Efficient Algorithms for Analyzing Cascading Failures in a Markovian Dependability Model

Mihir Sanghavi, Sashank Tadepalli, Timothy J. Boyle Jr., Matthew Downey and Marvin K. Nakayama

Abstract—We devise efficient algorithms to construct, evaluate, and approximate a Markovian dependability system with cascading failures. The model, which was previously considered in [22], represents a cascading failure as a tree of components that instantaneously and probabilistically fail. Constructing the Markov chain presents significant computational challenges because it requires generating and evaluating all such possible trees, but the number of trees can grow exponentially in the size of the model. Our new algorithm reduces runtimes by orders of magnitude compared to a previous method in [22]. Moreover, we propose some efficient approximations based on the idea of most likely paths to failure to further substantially reduce the computation time by instead constructing a model that uses only a subset of the trees. We also derive two new dependability measures related to the distribution of the size of a cascade. We present numerical results demonstrating the effectiveness of our approaches. For a model of a large cloud-computing system, our approximations reduce computation times by orders of magnitude with only a few percent error in the computed dependability measures.

Index Terms—Availability, reliability modeling, Markov processes, trees, cascading failures.

Acronyms

BFH	breadth-first-history data structure for computing tree rate
CTMC	continuous-time Markov chain
DCSUF	distribution of cascade size until failure
DECaF	Dependability Evaluator of Cascading Failures
DMT	dependability-measure (computation) time
DTMC	discrete-time Markov chain
FTT	failure-transition (computation) time
MTTF	mean time to failure
MTTFT	MTTF (computation) time
NFTT	non-failure-transition (computation) time
RTT	rate threshold (computation) time
SSDCS	steady-state distribution of cascade size
SSU	steady-state unavailability

Notation

Q	(infinitesimal) generator matrix of complete CTMC including all trees
Ω	set $\{1, 2, \ldots, N\}$ of component types
N	number of component types
r_i	redundancy of component type $i \in \Omega$
$\mathcal{E}$	set $\{0, 1, \ldots, L\}$ of environments
ν_e	(exponential) rate of leaving environment $e \in \mathcal{E}$
$\delta_{e,e'}$	transition probability of moving from environment e to e'
$\lambda_{i,e}$, $\mu_{i,e}$	failure and repair rates, resp., of a component of type i in environment e
Γ_i	(ordered) set of component types that can be caused to fail immediately when a type-i component fails
$\phi_{i,j}$	probability that a failure of a component of type i causes a component of type $j \in \Gamma_i$ to immediately fail.
Z	CTMC of dependability system
S	state space of CTMC Z
x_*	state with no comps failed and environment 0
$\Psi_\mathrm{e}, \Psi_\mathrm{r}, \Psi_\mathrm{f}$	sets of environment, repair, and failure transitions, resp.
$R(T)$	rate of tree T
ρ	product of component-affected probabilities $\phi_{i,j}$ of failed components in a tree
η	product of complements of component-affected probabilities of non-failing components in a tree
u_i	current number of up components of type i
U, F	sets of up and failed states, resp., in S
Y	embedded DTMC
P	transition probability matrix of DTMC
P_U	submatrix of P corresponding to U
$a(x, y)$	number of components failing in transition (x, y)
b	sum of redundancies of all component types
θ	steady-state distribution of cascade size (SSDCS)
χ	distribution of cascade size until failure (DCSUF)
Q'	CTMC generator matrix when trees omitted
$R'(T)$	approximate rate of tree T
$\bar{\lambda}_i$	max failure rate of type i over all environments
d_i	minimum number of components failed of type i needed for system to be down
$\tau_\mathrm{h}, \tau_\mathrm{n}, \tau_\mathrm{r}$	height, node, and rate thresholds, resp.
α	one factor in rate threshold
β	second factor in rate threshold
f_i	number of type-i comps failing in a transition on approximate most likely path to failure
G	weighted graph (V, E, W) used to build tree to approximate most likely path to failure

• The authors are affiliated with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, 07102. E-mail: marvin@njit.edu

P'	approximate transition probability matrix
Q''	CTMC generator matrix when trees omitted and corrected diagonal entries
ϵ_x	difference in diagonal entry (x, x) for Q and Q'
FS, MS	front-end and middle-end servers, resp.
BS, LB	back-end server and load balancer, resp.
HV	hypervisor

1 Introduction

Modern society relies on complex stochastic systems that operate in uncertain environments. These systems can suffer from cascading failures, in which the failure of one part of the system causes other parts to also fail. Examples include networks [13], electric power grids [12], national infrastructures [27], and transportation and communication systems [43]. Cascading failures in these systems can be catastrophic, causing widespread disruptions and damage.

Cascading failures occur in quite varied and complicated ways. For example, complex interdependencies have led to propagating failures in telecommunications networks initiated by disasters [31] and across infrastructure systems due to blackouts [29]. In a large study of cloud-system outages, [19] determines that cascades often arise from both hardware (e.g., failures corrupting data) and software issues ("killer bugs . . . that simultaneously affect multiple nodes or the entire cluster"). Moreover, [20] conducts another analysis of many recent large-scale service outages in cloud systems, and provides statistics on the frequency and durations of the outages; [20] further identifies prominent causes of cascades, including "cascading bugs" ("one bug simultaneously affects many or all of the redundancies, hence impossibility of failover") and redirected traffic from failed nodes overwhelming healthy ones. In addition, [18] also discusses several major cloud outages, where 25-30% of the machines went down.

In this paper we devise methods for analyzing and approximating a dependability system with cascading failures. We model such a system as a continuous-time Markov chain (CTMC; e.g., Chapter 5 of [35]), where the system is a collection of components operating in a randomly changing environment, and each component can fail and be repaired with specified failure and repair rates. We represent a cascading failure as a tree of probabilistically failing components that fail instantaneously, where the root of the tree is the failing component that triggers the cascade. The root probabilistically causes components from a specified set to fail immediately, with each component in the set having its own component-affected probability. Each of these secondary failures subsequently cause other components to probabilistically fail immediately, and so on. The rate of the resulting tree depends on the failure rate of the root and the component-affected probabilities of the failing and non-failing components in the cascade.

Analyzing such a Markovian dependability model with cascading failures presents significant computational challenges. Even simply building the (infinitesimal) generator matrix Q of the CTMC can be extremely time consuming, and indeed, the amount of work required can grow exponentially in the number of components in the system. For example, consider a simple system with components A, B and C, where the failure of any one component can cause each of the others to fail immediately with certain probabilities. For a CTMC transition (x, y) in which all three components fail instantaneously, there are 9 corresponding cascading-failure trees: A causing B to fail, and then B causes C to fail; A causing B and C to immediately fail; B causing A causing C to fail; and so on. If the different components' failure rates and the component-affected probabilities differ, then each of the 9 trees has a different rate, and computing the (x, y)-entry in Q requires summing the rates of all 9 trees. In general, the number of trees corresponding to a single collection of components instantaneously failing can grow exponentially in the size of the set, with up to m^{m-1} different trees corresponding to a collection of m components failing in a cascade in the worst case [22]. Moreover, we need to examine each possible set of components that can immediately fail, and the number of such sets is exponential in the number of components in the system. Thus, merely constructing the generator matrix Q for the CTMC presents significant computational hurdles.

The paper [22] considers the same dependability model and provides an algorithm to build Q, but that algorithm often requires enormous runtimes to generate and solve larger models. We now devise new algorithms, which are significantly more efficient and can decrease the computational effort by orders of magnitude. We have developed software implementing our techniques in Java, and we call the package the Dependability Evaluator of Cascading Failures (DECaF). DECaF reads in a user-created input file specifying a model's basic building blocks (e.g., component types, redundancies, failure and repair rates, component-affected probabilities), from which DECaF builds the CTMC and solves it analytically for various dependability measures.

Previous tree-generation algorithms exist (e.g., Section 2.3.4.4 of [24]) for enumerating all possible trees in which there are no limitations on a node's possible children. But in our model, we restrict the children a particular node can have, which is why [22] and the current paper needed to develop new tree-generation algorithms.

In addition, because the time to construct Q inherently grows exponentially because of the trees to build, we also propose efficient approximations to reduce the computational effort by generating only a subset of the trees. Our method exploits the idea of the most likely paths to failure, which has been previously utilized to design provably effective quick simulation methods based on importance sampling for analyzing systems with highly reliable components, i.e., failure rates are much smaller than repair rates [23], [33]. We apply the concept to try to generate only the cascading-failure trees that arise on the most likely paths to failure, because these paths typically contribute most to the dependability measures computed, and leave out those trees on significantly less likely paths. The omission leads to an approximate generator matrix, which incurs errors in the resulting dependability measures. We explore the trade-off of the time savings from skipping trees with the error in the dependability measures. We also present numerical results demonstrating that for a large model with significant amounts of cascading possible,

our techniques can reduce computation time by orders of magnitude while incurring only small error.

The rest of the paper has the following layout. Section 2 reviews related work on dependability models, with a particular focus on cascading failures and other component interactions. We describe the mathematical model in Section 3. Section 4 contains our new algorithms to build the CTMC's generator matrix and presents numerical results comparing its runtime to that of the implementation in [22]. Section 4.1 discusses dependability measures, including two new ones related to the cascade-size distribution. In Section 5 we develop the approximations that reduce runtime by instead building a model based on only a subset of the trees, which introduces inaccuracies in the dependability measures, and we explore the trade-off through experiments. We apply our methods to a large cloud-computing model in Section 6, and Section 7 provides some concluding remarks. An appendix gives a detailed example showing how our algorithms compute the rate of a tree.

2 Related Work

As mentioned in Section 1, the model we analyze was previously studied in [22], but our new algorithms can solve large models with orders-of-magnitude reductions in runtime. In addition, the current paper devises approximations to further substantially reduce computation times while incurring only small error; [22] does not consider such approximations. The SAVE (System Availability Estimator) package [4], developed at IBM, analyzes a similar Markovian dependability model with cascading failures having the restriction that there is only one level of cascading; i.e., the root of a tree can cause other components to immediately fail, but those subsequent failures cannot cause further instantaneous failures. Allowing for more than a single level of cascading makes the CTMC model we consider tremendously more difficult to construct.

Other modeling techniques, such as fault trees, reliability block diagrams (RBDs), and reliability graphs, have also been applied to study dependability systems, but these approaches do not allow for the level of details that are possible with CTMCs [32]. However, a notable drawback of CTMCs is the explosive growth in the size of the state space, which increases exponentially in the number of components in the system. Other packages for analyzing Markovian dependability models include SHARPE [36], SURF [11], SURF-2 [2], TANGRAM [3], and HIMAP [25].

Instead of assuming a CTMC model, some packages work with other mathematical models, such as stochastic Petri nets (SPNs), which are analyzed by SNPN [21]. OpenSESAME [39] also solves SPNs described via a high-level modeling language, which allows for cascading failures through failure dependency diagrams, but the complexity of the cascades that can be handled is not as great as in our model. The Galileo package [37] examines dynamic fault trees (DFTs), which can model certain types of cascading failures via functional dependency (FDEP) gates. One limitation of the FDEP gate is that it appears to allow for only deterministic cascading; i.e., the failure of one component deterministically causes other components to fail. Our framework permits probabilistic cascading, where the failure of one component causes other components to fail, each with a given probability. Moreover, modeling a cascading failure with a DFT FDEP gate can lead to an ambiguity in how a cascade progresses, complicating the construction of a CTMC model of the dependability system. For example, consider a system with components A and B. Suppose that the failure of A can immediately cause B to fail, and also that the failure of B can immediately cause A to fail. Then in moving from a state with both A and B up to a state with both failed, there are two possible ways in which this can occur: A first fails, causing B to fail immediately; and B first fails, causing A to fail immediately. A CTMC model needs to explicitly consider both possibilities, as is done in [22] and as we do in our development, but the DFT does not. The paper [6] instead uses an input/output interactive Markov chain (I/O-IMC) to formalize DFT, producing a continuous-time Markov decision process (CTMDP). An advantage of the approach in [6] is that the resulting I/O-IMC may be smaller than the corresponding CTMC. But analysis of a CTMDP provides only bounds for dependability measures rather than their exact values, as one can get by solving a CTMC.

The paper [26] considers Bayesian networks for studying reliability systems. The software tool RADYBAN [30] includes a generalization of the DFT FDEP gate called a probabilistic dependency (PDEP) gate, which allows for a type of probabilistic cascading failure that seems to differ from ours. Specifically, suppose that when a component of type A fails, it can cause a component of type B and a component of type C to fail instantaneously. In the PDEP gate, A causes B and C to both fail with a single specified probability. In our framework, if A fails, the immediate failures of B and C are independent events, each occurring with its own probability. Also, RADYBAN converts a DFT into a dynamic Bayesian network to compute reliability measures.

Other mathematical modeling techniques that allow some forms of cascading failures or component interactions include Boolean driven Markov processes (BDMP) [7], common-cause and common-mode failures [1], [8], and coverage [15]. DRBD [41] uses dynamic reliability block diagrams, which extend traditional RBDs to allow for certain component interactions.

3 Model

We now describe the mathematical model. We work with the stochastic model of [22], which considers the evolution over time of a repairable dependability system operating in a randomly changing environment. We start by explaining the basic building blocks of the model, which we then use to define a CTMC. The system consists of a collection $\Omega = \{1, 2, \ldots, N\}$ of $N < \infty$ component types. Each component type $i \in \Omega$ has a redundancy $1 \leqslant r_i < \infty$, and the r_i components of type i are assumed to be identical. A component can be either operational (up) or failed (down).

Environments

The environment changes randomly within a set $\mathcal{E} = \{0, 1, 2, \ldots, L\}$. For example, the environment might represent the current load on the system, and if there are

two possible environments, 0 and 1, then 0 (resp., 1) may represent a low (resp., high) load. Once the environment enters $e \in \mathcal{E}$, it remains there for an exponentially distributed amount of time with rate $\nu_e > 0$, after which the environment changes to e' with probability $\delta_{e,e'} \geqslant 0$, where $\delta_{e,e} = 0$ and $\sum_{e' \in \mathcal{E}} \delta_{e,e'} = 1$. We assume the matrix $\delta = (\delta_{e,e'} : e, e' \in \mathcal{E})$ is irreducible; i.e., for each $e, e' \in \mathcal{E}$, there exists $k \geqslant 1$ and a sequence $e_0 = e, e_1, e_2, \ldots, e_k = e'$ with each $e_i \in \mathcal{E}$ such that $\prod_{i=0}^{k-1} \delta_{e_i,e_{i+1}} > 0$. In other words, it is possible to eventually move from environment e to environment e'.

Failure and Repair Rates

The components in the system can randomly fail and then be repaired. When the environment is $e \in \mathcal{E}$, the failure rate and repair rate of each component of type i are $\lambda_{i,e} > 0$ and $\mu_{i,e} > 0$, respectively. If there is only one environment e, i.e., $|\mathcal{E}| = 1$, then the lifetimes and repair times of components of type i are exponentially distributed with rates $\lambda_{i,0}$ and $\mu_{i,0}$, respectively. Exponential distributions are frequently used to model lifetimes of hardware and software components; e.g., see [40]. We assume that all operating components of a type i have the same failure rate $\lambda_{i,e}$ in environment e. Thus, in a system with redundancies for which not all components of a type are needed for operation of the system, the extras are "hot spares" because they fail at the same rate as the main components.

Cascading Failures

Our model includes probabilistic, instantaneous cascading failures occurring as follows. The ordered set Γ_i specifies the types of components that a failure of a type-i component can cause to immediately fail. When a component of type i fails, it directly causes a single component of type $j \in \Gamma_i$ to fail immediately with probability $\phi_{i,j} > 0$ (if there is at least one component of type j up), and we call $\phi_{i,j}$ a "component-affected probability". The events that the individual components of types $j \in \Gamma_i$ fail immediately are statistically independent. Thus, when a component of type i fails, there are statistically independent "coin flips" to determine which components in Γ_i fail, where the coin flip for $j \in \Gamma_i$ comes up heads (one component of type j fails) with probability $\phi_{i,j}$ and tails (no component of type j fails) with probability $1 - \phi_{i,j}$.

A cascading failure can continue as long as there are still components operational in the system. For example, the failure of a component of type i may cause a component of type j to fail (with probability $\phi_{i,j}$), which in turn makes a component of type k fail (with probability $\phi_{j,k}$), and so on. As noted in [22], the SAVE package [4] allows for only one level of cascading, but the unlimited cascading in our model makes it significantly more difficult to analyze.

We can think of a cascading failure as a tree of instantaneously failing components. The root is the component, say of type i, whose failure triggers the cascade. The root's children, which are from Γ_i, are those components whose immediate failures were directly caused by the root's failure. At any non-root level of the tree, these components' failures were directly caused by the failures of their parents at the previous level. Although all the failing components in a cascade fail at the same time, we need to specify an order in which they fail for our problem to be well-defined, as we explain later in Section 3.2. We assume the components in a tree fail in breadth-first order.

Repair Discipline

There is a single repairman who fixes failed components using a processor-sharing discipline. Specifically, if the current environment is e and there is only one failed component, which is of type i, then the repairman fixes that component at rate $\mu_{i,e}$. If there are b components currently failed, then the repairman allocates $1/b$ of his effort to each failed component, so a failed component of type i is repaired at rate $\mu_{i,e}/b$. (While our model assumes a single repairman, we can easily extend the model to allow for multiple repairmen. On the other hand, instead assuming a first-come-first-served repair discipline would require the Markov chain (see Section 3.1) to keep track of the order in which components fail, leading to a much larger state space.)

3.1 Markov Chain

We want to analyze the behavior of the system as it evolves over time. Because of the processor-sharing repair discipline and the exponential rates for the event lifetimes, it will suffice to define the state of the system as a vector containing the number of failed components of each type and the current environment. Thus, let $S = \{x = (x_1, x_2, \ldots, x_N, x_{N+1}) : 0 \leqslant x_i \leqslant r_i \ \forall i \in \Omega, x_{N+1} \in \mathcal{E}\}$ be the state space, where x_i is the number of failed components of type i in state x and x_{N+1} is the environment. Let $Z = [Z(t) : t \geqslant 0]$ be the CTMC living on S keeping track of the current state of the system. (If we had instead assumed a first-come-first-served repair discipline, then the state space would need to be augmented to keep track of the order in which the current set of down components failed.) We assume that Z starts in environment $0 \in \mathcal{E}$ with no components failed, i.e., state $x_* \equiv (0, 0, \ldots, 0)$. As noted in [22] the CTMC is irreducible and positive recurrent.

Generator Matrix

We now describe the CTMC's (infinitesimal) generator matrix $Q = (Q(x, y) : x, y \in S)$, where $Q(x, y)$ is the rate that the CTMC Z moves from state $x = (x_1, \ldots, x_N, x_{N+1})$ to state $y = (y_1, \ldots, y_N, y_{N+1})$. If $y_i = x_i$ for each $i \in \Omega$ and $y_{N+1} \neq x_{N+1}$, then (x, y) is an "environment transition" with $Q(x, y) = \nu_{x_{N+1}} \delta_{x_{N+1},y_{N+1}}$. If $y_i = x_i - 1$ for one $i \in \Omega$, $y_j = x_j$ for each $j \in \Omega - \{i\}$, and $y_{N+1} = x_{N+1}$, then (x, y) is a "repair transition" corresponding to the repair of a component of type i, and $Q(x, y) = x_i \mu_{i,x_{N+1}} / (\sum_{j \in \Omega} x_j)$. If $y_i \geqslant x_i$ for all $i \in \Omega$ with $y_j > x_j$ for some $j \in \Omega$ and $y_{N+1} = x_{N+1}$, then (x, y) is a "failure transition" in which $y_i - x_i$ components of type i fail, $i \in \Omega$. Any other (x, y) with $x \neq y$ not falling into one of the above three categories is not possible, so $Q(x, y) = 0$. Let Ψ_{e}, Ψ_{r} and Ψ_{f} be the sets of environment, repair, and failure transitions, respectively. Each diagonal entry satisfies $Q(x, x) = -\sum_{y \neq x} Q(x, y)$, as required for a CTMC; e.g., see Chapter 5 of [35].

We now determine the rate $Q(x, y)$ of a failure transition (x, y). First consider the case when cascading failures are not possible, i.e., $\Gamma_i = \varnothing$ for each type i. Then the only

possible failure transitions (x, y) have $y_i = x_i + 1$ for one $i \in \Omega$, $y_j = x_j$ for each $j \in \Omega - \{i\}$, and $y_{N+1} = x_{N+1}$, and this transition corresponds to a single component of type i failing. Then $Q(x, y) = (r_i - x_i)\lambda_{i,x_{N+1}}$.

Generator Matrix When Cascading Failures Possible

Cascading failures complicate the computation of $Q(x, y)$ for a failure transition (x, y). As mentioned before, we model a cascading failure as a tree T built from the multiset B of instantaneously failing components, where B has $y_\ell - x_\ell \geqslant 0$ failing components of type ℓ, $\ell \in \Omega$. A tree T in a transition starting from a state x has a rate

$$R(T) \equiv R(T, x) = (r_i - x_i)\lambda_{i,x_{N+1}}\rho\,\eta, \tag{1}$$

where

- $(r_i - x_i)\lambda_{i,x_{N+1}}$ is the failure rate of the root (assumed here to be of type i),
- $\rho = \rho(T)$ is the product of the $\phi_{j,k}$ terms for a parent node of type j immediately causing a child of type $k \in \Gamma_j$ to fail in the tree T, and
- $\eta = \eta(T, x)$ is the product of $1 - \phi_{j,k}$ terms from a node of type j not causing a component of type $k \in \Gamma_j$ to fail when there are type-k components up.

We provide more details in Section 3.2.

A difficulty arises because there can be many such trees corresponding to the multiset B of components failing in (x, y), and calculating $Q(x, y)$ requires summing $R(T)$ over all possible trees T that can be constructed from B. The number of such trees grows exponentially in the number of failing components in the cascade; see [22].

Our model assumes that the component-affected sets Γ_i and the component-affected probabilities $\phi_{i,j}$ do not depend on the current state x of the system. This may limit our model's appropriateness for certain application domains. But the assumption can greatly reduce the amount of information that the user needs to specify in building a model. Because the state space S may be enormous, requiring the user to instead specify state-dependent $\Gamma_i(x)$ and $\phi_{i,j}(x)$ for each state $x \in S$ quickly becomes intractable. A simplification may be to specify particular functional forms for $\Gamma_i(x)$ and $\phi_{i,j}(x)$ as a function of the state x, but this may also be difficult to do.

3.2 Example of Computing a Tree's Rate

We now provide an example of computing the rate $R(T)$ of a tree T. Let $\Omega = \{A, B, C\}$, with redundancies $r_A = r_B = r_C = 4$. Also, define the component-affected sets $\Gamma_A = \{B, C\}$, $\Gamma_B = \{A, C\}$, and $\Gamma_C = \{A, B\}$. Suppose that the set of environments is $\mathcal{E} = \{0\}$, and consider the failure transition (x, y) with $x = (2, 2, 3, 0)$ and $y = (4, 4, 4, 0)$. Thus, (x, y) corresponds to 2 components each of types A and B failing and a single component of type C failing. One possible tree T corresponding to (x, y) is shown in Figure 1. We assume the nodes in T fail in breadth-first order.

The nodes depicted as double circles form the tree of failing components. The dashed circles correspond to components in some Γ_i but did not fail. A component type j in some Γ_i could have not failed because either there are components of type j up at this point but its coin flip came up tails (with probability $1 - \phi_{i,j}$), or there were no more components of type j up at this point. Each node has a label of the form i-ID, where i denotes the type of the component for that node, and ID is the position of the node in a breadth-first ordering of all the nodes (dashed circles and double circles). We include the IDs to simplify the discussion here. We call the tree of all nodes the "supertree" corresponding to the tree T of failing nodes.

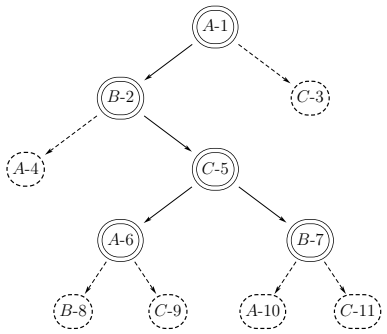

Fig. 1: An example of a supertree.

The supertree is used to compute $R(T)$ of T as follows. Let u_i be the number of components of type i currently up in the system. Because the root is a component of type A and there are $u_A = r_A - x_A = 2$ components of type A at the start of the transition (x, y), the rate of the trigger of the cascade is $2\lambda_{A,0}$. The root then causes a component of type B to fail at node ID 2, and this failure occurs with probability $\phi_{A,B}$. (The failure of A at ID 1 can cause only zero or one B to fail, with respective probabilities $1 - \phi_{A,B}$ and $\phi_{A,B}$, even if there is more than one B up at that point.) The node at ID 3 did not fail, and at this point there are $u_C = r_C - x_C = 1 > 0$ components of type C still up, so this non-failure occurs with probability $1 - \phi_{A,C}$. Instead of stepping through the rest of the supertree one node at a time, we note that $R(T)$ includes the product of all the $\phi_{i,j}$ terms for a type-i parent with a type-j child when both are double circles. Thus, in (1) we have $\rho = \phi_{A,B}\,\phi_{B,C}\,\phi_{C,A}\,\phi_{C,B}$ from IDs 2, 5, 6, and 7, respectively.

We observe the following when calculating the product η in (1):

- $1 - \phi_{i,j}$ factors are included if and only if there are still components of type j up at that point in the breadth-first traversal through the tree.
- Each time we encounter a node of type j that has failed in the breadth-first traversal of T, we decrement u_j by 1.

Keeping these observations in mind, we now calculate η. For component type A, we have $u_A = 2$ before the cascade begins. As we do a breadth-first traversal through T, at ID 1, it decrements to $u_A = 1$. We see that $u_A > 0$ until ID 6, so η includes factor $1 - \phi_{B,A}$ from ID 4, but η does not include the factor $1 - \phi_{B,A}$ at ID 10 as type A has been exhausted before that point. For component type B, we have $u_B = r_B - x_B = 2$ before we traverse through T. Because components of type B are exhausted at ID 7, we do not include the factor $1 - \phi_{A,B}$ at ID 8 in η. For component type C, we see that $u_C = 1$ before we traverse through

T, and $u_C = 0$ at ID 5. Hence, the only contribution to η from a component of type C not failing is $1 - \phi_{A,C}$ from ID 3; we do not include the factors $1 - \phi_{A,C}$ and $1 - \phi_{B,C}$ from IDs 9 and 11, respectively. Taking the product over all component types yields $\eta = (1-\phi_{A,C})\,(1-\phi_{B,A})$ from IDs 3 and 4. Therefore, $R(T)$ is $2\lambda_{A,0}\,\rho\,\eta$. In our implementation, we calculate η through a data structure called the breadth-first history (BFH), which is described later in Section 4.

We previously stated that the order in which the components fail in a tree must be specified for the tree's rate to be well defined. To see why, suppose instead that the components in Figure 1 fail in depth-first order. The depth-first traversal of T is A-1, B-2, A-4, C-5, A-6, B-8, C-9, B-7, A-10, C-11, C-3. Zero time elapses for the entire tree to occur, but the depth-first traversal specifies the ordering of the nodes. Initially, the number of components up of each type are $u_A = 2$, $u_B = 2$ and $u_C = 1$, as before. In this traversal a component of type C first fails at ID 5, which makes $u_C = 0$. Thus, η for the depth-first traversal does not include the factors $1-\phi_{A,C}$, $1-\phi_{B,C}$ and $1-\phi_{A,C}$ from the subsequent type-C nodes at IDs 9, 11 and 3, respectively. In contrast, the breadth-first traversal includes one $1 - \phi_{A,C}$ factor at ID 3. Moreover, the depth-first traversal also includes a factor $1 - \phi_{A,B}$ at ID 8, which is not included in the breadth-first traversal. Overall, the depth-first traversal has $\eta = (1 - \phi_{B,A})(1 - \phi_{A,B})$ from IDs 4 and 8, as opposed to $\eta = (1 - \phi_{A,C})(1 - \phi_{B,A})$ for the breadth-first traversal. Thus, even though the components in a cascading tree fail instantaneously, this example demonstrates the necessity of defining an order in which they fail for the tree rate (and the CTMC) to be well-defined, as η (but not ρ) depends on the order.

4 Algorithms to Construct the Exact Generator Matrix

We now provide efficient algorithms for generating all possible trees and constructing the exact generator matrix Q of the CTMC. A tree corresponds to a multiset of particular components failing, and cascading failures starting from different states can have the same multiset of components failing. Hence, a particular tree may correspond to several different transitions (x, y). Our algorithm generates each possible tree only once and determines all the transitions to which this tree corresponds. The approach avoids generating the same tree numerous times for each corresponding transition, as was originally done in [22]. Moreover, rather than building each new tree from scratch, as was done in [22], our current algorithm builds larger trees from smaller ones already considered, leading to substantial additional savings in the overall computational effort.

Computing the rate (1) of a given tree depends on the state from which the cascading failure began and the multiset of components that fail in the cascade. We do not actually construct the supertree in our algorithm to compute a tree's rate, but instead build a data structure called a breadth-first history to keep track of the information necessary to compute η in (1). The breadth-first history concisely recounts the creation of the tree and thus allows us to obtain η without having to build supertrees per transition as was done in [22]. All of the computations are done in Algorithms 1 (`SeedTrees`), 2 (`AddTreeLevel`) and 3 (`ComputeTreeRate`), which we describe below. In the Appendix we re-examine the example tree from Section 3.2 to show how the breadth-first history is constructed and used to evaluate η.

`SeedTrees` starts the tree generation and initializes the necessary data structures for `AddTreeLevel`. `AddTreeLevel` introduces a new level to an existing tree in a recursive fashion, updates ρ in (1) to include the component-affected probabilities of failed components, and builds the tree's breadth-first history. `ComputeTreeRate` calculates the rate of a completed tree for all the transitions it corresponds to using ρ and η computed from the breadth-first history populated in `AddTreeLevel`. Later references to line numbers in the algorithms are given within angled brackets $\langle\ \rangle$. Section 4.2 will work through an example, including constructing BFH, using Algorithms 1 and 2.

Algorithm 1: `SeedTrees`

```
   Input: Γ
   // an array of ordered sets that
      describes which components can cause
      which other components to fail
1  for root.type ∈ Ω do
2  |  level = [ ];
   |  // Dynamic array of failed components
   |     at tree's current bottom level;
   |     initially empty
3  |  nFailed = [0, 0, ..., 0];
   |  // Array that counts failed
   |     components of each type in the
   |     tree
4  |  BFH = [( ), ( ), ..., ( )];
   |  // Array of linked lists that keeps a
   |     history of parent component types
   |     in breadth-first order; BFH is
   |     indexed by component types; each
   |     linked list is initially empty
5  |  add root.type to level;
6  |  nFailed [root.type ] = 1;
7  |  add @ to BFH [root.type ];
   |  // Signifies one component of type
   |     root.type has failed
8  |  ρ = 1;
   |  // initialize product of
   |     component-affected probabilities
9  |  if Γ_nFailed[root.type] == ∅ then
10 |  |  ComputeTreeRate(nFailed,BFH,ρ,root.type );
11 |  else
12 |  |  AddTreeLevel(level,nFailed,BFH,ρ,root.type );
```

4.1 Dependability Measures

Once the generator matrix Q has been constructed, we can use it to compute various dependability measures. We first partition the state space $S = U \cup F$, where U (resp., F) is the set of states for which the system is operational

Algorithm 2: `AddTreeLevel`

Input: level, nFailed, BFH, ρ, root.type

```
// where level is the current level of failed components, nFailed counts failed components by type in the tree, BFH is breadth-first history, ρ is a cumulative product of component-affected probabilities, root.type is the root component's type in the current tree
```

1 $\text{nextLevelPossibilities} = \bigtimes_{\substack{i=1:\\ \Gamma_{\text{level}[i]} \neq \emptyset}}^{|\text{level}|} \mathcal{P}(\Gamma_{\text{level}[i]})$;

```
// Builds all possibilities for the next level (given the current level) by taking a Cartesian product of the power sets 𝒫 of non-empty Γ sets of failed nodes in the current level
```

2 for oneNextLevelPossibility ∈ nextLevelPossibilities do
3 addedAChildFlag = False;
4 validTree = True;
5 for parent ∈ level do
6 for $i \in \Gamma_{\text{parent.type}}$ do
7 if ∃ child ∈ oneNextLevelPossibility : child.type == i && child.parentID == parent.ID && validTree then
8 addedAChildFlag = True;
9 validTree = True;
10 if nFailed[child.type] == $r_{\text{child.type}}$ then
// Invalid tree, it requires more components of type child.type than available
11 validTree = False;
12 if validTree then
13 nFailed [child.type] = nFailed [child.type] + 1;
14 add @ to BFH[child.type];
// @ denotes a component of type child.type has failed
15 $\rho = \rho \cdot \phi_{\text{parent.type, child.type}}$;
// Update rate with appropriate component-affected probability
16 else if validTree then
17 add parent.type to BFH[child.type];
// One component of type child.type has not failed, but was present in $\Gamma_{\text{parent.type}}$
18 if validTree then
19 if addedAChildFlag then // Tree can be grown further
20 AddTreeLevel(oneNextLevelPossibility, nFailed, BFH, ρ, root.type);
21 else // Current tree is complete because it cannot be grown further
22 ComputeTreeRate(nFailed, BFH, ρ, root.type);

Algorithm 3: `ComputeTreeRate`

Input: nFailed, BFH, ρ, root.type

```
// where nFailed is the number of failed components of each type in the tree, BFH is breadth-first history, ρ is a cumulative product of component-affected probabilities, root.type is the root's type in the tree
```

1 for $x' \in S'$ do
2 $\eta = 1$;
// Cumulative product of complement probabilities of components that could have failed but did not
3 for $i \in \Omega$ do
4 $u_i = r_i - x'_i$;
5 for parent.type ∈ BFH[i] do
6 if parent.type == @̄ then
7 $u_i = u_i - 1$;
8 else if $u_i > 0$ then
9 $\eta = \eta \cdot (1 - \phi_{\text{parent.type}, i})$;
// need $u_i > 0$ or else there cannot be any more failed nodes of type i
10 for $e \in \mathcal{E}$ do
11 $x = (x', e)$;
12 $y = (x' + \text{nFailed}, e)$;
13 if y is a valid state then
14 $Q(x, y) = (r_{\text{root.type}} - x[\text{root.type}]) \cdot \lambda_{\text{root.type},e} \cdot \rho \cdot \eta$;

(resp., failed). We assume that the initial state $x_* \in U$ and that $F \neq \emptyset$. The partition is determined by a model specification giving conditions under which the system is considered to be operational; e.g., at least υ_i components of type i are up for each type $i \in \Omega$.

4.1.1 Steady-State Unavailability

One dependability measure is the steady-state unavailability (SSU), which we define as follows. Let $\pi = (\pi(x) : x \in S)$ be the nonnegative row vector defined such that $\pi Q = 0$ and $\pi e = 0$, where e is the column vector of all 1s; i.e., π is the steady-state probability vector of the CTMC; e.g., see Chapter 5 of [35]. The vector π exists and is unique because, as shown in [22], our CTMC is irreducible and positive recurrent. We then define the SSU as $\sum_{x \in F} \pi(x)$, which is the long-run fraction of time the CTMC is in F.

4.1.2 Mean Time to Failure

Another dependability measure is the mean time to failure (MTTF), which can be defined as follows. Define $T_F = \inf\{t > 0 : Z(t) \in F\}$, so the MTTF is $\mathbf{E}[T_F \mid Z(0) = x_*]$, where $\mathbf{E}$ denotes statistical expectation. We can compute the MTTF in terms of Q as follows. Define the transition probability matrix $P = (P(x, y) : x, y \in S)$ of the embedded discrete-time Markov chain (DTMC) $Y = [Y_n : n = 0, 1, 2, \ldots]$ with $P(x, y) = -Q(x, y)/Q(x, x)$

for $x \neq y$, and $P(x,x) = 0$ (Chapter 5 of [35]). Also define the $|U| \times |U|$ matrix $P_U = (P(x,y) : x, y \in U)$ and $|U| \times |U|$ identity matrix I, and let $h = (h(x) : x \in U)$ be the column vector such that $h(x) = -1/Q(x,x)$, which is the mean holding time that the CTMC spends in each visit to state x. Because Y is irreducible, $|S| < \infty$, and $F \neq \varnothing$, we have that $I - P_U$ is nonsingular. Then let $m = (I - P_U)^{-1}h$, where $(I - P_U)^{-1}$ is known as the "fundamental matrix" of the DTMC, and the MTTF equals $m(x_*)$; e.g., see Section 7.9 of [38].

4.1.3 Steady-State Distribution of Cascade Size

We introduce a new dependability measure, the "steady-state distribution of cascade size" (SSDCS). To define the SSDCS, recall that Ψ_{f} is the set of failure transitions. For $(x,y) \in \Psi_{\mathrm{f}}$, let $a(x,y) = \sum_{i=1}^{N}(y_i - x_i)$ be the total number of components (of all types) that fail in transition (x,y). Also, let $a(x,y) = 0$ for a non-failure transition $(x,y) \notin \Psi_{\mathrm{f}}$. The maximum number of components failing in a cascade is $b = \sum_{i=1}^{N} r_i$. For each integer $1 \leqslant l \leqslant b$, let $\Psi_{\mathrm{f}}(l) = \{(x,y) \in \Psi_{\mathrm{f}} : a(x,y) = l\}$, which is the set of failure transitions in which exactly l components fail. Let $\bar{S} = \{x = (x_1, \ldots, x_N, x_{N+1}) \in S : x_i < r_i \text{ for some } i = 1, 2, \ldots, N\}$, which is the set of states having at least one nonfailed component. Define $\xi = (\xi(x) : x \in S)$ as the nonnegative row vector with $\xi(x) = \pi(x)Q(x,x)/[\sum_{y \in S} \pi(y)Q(y,y)]$ for each $x \in S$, so ξ is the steady-state distribution of the embedded DTMC Y; i.e., $\xi P = \xi$, $\xi e = 1$, and $\xi \geqslant 0$. Then we have the following:

Theorem 1. The SSDCS $\theta = (\theta(l) : 1 \leqslant l \leqslant b)$ satisfies

$$\theta(l) = \frac{1}{\sum_{w \in \bar{S}} \xi(w)} \sum_{x \in \bar{S}} \xi(x) \frac{\sum_{(x,y) \in \Psi_f(l)} P(x,y)}{\sum_{(x,z) \in \Psi_f} P(x,z)}. \quad (2)$$

Proof. Let H be a random variable denoting the number of failing components in a cascade in steady state. In any state $x \in S$ such that $x_i = r_i$ for all $i = 1, 2, \ldots, N$, no components are operational, so there cannot be any cascades out of such a state x. Thus, a cascade (possibly with just a single component) can only start from a state in $\bar{S}$. Define the row vector $\bar{\xi} = (\bar{\xi}(x) : x \in \bar{S})$ with $\bar{\xi}(x) = \xi(x)/[\sum_{y \in \bar{S}} \xi(y)]$, which is the steady-state distribution of the DTMC conditioned to lie in $\bar{S}$. Let $P_{\bar{\xi}}$ be the conditional probability measure, given that the initial state Y_0 of the DTMC is chosen using distribution $\bar{\xi}$. For $1 \leqslant l \leqslant b$, we have that $\theta(l) = P(H = l)$ satisfies

$$\begin{aligned} \theta(l) &= P_{\bar{\xi}}((Y_0, Y_1) \in \Psi_{\mathrm{f}}(l) \,|\, (Y_0, Y_1) \in \Psi_{\mathrm{f}}) \\ &= \sum_{x \in \bar{S}} \bar{\xi}(x) \frac{\sum_{(x,y) \in \Psi_{\mathrm{f}}(l)} P(x,y)}{\sum_{(x,z) \in \Psi_{\mathrm{f}}} P(x,z)}, \end{aligned} \quad (3)$$

from which (2) follows. □

4.1.4 Distribution of Cascade Size Until Failure

We next introduce another new dependability measure $\chi = (\chi(l) : 1 \leqslant l \leqslant b)$, which we call the "distribution of cascade size until failure" (DCSUF). For $1 \leqslant l \leqslant b$, let J_l be the number of cascades of size exactly l until the system first fails. Specifically, let $T'_F = \inf\{n \geqslant 0 : Y_n \in F\}$, which is the number of transitions that the DTMC Y takes to first enter F. Let $\mathcal{I}(\cdot)$ denote the indicator function, which takes on value 1 (resp., 0) when its argument is true (resp., false). For $1 \leqslant l \leqslant b$, we have that

$$J_l = \sum_{n=1}^{T'_F} \mathcal{I}(a(Y_{n-1}, Y_n) = l).$$

Also, let $J = \sum_{l=1}^{b} J_l$ be the total number of cascades (of any size) until the system first fails. We then define the distribution χ of cascade size until failure, given the DTMC starts in state x_*, with

$$\chi(l) = \frac{\mathbf{E}[J_l \,|\, Y_0 = x_*]}{\mathbf{E}[J \,|\, Y_0 = x_*]}. \quad (4)$$

Thus, $\chi(l)$ is the fraction of the expected number of cascades until failure that have size exactly l.

We next derive a computable expression for $\chi(l)$. Let

$$\zeta_l(x) = \mathbf{E}\left[\, J_l \mid Y_0 = x \,\right], \quad (5)$$

which is the conditional expectation of J_l, given the DTMC starts in state $x \in U$. Then we have the following result.

Theorem 2. The DCSUF $\chi = (\chi(l) : 1 \leqslant l \leqslant b)$ has

$$\chi(l) = \frac{\zeta_l(x_*)}{\sum_{i=1}^{b} \zeta_i(x_*)}, \quad (6)$$

where $\zeta_l = (\zeta_l(x) : x \in U)$ satisfies

$$\zeta_l = (I - P_U)^{-1} \kappa_l, \quad (7)$$

with $\kappa_l = (\kappa_l(x) : x \in U)$ and

$$\kappa_l(x) = \sum_{\substack{z \in S: \\ (x,z) \in \Psi_f(l)}} P(x,z). \quad (8)$$

Proof. By conditioning on the first step of the DTMC Y, we can express (5) as

$$\begin{aligned} \zeta_l(x) &= \sum_{y \in U} P(x,y) \left[\mathcal{I}(a(x,y) = l) + \zeta_l(y)\right] \\ &\quad + \sum_{z \in F} P(x,z) \mathcal{I}(a(x,z) = l) \\ &= \sum_{y \in U} P(x,y) \zeta_l(y) + \sum_{z \in S} P(x,z) \mathcal{I}(a(x,z) = l) \end{aligned} \quad (9)$$

because $S = U \cup F$. Note that $\kappa_l(x)$ in (8) is the probability of having a cascade of exactly size l from state x, which we can write as

$$\kappa_l(x) = \sum_{z \in S} P(x,z) \mathcal{I}(a(x,z) = l).$$

Then we can express (9) in matrix form as $\zeta_l = P_U \zeta_l + \kappa_l$, or equivalently, $(I - P_U)\zeta_l = \kappa_l$. We previously argued (Section 4.1.2) that $I - P_U$ is nonsingular, so (7) holds. Thus, in (4), the numerator is $\mathbf{E}[J_l \,|\, Y_0 = x_*] = \zeta_l(x_*)$, and the denominator is

$$\mathbf{E}[J \,|\, Y_0 = x_*] = \sum_{l=1}^{b} \mathbf{E}[J_l \,|\, Y_0 = x_*] = \sum_{l=1}^{b} \zeta_l(x_*).$$

Therefore, we obtain (6) to complete the proof. □

4.2 Example Demonstrating Tree Generation

We now provide an example illustrating our tree-generation algorithms. We consider a system with $\Omega = \{A, B, C\}$, so there are $N = 3$ types of components, with redundancies $r_A = r_B = 4$, and $r_C = 1$. There is a single environment, i.e., $\mathcal{E} = \{0\}$, so the system has $(r_A + 1)(r_B + 1)(r_C + 1)|\mathcal{E}| = 50$ states. The component repair rates are $\mu_{A,0} = \mu_{B,0} = \mu_{C,0} = 1$, and components B and C have failure rates $\lambda_{B,0} = 2\text{E}{-}4$ and $\lambda_{C,0} = 1\text{E}{-}10$. For cascading, the component-affected sets are $\Gamma_A = \{B, C\}, \Gamma_B = \{A\}, \Gamma_C = \varnothing$, and $\phi_{A,C} = 1\text{E}{-}08$. For the other parameters ($\lambda_{A,0}$, $\phi_{A,B}$, and $\phi_{B,A}$), we considered three versions of the model, called Cases 1–3, that differ in their values, which are given at the top of Table 1. We chose the cases' parameter values to illustrate other aspects of our approaches, as we will see later in Section 5.4.

The structure of the trees built by Algorithms 1 and 2 is described in Table 1, between the two sets of horizontal double lines and to the left of the vertical double lines. (The other parts of the table will be explained in Section 5.4.) Each row with depth equal to 0 corresponds to a new iteration of the loop in line ⟨1⟩ of Algorithm 1, which starts building a new tree with a particular root type. After initializing the data structures in lines ⟨2⟩–⟨8⟩, Algorithm 1 then calls Algorithm 2 in ⟨12⟩ to further build the tree. Each row of Table 1 represents one iteration of the outer loop (line ⟨2⟩) of Algorithm 2, which recursively constructs trees by adding a new level onto a previously built tree, increasing the depth by 1. For each tree, the column labeled "Nodes" in the table gives the nodes corresponding to the components belonging to the component-affected sets of the failed nodes from the previous level. Nodes that represent components that have not failed, but belong to the component-affected set Γ_i of a node of type i in the previous level, are prefaced with "-". For each tree, there is one additional hidden level one level deeper where no components have failed. These levels are not depicted in the table, and do not contribute to the structure of the failed components in each tree. But they are important for calculating the tree's exact rate in (1) because they may contribute $1-\phi_{i,j}$ factors to η from non-failing components. As previously noted in the first paragraph of Section 4, each constructed tree may correspond to several (x, y) transitions of the CTMC, and the column "Trees Eval." in Table 1 gives the number of transitions in which each tree is used. For example, tree $t = 2$ corresponds to 20 different transitions in the CTMC's generator matrix.

A tree with depth 0 is a tree with a single node, and it is not built from any previous tree. The node in such a tree will be the root for any tree that is built from it. For example, in Algorithm 1, the first iteration of the loop at line ⟨1⟩ starts building a tree with root of type A, which is at depth 0 in the tree; ⟨3⟩ and ⟨4⟩ initialize $\mathsf{nFailed}[i] = 0$ and $\mathsf{BFH}[i] = ()$ for each component type i; ⟨6⟩ sets $\mathsf{nFailed}[A] = 1$; ⟨7⟩ sets $\mathsf{BFH}[A] = (@)$ to denote a type-A component failed; ⟨8⟩ initializes $\rho = 1$; and ⟨12⟩ calls `AddTreeLevel` to try to further grow the tree.

Then in Algorithm 2, line ⟨1⟩ builds all subsets of $\Gamma_A = \{B, C\}$ as the possible children (at depth 1) of the root, where each subset is considered separately in the loop at

				Case 1	Case 2	Case 3
			$\lambda_{A,0}$	0.002	0.002	0.02
			$\phi_{A,B}$	0.06	0.042	0.0042
			$\phi_{B,A}$	0.08	0.088	0.0088
			τ_{r}	4.82E-13	1.39E-08	2.00E-02
Tree t	Depth	Nodes	Trees Eval.	$R'(T_t)$	$R'(T_t)$	$R'(T_t)$
1	0	A	40	2.00E-03*	2.00E-03*	2.00E-02*
2	1	-B, C	20	2.00E-11*	2.00E-11	2.00E-10
3	1	B, -C	32	1.20E-04*	8.40E-05*	8.40E-05
4	2	A	24	9.60E-06*	7.39E-06*	7.39E-07
5	3	-B, C	12	9.60E-14	7.39E-14	7.39E-15
6	3	B, -C	18	5.76E-07*	3.10E-07*	3.10E-09
7	4	A	12	4.61E-08*	2.73E-08*	2.73E-11
8	5	-B, C	6	4.61E-16	2.73E-16	2.73E-19
9	5	B, -C	8	2.76E-09*	1.15E-09	1.15E-13
10	6	A	4	2.21E-10*	1.01E-10	1.01E-15
11	7	-B, C	2	2.21E-18	1.01E-18	1.01E-23
12	7	B, -C	2	1.33E-11*	4.24E-12	4.24E-18
13	7	B, C	1	1.33E-19	4.24E-20	4.24E-26
14	5	B, C	4	2.76E-17	1.15E-17	1.15E-21
15	6	A	2	2.21E-18	1.01E-18	1.01E-23
16	7	B, -C	1	1.33E-19	4.24E-20	4.24E-26
17	3	B, C	9	5.76E-15	3.10E-15	3.10E-17
18	4	A	6	4.61E-16	2.73E-16	2.73E-19
19	5	B, -C	4	2.76E-17	1.15E-17	1.15E-21
20	6	A	2	2.21E-18	1.01E-18	1.01E-23
21	7	B, -C	1	1.33E-19	4.24E-20	4.24E-26
22	1	B, C	16	1.20E-12*	8.40E-13	8.40E-13
23	2	A	12	9.60E-14	7.39E-14	7.39E-15
24	3	B, -C	9	5.76E-15	3.10E-15	3.10E-17
25	4	A	6	4.61E-16	2.73E-16	2.73E-19
26	5	B, -C	4	2.76E-17	1.15E-17	1.15E-21
27	6	A	2	2.21E-18	1.01E-18	1.01E-23
28	7	B, -C	1	1.33E-19	4.24E-20	4.24E-26
29	0	B	40	2.00E-04*	2.00E-04*	2.00E-04
30	1	A	32	1.60E-05*	1.76E-05*	1.76E-06
31	2	-B, C	16	1.60E-13	1.76E-13	1.76E-14
32	2	B, -C	24	9.60E-07*	7.39E-07*	7.39E-09
33	3	A	18	7.68E-08*	6.50E-08*	6.50E-11
34	4	-B, C	9	7.68E-16	6.50E-16	6.50E-19
35	4	B, -C	12	4.61E-09*	2.73E-09	2.73E-13
36	5	A	8	3.69E-10*	2.40E-10	2.40E-15
37	6	-B, C	4	3.69E-18	2.40E-18	2.40E-23
38	6	B, -C	4	2.21E-11*	1.01E-11	1.01E-17
39	7	A	2	1.77E-12*	8.89E-13	8.89E-20
40	8	-B, C	1	1.77E-20	8.89E-21	8.89E-28
41	6	B, C	2	2.21E-19	1.01E-19	1.01E-25
42	7	A	1	1.77E-20	8.89E-21	8.89E-28
43	4	B, C	6	4.61E-17	2.73E-17	2.73E-21
44	5	A	4	3.69E-18	2.40E-18	2.40E-23
45	6	B, -C	2	2.21E-19	1.01E-19	1.01E-25
46	7	A	1	1.77E-20	8.89E-21	8.89E-28
47	2	B, C	12	9.60E-15	7.39E-15	7.39E-17
48	3	A	9	7.68E-16	6.50E-16	6.50E-19
49	4	B, -C	6	4.61E-17	2.73E-17	2.73E-21
50	5	A	4	3.69E-18	2.40E-18	2.40E-23
51	6	B, -C	2	2.21E-19	1.01E-19	1.01E-25
52	7	A	1	1.77E-20	8.89E-21	8.89E-28
53	0	C	25	1.00E-10*	1.00E-10	1.00E-10
			MTTF(Q)	1.51E+08	2.28E+08	2.84E+05
			MTTF(Q')/MTTF(Q)	1.000214	1.189983	1.032338
			MTTF(Q'')/MTTF(Q)	1.000000	1.138655	0.936967
			SSU(Q)	8.93E-09	5.73E-09	3.61E-06
			SSU(Q)/SSU(Q')	1.000367	1.231557	1.037857
			SSU(Q)/SSU(Q'')	1.000049	1.173329	0.941876
			Trees: unique (eval.)	19 (341)	9 (240)	1 (40)

TABLE 1: Example with three cases to illustrate the exact tree-generation methods in Algorithms 1 and 2, and the computed dependability measures for the exact generator matrix Q. Other aspects of the table, including τ_{r}, $R'(T_t)$, Q', and Q'', will be explained later in Section 5.4.

$\langle 2 \rangle$. In Table 1, the subset $\varnothing$ (resp., $\{C\}$, $\{B\}$, and $\{B, C\}$) of Γ_A corresponds to row $t = 1$ (resp., 2, 3, and 22). We next explain how Algorithm 2 handles each of the subsets of Γ_A to be added at depth 1, including updating BFH.

- For the subset $\varnothing \subseteq \Gamma_A$, the first iteration of the loop in line $\langle 6 \rangle$ considers $B \in \Gamma_A$, and $\langle 17 \rangle$ updates $\mathsf{BFH}[B] = (A)$ to denote that a type-B component (with parent type A) did not fail. The next iteration of the loop in $\langle 6 \rangle$ considers $C \in \Gamma_A$, and $\langle 17 \rangle$ updates $\mathsf{BFH}[C] = (A)$ to denote that a type-C component (with parent type A) did not fail. As this tree cannot be further grown, $\langle 22 \rangle$ calls `ComputeTreeRate`.
- For the subset $\{C\} \subseteq \Gamma_A$, the first iteration of the loop in $\langle 6 \rangle$ considers $B \in \Gamma_A$, and $\langle 17 \rangle$ updates $\mathsf{BFH}[B] = (A)$ to denote that a type-B component (with parent type A) did not fail. The next iteration of the loop in $\langle 6 \rangle$ considers $C \in \Gamma_A$. Because a type-C component fails (at depth 1) in the current subset, $\langle 13 \rangle$ increments $\mathsf{nFailed}[C]$ to 1, $\langle 14 \rangle$ updates $\mathsf{BFH}[C] = (@)$ to denote that a type-C component failed, and $\langle 15 \rangle$ updates ρ by multiplying it by $\phi_{A,C}$.
- The subset $\{B\} \subseteq \Gamma_A$ is handled similarly, but instead with $\mathsf{nFailed}[B]$ incremented to 1, $\mathsf{BFH}[B] = (@)$, ρ is multiplied by $\phi_{A,B}$, and $\mathsf{BFH}[C] = (A)$.
- For the subset $\{B, C\} \subseteq \Gamma_A$, we instead have both $\mathsf{nFailed}[B]$ and $\mathsf{nFailed}[C]$ incremented to 1, $\mathsf{BFH}[B] = (@)$, $\mathsf{BFH}[C] = (@)$, and ρ multiplied by $\phi_{A,B}\phi_{A,C}$.

Because at least one node was added (at depth 1) to the tree for each of the last three subsets, line $\langle 20 \rangle$ recursively calls `AddTreeLevel` to try to further grow the tree for each of those subsets.

We next continue to depth 2 in Algorithm 2 for each of the last three Γ_A subsets at depth 1 considered above.

- For the subset $\{C\}$ from depth 1, we have that $\Gamma_C = \varnothing$. Thus, this tree cannot be grown any further, and $\langle 22 \rangle$ calls `ComputeTreeRate`.
- For the subset $\{B\}$ from depth 1, we need to consider each of the subsets (at depth 2) of $\Gamma_B = \{A\}$.
  - For the subset $\varnothing \subseteq \Gamma_B$ at depth 2, $\langle 17 \rangle$ updates $\mathsf{BFH}[A] = (@, B)$ to denote that a type-A component (with parent type B) did not fail. Because the tree cannot be grown any further, $\langle 22 \rangle$ calls `ComputeTreeRate`.
  - For the subset $\{A\} \subseteq \Gamma_B$ at depth 2, a type-A component fails (at depth 2), $\langle 13 \rangle$ increments $\mathsf{nFailed}[A]$ to 2, $\langle 14 \rangle$ updates $\mathsf{BFH}[A] = (@, @)$ to denote that another type-A component failed, and $\langle 15 \rangle$ updates ρ by multiplying it by $\phi_{B,A}$. The resulting tree corresponds to row $t = 4$ in Table 1. Because at least one node was added (at depth 2) to the tree, line $\langle 20 \rangle$ recursively calls `AddTreeLevel` to try to further grow the tree.
- For the subset $\{B, C\}$ from depth 1, we need to consider each of the subsets (at depth 2) of only $\Gamma_B = \{A\}$ because $\Gamma_C = \varnothing$. We handle the subsets at depth 2 of Γ_B as in the previous bullet, but instead the resulting tree for the subset $\{A\} \subseteq \Gamma_B$ at depth 2 corresponds to row $t = 23$ in Table 1.

Rather than going through the details of the rest of the example, we note that the full structure of any tree in Table 1 can be gleaned from the table by following the rows backwards until reaching depth 0. Each tree with depth greater than 0 builds on a tree that appears previously in the table by adding an additional level of nodes, increasing the depth by one. The immediate-predecessor tree from which one tree is directly built is the nearest tree that appears previously in the table with a depth one less than its own. For example, the tree T_{18} (in row 18) has the following structure.

- At depth 4, a type-A component fails because Nodes for the row for T_{18} is "A".
- Row $t = 17$ is the nearest row with depth 3 above row 18, and one component each of types B and C fail at depth 3 because Nodes is "B, C" in row 17. The type-A component from depth 4 in row 18 is the child of the type-B component that fails from T_{17} at depth 3 because $A \in \Gamma_B$. (The other component, of type C, in row 17 has $\Gamma_C = \varnothing$.)
- Row $t = 4$ is the closest row with depth 2 above row 17, and one component of type A is the only node to fail at depth 2 because Nodes is "A" in row 4. Both components from depth 3 in row 17 are children of the type-A component from T_4 at depth 2 because $B, C \in \Gamma_A$.
- Row $t = 3$ is the closest row with depth 1 above row 4, and a component of type B (resp., C) fails (resp., does not fail) at depth 1 because Nodes is "B, -C" in row 3. The component of type A from depth 2 is the child of the B at depth 1 because $A \in \Gamma_B$.
- Row $t = 1$ is the closest row with depth 0 above row 3, and a component of type A fails at depth 0 because Nodes is "A" in row 1. The B at depth 1 is a child of the A at depth 0 because $B \in \Gamma_A$. (A component of type C did not fail at level 1, even though $C \in \Gamma_A$.)

4.3 Comparison of Runtimes

We now compare the runtimes of the original version of the code [22] with our current implementation of DECaF, as described in Section 4. Both versions are implemented in Java, where the current code is a complete overhaul of the original. We carry out the comparison on a set of different models described in Table 2, which gives for each model the cardinality of its state space S, the set of component types, the redundancies of each component type, the component-affected sets Γ_i, and the number of environments. In the text below we refer to each model by its number of states, e.g., the "125-state model." Note that the number of trees does not always grow as the number of states increases, but rather the relationships among the Γ sets and the component redundancies determine the amount of cascading possible. The experiments were conducted on the Amazon EC2 c1.xlarge cloud service, with 64-bit Intel Xeon E5-2650 CPU (2Ghz, 8 cores), 8 virtual CPUs and 7GB of memory, running Windows Server 2012.

States	Comp. Types	Redundancies	Component-Affected Sets	Env.
81	A, B, C, D	$r_A = 2$, $r_B = 2$, $r_C = 2$, $r_D = 2$	$\Gamma_A = \{B, C\}$, $\Gamma_B = \{A, D\}$, $\Gamma_C = \varnothing$, $\Gamma_D = \{A, B, C\}$	1
288	A, B, C, D	$r_A = 3$, $r_B = 3$, $r_C = 2$, $r_D = 2$	$\Gamma_A = \{B, C\}$, $\Gamma_B = \{A, C\}$, $\Gamma_C = \{B, D\}$, $\Gamma_D = \{C\}$	2
640	A, B, C, D	$r_A = 4$, $r_B = 3$, $r_C = 3$, $r_D = 3$	$\Gamma_A = \{B, C\}$, $\Gamma_B = \{A, C\}$, $\Gamma_C = \{B, D\}$, $\Gamma_D = \{B\}$	2
125	A, B, C	$r_A = 4$, $r_B = 4$, $r_C = 4$	$\Gamma_A = \{B, C\}$, $\Gamma_B = \{A, C\}$, $\Gamma_C = \{A, B\}$	1
1944	A, B, C, D, E, F	$r_A = 2$, $r_B = 2$, $r_C = 2$, $r_D = 3$, $r_E = 2$, $r_F = 2$	$\Gamma_A = \Gamma_B = \{C, D\}$, $\Gamma_C = \{A, E\}$, $\Gamma_D = \{B, F\}$, $\Gamma_E = \Gamma_F = \varnothing$	2

TABLE 2: Description of the various models we used to analyze our algorithm

States	Trees	Previous Version		New Version		
		FTT	NFTT	FTT	NFTT	DMT
81	978	1.30	0.30	0.12	0.10	0.08
288	4507	19.00	0.62	0.26	0.10	0.16
640	27746	137.29	5.98	1.56	0.11	0.61
125	321372	114.25	0.33	6.01	0.10	0.09
1944	6328	6124.01	234.84	1.35	0.16	8.27

TABLE 3: Number of trees, failure-transition time (FTT), non-failure-transition time (NFTT), and dependability-measure time (DMT) across several models for the previous and new versions of the code.

Table 3 gives the running times (in seconds) for various parts of the overall algorithms of the original code [22] and the current implementation. (We ran each model several times and observed very little difference in run times. Table 3 contains the averages across the runs.) We compare the two versions in terms of the failure-transition time (FTT) and non-failure-transition time (NFTT). The FTT is the time to generate all of the trees and to fill in all of the failure transitions in generator matrix Q. The NFTT is the time to fill in the rates for the repair and environment transitions. For the current implementation, we also give the dependability-measure time (DMT), which includes the time to compute the MTTF and SSU (but not the SSDCS and DCSUF), as described in Section 4.1, after Q is built. While the MTTF and SSU computations are performed using the OJAlgo package [34], the solving of the measures once Q has been constructed is not our paper's focus, and we can swap the current solver with another.

Table 3 shows the enormous increases in efficiency that we get from the new version of the code. The current implementation decreases the FTT by about one order of magnitude on the 81-state model and by over a factor of 4500 on the 1944-state model. The efficiency gains are due to the changes described in Section 4, as well as other improvements in the design of the algorithms and data structures developed. For the new version of the code, the FTT mainly grows as a function of the number of trees.

5 Constructing Approximate Models

The number of trees can grow exponentially in the number of components in the cascade [22], which limits the size of the models that our algorithms in Section 4 can handle. To address this issue, we explored efficient approximations that reduce the computational effort by selectively constructing only certain trees. We implement this idea by enclosing lines ⟨18⟩–⟨22⟩ of Algorithm 2 within an if statement that checks whether a given condition, which we call a "growing criterion," is satisfied. Thus, if the growing criterion does not hold, the algorithm skips over to the next enumeration of the bottom-most tree level, so we do not generate certain trees; the omitted trees' rates are not computed and not included in the generator matrix. This saves computation time, but the resulting matrix $Q' = (Q'(x, y) : x, y \in S)$ (which includes all repair and environment transitions but for the failure transitions, sums the rates of only the built trees) can differ from the matrix Q that includes all trees. Solving for the dependability measures with Q' rather than Q leads to inaccuracies in the values for the measures.

Omitting trees often results in the MTTF being greater or equal to the MTTF when all trees are considered. To see why, observe that $Q(x, y) = Q'(x, y)$ for all non-failure transitions (x, y) with $x \neq y$. But $Q(x, y) \geqslant Q'(x, y)$ for all failure transitions (x, y) because Q' only considers a subset of the trees used in computing Q. Thus, from each state, the CTMC with Q' is less likely to make a failure transition than the CTMC for Q, which typically leads to the MTTF being at least as large when trees are left out. Similarly, the SSU for Q' is usually no greater than the SSU for Q.

We investigated the trade-off in the time saved by omitting some trees versus the resulting error in the MTTF and SSU computed from Q' instead of Q. In designing a growing criterion specifying if the algorithm should enlarge the current tree, we want to allow trees with large rates to be generated, as these often have a big impact on the MTTF and SSU, and skip small-rate trees. We considered three criteria based on different types of thresholds.

We first examine a "height threshold," and the growing criterion is $\mathsf{height} \leqslant \tau_{\mathrm{h}}$, where the global variable height keeps track of the current tree height and τ_{h} is the threshold. Our implementation requires a slight change to `AddTreeLevel` (Algorithm 2), where we introduce height as a method parameter. We also modified line ⟨20⟩ of Algorithm 2 to increment height on every successive recursive call.

We also consider a "node threshold" τ_{n} in the growing criterion $\sum_{i \in \Omega} \mathsf{nFailed}[i] \leqslant \tau_{\mathrm{n}}$. Hence, constructed trees will contain a maximum of τ_{n} failed components.

The third growing criterion uses a "rate threshold" to only generate trees with rates above a certain value. Define

$$R'(T) = \bar{\lambda}_i \rho \tag{10}$$

as the approximate rate for a tree T, where $\bar{\lambda}_i = \max_{e \in \mathcal{E}} \lambda_{i,e}$ is the maximum failure rate for the type i of the root of the tree T over all the different environments, and ρ is as defined in (1). Note that $R'(T)$ differs from the tree rate $R(T)$ in (1) because $R'(T)$ omits both η, which is the product of the $1 - \phi_{i,j}$ factors for components of types

j that did not fail in the cascade but could have (because $j \in \Gamma_i$ of a component of type i that did fail and there are still type-j components up at that point), and the current number of up components of the root type. Also, $R'(T)$ uses the maximum failure rate $\bar{\lambda}_i$ of the root type i instead of the environment-specific failure rate. Then the growing criterion is $R'(T) \geqslant \tau_{\mathrm{r}}$, where τ_{r} is the specified threshold.

For each of the three growing criteria, once a given tree T does not satisfy the criterion, the current tree will not be grown any further. This point is clear for the node and height thresholds. For the rate threshold, adding additional nodes to T decreases ρ in (10) because it is multiplied by additional factors $\phi_{j,k} \leqslant 1$, so once the rate growing criterion is not satisfied, T will not be further enlarged.

Increasing τ_{h} and τ_{n} leads to a monotonic increase in the number of generated trees, whereas the number of generated trees decreases monotonically in τ_{r}. Setting $\tau_{\mathrm{h}} = \tau_{\mathrm{n}} = \infty$ or $\tau_{\mathrm{r}} = 0$ results in generating all trees, so the computed generator matrix is exact and there is then no error in the computed MTTF and SSU.

We now present numerical results when applying one growing criterion at a time for the 125-state model (see Table 2), where the failure rates are $\lambda_{A,0} = \lambda_{B,0} = 0.02$, $\lambda_{C,0} = 0.01$, and component-affect probabilities $\phi_{A,B} = \phi_{B,C} = 0.2$, $\phi_{C,A} = \phi_{B,A} = 0.3$, $\phi_{A,C} = \phi_{C,B} = 0.4$. Also, the repair rates $\mu_{i,0} = 1$ for all types i, and the system is operational as long as at least 1 component is up of each type. Figure 2 plots pairs (% FTT, % Error), where % FTT is the percentage of the FTT when omitting trees relative to the FTT when all trees are constructed, and % Error is the percent error in the computed dependability measure (MTTF or SSU) relative to the measure with all trees. For example, if % FTT is 25, then the time to generate all of the failure transitions when using the threshold was a quarter of the corresponding time when generating all trees. The experiments were performed on a PC with an Intel Core i7 4770k processor with 8 virtual cores operating at 3.5–3.9 GHz and 32GB of memory, running 64-bit Windows 10. As we expect, for each threshold, the magnitude of the error decreases as the FTT increases. For a fixed FTT, the growing criterion based on the rate threshold leads to smaller absolute error than the two other criteria; similarly, if we fix a level of error, the rate criterion requires less FTT than the other two criteria to achieve that error. Hence, the points from using the rate threshold define the "efficient frontier," analogous to the idea introduced by [28] in the context of financial portfolio selection. (Results for the other models in Table 2 are not shown but are similar. Section 6 contains results for a much larger model using a rate threshold computed from the model building blocks and a further correction described in Section 5.2, and our methods reduce FTT by up to a factor of over 600 with just a few percent error.)

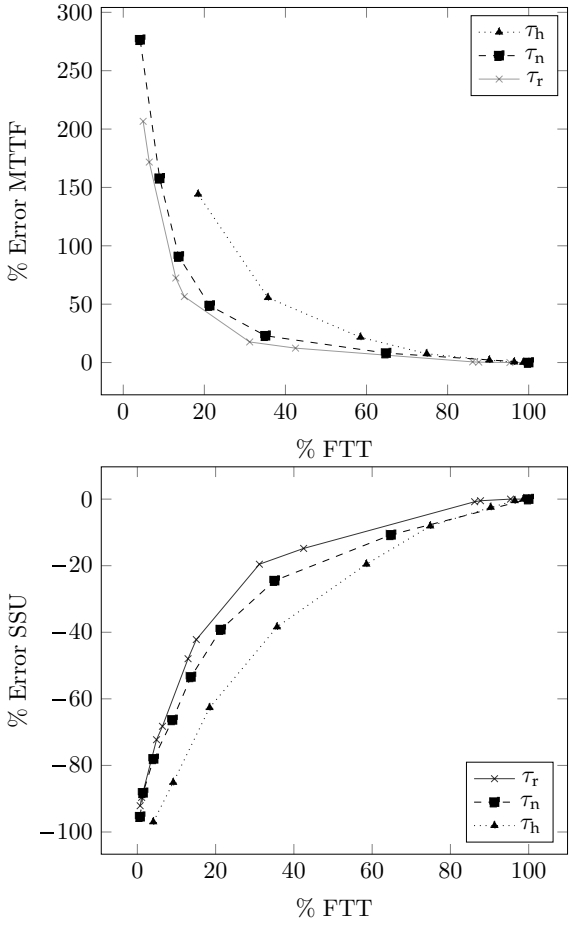

Fig. 2: % Error in MTTF and SSU versus % FTT when iterating each threshold τ

5.1 Computing a rate threshold

The previous discussion shows that a growing criterion based on a rate threshold appears to outperform the other thresholds we considered. We now discuss an efficient approach that solely uses the building blocks of the model to try to select an appropriate value for the rate threshold τ_{r} so that only a relatively small number of trees are generated but the resulting error in the dependability measures is small. The method exploits the idea that trees occurring in the most likely way the system fails should be the ones whose rates contribute most to the computed dependability measures. Exactly identifying these trees is complicated, so we instead use various approximations and simplifying assumptions to roughly determine a value for τ_{r} that allows such trees to be built while precluding trees that only occur on significantly less likely paths to failure.

We first give an overview of our approach. Assume that the system consists of highly reliable components [17] in the sense that component failure rates are much smaller than the repair rates. Suppose the system-operational conditions require that at least υ_i components of each type i are up for the system to be operational, so the system is failed when at least $d_i = r_i - \upsilon_i + 1$ components have failed for some type i. We will focus on sequences of states (i.e., paths of the embedded DTMC) for which the first state in the sequence has all components up, the last state in the sequence is a failed state (i.e., in F), all states in between are operational (i.e., in U), and each successive pair of states is a failure transition (possibly with more than one component failing). Such a sequence of states is a path to

system failure, and the most likely way the system fails is usually when exactly d_i components of some type i fail along the path. (There may be other component types that also fail in cascades along the path. If the path has no multi-component cascades, then each failure transition is just a tree with a single node, which is of type i.)

For each component type i and each $1 \leqslant k \leqslant d_i$, we examine paths consisting of exactly k (failure) transitions over which a total of d_i components of type i fail. For each failure transition in the path, we build only one tree out of the collection of failing components on the transition, even though there might be multiple trees corresponding to the transition. If there is more than one tree corresponding to a particular transition in the path, we want the single tree that we construct to be the one with the largest rate. We then use that tree in a rough approximation for the probability of the transition for the embedded DTMC. The product of the approximate transition probabilities along the path then gives an approximate probability of the entire path. The path that maximizes the approximate path probability over all types i and numbers k of transitions in the path provides an approximation to the most likely path to failure. Finally we set the rate threshold as

$$\tau_{\mathrm{r}} = \alpha\beta, \tag{11}$$

where α is the smallest approximate rate R' from (10) of a tree along the approximate most likely path to failure, and $0 < \beta \leqslant 1$ is a correction factor that is included to allow trees with approximate rates somewhat below α to be generated. We next provide details of the approach. (In Section 5.1.4 we will demonstrate the algorithms using an example.)

5.1.1 *Computing* α

`FindRateThreshold` (Algorithm 4) determines τ_{r} in (11). We start by explaining how it specifies α. Line $\langle 3 \rangle$ calls `BuildBestTrees` (Algorithm 5, which we will explain in Section 5.1.2) to build and store, for each $i \in \Omega$, $j \in \Omega$ and $1 \leqslant f \leqslant d_i$, a tree $T_{i,j,f}$ with a root of type j and having f components of type i failing, where $T_{i,j,f}$ has approximately the largest rate among those trees with these characteristics. Lines $\langle 4 \rangle$, $\langle 5 \rangle$, and $\langle 7 \rangle$ loop over all possible values of i, j, and k, where i is the component type whose d_i failures will cause the system to fail, j is the type for the root of the trees along the path to failure of the embedded DTMC, and k is the number of transitions along the path. Each transition along the path has exactly $f_i \equiv d_i/k$ type-i components failing. (When d_i/k is not an integer, we first allocate $\lfloor d_i/k \rfloor$ failing components of type i to each of the k transitions, where $\lfloor \cdot \rfloor$ denotes the floor function. Then for the remaining $b_i \equiv d_i - k\lfloor d_i/k \rfloor$ type-i components to fail along the path, we allocate one additional failing component of type i to the first b_i transitions. Thus, the first b_i transitions along the path each have $\lfloor d_i/k \rfloor + 1$ type-i components failing, and the other $k - b_i$ transitions each have $\lfloor d_i/k \rfloor$ type-i failures. In this case, we let f_i be either $\lfloor d_i/k \rfloor$ or $\lfloor d_i/k \rfloor + 1$; this is done in lines $\langle 10 \rangle$–$\langle 15 \rangle$ of Algorithm 4.)

Lines $\langle 11 \rangle$–$\langle 18 \rangle$ loop over the k transition indices $l = 1, 2, \ldots, k$ to build a path $(x^{(0)}, x^{(1)}, \ldots, x^{(k)})$ to failure of the embedded DTMC, where the initial state $x^{(0)}$ is

Algorithm 4: `FindRateThreshold`

```
   Output: τ_r
   // Calculate α
1  maxApproxProb = 0;
2  bestPathTrees = ∅;  // trees along best path
3  BuildBestTrees();      // Build trees T_{i,j,f}
    with approx largest rate with root type
    j and f type-i components.
4  for i ∈ Ω do
5      for j ∈ Ω do
6          x^(0) = x_*[j];      // initial state has
            environment e that maximizes λ_{j,e}
7          for k ∈ [1, d_i] do
8              pathApproxProb = 1;
9              pathTrees = ∅;
10             b_i = d_i − k⌊d_i/k⌋;
11             for l ∈ [1, k] do
12                 if l ⩽ b_i then //adjust first b_i
13                     f_i = ⌊d_i/k⌋ + 1;
14                 else
15                     f_i = ⌊d_i/k⌋;
16                 x^(l) =
                    resultingState(x^(l−1), T_{i,j,f_i});
17                 pathTrees = pathTrees ∪ {T_{i,j,f_i}};
18                 pathApproxProb =
                    pathApproxProb · P′(x^(l−1), x^(l));
                    // P′ defined in eq. (12)
19             if pathApproxProb > maxApproxProb then
20                 bestPathTrees = pathTrees;
21                 maxApproxProb = pathApproxProb;
22                 k_* = k;
23 T_* = arg min_{T∈bestPathTrees} R′(T);
24 α = R′(T_*);
   // Calculate β
25 M = {(i, j) : φ_{i,j} > 0};
26 β = ( (Σ_{(i,j)∈M} φ_{i,j}) / |M| )^{min[|T_*|, Σ_{l∈Ω}(r_l − x_l^{(k_*)})]};
27 return τ_r = α β;
```

defined in line $\langle 6 \rangle$ as the state $x_*[j]$ with all components operational and the environment e is chosen to maximize the failure rate $\lambda_{j,e}$ of the component type j at the root, as was also done in (10). Line $\langle 16 \rangle$ uses the tree T_{i,j,f_i} precomputed by `BuildBestTrees` in line $\langle 3 \rangle$ for the current values of i, j, and f_i, and the function `resultingState` determines the next state $x^{(l)}$ that follows the previous state $x^{(l-1)}$ after a cascade with tree T_{i,j,f_i} occurs. Line $\langle 18 \rangle$ updates the approximate probability of the constructed path by multiplying by the approximate DTMC probability $P'(x^{(l-1)}, x^{(l)})$ of the current transition, where we define

$$P'(x, y) = \frac{R(T, x)}{\sum_{j' \in \Omega}(r_{j'} - x_{j'})\lambda_{j',e} + \sum_{j' \in \Omega} x_{j'}\mu_{j',e}/(\sum_{l \in \Omega} x_l)}, \tag{12}$$

for a transition (x, y) corresponding to a tree T. The numerator in (12) is the (exact) rate from (1) for the transition (x, y) corresponding to the tree T, and the

denominator is the total failure and repair rate out of state x. (When $\sum_{l\in\Omega} x_l = 0$, there are no components failed in state x, so $\sum_{j'\in\Omega} x_{j'}\mu_{j',e} = 0$, and the second term in the denominator is $0/0$, which we define to be 0.) We omit the environment-change rate from the denominator of $P'(x, y)$ as we are only focusing on failure and repair transitions in our approximation. If, for some combination of loop indices i, j, and k, the approximate probability of one of the transitions is zero because `BuildBestTrees` did not identify the necessary corresponding tree T_{i,j,f_i}, then that combination is not considered. A tree can always be found for $f_i = 1$ when in a non-failed state because a tree of just single component of type i can always occur; therefore, a path of $k = d_i$ transitions where each transition has size $f_i = 1$ is always possible.

After lines ⟨19⟩–⟨22⟩ of Algorithm 4 identify the path with the highest approximate probability, ⟨23⟩–⟨24⟩ computes α as the minimum approximate tree rate along that path. If we set the rate threshold as $\tau_{\mathrm{r}} = \alpha$, then the algorithm would generate all of the trees used to construct the approximate most likely path to failure but trees with smaller approximate rates are omitted. Instead, we include additional trees by further multiplying the threshold by $0 < \beta \leqslant 1$ (computed in ⟨25⟩–⟨26⟩), which we will explain in Section 5.1.3.

5.1.2 *BuildBestTrees*

We now discuss Algorithm 5, which, for each $i \in \Omega$, $j \in \Omega$, and $1 \leqslant f \leqslant d_i$, constructs a tree $T_{i,j,f}$ with f type-i components and type-j root, where each tree built has roughly the largest rate among those trees with the specified characteristics. For any tree T, recall $R'(T)$ in (10) is the product of the maximum failure rate of the root and the product ρ of the component-affected probabilities $\phi_{l,m}$ of components that fail in the cascade. As adding more nodes to a tree will multiply its approximate rate by additional $\phi_{l,m}$ factors, each of which is no greater than 1, a tree T with large $R'(T)$ will typically have not too many nodes and the $\phi_{l,m}$ factors included in ρ from (1) will often be relatively large. We equivalently try to find such a tree T with large $\ln(R'(T))$, which converts the product $R'(T)$ into a sum of logs. This transformation allows us to use a shortest-path algorithm on an appropriately defined graph to approximately identify such a tree. To simplify the search, we restrict ourselves to trees in which only the root can have more than one child; we call such a tree a "broom."

Specifically, construct a weighted graph $G = (V, E, W)$, where $V = \Omega$ is the set of vertices, $E = \{(l, m) : l \in \Omega, m \in \Gamma_l\}$ is its set of edges, and $W = \{w_{l,m} : (l, m) \in E\}$ is the set of weights (costs), with $w_{l,m} = -\ln \phi_{l,m}$. Thus, large $\phi_{l,m}$ corresponds to small $w_{l,m}$. We next explain how Algorithm 5 tries to identify a broom $T_{i,j,f}$ with large $\ln(R'(T_{i,j,f}))$ for each i, j, and f.

In line ⟨3⟩ of Algorithm 5, the function `Dijkstra` returns the lowest-cost i'-to-j' path $g_{i',j'}$ and its cost $c_{i',j'}$. Then lines ⟨4⟩–⟨8⟩ compute for each possible i' and j' the optimal path from i' to j' (by considering the first step to each possible $l \in \Gamma_{i'}$) and the associated cost, and stores them in a priority queue $\mathsf{pq}_{i',j'}$. The data in the priority queue for each i', j' is later used as we iteratively add in the current lowest-cost branch from i' to j' and then remove it from the priority queue. Lines ⟨9⟩–⟨11⟩ loop over all i, j, and f to build the best broom with f type-i components and type-j root, using the variable failed to count the number of type-i components in the broom so far. The while loop at line ⟨22⟩ iteratively removes the lowest-cost j-to-i branch from the priority queue $\mathsf{pq}_{j,i}$ and attaches it to the root, as long as its cost is lower than that of the best i-to-i cycle. (The paths in the priority queue are stored in line ⟨8⟩ to not include the starting node, so attaching a branch does not incorrectly have the first step going from j to j.) This continues as long as the broom does not have enough type-i components and the priority queue is not empty. Once the next lowest-cost j-to-i branch is more expensive than the best i-to-i cycle, the while loop at line ⟨30⟩ only appends the (same) best i-to-i cycle to the leaf of the first branch of the root, stopping when the broom has enough type-i components. (In the first iteration of line ⟨30⟩, if the broom so far has only the root, which then must be of type i because of ⟨27⟩, then the best i-to-i cycle is appended to the root.)

5.1.3 *Computing β*

In our initial numerical experiments using a computed rate threshold τ_{r}, we first set $\tau_{\mathrm{r}} = \alpha$ but found that it did not work well on some models. We determined that $\tau_{\mathrm{r}} = \alpha$ was such a high threshold that too few trees were being generated. Although it led to an enormous decrease in the computation time needed to construct trees, the resulting errors in the MTTF and SSU were unacceptably large. One reason is that in addition to the approximate most likely path to failure, there may be many other paths whose approximate probabilities are only slightly smaller. These additional paths also significantly contribute to the computed dependability measures, and omitting the trees in those paths causes substantial errors in the MTTF and SSU. Compounding this issue is the coarseness of the approximations applied to determine the approximate most likely path to failure. Hence, we adjust τ_{r} by including another factor $0 < \beta \leqslant 1$ (see (11)) to permit more of the important trees to be generated.

To explain how we compute β in lines ⟨25⟩–⟨26⟩ of Algorithm 4, consider the tree T_* identified in line ⟨23⟩ of Algorithm 4, whose approximate rate $R'(T_*)$ is minimal along the approximate most likely path to failure, i.e., $R'(T_*) = \alpha$ (see line ⟨24⟩). Let T'_* be another tree obtained by adding some extra nodes to T_* below the root; we compute its approximate rate $R'(T'_*)$ by multiplying α and the product of additional component-affected probabilities $\phi_{l,m}$ for the new nodes where they are attached to T_*. The value of $R'(T'_*)$ may be only slightly smaller than α when the number of additional new nodes is not too large and when the extra $\phi_{l,m}$ factors are relatively large. But if we set the rate threshold $\tau_{\mathrm{r}} = \alpha$, then the tree T'_* would be eliminated by the rate threshold. Thus, we adjust τ_{r} by further multiplying it by β to approximate the extra factors by which we multiply α to obtain $R'(T'_*)$. This is done in lines ⟨25⟩–⟨26⟩ of Algorithm 4. (In line ⟨26⟩, when the component-affected sets of all component types are empty, we have that $|M| = 0$, and $\sum_{(i,j)\in M} \phi_{i,j} = 0$. In this case, we define $\beta = 1$.) Rather than multiplying

Algorithm 5: `BuildBestTrees`

```
   // Precompute optimal paths and costs
1  for i' ∈ Ω do
2      for j' ∈ Ω do
3          (g_{i',j'}, c_{i',j'}) = Dijkstra(G, i', j');
           // Function Dijkstra returns
           lowest-cost i'-to-j' path and its
           cost.
4  for i' ∈ Ω do
5      for j' ∈ Ω do
6          for l ∈ Γ_{i'} do
7              if w_{i',l} + c_{l,j'} < ∞ then
8                  pq_{i',j'}.insert(g_{l,j'}, w_{i',l} + c_{l,j'});
                   // i'-to-j' path omits i' but
                   cost includes cost of first
                   step from i' to l
   // For each i, j, f, build "best" tree
      with f type-i comps and type j as
      root
9  for i ∈ Ω do
10     for j ∈ Ω do
11         for f ∈ [1, d_i] do
12             if |pq_{i,i}| ≠ 0 then
13                 (cycPath_i, cycCost_i) = pq_{i,i}.peek();
14             else
15                 cycCost_i = ∞;
16             failed = 0;
17             T_{i,j,f}.root = j;
18             if j == i then
19                 failed = failed + 1;
20             else
21                 pqBranch = pq_{j,i};
22                 while failed < f && |pqBranch| > 0
                    do
23                     (branch, cost) =
                        pqBranch.remove();
24                     if cost < cycCost_i or failed ==0
                        then
25                         attach branch to root of T_{i,j,f};
26                         failed = failed + 1;
27             if failed == 0 || (failed < f &&
                cycCost_i==∞) then
28                 T_{i,j,f} = ∅;
29             else
30                 while failed < f do
31                     attach cycPath_i to leaf of
                        T_{i,j,f}.firstBranch;
32                     failed = failed + 1;
```

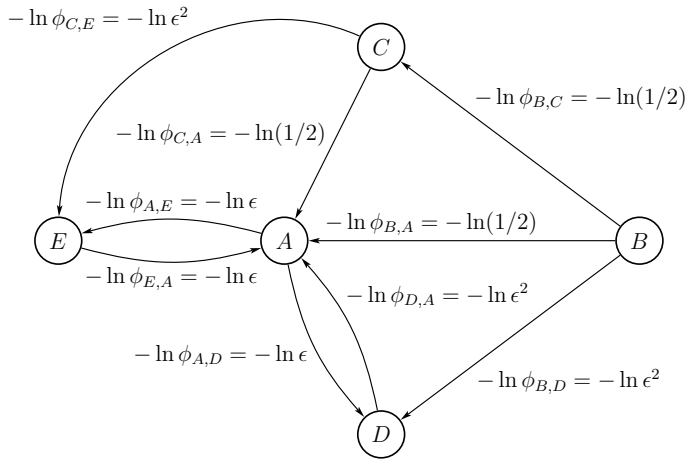

Fig. 3: Example of weighted graph $G = (V, E, W)$ used by Algorithm 5.

α by the specific $\phi_{l,m}$ to obtain $R'(T'_*)$, we simplify the calculations by instead using the average of the component-affected probabilities, which is the base of β in line $\langle 26 \rangle$ of Algorithm 4. If we attach one new node to each existing node in T_* to obtain T'_*, then the number of new nodes is $|T_*|$. But when adding the new nodes to T_*, we still want the resulting new path to failure to be possible, so we cannot add more nodes to T_* than there are remaining up components at the end of the path to failure, i.e., $\sum_{l\in\Omega}(r_l - x_l^{(k_*)})$. Hence, we take the minimum of this and $|T_*|$ to obtain the exponent of β. Finally given α and β, we compute $\tau_{\mathrm{r}} = \alpha\beta$ as in (11) and line $\langle 27 \rangle$ of Algorithm 4.

5.1.4 Example Demonstrating Computing Rate Threshold

We now use an example to demonstrate how Algorithms 4 and 5 compute the rate threshold τ_{r} in (11). Consider a system with $\Omega = \{A, B, C, D, E\}$ and $\mathcal{E} = \{0\}$. Each component type $i \in \Omega$ has redundancy $r_i = 6$, and the system fails when any type's redundancy is exhausted, i.e., $d_i = 6$. Let ϵ be a small positive constant, e.g., $\epsilon = 10^{-5}$, and the component failure rates are $\lambda_{B,0} = \epsilon$ and $\lambda_{i,0} = \epsilon^4$ for types $i \neq B$. The repair rate is $\mu_{i,0} = 1$ for each type $i \in \Omega$. For cascading, we have $\Gamma_A = \{D, E\}$, $\Gamma_B = \{A, C, D\}$, $\Gamma_C = \{A, E\}$, $\Gamma_D = \Gamma_E = \{A\}$, and $\phi_{A,D} = \phi_{A,E} = \phi_{E,A} = \epsilon$, $\phi_{B,A} = \phi_{B,C} = \phi_{C,A} = 1/2$, $\phi_{B,D} = \phi_{C,E} = \phi_{D,A} = \epsilon^2$.

`BuildBestTrees`

We first describe how Algorithm 5 constructs trees $T_{i,j,f}$ with f components of type i failing and root type j. We only consider the situation for $j = B$ and $i = A$, which turn out to be the values for the approximate most likely paths to failure. Figure 3 shows the weighted graph G constructed from the component-affect sets and probabilities, which is used in line $\langle 3 \rangle$. The graph G has the following B-to-A and A-to-A paths, with corresponding costs:

- $B \to A$ has cost $-\ln\phi_{B,A} = -\ln(1/2)$,
- $B \to C \to A$ has cost $-\ln\phi_{B,C} - \ln\phi_{C,A} = -2\ln(1/2)$,

- $B \to C \to E \to A$ has cost $-\ln\phi_{B,C} - \ln\phi_{C,E} - \ln\phi_{E,A} = -\ln(1/2) - 3\ln\epsilon$,
- $B \to D \to A$ has cost $-\ln\phi_{B,D} - \ln\phi_{D,A} = -4\ln\epsilon$,
- $A \to D \to A$ has cost $-\ln\phi_{A,D} - \ln\phi_{D,A} = -3\ln\epsilon$,
  $A \to E \to A$ has cost $-\ln\phi_{A,E} - \ln\phi_{E,A} = -2\ln\epsilon$.

Thus, $\langle 8\rangle$ builds the priority queue $\mathsf{pq}_{B,A} = ((A, -\ln(1/2)), (C \to A, -2\ln(1/2)), (D \to A, -4\ln\epsilon))$ as the best branches from B to A (omitting the initial B) for each possible first step, where the entries in $\mathsf{pq}_{B,A}$ are sorted with ascending costs. Also, we have $\mathsf{pq}_{A,A} = ((E \to A, -2\ln\epsilon), (D \to A, -3\ln\epsilon))$ has the best A-to-A cycles (without the initial A) for each possible first step, sorted with increasing costs. Then $\langle 13\rangle$ sets $\mathsf{cycPath}_A$ as $E \to A$ as the best A-to-A cycle, which has cost $\mathsf{cycCost}_A = -2\ln\epsilon$. The while loop at $\langle 22\rangle$ will attach to the root B the B-to-A branches from $\mathsf{pq}_{B,A}$ in order of cost as long as those branches have lower cost than $\mathsf{cycCost}_A$, at which point the algorithm repeatedly attaches the same best A-to-A cycle until the counter failed of type-A nodes in the tree equals the required number f.

Now we consider each iteration of the loop over $f \in [1, 6]$ in $\langle 11\rangle$ for $i = A$ and $j = B$ in $\langle 9\rangle$ and $\langle 10\rangle$, respectively. In each iteration, we begin with variable $\mathsf{failed} = 0$ in $\langle 16\rangle$, and the tree root as B in $\langle 17\rangle$.

- First consider $f = 1$ in the loop at line $\langle 11\rangle$. The first iteration of the while loop at line $\langle 22\rangle$ removes the best B-to-A branch, $(A, -\ln(1/2))$, from $\mathsf{pq}_{B,A}$ in $\langle 23\rangle$, and $\langle 25\rangle$ attaches a child A to the root. We now have $\mathsf{pq}_{B,A} = ((C \to A, -2\ln(1/2)), (D \to A, -4\ln\epsilon))$ and $\mathsf{failed} = 1$. Thus, the tree has the required number f of A nodes, so the tree is complete. The resulting tree $T_{A,B,1}$, which appears in Figure 4, has approximate rate
$$R'(T_{A,B,1}) = \lambda_{B,0}\phi_{B,A} = (1/2)\epsilon$$
computed from (10).
- Now consider $f = 2$ in the loop at line $\langle 11\rangle$. The first iteration of the while loop at $\langle 22\rangle$ is the same as above for $f = 1$. The second iteration of the loop at $\langle 22\rangle$ finds that the new best B-to-A branch, $C \to A$, in $\mathsf{pq}_{B,A}$, has cost, $-2\ln(1/2))$, that is lower than the $\mathsf{cycCost}_A = -2\ln\epsilon$ of the best A-to-A cycle. Thus, $\langle 23\rangle$ removes $(C \to A, -2\ln(1/2))$ from $\mathsf{pq}_{B,A}$, and $\langle 25\rangle$ attaches a branch $C \to A$ to the root. We now have $\mathsf{pq}_{B,A} = ((D \to A, -4\ln\epsilon))$ and $\mathsf{failed} = 2$. Thus, the tree has the required number f of A nodes, so the tree is complete. The resulting tree $T_{A,B,2}$ in Figure 4 has approximate rate
$$R'(T_{A,B,2}) = \lambda_{B,0}\phi_{B,A}\phi_{B,C}\phi_{C,A} = (1/8)\epsilon.$$
- Now consider $f = 3$ in the loop at $\langle 11\rangle$. The first two iterations of the while loop at $\langle 22\rangle$ are the same as above for $f = 1$ and $f = 2$. The third iteration of the while loop at $\langle 22\rangle$ finds that the cost, $-4\ln\epsilon$, of the new best B-to-A branch, $D \to A$, in $\mathsf{pq}_{B,A}$ is higher than $\mathsf{cycCost}_A = -2\ln\epsilon$. Thus, the while loop at $\langle 22\rangle$ will not attach anymore B-to-A branches from $\mathsf{pq}_{B,A}$ to the root. Instead, the while loop at line $\langle 30\rangle$ will grow the first branch in $T_{A,B,2}$ by attaching the best A-to-A cycle, $E \to A$, to that branch's leaf, which is an A. Thus, as we now have $\mathsf{failed} = 3$, the tree has the required number f of A nodes, so the tree is complete. The resulting tree $T_{A,B,3}$ in Figure 4 has approximate rate
$$\begin{aligned} R'(T_{A,B,3}) &= \lambda_{B,0}\phi_{B,A}\phi_{B,C}\phi_{C,A}\phi_{A,E}\phi_{E,A} \\ &= (1/8)\epsilon^3. \end{aligned}$$
- Each of the remaining iterations for $f = 4, 5, 6$ in line $\langle 11\rangle$ will further grow the branch from the first iteration of the while loop at $\langle 22\rangle$ by attaching the same A-to-A cycle, $E \to A$, to that branch's leaf, which is of type A. The resulting trees, $T_{A,B,4}$, $T_{A,B,5}$, and $T_{A,B,6}$, appear in Figure 4, and they have approximate rate
$$\begin{aligned} R'(T_{A,B,f}) &= \lambda_{B,0}\phi_{B,A}\phi_{B,C}\phi_{C,A}(\phi_{A,E}\phi_{E,A})^{f-2} \\ &= (1/8)\epsilon^{2f-3}. \end{aligned}$$

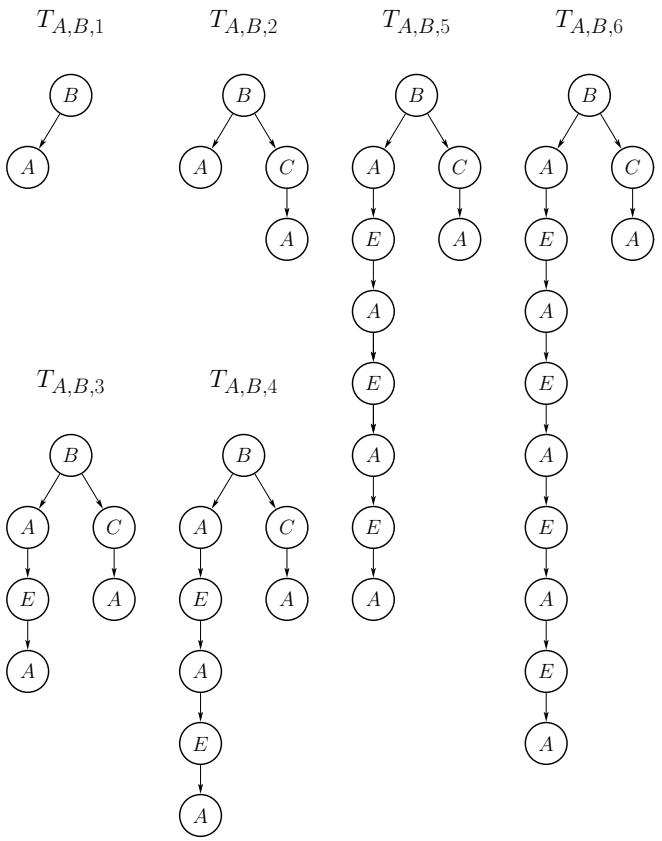

Fig. 4: Trees $T_{A,B,f}$ constructed by Algorithm 5 having f type-A components failing and type-B root.

`FindRateThreshold`

We next describe how Algorithm 4 computes the rate threshold τ_{r}. We only consider the loops for $i = A$ and $j = B$ in lines $\langle 4\rangle$ and $\langle 5\rangle$; i.e., when the system fails from exhausting type A with trees having a type-B root. We now examine what happens for each iteration of k in the loop at $\langle 7\rangle$, where k is the number of transitions in a path to failure that is to be constructed, and we recall that $d_A = 6$. When we compute $\mathsf{pathApproxProb}$ in $\langle 18\rangle$, the first transition along the path has that the denominator in (12)

is $O(\epsilon)$ because there are only failure transitions and no repair transitions out of the initial state. Each subsequent transition along the path has that the denominator in (12) is $O(1)$ because at least one component is failed so there is an ongoing repair.

- For $k = 1$, line $\langle 10 \rangle$ has $b_A = 0$, so the single transition in the constructed path to failure has $f_A = 6$ type-A components failing. We use the tree $T_{A,B,6}$ in Figure 4 for that transition. The resulting DTMC path has approximate probability

$$\mathsf{pathApproxProb} = O(\epsilon^8)$$

  computed using $\langle 18 \rangle$.
- For $k = 2$, line $\langle 10 \rangle$ has $b_A = 0$, so by $\langle 15 \rangle$, each of the k transitions in the constructed path to failure has $f_A = 3$ type-A components failing. The resulting DTMC path of k transitions, each corresponding to $T_{A,B,3}$, has approximate probability

$$\mathsf{pathApproxProb} = O(\epsilon^5).$$

- For $k = 3$, line $\langle 10 \rangle$ has $b_A = 0$, so by $\langle 15 \rangle$, each of the k transitions in the constructed path to failure has $f_A = 2$ type-A components failing. The resulting DTMC path of k transitions, each corresponding to $T_{A,B,2}$, has approximate probability

$$\mathsf{pathApproxProb} = O(\epsilon^2).$$

- For $k = 4$, line $\langle 10 \rangle$ has $b_A = 2$, so in the loop at $\langle 11 \rangle$, each of the first two transitions in the constructed path to failure has $f_A = 2$ type-A components failing, and each of the last two transitions has $f_A = 1$. These correspond to trees $T_{A,B,2}$ and $T_{A,B,1}$ in Figure 4. The resulting DTMC path of k transitions has approximate probability

$$\mathsf{pathApproxProb} = O(\epsilon^3).$$

- For $k = 5$ and 6, we can similarly show that $\mathsf{pathApproxProb} = O(\epsilon^4)$ and $O(\epsilon^5)$ respectively.

Hence, $\langle 19 \rangle$–$\langle 22 \rangle$ of Algorithm 4 identify the constructed path with the highest approximate probability as having length $k_* = 3$, so $\langle 23 \rangle$–$\langle 24 \rangle$ result in $T_* = T_{A,B,2}$ and $\alpha = R'(T_{A,B,2}) = (1/8)\epsilon$ as the minimum approximate tree rate along the identified path with k_* transitions.

Finally, we compute the other factor β in (11). First, we have that $M = \{ (A, D), (A, E), (B, A), (B, C), (B, D), (C, A), (C, E), (D, A), (E, A) \}$ in line $\langle 25 \rangle$ of Algorithm 4. In $\langle 26 \rangle$ the base averages the component-affected probabilities: $(\epsilon + \epsilon + 1/2 + 1/2 + \epsilon^2 + 1/2 + \epsilon^2 + \epsilon^2 + \epsilon)/9 \approx 1/6$. For the exponent in $\langle 26 \rangle$, we have that $|T_*| = |T_{A,B,2}| = 4$, as seen in Figure 4. Also, after the $k_* = 3$ transitions in the constructed approximate most likely path to failure, the numbers of remaining up components of types $A, B, \ldots, E$ are 0, 3, 3, 6, 6, so the second term in the exponent for β is their sum, 18. Hence, the exponent for β is $\min(4, 18) = 4$, so $\beta \approx (1/6)^4$, resulting in $\tau_{\mathrm{r}} = \alpha\beta \approx (1/8)(1/6)^4\epsilon$ as the rate threshold.

5.2 Correcting the Generator Matrix's Diagonal Entries

As we will see in Section 6, the methods developed in Sections 5 and 5.1 can drastically reduce the number of trees constructed and the computation time. But because the resulting generator matrix Q' includes only a subset of the trees used to construct Q, errors arise in the computed dependability measures. We next try to reduce the error by modifying Q' to obtain another generator matrix $Q'' = (Q''(x, y) : x, y \in S)$ that has the same diagonal entries as the original matrix Q. Because $-1/Q(x, x)$ is the mean time that the original CTMC spends in state x on each visit there, matching the diagonal entries to those of Q can help by ensuring the approximate CTMC spends the same amount of time on average in each state as the complete model. Moreover, we can view the diagonal correction of Q'' as a way of compensating for not generating all of the trees.

In the complete matrix Q, the sum of the rates (1) of all trees originating in a state x and having a root of type i is $(r_i - x_i)\lambda_{i,x_{N+1}}$, the factor from the root in each tree rate. This holds because the set of all those trees includes every possible combination of failures and non-failures of components that could be affected in a cascade triggered by the failure of a component of type i. Because a failure transition out of state x can be triggered by any component type that still has operating components in the state, the total failure rate out of state x satisfies

$$\sum_{y:(x,y)\in\Psi_{\mathrm{f}}} Q(x, y) = \sum_{i=1}^{N} (r_i - x_i)\lambda_{i,x_{N+1}} \geqslant \sum_{y:(x,y)\in\Psi_{\mathrm{f}}} Q'(x, y),$$

where we recall that Ψ_{f} was defined in Section 3 as the set of failure transitions, and the inequality holds because Q' was computed by omitting some trees.

For the complete matrix Q, the total rate out of x is $-Q(x, x)$, which equals

$$\sum_{y:(x,y)\in\Psi_{\mathrm{f}}} Q(x, y) + \sum_{y:(x,y)\in\Psi_{\mathrm{e}}} Q(x, y) + \sum_{y:(x,y)\in\Psi_{\mathrm{r}}} Q(x, y),$$

where we recall from Section 3 that Ψ_{e} and Ψ_{r} are the sets of environment and repair transitions, respectively. Because Q' includes all environment and repair transitions, we have that $\sum_{y:(x,y)\in\Psi_{\mathrm{e}}} Q'(x, y) = \sum_{y:(x,y)\in\Psi_{\mathrm{e}}} Q(x, y)$ and $\sum_{y:(x,y)\in\Psi_{\mathrm{r}}} Q'(x, y) = \sum_{y:(x,y)\in\Psi_{\mathrm{r}}} Q(x, y)$. Thus, for each x, the difference in the diagonal entries, which is

$$\epsilon_x \equiv -Q(x, x) + Q'(x, x) \geqslant 0, \tag{13}$$

results solely from the failure transitions, and we define the new matrix Q'' such that $Q''(x, x) = Q(x, x)$ for each $x \in S$. For $y \neq x$, we let $Q''(x, y) = Q(x, y)$ for $(x, y) \notin \Psi_{\mathrm{f}}$, so Q'' shares the same rates for non-failure transitions as Q (and Q'). We then need to define $Q''(x, y)$ for failure transitions $(x, y) \in \Psi_{\mathrm{f}}$ so that $\sum_{y:(x,y)\in\Psi_{\mathrm{f}}} Q''(x, y) = \sum_{y:(x,y)\in\Psi_{\mathrm{f}}} Q(x, y)$ for each state $x \in S$.

To do this, for a failure transition $(x, y) \in \Psi_{\mathrm{f}}$, we add to each $Q'(x, y)$ a portion of the difference ϵ_x in (13); i.e.,

$$Q''(x, y) = Q'(x, y) + w'(x, y)\epsilon_x, \tag{14}$$

where $w'(x,y) \geqslant 0$ is a weighting function such that $\sum_{y:(x,y)\in\Psi_{\mathrm{f}}} w'(x,y) = 1$. Through experimentation with various weighting schemes, we found that setting

$$w'(x,y) = \frac{Q'(x,y)^v}{\sum_{z:(x,z)\in\Psi_{\mathrm{f}}} Q'(x,z)^v}$$

with $v = -2$ worked well across different models. Observe that $w'(x,y) = 0$ whenever $Q'(x,y) = 0$, so only failure transitions that are possible in Q' are modified by (14).

As noted in Section 5, Q' typically has the property that its corresponding MTTF and SSU are no worse than those for Q (because omitting trees usually makes the system more dependable), but no such trends seem to hold when comparing the dependability measures of Q'' and Q. However, we often have that the system with Q'' is less dependable than the system with matrix Q' because Q'' has larger rates for the failure transitions than Q' does (with the same environment and repair rates). It is often the case that the system with Q'' is more dependable than with Q; however, it is possible that Q'' overcompensates, and the system with Q'' can be less dependable than with Q; e.g., see Case 3 of Table 1.

5.3 Errors from Q' and Q''

Our approach in Section 5.1 to compute τ_{r} relies on a number of approximations, some of which were developed under the assumption that the component-affected probabilities $\phi_{i,j}$ are small. To test the effect of the size of the $\phi_{i,j}$ on the resulting errors in the MTTF and SSU, we ran numerical experiments on various models, where we systematically increased the $\phi_{i,j}$. Figure 5 shows the results for a modified version of the 125-state model from Table 2 with $\Gamma_A = \Gamma_B = \{C\}$ and $\Gamma_C = \{A, B\}$. We started with $\phi_{A,C} = 0.1$, $\phi_{B,C} = 0.3$, $\phi_{C,A} = 0.05$, $\phi_{C,B} = 0.4$, and the other $\phi_{i,j} = 0$, and then simultaneously changed all nonzero $\phi_{i,j}$ to $\phi_{i,j} + \Delta$ for $\Delta = 0.1, 0.2, \ldots, 0.5$.

For a generator matrix $\bar{Q}$, let $\mathrm{MTTF}(\bar{Q})$ and $\mathrm{SSU}(\bar{Q})$ be the MTTF and SSU, respectively, computed from $\bar{Q}$. Figure 5 plots the ratios $\mathrm{MTTF}(\hat{Q})/\mathrm{MTTF}(Q)$ and $\mathrm{SSU}(Q)/\mathrm{SSU}(\hat{Q})$, for $\hat{Q} = Q'$ and Q''. The reason we inverted the ratios for MTTF and SSU is that increasing Δ leads to the system becoming less dependable, so the MTTF and SSU then move in opposite directions. A ratio of 1 indicates no error. As Δ grows, the ratios worsen, so using the rate threshold τ_{r} alone (i.e., Q') seems to work best when the $\phi_{i,j}$ are relatively small. But Q'' from Section 5.2 substantially reduces the error by correcting the diagonal entries of the generator matrix. Results for other models (not shown) are similar.

5.4 Error from Eliminating a Single Tree

When computing the dependability measures of a system for which trees have been eliminated (e.g., by a rate threshold), the computed dependability measures typically differ from the exact values that correspond to the original system with generator Q. We now study to what extent the values of dependability measures change by eliminating just a single tree at a time, and as we will see, this provides further numerical evidence supporting our decision of using a rate threshold to omit trees. We carry out the analysis

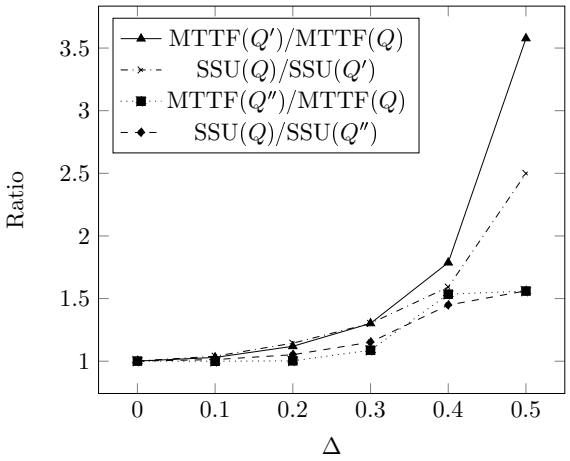

Fig. 5: Ratios comparing dependability measures with Q' and Q'' to Q as non-zero component-affected probabilities $\phi_{i,j} + \Delta$ vary.

on the model previously considered in Section 4.2. For the three cases of the model, we chose the values of $\lambda_{A,0}$ $\phi_{A,B}$, and $\phi_{B,A}$, listed at the top of Table 1, so that Algorithm 4 identifies fundamentally different approximate most likely paths to failure.

In each case, component type A is the one whose $d_A = r_A - \upsilon_A + 1 = 4$ failures lead to system failure in the approximate most likely path to failure, but the cases have a different optimal number k_* of transitions in that path (see line ⟨22⟩ of Algorithm 4).

- Case 1 has $k_* = 1$ transition, and the single tree on that path to failure is tree $t = 10$ (i.e., T_{10}) in Table 1; tree T_{10} has $f_A = d_A/k_* = 4$ type-A components (along with other components) failing. (Section 4.2 explains the structure of the trees that appear in the table.)
- Case 2 has $k_* = 2$ transitions, and its approximate most likely path to failure has two successive occurrences of tree T_4, which has $f_A = d_A/k_* = 2$ type-A failures.
- Case 3 has $k_* = 4$ transitions, where tree T_1, which has $f_A = d_A/k_* = 1$ type-A failure, is repeated four times on the path to failure.

The row labeled τ_{r} in Table 1 gives the value of the resulting rate threshold obtained by Algorithm 4 for each case. Table 1 also gives the approximate rate $R'(T_t)$ from (10) of each tree T_t, where an entry with a * denotes that $R'(T_t) \geqslant \tau_{\mathrm{r}}$; the values of the MTTF and SSU for Q; and the ratios of MTTF and SSU for Q compared to Q' and Q''. The last row of Table 1 gives for each case, the number of trees constructed (labeled "unique") and total number of times those built trees were used in transitions ("eval.") in Q'.

Let $Q_t^* = (Q_t^*(x,y) : x, y \in S)$ be the infinitesimal generator matrix when all trees except T_t are included. To isolate the impact of just T_t, larger trees built from T_t are still included in Q_t^*. This is in contrast to the study in Figure 2, where all larger trees built from T_t

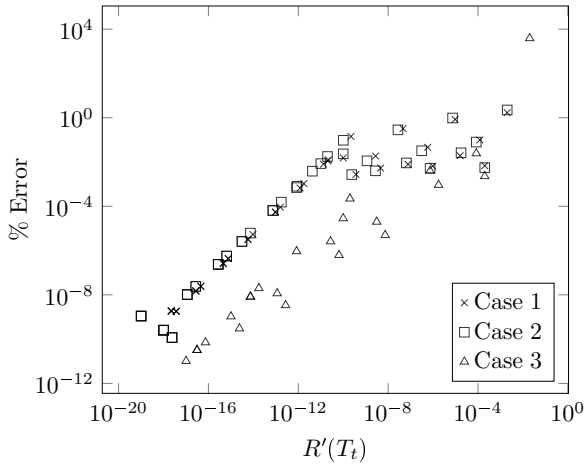

Fig. 6: Absolute value of percent error in MTTF versus approximate rate when removing each distinct failure tree

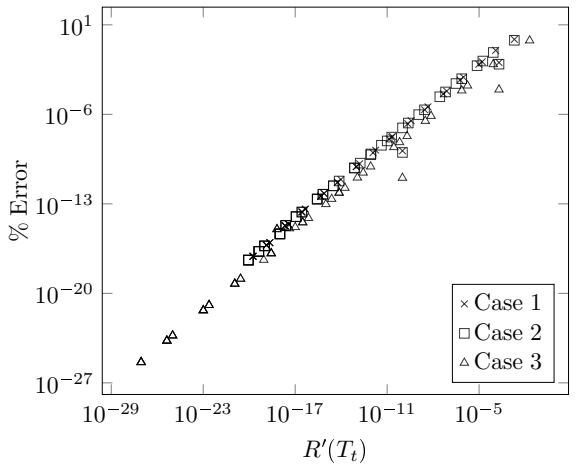

Fig. 7: Sum over all cascade sizes of absolute values of errors in DCSUF versus approximate rate when removing each distinct failure tree

would also be skipped because if T_t does not satisfy the growing condition, then each larger tree built from T_t would also not satisfy the growing condition. Thus, using the tree-rate function R defined in (1), we have that $Q_t^*(x,y) = Q(x,y) - R(T_t,x)$ for each failure transition (x,y) that includes T_t, and $Q_t^*(x,y) = Q(x,y)$ for each transition (x,y) with $x \neq y$ not including T_t. Also, the diagonal entry $Q_t^*(x,x) = -\sum_{y\neq x} Q_t^*(x,y)$. We next compare for each tree T_t the values of our dependability measures for Q_t^* and for the original generator Q. This gives us a way to assess each tree's importance.

For each tree T_t, Figure 6 plots the % error in the MTTF for Q_t^* as a function of the approximate rate $R'(T_t)$, where the % error is given by $|\text{MTTF}(Q_t^*) - \text{MTTF}(Q)|/\text{MTTF}(Q)$. Though not shown, the plots of the steady-state unavailability error follow the same pattern. Similarly, Figure 7 (resp., 8) plots the % error in the computed DCSUF χ in (6) as a function of $R'(T_t)$, where the % error is given by $\sum_{l=1}^{b} |\chi(l,Q_t^*) - \chi(l,Q)|$ (resp., $\sum_{l=1}^{b} |\chi(l,Q_t^*) - \chi(l,Q)|/\chi(l,Q)$), and $\chi(l,\bar{Q})$ is the value of $\chi(l)$ for a generator matrix $\bar{Q}$. Though not shown,

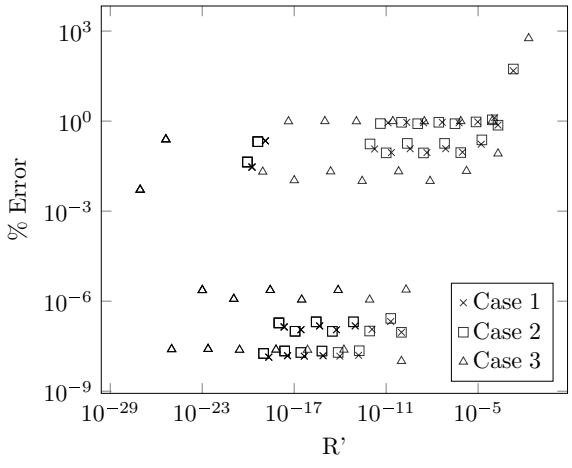

Fig. 8: Sum over all cascade sizes of absolute values of relative errors in DCSUF versus approximate rate when removing each distinct failure tree

the plots of the error in the SSDCS θ in (2) follow the same pattern. In all three cases, a clear trend shows that eliminating a tree with a higher approximate rate from all appropriate transitions in the generator matrix tends to lead to greater error in the computed dependability measures. This leads us to conclude that trees with a higher approximate rate have a greater importance on the accuracy of the dependability measures, which is in accordance with our previous findings from Figure 2.

Figure 8 shows a clear grouping of plotted points that is not found in Figures 6 or 7. In Figure 8, with the exception of the two leftmost points, which are in fact eight overlapping points, those in the upper half of the plot correspond to trees that do not contain a failed component of type C, while those in the lower half do. This is significant because the models have only one component of type C, and there is very small chance that another component type causes a C to fail (i.e., $\phi_{A,C} = 10^{-8}$ and $\phi_{B,C} = 0$). This means that whenever a component of type C fails in some tree T_t, and it is not the root of T_t, the exact rate of T_t becomes extremely small. When the rate of T_t is significantly smaller than the rate of some other tree of the same cascade size, T_t seems to contribute little to $\chi(l)$ relative to other trees with larger rates. Note that the two left-most points do not seem to follow this trend, as the trees that contribute to those points all contain a failed component of type C. This is because for those eight trees, the cascade has the largest possible size. These eight trees are the only trees with every single component failing, including one of type C. This means that no tree has a rate that is significantly smaller than that of other trees of the same size due to the effect of including one failed component of type C. Furthermore, within each of those groupings (i.e., the top group, the bottom group, and the group containing the two leftmost points), there is a further subgrouping, with one band of points above and one below. The points in the upper (resp., lower) grouping correspond to trees with a type-A (resp., type-B) component at the root. For these models we have that $\lambda_A/\lambda_B = 10$ or 100, and a similar argument as before explains how eliminating

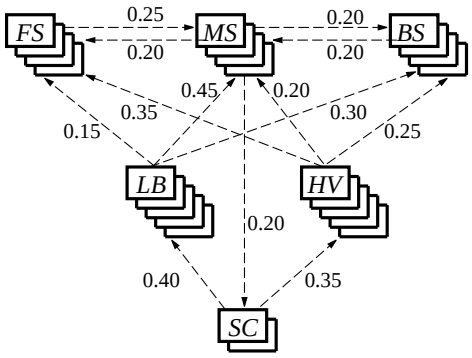

Fig. 9: Component-affected probabilities $\phi_{i,j}$ in a cloud-computing model.

a tree T_t seems to lead to the error being smaller when T_t has a rate significantly smaller than some other tree of the same cascade size.

6 Cloud-Computing Model

To test the efficacy and efficiency of our approaches, we ran numerical experiments on a large-scale model. High dependability is crucial for cloud-computing services [14], and we considered a dependability model of a three-tier cloud-computing system in Figure 9. Each group of boxes represents a component type, where the labels FS, MS, BS, LB, HV and SC, respectively, denote front-end servers, middle-end servers, back-end servers, load balancers, hypervisors, and system controllers. There is a directed edge from component type i to type j if the failure of a component of type i can probabilistically cause a component of type j to immediately fail, i.e., $j \in \Gamma_i$. The label on an edge from i to j is the component-affected probability $\phi_{i,j}$. For example, there are edges from HV to FS, MS and BS because according to [42], hypervisors often "cause other system components to fail and certainly cause server racks to fail because of state corruption."

The redundancies for the different component types are $r_{FS} = r_{MS} = r_{BS} = 4$, $r_{LB} = r_{HW} = 5$, and $r_{SC} = 2$, which are depicted in Figure 9 by the multiple boxes for a component type. We assume the system is operational if and only if there is at least one component up of each type. Additionally, the system operates in two environments: high demand ($e = 1$) and low demand ($e = 0$). The resulting state space S of the CTMC has size $|S| = 27{,}000$.

For the high-demand environment, the component types have failure rates $\lambda_{FS,1} = \lambda_{MS,1} = \lambda_{BS,1} = 1/8760$, $\lambda_{LB,1} = \lambda_{HV,1} = 1/4380$, and $\lambda_{SC,1} = 1/43800$, where the time unit is hours. Thus, in the high-demand environment, the mean component lifetime of a server is one year, each load balancer and hypervisor has a mean lifetime of 0.5 years, and a system controller has a mean lifetime of 5 years. In the low-demand environment 0, we set $\lambda_{SC,0} = \lambda_{SC,1}/2$ and $\lambda_{i,0} = \lambda_{i,1}/4$ for each other component type $i \neq SC$. These values are roughly comparable to failure rates given in [16], in which the authors state, based on discussions with vendor personnel, their numbers are "reasonable" with respect to actual proprietary values.

In environment 1, the repair rate for failed servers is $1/12$, and the load balancer and hypervisor (resp., system controller) have double (resp., half) that repair rate. In environment 0, the repair rate is halved for each type, except for the system controller, which has the same repair rate in both environments. These values are roughly comparable to repair rates used in [16]. The environment switches once every 12 hours on average to the other environment, so $\nu_0 = \nu_1 = 1/12$ and $\delta_{0,1} = \delta_{1,0} = 1$.

Table 4 contains results from running the new version of DECaF to compute the MTTF and DCSUF of various versions of the cloud-computing model. The versieons differ in the amount of cascading possible: high, low, and none. The high-cascading version is as described above, with component-affected sets $\Gamma_{FS} = \{MS\}$, $\Gamma_{MS} = \{FS, BS, SC\}$, $\Gamma_{BS} = \{MS\}$, $\Gamma_{LB} = \{FS, MS, BS\}$, $\Gamma_{HV} = \{FS, MS, BS\}$, and $\Gamma_{SC} = \{LB, HV\}$. Some sets are smaller in the low-cascading version: $\Gamma_{LB} = \Gamma_{HV} = \{MS\}$ and all other Γ_i are unchanged from the high-cascading version. Also, the low-cascading version has the same component-affected probabilities $\phi_{i,j}$ as shown in Figure 9 for $j \in \Gamma_i$, and $\phi_{i,j} = 0$ for $j \notin \Gamma_i$. The no-cascading version has all $\Gamma_i = \varnothing$ and $\phi_{i,j} = 0$. For the high- and low-cascading models, we ran DECaF three times: once with the complete generator matrix Q that includes all cascading-failure trees, using the algorithms from Section 4; a second time with generator matrix Q', which implements only the rate threshold τ_{r} from Sections 5 and 5.1; and the third with generator matrix Q'', which further corrects the diagonal entries to equal those in Q, as described in Section 5.2. (For the model with no cascading, the methods from Sections 5.1 and 5.2 do not eliminate any trees, so we only report the results for Q.) The third column of Table 4 shows the number of unique trees that are constructed by

TABLE 4: Numerical results for different versions of the cloud-computing model, including the failure-transition time (FTT), non-failure transition time (NFTT), fundamental-matrix time (FMT), MTTF time (MTTFT), DCSUF time (DCSUFT), and rate-threshold time (RTT).

Casc. Amt.	Gen. Mat.	Unique Trees	Trees Eval.	MTTF	% Error MTTF	CPU Times (seconds) Total	FTT	NFTT	FMT	MTTFT	DCSUFT	RTT
High	Q	2.00E08	1.93E10	26851	0.0	269957.0	266205.2	2.1	3733.7	0.15	15.87	0
High	Q'	2.67E05	3.82E08	42796	59.4	4160.0	442.1	2.2	3699.7	0.15	15.86	0.02
High	Q''	2.67E05	3.82E08	26046	−3.0	4089.9	437.8	1.9	3633.4	0.14	16.59	0.02
Low	Q	1.04E06	3.17E08	49453	0.0	5039.9	1384.0	1.9	3638.7	0.14	15.13	0
Low	Q'	2.92E04	4.70E07	57674	16.6	4012.0	47.2	2.2	3943.0	0.25	19.33	0.03
Low	Q''	2.92E04	4.70E07	49353	−0.2	3688.7	44.9	1.9	3626.8	0.14	14.93	0.02
None	Q	6.00E00	1.28E05	67606175	0.0	3445.5	0.04	1.9	3428.4	0.14	14.98	0

DECaF, where each unique tree may be used several times for different transitions, as explained in Section 4. Summing up the number of times each unique tree is used in some failure transition over all unique trees results in the entries in the column labeled "Trees Eval."; thus, the ratio of Trees Evaluated over Unique Trees gives the average number of failure transitions for which each unique tree was used. The fifth column of Table 4 contains the MTTF for the specified generator matrix Q, Q' or Q''. As expected, MTTF(Q) increases (i.e., the system becomes more dependable) as the amount of cascading decreases. For each model version, the sixth column shows that the percent error in the MTTF for Q' is always nonnegative in this model, so Q' results in a more dependable system, as we had noted is usually the case in Section 5. Note that Q' produces MTTFs that are reasonably close to the true value from Q, but Q'' does substantially better, resulting in errors of at most a few percent.

The last seven columns of Table 4 present the CPU times it took DECaF to perform various computations. The experiments were performed on the same computer used to generate the results in Figure 2 (but different from the one employed for Table 3). The "Total" column gives the amount of time required to complete an entire run of DECaF to compute the MTTF and DCSUF. (We did not compute the SSU nor the SSDCS in this set of experiments.) As in Table 3, the FTT (resp., NFTT) is the failure-transition (resp., non-failure transition) time. Note that NFTT is insignificant compared to Total Time for all rows in Table 4. The FTT includes the time to perform the computations corresponding to the columns Unique Trees and Trees Eval. The FMT is the fundamental-matrix time, which is the time to compute $(I - P_U)^{-1}$, needed for computing both the MTTF and DCSUF; see Sections 4.1.2 and 4.1.4. MTTFT is the additional CPU time required within a run to compute the MTTF after computing the fundamental matrix. DCSUFT is the extra time to compute DCSUF after $(I - P_U)^{-1}$ has been computed. RTT in the last column of Table 4 is the rate-threshold time: how long it took to compute the rate threshold τ_{r}, which we note is minuscule compared to FMT and the Total Time. Moreover, RTT is always much smaller than FTT, with orders of magnitude difference when there are many trees. Thus, the algorithm in Section 5.1 to compute τ_{r} is very efficient and incurs virtually no overhead.

In the high-cascading version of the model, the use of the rate threshold decreased the number of unique trees by a factor of 761, with a 51-fold drop in the trees evaluated. FTT was lowered by a factor of about 600, and the total time shrank by over 60-fold. The error in the MTTF from using only the rate threshold (i.e., Q') is about 60%, and further applying the diagonal correction (i.e., Q'') produced less than 3% error. Thus, the methods of Sections 5.1 and 5.2 can dramatically reduce computation time with small error in models with a high level of cascading. For the low-cascading model version, the unique trees generated decreased by a factor of 35.6, resulting in a roughly 30-fold reduction in FTT. The total time does not shrink as dramatically because FMT now becomes the bottleneck. (We are currently investigating more efficient techniques to compute the fundamental matrix.) Compared to the high-cascading model, Q' and Q'' for low cascading produce substantially more accurate values for MTTF, with only 0.2% error for Q''.

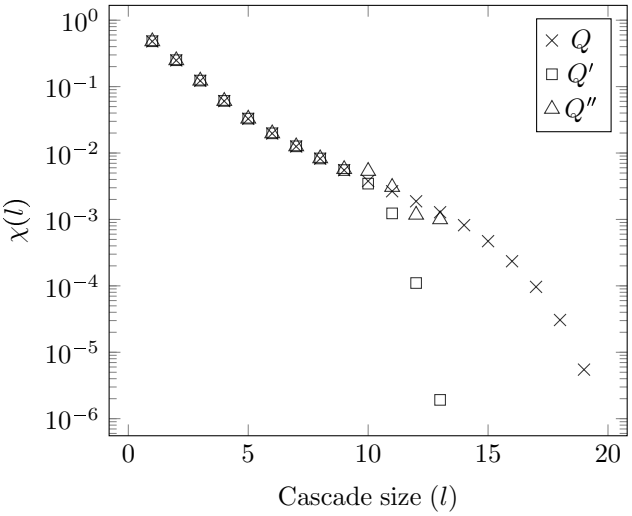

Fig. 10: DCSUF $\chi(l)$, $l \in [1, 24]$, for cloud model with high cascading, where only non-zero values of $\chi(l)$ are plotted

For the cloud model with the high (resp., low) level of cascading, Figure 10 (resp., 11) plots (on semilog scale) the values of the DCSUF $\chi(l)$ from Section 4.1.4 for the possible cascade (tree) sizes l for the generator matrices Q, Q' and Q''. Although the theoretical largest possible value of l is 24, which is the sum of all component redundancies, we have that $\chi(l) = 0$ for Q when $l \geqslant 20$ (resp., $l \geqslant 13$) for the model with high (resp., low) cascading as those larger trees cannot occur because of limitations imposed by the component-affected sets Γ_i and the redundancies. In Figure 10 we see that the values of $\chi(l)$ closely match for all three generators when l is small; the values for Q' and Q start diverging for the middle range of l; and there are no values for Q' and Q'' for large l. The apparent reason for this arises from the way we apply the growing criterion with rate threshold τ_{r}, which we recall generates a tree T if and only if its approximate rate $R'(T) \geqslant \tau_{\mathrm{r}}$. Cascades with small size l correspond to trees T with relatively large $R'(T)$, so most of those trees are not eliminated by τ_{r}. This leads to $\chi(l)$ for Q' and Q'' being close to that for Q for small l. The middle values of l result in trees whose approximate rate straddle τ_{r}, so some portion of them are constructed and others not. This leads to Q' having substantial error for the middle range of l, but Q'' largely corrects this. Finally, trees with large size l are completely eliminated by τ_{r}, so Q' and Q'' give $\chi(l) = 0$. Figure 11, which is for the low-cascading version, exhibits a similar pattern but without the behavior on the far right of Figure 10 because, although Q' and Q'' correspond to eliminating some trees in the low-cascading model, there are still others that are built for each value of l for which $\chi(l) > 0$ for Q.

7 Concluding Remarks

We developed efficient algorithms and data structures to construct, analytically solve, and approximate a CTMC model of a dependability system having cascading failures.

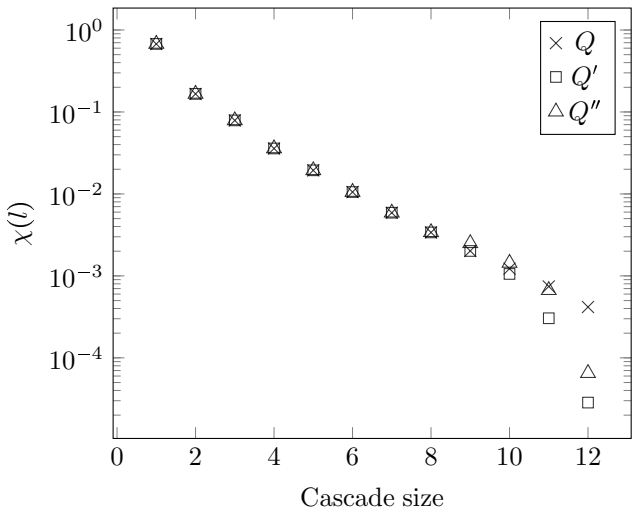

Fig. 11: DCSUF $\chi(l)$, $l \in [1, 24]$, for cloud model with low cascading, where only non-zero values of $\chi(l)$ are plotted

We implemented the ideas in a software package called the Dependability Evaluator of Cascading Failures, DECaF, which builds the CTMC from basic building blocks describing the system and then solves it to compute various dependability measures. In addition to the SSU and MTTF, we also derive two new dependability measures: the steady-state distribution of cascade size (SSDCS), and the distribution of cascade size until failure (DCSUF).

In contrast to many studies of CTMCs, simply building the CTMC in our setting poses tremendous computational hurdles. The problem arises from the complexity in generating the cascading-failure trees, and we provided efficient methods to quickly construct the trees. The new algorithms led to decreasing the runtime of DECaF by orders of magnitude compared to the previous version in [22].

But even with efficient methods for generating trees, the exponential growth in the number of trees limits the size of models that can be analyzed exactly. Thus, we also proposed a technique that judiciously generates only a subset of the trees by using a rate threshold τ_{r}. Exploiting the idea of most likely paths to failure, the approach tries to generate trees that arise on such paths but omits those on significantly less probable paths. Because not all trees are generated, the resulting dependability measures have some error, but our numerical experiments indicate the approach can dramatically reduce computation time and still have very accurate results when we further correct for the diagonal entries in the generator matrix, especially when the component-affected probabilities are relatively small (see Section 5.3).

Our approach in Section 5.1 to specify τ_{r} exploits approximations based on the system comprising highly reliable components, i.e., failure rates are much smaller than repair rates. Section 6 presented numerical results for a large cloud model with this characteristic, as well as the $\phi_{i,j}$ not being too large, and our methods reduced computation time by orders of magnitude with just a few percent error. Our techniques can also work with models of other systems satisfying these assumptions in different applications areas.

It would be interesting to see if additional problem structure can be exploited to obtain more efficient methods by using e.g., lumping [9], symmetries [10], or continuous approximations [5].

Another topic for future work is to adapt the approximation based on a subset of the trees for use in a quick simulation method using importance sampling (IS) [33]. Previously developed IS schemes have applied the concept of most likely paths to failure in their design for efficiently simulating dependability systems with limited cascading failures [23], [33]. We plan to investigate expanding the idea in our model with more general cascading failures.

Acknowledgments

This work has been supported in part by the National Science Foundation under Grants No. CMMI-0926949 and CMMI-1200065. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. Additional funding came from the NJIT Provost Undergraduate Summer Research program. The authors thank Prasad Tendolkar for his contributions to the early efforts of the project. Finally, the authors thank the associate editor and referees for making numerous suggestions that led to an improved paper.

Appendix

Computing a Tree's Rate Revisited

We reconsider our example in Section 3.2 to show how to compute η from (1) for Figure 1 using the data structure BFH (breadth-first history). We assume the tree in Figure 1 was first constructed from several recursive calls to `AddTreeLevel` (Algorithm 2), which also built BFH as follows.

A	@-1	B-2	@-6	B-7
B	@-2	@-7	A-6	
C	A-1	@-5	A-6	B-7

We have also included the node IDs in BFH to aid in the following discussion, although the IDs are not part of BFH.

We then proceed to `ComputeTreeRate` (Algorithm 3) to compute η. As in Section 3.2, prior to iterating through BFH, we have $u_A = 2$, $u_B = 2$ and $u_C = 1$. Iteration through BFH occurs as specified in `ComputeTreeRate` in lines $\langle 3 \rangle$–$\langle 9 \rangle$.

- Iteration through the linked list at index A is as follows. The @-1 means a component of type A has failed at ID 1, so we then decrement u_A to 1. Then for the next entry, because u_A is still positive, η includes a factor $(1 - \phi_{B,A})$ from B-2 in its product from the A not failing at ID 4. Next the @-6 means that a component of type A failed at ID 6, so we then decrement u_A to 0. Finally, the B-7 means that the node of type A at ID 10 has as its parent the node of type B at ID 7; but because $u_A = 0$ at this point, η does not include a factor $(1 - \phi_{B,A})$ for B-7.
- Iteration through the linked list at index B is as follows: @-2 and @-7 mean components of type B have failed at IDs 2 and 7 respectively, so we

decrement u_B from 2 to 1 and then from 1 to 0. Because u_B is now 0, η does not include the factor $(1 - \phi_{A,B})$ for A-6.

- Iteration through the linked list at index C is as follows. Because u_C starts out positive (i.e., $u_C = 1$), we include the factor $(1 - \phi_{A,C})$ from A-1. Next, @-5 means a component of type C has failed at ID 5, so we then decrement u_C to 0. Because $u_C = 0$ now, we do not include any further factors from this row in η.

Multiplying the contributions from each index results in $\eta = (1 - \phi_{B,A})(1 - \phi_{A,C})$.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1:11–33, 2004.

[2] C. Beounes, M. Aguera, J. Arlat, S. Bachmann, 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. In The Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS-23) Digest of Papers, pages 668–673, 1993.

[3] S. Bernson, E. de Souza e Silva, and R. Muntz. A methodology for the specification of Markov models. In W. Stewart, editor, Numerical Solution to Markov Chains, pages 11–37, 1991.

[4] A. Blum, P. Heidelberger, S. S. Lavenberg, M. K. Nakayama, and P. Shahabuddin. Modeling and analysis of system availability using SAVE. In Proceedings of the 23rd International Symposium on Fault Tolerant Computing, pages 137–141, 1994.

[5] L. Bortolussi, J. Hillston, D. Latella, and M. Massink. Continuous approximation of collective system behaviour: A tutorial. Performance Evaluation, 70:317–349, 2013.

[6] H. Boudali, P. Crouzen, and M. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Transactions on Dependable and Secure Computing, 7(2):128–143, 2010.

[7] M. Bouissou and J. L. Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. Reliability Engineering and System Safety, 82:149–163, 2003.

[8] W. G. Bouricius, W. C. Carter, and P. R. Schneider. Reliability modeling techniques for self-repairing computer systems. In Proceedings of the 1969 24th ACM National Conference, pages 295–309. ACM, 1969.

[9] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. Journal of Applied Probability, 31:59–75, 1994.

[10] P. Buchholz. Hierarchical Markovian models: symmetries and reduction. Performance Evaluation, 22(1):93–110, 1995.

[11] R. W. Butler. The SURE reliability analysis program. In AIAA Guidance, Navigation, and Control Conference, pages 198–204, 1986.

[12] B. A. Carreras, V. E. Lynch, I. Dobson, and D. E. Newman. Critical points and transitions in an electric power transmission model for cascading failure blackouts. Chaos, 12:985–1076, 2002.

[13] P. Crucitti, V. Latora, and M. Marchiori. Model for cascading failures in complex networks. Physical Review E, 69:045104, 2004.

[14] S. Distefano, A. Puliafito, and K. S. Trivedi. Guest editors' introduction: Special section on cloud computing assessment: Metrics, algorithms, policies, models, and evaluation techniques. IEEE Transactions on Dependable and Secure Computing, 10:251–252, 2013.

[15] J. B. Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. IEEE Transactions on Computers, 28:775–787, 1989.

[16] S. Goddard, R. Kieckhafer, and Y. Zhang. An unavailability analysis of firewall sandwich configurations. In Sixth IEEE International Symposium on High Assurance Systems Engineering, pages 139–148, 2001.

[17] A. Goyal, P. Shahabuddin, P. Heidelberger, V. Nicola, and P. W. Glynn. A unified framework for simulating Markovian models of highly dependable systems. IEEE Transactions on Computers, C-41:36–51, 1992.

[18] H. S. Gunawi, T. Do, J. M. Hellerstein, I. Stoica, D. Borthakur, and J. Robbins. Failure as a Service (FaaS): A cloud service for large-scale, online failure drills. Technical Report UCB/EECS-2011-87, Electrical Engineering and Computer Sciences, U. C. Berkeley, 2011.

[19] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In E. Lazowska, D. Terry, R. H. Arpaci-Dusseau, and J. Gehrke, editors, Proceedings of the ACM Symposium on Cloud Computing, pages 7:1–7:14, 2014.

[20] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar. Why does the cloud stop computing? Lessons from hundreds of service outages. In M. K. Aguilera, B. Cooper, and Y. Diao, editors, Proceedings of the 7th ACM Symposium on Cloud Computing, pages 1–16, 2016.

[21] C. Hirel, B. Tuffin, and K. S. Trivedi. SPNP version 6.0. Lecture Notes in Computer Science, 1786:354–357, 2000.

[22] S. M. Iyer, M. K. Nakayama, and A. V. Gerbessiotis. A Markovian dependability model with cascading failures. IEEE Transactions on Computers, 139:1238–1249, 2009.

[23] S. Juneja and P. Shahabuddin. Rare event simulation techniques: An introduction and recent advances. In S. G. Henderson and B. L. Nelson, editors, Elsevier Handbooks in Operations Research and Management Science: Simulation, pages 291–350. Elsevier, Amsterdam, 2006.

[24] D. E. Knuth. The Art of Computer Programming: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts, third edition, 1997.

[25] G. Krishnamurthi, A. Gupta, and A. K. Somani. The HIMAP modeling environment. In Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, pages 254–259, 1996.

[26] H. Langseth and L. Portinale. Bayesian networks in reliability. Reliability Engineering and System Safety, 92:92–108, 2007.

[27] R. G. Little. Controlling cascading failure: Understanding the vulnerabilities of interconnected infrastructures. Journal of Urban Technology, 9:109–123, 2002.

[28] H. M. Markowitz. Portfolio selection. Journal of Finance, 7:77–91, 1952.

[29] T. McDaniels, S. Chang, K. Peterson, J. Mikawoz, and D. Reed. Empirical framework for characterizing infrastructure failure interdependencies. Journal of Infrastructure Systems, 13(3):175–184, 2007.

[30] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri. RADYBAN: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. Reliability Engineering and System Safety, 93:922–932, 2008.

[31] B. Mukherjee, F. Habib, and F. Dikbiyik. Network adaptability from disaster disruptions and cascading failures. IEEE Communications Magazine, 52(5):230–238, May 2014.

[32] J. K. Muppala, R. M. Fricks, and K. S. Trivedi. Techniques for system dependability evaluation. In W. K. Grassmann, editor, Computational Probability, pages 445–480, The Netherlands, 2000. Kluwer.

[33] V. F. Nicola, P. Shahabuddin, and M. K. Nakayama. Techniques for fast simulation of models of highly dependable systems. IEEE Transactions on Reliability, 50:246–264, 2001.

[34] A. Peterson. oj! algorithms. http://ojalgo.org/, 2013.

[35] S. M. Ross. Stochastic Processes. Wiley, New York, second edition, 1995.

[36] R. A. Sahner, K. S. Trivedi, and A. Puliafito. Performance and Reliability Analysis of Computer Systems. Kluwer, Boston, 1996.

[37] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, pages 232–235, 1999.

[38] K. S. Trivedi. Probability and Statistics with Reliability, Queueing, and Computer Science Applications. Wiley, New York, second edition, 2001.

[39] M. Walter, M. Siegle, and A. Bode. Opensesame–the simple but extensive, structured availability modeling environment. Reliability Engineering and System Safety, 93:857–873, 2008.

[40] M. Xie, Y. S. Dai, and K.L. Poh. Computing Systems Reliability: Models and Analysis. Kluwer Academic, New York, 2004.
[41] H. Xu, L. Xing, and R. Robidoux. DRBD: Dynamic reliability block diagrams for system reliability modeling. International Journal of Computers and Applications, 31, 2009. DOI: 10.2316/Journal.202.2009.2.202-2552.
[42] M. Ye and Y. Tamir. Rehype: Enabling vm survival across hypervisor failures. ACM SIGPLAN Notices, 46(7):63–74, 2011.
[43] J.-F. Zheng, Z.-Y. Gao, and X.-M. Zhao. Clustering and congestion effects on cascading failures of scale-free networks. Europhysics Letters, 79:58002, 2007.

Mihir Sanghavi received his BS degree in Computer Science and Applied Mathematics from the New Jersey Institute of Technology in 2013. He is currently a technology associate at Morgan Stanley rewriting the platform that supports ultra high-net-worth individuals. He is responsible for gathering business requirements, designing, proofing and developing web and mobile software solutions. His current research interest is in natural language processing and algorithmic trading through dark pools.

Sashank Tadepalli received the BS degree in Computer Science from the New Jersey Institute of Technology in 2013. He is currently a lead systems engineer at Kydia Inc, where he is responsible for designing, engineering and developing web-based software solutions. He has held a previous position in Tata Consultancy Services as a solutions engineer and technology consultant. His current research interest is in the study of high-performance applications in the mobile-web domain.

Timothy J. Boyle Jr. received BS and MS degrees in Computer Science from the New Jersey Institute of Technology.

Matthew Downey received a BS in Computer Science from the New Jersey Institute of Technology.

Marvin K. Nakayama is a professor in the Department of Computer Science at the New Jersey Institute of Technology. He has previously held positions at Rutgers University's Graduate School of Management, Columbia Business School in New York, and at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York. He received a B.A. in mathematics/computer science from University of California, San Diego, and an M.S. and Ph.D. in operations research from Stanford University. Dr. Nakayama won second prize in the 1992 George E. Nicholson Student Paper Competition sponsored by INFORMS and is a recipient of a CAREER Award from the National Science Foundation. He was the simulation area editor for INFORMS Journal on Computing from 2007–2016, and is an associate editor for ACM Transactions on Modeling and Computer Simulation. His research interests include simulation and modeling, applied probability, statistics, dependability modeling, energy and risk analysis.