend on $\widetilde{C}^{(m')}(k,d)$, $0 \leq m' \leq m$, $k \neq k'$, which, by Proposition 6.3, are increasing on k. Propositions 6.1 and 6.2 guarantee, respectively, that $\widetilde{C}'(k',d)$ upper bound C_U^s, $s \in \widetilde{U}_{k',d}$ and are decreasing on d. Therefore, $\widetilde{C}^{(m+1)}(k',d) = \min\{\widetilde{C}'(k',d), \widetilde{C}^{(m)}(k',d)\}$ upper bound C_U^s, $s \in \widetilde{U}_{k',d}$ and are decreasing on d. $\square$

6.2 Failure Rate Bounding Structures

In this section we derive suitable upper bounds $\widetilde{F}_{i,r}(k,d)$, $(k,d,i,r) \in \widetilde{\mathcal{R}'}$ for $\sum_{d'=0}^{r} \lambda_{s,\widetilde{U}_{k+i,d'}}$, $s \in \widetilde{U}_{k,d}$.

For $r = \min\{d, N-k-i\}$ we take the same upper bound as for the method derived in Chapter 5:

$$\widetilde{F}_{i,\min\{d,N-k-i\}}(k,d) = f_i\,.$$

Let $\widetilde{\eta}(e)$, $e \in E$ be the lower bound for the failure distance from a state whose bag of failed components is e. Let $s' \in U$ be a state reached from $s \in \widetilde{U}_{k,d}$ in a single-step transition associated with failure bag e, i.e $F(s') = F(s) + e$. Note that from the bag of failed components point of view, nothing prevents us from regarding s' as the result of two single-step "transitions": the first one, associated with e, from o to a dummy state s'' with $F(s'') = e$, and the second transition, involving $F(s)$ components, from s'' to s. Since $\widetilde{d}(s'') = \widetilde{\eta}(e)$, we have (6.8) $\widetilde{d}(s') \geq \widetilde{\eta}(e) - |F(s)| = \widetilde{\eta}(e) - k$. Therefore, for $(k, d, i, r) \in \widetilde{\mathcal{R}'}$, $r < \min\{d, N - k - i\}$ we can write

$$\widetilde{F}_{i,r}(k, d) = \sum_{\substack{e \in E_i \\ \widetilde{\eta}(e) \leq k+r}} \lambda_{\text{ub}}(e) \geq \sum_{d'=\max\{0,d-i\}}^{r} \lambda_{s,\widetilde{U}_{k+i,d'}} .$$

Since neither $\widetilde{F}_{i,r}(k, d)$, $(k, d, i, r) \in \widetilde{\mathcal{R}'}$ nor $\widetilde{F}_{i,d}(k, d)$, $(k, d, i, d) \in \widetilde{\mathcal{R}'}$ depend on d, they are (non-strictly) decreasing on d and, thereby, fulfill the conditions imposed by Theorem 6.3.

6.3 Analysis

In this section we analyze the performance of the bounding method described in this chapter and compare it with the bounding method developed in Chapter 5 and the methods described in [22] and [70]. In all cases the subset G is incrementally generated until the relative unavailability band, $([UA]_{\text{ub}} - [UA]_{\text{lb}})/[UA]_{\text{lb}}$, is smaller than or equal to the desired one. For the first three methods we use the state space exploration algorithm *CONT_TG_W* reviewed in Section 5.3. For the last method we use an analogous state exploration algorithm with the modifications described in Section 5.4.

The analysis and comparison will be made using two examples. The architecture of the first example is depicted in Figure 6.2. The system is exactly as the last example we used in Section 5.4, but with the number of nodes of both networks reduced to six and the number of processing clusters reduced to three. The system has 48 components and 8,653 minimal cuts, 4 with cardinality 3, 6,561 with cardinality 4, 1,944 with cardinality 5, and 144 with cardinality 6. As a second example we use the last one of Section 5.4, which has 78 components and 87,031 minimal cuts, 6 with cardinality 3, 75,625 with cardinality 4, 11,000 with cardinality 5, and 400 with cardinality 6. For both examples, $L = 3$ and $\widetilde{L} = 2$. All results have been obtained in a 128 MB UltraSparc 1 workstation with the control parameter BR of algorithm *CONT_TG_W* set to 0.1. For the method described in Chapter 5, the control parameter R to compute failure distances has been

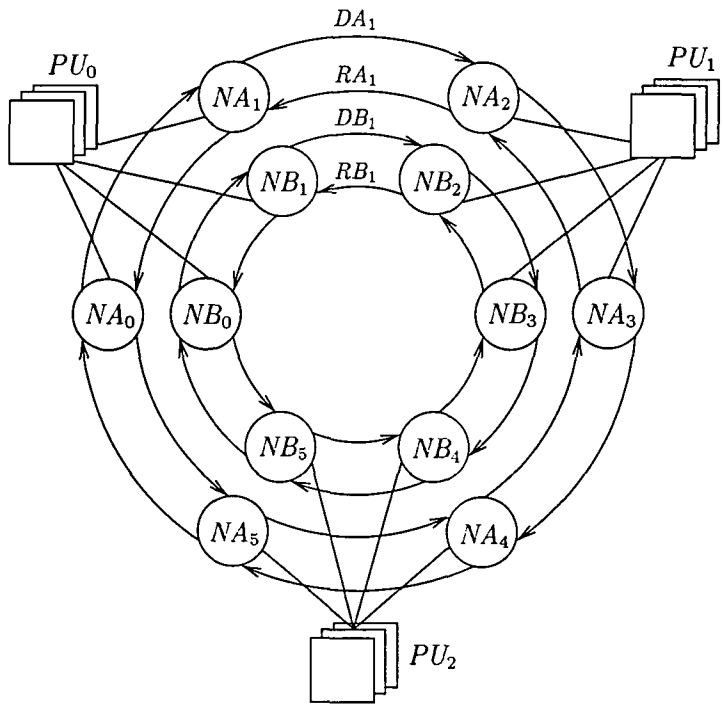

Figure 6.2: Architecture of the first example.

set to 2 and the control parameter R' to compute subbags of minimal cuts has been set to 6.

We show in Figure 6.3 the relative unavailability band as a function of the number of states in G for the first example. The proposed method clearly outperforms the method described in [70]. Hence, the number of states required by that method to achieve a given relative unavailability band ranges from 4.6 to 12.2 times the number of states required by the method developed in this chapter. Regarding the method described in [22], the proposed method requires a number of states between 1.6 and 30.6 times larger. The comparison is even worse for the method described in Chapter 5, which requires a number of states between 2.1 and 42 times smaller than the method proposed in this chapter. Those figures are somewhat misleading since they do not take into account the amount of memory required in the methods described in [22] and Chapter 5 to store all minimal cuts and related structures. In Figure 6.4 we compare the methods in terms of memory consumption, which is really the figure of interest. The proposed method is again better than the method described in [70], with a memory consumption for a given relative band between 4.1 and 7.6 times smaller. Regarding the two methods that use exact failure distances, the proposed method is more efficient for moderate values of the desired relative unavailability band. For this example, storing the minimal cuts takes about 4.5 MB of

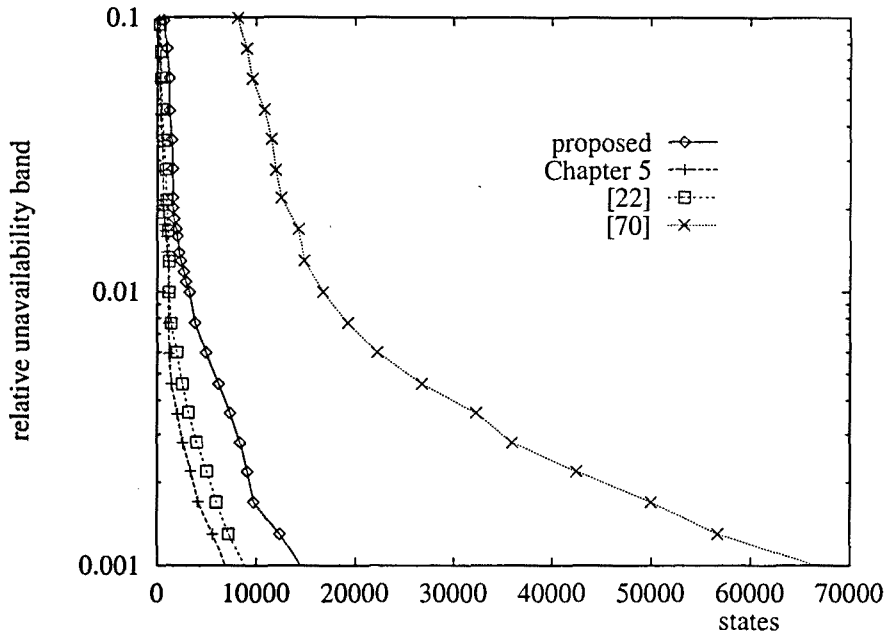

Figure 6.3: Relative unavailability band as a function of the number of states in G for the first example.

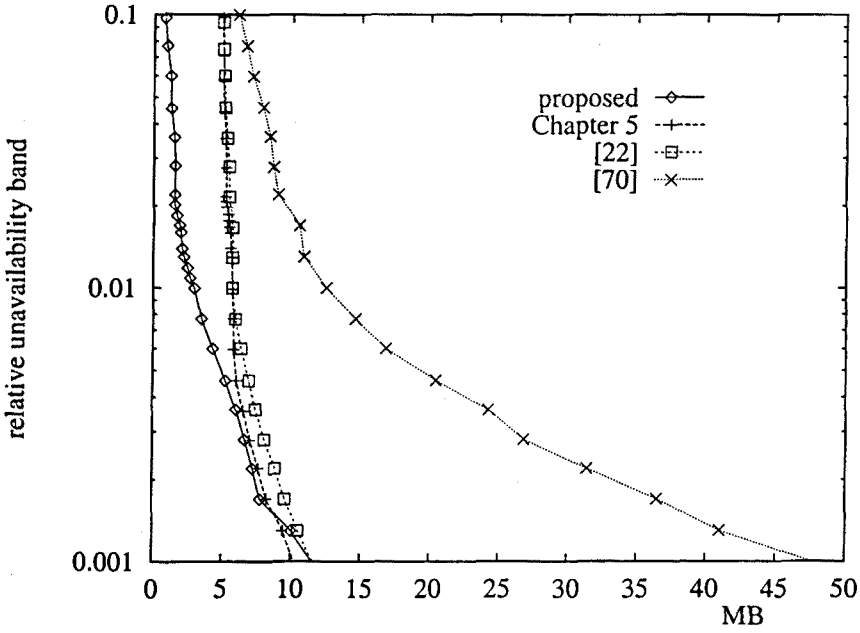

Figure 6.4: Relative unavailability band as a function of memory consumption for the first example.

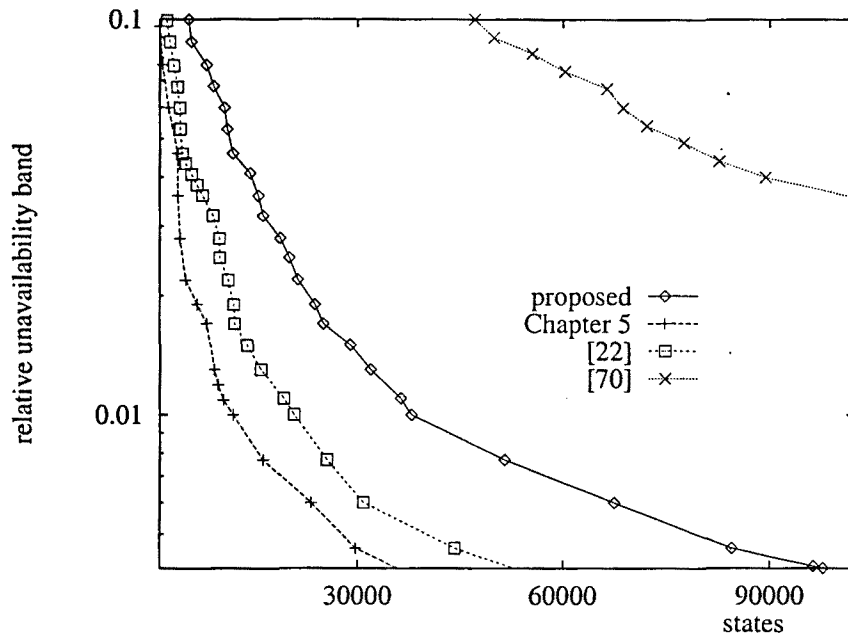

Figure 6.5: Relative unavailability band as a function of the number of states in G for the second example.

memory. As it can be seen in Figure 6.4, this is approximately the difference in terms of memory consumption between the proposed method and the methods described in [22] and Chapter 5 for a relative band equal to 0.1. The smaller the relative band, however, the larger the generated subset and, thereby, the memory consumption due to storing state descriptions and list of contributions becomes relatively more important than the overhead introduced by storing the minimal cuts and related data structures. This explains why the difference in terms of memory consumption between the proposed method and the methods described in [22] and Chapter 5 decreases as the target band gets smaller and, eventually, the proposed method becomes more memory consuming.

In figures 6.5 and 6.6 we show the relative unavailability band for the second example in terms of, respectively, size of the generated subset and memory consumption. The same comments we did for the previous example apply here: the proposed method outperforms the method described in [70] and, in terms of memory consumption, it is more efficient than the methods described in [22] and Chapter 5 for large to moderate relative unavailability bands.

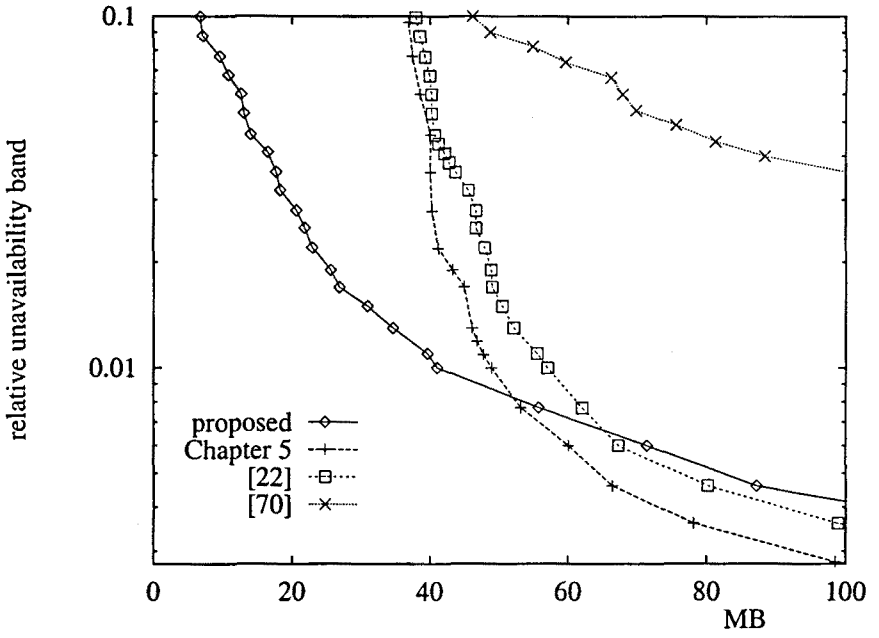

Figure 6.6: Relative unavailability band as a function of memory consumption for the second example.

6.4 Conclusions

In this chapter we have developed a bounding method for the steady-state unavailability that is based on lower bounds for failure distances and, therefore, does not require the knowledge of the minimal cuts of the system. The method seems to be better than the method described in [70], which is not based in the failure distance concept. Regarding the methods described in [22] and Chapter 5, which require to know and hold in memory the set of minimal cuts of the system, the proposed method can outperform both of them for moderate values of the relative unavailability band when the number of minimal cuts is large.

Chapter 7

Conclusions and Future Work

The goal of the dissertation was the development of failure distance based bounding methods for two dependability measures/scenarios: 1) the unreliability at time t, $ur(t)$, for non-repairable fault-tolerant systems, and 2) the steady-state unavailability, UA, for repairable fault-tolerant systems. The goal has been achieved for a quite wide class of models and the required high-level knowledge about the model is reasonable and modest. We have developed four methods: two using failure distances, and two using lower bounds for failure distances that can be inexpensively computed on the fault tree of the system. For the same number of generated states, the first two give, in general, tighter bounds, but those methods have the potential limitation that computing failure distances is an NP-hard problem. The algorithms we have used to compute the failure distances assume the knowledge of the minimal cuts of the fault tree. The algorithms seem to be very efficient from a time point of view even when the number of minimal cuts is large (tens of thousands). However, computation of the minimal cuts in a reasonable amount of time may be impossible and, then, the methods could not be used. Also, the number of minimal cuts can be very large and, in that case, the memory overhead associated with them may be important, making the other bounding methods more efficient from a memory usage point of view. As the examples we have presented show, all four methods can outperform significantly previously available methods and extend significantly the complexity of the fault-tolerant systems for which tight bounds for the considered dependability measures can be computed. In that regard, we want to note that the method described in Chapter 5 has obtained very tight bounds for a fault-tolerant system with 372 components using 100 MB of memory. Workstations with 1 GB of memory are common today and, we estimate, with such a workstation tight bounds should be attainable for fault-tolerant systems having of the order of 1,000 components. Fault-tolerant

interconnection networks [36] are fault-tolerant systems having that complexity. Once solving the difficulty of building a fault tree for those systems, their dependability could be analyzed precisely using the methods described in the dissertation.

Much research work related with the dissertation remains and the author wants to undertake it in the future. First, it is tempting to develop bounding methods for $ur(t)$ for non-repairable fault-tolerant systems using structures $F_{i,r}(k,d)$ similar to those used by the proposed bounding methods for UA. Secondly, efficient state space exploration algorithms for those methods could be developed. Finally, other dependability measures/scenarios could be considered such as $ur(t)$ for repairable systems, the point unavailability and the expected interval unavailability for repairable systems, and the distribution of the interval availability for repairable systems. The latter would have special interest, since available numerical methods for computing that measure [23, 84, 85, 94] are expensive and keeping the size of the generated state space as small as possible is an important issue. Finally, all these methods should be offered in a tool with an as flexible as possible modeling language encompassing the class of models for which the bounding methods have been developed.

Bibliography

[1] J. A. Abraham, "An improved algorithm for network reliability," *IEEE Trans. on Reliability*, vol. R-28, April 1979, pp. 58–61.

[2] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test Generation for VLSI Scan-Design Circuits,", *IEEE Design and Test of Computers*, vol. 3, no. 4, August 1986, pp. 43–54.

[3] K. K. Aggarwal, K. B. Misra, and J. S. Gupta, "A fast algorithm for realiability evaluation," *IEEE Trans. on Reliability*, vol. R-24, April 1975, pp. 83–85.

[4] Ch. Alexopoulos and B. C. Shultes, "The Balanced Likelihood Ratio Method for Estimating Performance Measures of Highly Reliable Systems," in *Proc. Winter Simulation Conference*, Piscataway, New Jersey, 1998.

[5] Ch. Alexopoulos and B. C. Shultes, "Estimating Reliability Measures for Highly-reliable Markovian Systems Using Balanced Likelihood Ratios," Technical Report, Georgia Institute of Technology, March 1999.

[6] S. V. Amari, J. B. Dugan and R. B. Misra, "A separable method for incorporating imperfect fault-coverage into combinatorial models," *IEEE Trans. on Reliability*, vol. 48, no. 3, September 1999, pp. 267–274.

[7] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation—A Methodology and Some Applications," *IEEE Trans. on Software Engineering*, vol. 16, no. 2, pp. 166-182, February 1990.

[8] R.E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing. Probability Models*, McArdle Press, Silver Spring, 1981.

[9] J. H. Barton, E. W. Czeck, Z. Z. Segall, and D. P. Siewiorek, "Fault Injection Experiments Using FIAT," *IEEE Trans. on Computers*, vol. 39, no. 4, pp. 575–582, April 1990.

[10] N. N. Bengiamin, B. A. Bowen, and K. F. Schenk, "An Efficient Algorithm for Reducing the Complexity of Computation in Fault Tree Analysis," *IEEE Trans. on Nuclear Science*, vol. NS-23, no. 5, October 1976, pp. 1442–1446.

[11] R. G. Bennets, "On the analysis of fault trees," *IEEE Trans. on Reliability*, vol. R-24, August 1975, pp. 175–185.

[12] C. Béounes, M. Aguéra, J. Arlat, S. Bachman, C. Bourdeau, J. E. Doucet, K. Kanoun, J. C. Laprie, S. Metge. J. Moreira de Souza, D. Powell, and P. Spiesser, "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems," *Proc. 23rd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, 1993, pp. 668–673.

[13] U. N. Bhat, *Elements of Applied Stochastic Processes*, John Wiley and Sons, 1984.

[14] W. G. Bouricious, W. C. Carter and P. R. Scheider, "Reliability modeling techniques for self-repairing computer systems," in *Proc. 24th Annual ACM Nat. Conf.*, 1969, pp. 295-309.

[15] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, vol. 24, 1992, pp. 293–318.

[16] J.A. Carrasco and J. Figueras, "METFAC: Design and Implementation of a Software Tool for Modeling and Evaluation of Complex Fault-Tolerant Computing Systems," *Proc. of the 16th Int. Symp. on Fault-Tolerant Computing FTCS-16*, 1986, pp. 424–429.

[17] J. A. Carrasco, "Efficient Transient Simulation of Failure/Repair Markovian Models," in *Proc. 10th IEEE Symp. on Reliable Distributed Systems*, 1991, pp. 152–161.

[18] J. A. Carrasco , "Failure Distance-based Simulation of Repairable Fault-Tolerant Systems," in *Computer Performance Evaluation*, Elsevier, 1992, pp. 351–365.

[19] J. A. Carrasco, A. Calderón, and J. Escrivá, "Two new algorithms to compute steady-state bounds for Markov models with slow forward and fast backward transitions," in *Proc. 4th Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'96)*, February 1996, pp. 89–95.

[20] J.A. Carrasco, J. Escribá and A. Calderón, "Efficient Exploration of Availability Models Guided by Failure Distances", *Performance Evaluation Review*, vol. 24, no. 1, May 1996.

[21] J. A. Carrasco and J. L. Domingo, "METFAC-2: A Tool for Specification and Solution of Markov Performance, Dependability and Performability Models," in *Proc. XII Design of Circuits and Integrated Systems Conference*, Sevilla, Spain, November 1997, pp. 195–200.

[22] J. A. Carrasco, "Tight Steady-state Availability Bounds using the Failure Distance Concept," *Performance Evaluation*, vol. 34, September 1998, pp. 27–64.

[23] J. A. Carrasco, "Solving Large Interval Availability Models using a Model Transformation Approach," Technical Report, Universitat Politècnica de Catalunya, March 1999, available at `ftp://ftp-eel.upc.es/techreports` under the name `DMSD_99_3.ps`

[24] J. A. Carrasco, "Bounding steady-state availability models with group repair and phase type repair distributions," *Performance Evaluation*, vol. 35, no. 4, 1999, pp. 193–214.

[25] J. A. Carrasco, "Failure-distance-based importance sampling schemes for simulation of repairable fault-tolerant computer systems," 1998, under review for *IEEE Trans. on Computers*.

[26] J. Carreira, H. Madeira and J. G. Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units," in *Proc. 5th Int'l Working Conf. on Dependable Computing for Critical Applications (DCCA-5)*, September 1995, pp. 135–149.

[27] P. Chatterjee, "Modularization of fault trees: A method to reduce the cost of analysis," *Reliability and Fault Tree Analysis*, SIAM, 1975, pp. 101–137.

[28] G. Chiola. C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications," *IEEE Trans. on Computers*, vol. 42, no. 11, November 1993, pp. 1343–1360.

[29] G. S. Choi, R. K. Iyer and V. Carreno, "FOCUS: An Experimental Environment for Fault Sensitivity Analysis," *IEEE Trans. on Computers*, vol. 41, no. 12, pp. 1,515–1,526, December 1992.

[30] G. Ciardo, J. Muppala and K. Trivedi, "SPNP: Stochastic Petri Net Package," *Proc. 3rd IEEE Int. Workshop on Petri Nets and Performance Models (PNPM89)*, Kyoto, 1989, pp. 142–150.

[31] E. Çinlar, *Introduction to Stochastic Processes*, Prentice-Hall, 1975, Englewood Cliffs, NJ, USA.

[32] A. E. Conway and A. Goyal, "Monte Carlo Simulation of Computer Systems Availability/Reliability Models," in *Proc. 17th IEEE Int. Symp. on Fault-Tolerant Computing*, 1987, pp. 230–235.

[33] O. Coudert and J. C. Madre, "MetaPrime: An Interactive Fault-Tree Analyzer," *IEEE Trans. on Reliability*, vol. 43, no. 1, March 1994, pp. 121–127.

[34] J. Couvillion, R. Freire, R. Johnson, W. Obal II, A. Qureshi, M. Rai, W. Sanders, and J. Tvedt, "Performability modelling with UltraSAN," *IEEE Software*, September 1991, pp. 69–80.

[35] S. A. Doyle, J. B. Dugan and F. A. Patterson-Hine, "A combinatorial approach to modeling imperfect coverage," *IEEE Trans. on Reliability*, vol. 44, March 1995, pp. 87–94.

[36] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks, an engineering approach*, IEEE Computer Society, 1997.

[37] J. B. Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist, "The hybrid automated reliability predictor," *AIAA J. Guidance, Contr., Dynam.*, vol. 9, May-June 1986, pp. 319–331.

[38] J. B. Dugan and K. S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," *IEEE Trans. on Computers*, vol. 38, no. 6, June 1989, pp. 775–787.

[39] J. B. Dugan, "Fault trees and imperfect coverage," *IEEE Trans. on Reliability*, vol. 38, no. 2, June 1989, pp. 177–185.

[40] Y. Dutuit and A. Rauzy, "A Linear-Time Algorithm to Find Modules of Fault Trees", *IEEE Trans. on Reliability*, vol. 45, no. 3, September 1996, pp. 422–425.

[41] H. Fujiwara and T. Shimano, "On the Acceleration of Test Generation Algorithms", *IEEE Transactions on Computers*, vol. C-32, no. 12, December 1983, pp. 1137–1144.

[42] J. B. Fussell and W. E. Vesely, "A New Methodology for Obtaining Cut Sets for Fault Trees," *Trans. of the American Nuclear Society*, vol. 15, June 1972, pp. 262–263.

[43] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[44] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, March 1981, pp. 215–222.

[45] L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Transactions on Circuits and Systems*, vol. CAS-26, no. 9, September 1979, pp. 685–693.

[46] A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg, and K.S. Trivedi, "The System Availability Estimator," *Proc. of the 16th Int. Symp on Fault-Tolerant Computing FTCS-16*, 1986, pp. 84–89.

[47] A. Goyal, P. Heidelberger and P. Shahabuddin, "Measure Specific Dynamic Importance Sampling for Availability Simulations," in *Proc. 1987 Winter Simulation Conference*, A. Thesen, H. Grant and W. D. Kelton (eds.), 1987, pp. 351–357.

[48] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola, and P. W. Glynn, "A Unified Framework for Simulating Markovian Models of Highly Dependable Systems," *IEEE Trans. on Computers*, vol. 42, no. 1, January 1992, pp. 36–51.

[49] D. Gross and D. Miller, "The randomization technique as a modeling tool and solution procedure for transient Markov processes", *Operations Research*, vol. 38, no. 2, 1984, pp 334–361

[50] U. Gunneflo, J. Karlsson and J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation," in *Proc. 19th Int'l Symp. on Fault-Tolerant Computing (FTCS-19)*, June 1989, pp. 340–347.

[51] S. Han, K. G. Shin and H. A. Rosenberg, "DOCTOR: An IntegrateD SOftware Fault InjeCTiOn Environment for Distributed Real-Time Systems," in *Proc. 1st Int'l Computer Performance and Dependability Symp.*, April 1995, pp. 204–213.

[52] W. Hennings and N. Kuznetsov, "FAMOCUTN & CUTQN: Programs for Fast Analysis of Large Fault Trees with Replicated & Negated Gates," *IEEE Trans. on Reliability*, vol. 44, no. 3, September 1995, pp. 368–376.

[53] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," in *Proc. 24th Int'lSymp. on Fault-Tolerant Computing (FTCS-24)*, 1994.

[54] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, 1989.

[55] G. A. Kanawati, N. A. Kanawati and J. A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties," in *Proc. 22nd Int'l Symp. on Fault-Tolerant Computing (FTCS-22)*, July 1992, pp. 336–344.

[56] W. L. Kao and R. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment," *Fault-Tolerant Parallel and Distributed Systems*, D. K. Pradhan and D. R. Avresky, eds., IEEE CS Press, 1995, pp. 252–259.

[57] T. Kohda, E. J. Henley and K. Inoue, "Finding Modules in Fault Trees," *IEEE Trans. on Reliability*, vol. 38, no. 2, June 1989, pp. 165–176.

[58] C. Landrault and J. C. Laprie, "SURF—A program for modeling and reliability prediction for fault-tolerant computing systems," *Information Technology*, J. Moneta, ed., North-Holland, 1978.

[59] N. Limnios and R. Ziani, "An Algorithm for Reducing Cut Sets in Fault-Tree Analysis," *IEEE Trans. on Reliability*, vol. R-35, no. 5, December 1986, pp. 559–561.

[60] W.S. Lee, D.L. Grosh, F.A. Tillman, and C.H. Lie, " Fault Tree Analysis, Methods and Applications –A Review," *IEEE Trans. on Reliability*, vol. R-34, no. 3, August 1985, pp. 194–203.

[61] D. Lee, J. Abraham, D. Rennels and G. Gilley, "A Numerical Technique for the Hierarchical Evaluation of Large, Closed Fault-Tolerant Systems," in *Dependable Computing for Critical Applications*, Springer-Verlag, 1992, pp. 95–114.

[62] D. Lubell, "A Short Proof of Sperner's Lemma," *Journal of Combinatorial Theory*, 1996, no. 1, p. 299.

[63] J.C.S. Lui and R. R. Muntz, "Evaluating Bounds on Steady-State Availability of Repairable Systems from Markov Models," in *Numerical Solution of Markov chains*, Marcel Dekker, New York, pp. 435–454, 1991.

[64] J.C.S. Lui and R.R. Muntz, "Computing Bounds on Steady State Availability of Repairable Computer Systems," *Journal of the ACM*, vol. 41, no. 4, July 1994, pp. 676–707.

[65] E. E. Lewis and F. Böhm, "Monte Carlo Simulation of Markov Unreliability Models," *Nuclear Engineering and Design*, vol. 77, 1984, pp. 49–62.

[66] S. Mahévas and G. Rubino, "Bounding asymptotic dependability and performance measures," in *Proc. 2nd IEEE Int. Performance and Dependability Symp.*, Urbana-Champaign, USA, September 1996, pp. 176–186.

[67] S. Mahévas and G. Rubino, "Bound computation of dependability and performance measures," Technical report no. 3135, IRISA, March 1997, to appear in *IEEE Trans. on Computers*.

[68] S.V. Makam and A. Avizienis, "ARIES 81: A reliability and life-cycle evaluation tool for fault-tolerant systems," in *Proc. 12th Int. Symp. on Fault-Tolerant Computing FTCS-12*, June 1982, pp. 266–274.

[69] J. F. Meyer, "On Evaluating the Peformability of Degradable Computing Systems," *IEEE Trans. on Computers*, vol. C-29, no. 8, August 1980, pp. 720–731.

[70] R.R. Muntz, E. de Souza e Silva and A. Goyal, "Bounding Availability of Repairable Computer Systems," *IEEE Trans. on Computers*, vol. 38, no. 12, pp. 1714–1723, December 1989.

[71] K. Nakashima and Y. Hattori, "An Efficient Bottom-up Algorithm for Enumerating Minimal Cut Sets of Fault Trees," *IEEE Trans. on Reliability*, vol. R-28, no. 5, December 1979, pp. 353–357.

[72] M. K. Nakayama, "General Conditions for Bounded Relative Error in Simulations of Highly Reliable Markovian Systems," *Advances in Applied Probability*, vol. 28, 1996, pp. 687–727.

[73] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*, Dover Publications Inc., New York, 1994, chapter 2.

[74] V. F. Nicola, P. Heidelberger and P. Shahabuddin, "Uniformization and Exponential Transformation: Techniques for Fast Simulation of Highly Dependable Non-Markovian Systems," in *Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing*, 1992, pp. 130–139.

[75] V. F. Nicola, P. Shahabuddin, P. Heidelberger, and P. W. Glynn, "Fast Simulation of Steady-State Availability in Non-Markovian Highly Dependable Systems," in *Proc. 23th IEEE Int. Symp. on Fault-Tolerant Computing*, 1993, pp. 38–47.

[76] V. F. Nicola, M. K. Nakayama, P. Heidelberger, and A. Goyal, "Fast Simulation of Highly Dependable Systems with General Failure and Repair Processes," *IEEE Trans. on Computers*, vol. 42, no. 12, December 1993, pp. 1440–1452.

[77] K. Odeh and N. Limnios, "A new algorithm for fault trees prime implicant computations," *ESREL'96*, Crete (Greece), June 1996, pp. 1085–1090.

[78] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[79] D. M. Rasmuson and N. H. Marshall, "FATRAM—A Core Efficient Cut-Set Algorithm," *IEEE Trans. on Reliability*, vol. R-27, no. 4, October 1978, pp. 250–253.

[80] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering and System Safety*, vol. 40, 1993, pp. 203–211.

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

[82] A. Rosenthal, "Decomposition Methods for Fault Tree Analysis," *IEEE Trans. on Reliability*, vol. R-29, no. 2, June 1980, pp. 136–138.

[83] S.M. Ross, *Stochastic Processes*, John Wiley & Sons, New York, 1983.

[84] G. Rubino and B. Sericola, "Interval Availability Distribution Computation," in *Proc. 23th Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, June 1993, pp. 48–55.

[85] G. Rubino and B. Sericola, "Interval Availability Analysis Using Denumerable Markov Processes: Application to Multiprocessor Subject to Breakdowns and Repair," *IEEE Trans. on Computers*, vol. 44, no. 2, February 1995, pp. 286–291.

[86] R.A. Sahner and K.S. Trivedi, "Reliability Modeling Using SHARPE," *IEEE Trans. on Reliability*, vol. R-36, no. 2, pp. 186–193, June 1987.

[87] W. H. Sanders and J. F. Meyer, "A Unified Approach for Specifying Measures of Performance, Dependability, and Performability," *Dependable Computing for Critical Applications*, vol. 4, A. Avizienis and J. C. Laprie, eds., Springer-Verlag, 1991, pp. 215–237.

[88] W. H. Sanders and J. F. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks," *IEEE Journal on Selected Areas in Communications*, special issue on *Computer-Aided Modeling, Analysis, and Design of Communication Networks*, vol. 9, no. 1, January 1991, pp. 25–36.

[89] P. Semal, "Refinable Bounds for Large Markov Chains," *IEEE Trans. on Computers*, vol. 44, no. 10, October 1995, pp. 1216–1222.

[90] S. N. Semanders, "'ELRAFT: A Computer Program for the Efficient Logic Reduction Analysis of Fault Trees," *IEEE Trans. on Nuclear Science*, vol. NS-18, February 1971, pp. 481–487.

[91] P. Shahabuddin, "Importance Sampling for the Simulation of Highly Reliable Markovian Systems," *Management Science*, vol. 40, no. 3, March 1994, pp. 333–352.

[92] P. Shahabuddin, V. F. Nicola, P. Heidelberger, A. Goyal, and P. W. Glynn, "Variance Reduction in Mean Time to Failure Simulations," in *Proc. 1988 Winter Simulation Conference*, M. Abrams, P. Haigh and J. Comfort (eds.), 1988, pp. 491–499.

[93] R. M. Sinnamon and J. D. Andrews, "Fault Tree Analysis and Binary Decision Diagrams," in *Proc. Annual Reliability and Maintainability Symp.*, 1996, pp. 215–222.

[94] E. de Souza e Silva and H. R. Gail, "Calculating Cumulative Operational Time Distributions of Repairable Computer Systems," *IEEE Trans. on Computers*, vol. C-35, no. 4, April 1986, pp. 322–332.

[95] E. de Souza e Silva and P.M. Ochoa, "State Space Exploration in Markov Models," *Performance Evaluation Review*, vol. 20, no. 1, June 1992, pp. 152–166.

[96] K. J. Sullivan, D. Coppit and J. B. Dugan, "The Galileo Fault Tree Analysis Tool," in *Proc. 29th Fault Tolerant Computing Symp. (FTCS-29)*, 1999.

[97] K. S. Trivedi, R. Geist, M. Smotherman, and J. B. Dugan, "Hybrid modeling of fault-tolerant systems," *Comput. Elec. Eng. Int. J.*, vol. 11, no. 2 & 3, 1985, pp. 87–108.

[98] J. M. Wilson, "Modularizing and Minimizing Fault Trees," *IEEE Trans. on Reliability*, vol. R-34, no. 4, October 1985, pp. 320–322.

[99] T. Zhuguo and E. E. Lewis, "Component Dependency Models in Markov Monte Carlo Simulation," *Reliability Engineering*, vol. 13, 1985, pp. 45–62.