Les Cahiers du GERAD ISSN: 0711–2440

Static network reliability estimation under the Marshall-Olkin copula

Z.I. Botev, P. L'Ecuyer,
R. Simard, B. Tuffin

G–2015–12

February 2015

Static network reliability estimation under the Marshall-Olkin copula

Zdravko I. Botev[a]

Pierre L'Ecuyer[b,c]

Richard Simard[c]

Bruno Tuffin[d]

[a] *School of Mathematics and Statistics, The University of New South Wales, Sydney, NSW 2052, Australia*

[b] *GERAD, HEC Montréal, Montréal (Québec) Canada, H3T 2A7*

[c] *DIRO, Université de Montréal, Montréal (Québec) Canada, H3C 3J7*

[d] *Inria Rennes Bretagne Atlantique, 35042 Rennes Cedex, France*

botev@unsw.edu.au
lecuyer@iro.umontreal.ca
richard.simard.1@umontreal.ca
bruno.tuffin@inria.fr

February 2015

Les Cahiers du GERAD

G–2015–12

Abstract: In a static network reliability model one typically assumes that the failures of the components of the network are independent. This simplifying assumption makes it possible to estimate the network reliability efficiently via specialized Monte Carlo algorithms. Hence, a natural question to consider is whether this independence assumption can be relaxed, while still attaining an elegant and tractable model that permits an efficient Monte Carlo algorithm for unreliability estimation. In this article we provide one possible answer by considering a static network reliability model with dependent link failures, based on a Marshall-Olkin copula, which models the dependence via shocks that take down subsets of components at exponential times, and propose a collection of adapted versions of permutation Monte Carlo (PMC, a conditional Monte Carlo method), its refinement called the turnip method, and generalized splitting (GS) methods, to estimate very small unreliabilities accurately under this model. The PMC and turnip estimators have bounded relative error when the network topology is fixed while the link failure probabilities converge to 0. When the network (or the number of shocks) becomes too large, PMC and turnip eventually fail, but GS works nicely for very large networks, with over 5000 shocks in our examples.

Key Words: Static network reliability, dependent components, Marshal-Olkin copula, splitting, conditional Monte Carlo, permutation Monte Carlo, turnip.

Acknowledgments: This work has been supported by the Australian Research Council under DE140100993 Grant to Z. Botev, an NSERC-Canada Discovery Grant, a Canada Research Chair, and an Inria International Chair to P. L'Ecuyer, the EuroNF Network of Excellence to B. Tuffin, and INRIA's associated team MOC-QUASIN to all authors. P. L'Ecuyer acknowledges the support of the Faculty of Science Visiting Researcher Award at UNSW: most of the paper was written during his visit. We benefited from the computing facilities of the Réseau québécois de calcul haute performance (RQCHP).

1 Introduction

Network reliability estimation problems occur in a wide range of situations and applications, including telecommunications, transportation, energy supply, and many others (Barlow and Proschan, 1975; Gertsbakh and Shpungin, 2010). In this paper, we focus on a classical *static network reliability problem* in which a given set of nodes of the network is selected a priori, a random subset of the links in the network fail, and we want to estimate the reliability of the network, defined as the probability $1-u$ that the selected nodes are all connected by operational links. For large networks, an exact computation of the reliability $1-u$, or equivalently of the *unreliability* u, is usually impractical and one has to rely on Monte Carlo techniques (Cancela et al., 2009; Gertsbakh and Shpungin, 2010). In fact, computing u is known as a #P-complete computational problem (Colbourn, 1987). When the network is highly reliable, u becomes a rare-event probability (it gets very small) and direct (crude) Monte Carlo is also impractical. Various rare-event simulation methods have been developed to address that problem. They include conditional Monte Carlo methods, importance sampling, use of control variates, splitting techniques, and combinations of these; see Alexopoulos and Shultes (2001), Botev et al. (2013), Botev et al. (2013), Cancela and El Khadiri (1995), Cancela and El Khadiri (2003), Cancela et al. (2009), Cancela et al. (2014), Elperin et al. (1991), Gertsbakh and Shpungin (2010), L'Ecuyer et al. (2011), Lomonosov and Shpungin (1999), Sahinoglu and Rice (2010), Tuffin et al. (2014), and the references given there. All these methods were originally developed for the special case where the links fail independently of each other.

In this setting where links are independent, both theory and empirical experiments tell us that for networks of moderate size and extremely small unreliability u, the approximate zero-variance importance sampling scheme of L'Ecuyer et al. (2011) and the turnip method (Gertsbakh and Shpungin, 2010), which is a refinement of the permutation Monte Carlo (PMC) method of Elperin et al. (1991) and Lomonosov and Shpungin (1999), are generally the best performers. In fact, these methods have been proved to give estimators with bounded relative error (BRE), which means that their standard deviation divided by the mean u remains bounded, when the link unreliabilities and u converge to 0 while the network is fixed. But when the size of the network increases, these methods eventually become inefficient (unless the network has special structure). They do not have BRE in an asymptotic regime where the number of links increases to infinity while u remains of the same order. For very large networks where the link unreliabilities are not so small but u is small because the nodes are connected by a huge number of paths (high redundancy), the best method we know is the generalized splitting (GS) algorithm of Botev et al. (2013). It works for general networks having several thousand links and $u < 10^{-15}$, for example.

The PMC, turnip, and GS methods all rely on a vector $\mathbf{Y}$ of continuous latent variables, which represent the repair times of all the links; that is, we turn the static system into a dynamic one in which we assume that each link is initially failed, gets repaired at some random time, and the set of links that are repaired at time 1 are those that are considered operational in the static network. PMC and turnip only look at the *order* in which the links are repaired (only the permutation, not the repair times) and compute the probability that the network is failed at time 1, conditional on this order, as an estimator of u. For very large networks, the important permutations, that contribute significantly to the unreliability, often are sampled much too rarely, so we are again in a rare-event situation. To address this problem, GS learns adaptively the regions where it is important to sample more, in the space of values of $\mathbf{Y}$. There is also a dual scheme in which one assumes that all links are initially operational, and one uses a latent vector $\mathbf{Y}$ of exponential *failure times* instead of repair times; it also works for PMC, turnip, and GS.

Botev et al. (2013) use GS to construct a kernel density estimate that mimics the distribution conditional on the rare event of network failure, and then use it as an importance sampling density. Empirically, this method often performs better than GS. Botev et al. (2013, Section 8) adapted GS to a situation of dependent links. They modeled the dependence via a normal or a t copula, and used a hit-and-run re-sampler in the GS method. They were able to estimate very small unreliabilities accurately for a (classical) dodecahedron network example with 20 nodes and 30 links. However, this approach becomes very time-consuming for large networks and these copulas may not be always appropriate to model the dependence in real-life networks.

In this paper, we consider a different way of modeling the dependence, via the exponential Marshall-Olkin (MO) copula (Marshall and Olkin, 1967; Nelsen, 2006). In our context, this is equivalent to assuming that components (links) can fail simultaneously in groups. This is very natural, as it may represent a situation where a subset of components fail together due to a common cause or by a cascading effect (Iyer et al., 2009; Kalyoncu and Sankur, 1992; Nelsen, 2006). In fact, this interpretation is already at the basis of the definition of the MO copula. This being said, our goal is not to study the relevance of the MO copula for static reliability networks but to develop effective rare-event simulation methods for this model.

With the MO model, a direct adaptation of PMC, turnip, and GS by using the vector of link repair times as latent variables becomes too complicated and ineffective, because it involves complicated conditional distributions. Our main contribution is to show how these methods can be adapted by using different sets of latent variables $\mathbf{Y}$. We develop corresponding algorithms and compare them numerically. These adaptations can handle very small unreliabilities and large networks.

By definition, the exponential MO model is specified in terms of a vector $\mathbf{Y}$ of latent variables that represent independent exponential shock times (Marshall and Olkin, 1967). Each shock takes down simultaneously a given subset $\mathbf{s}$ of components (those in $\mathbf{s}$ that are already down remain so) and occurs at an exponentially-distributed random time with rate $\lambda_{\mathbf{s}}$. In one of their variants, the PMC, turnip, and GS algorithms use this $\mathbf{Y}$ as a vector of latent variables in a similar way as for the independent case, except that link failure times are replaced by shock times. In the dual version, link repairs cannot be replaced by repairs of groups of components; but we show how they can be replaced by anti-shocks (shock removals) which also occur at exponential times with appropriate rates. Initially, we assume that all the shocks have occurred, and we remove them one by one when their corresponding anti-shocks occur. Removing a shock does not necessarily repair (some of) the affected links, because other shocks may have also taken down these links. To find the repair time of each link, we initialize a counter to the total number of shocks that affect this link, and decrease the counter by one each time one of these shocks is removed. The link is repaired when the counter reaches 0. These constructions provide elegant and efficient algorithms.

Although the problem and the algorithms are defined in this paper in terms of the connectivity of a subset of nodes in a network, everything generalizes easily to a multicomponent system where each component has a binary state (operating or failed), and the binary system state is a monotone increasing function of the component states, called the structure function (Barlow and Proschan, 1975). In the algorithms, "network" is replaced by "system" and the links are replaced by the system components. A key requirement for the implementation is to be able to quickly find if there is a change in the value of the structure function when one or more binary states are changed (when a shock is added or removed).

For the network connectivity problem studied in this paper, we maintain graph data structures to represent the state of the network and anticipate efficiently what happens to the structure function value when a link is added or removed. As an example of a generalization that could be handled, one may consider that the links in the network have a length and that each pair of selected nodes must be connected by a path of length no larger than a given number. Another example is if the links have a capacity, and the structure function indicates if the maximum flow that we can send from a source to a destination (two given nodes) reaches a given threshold. A further generalization would be to consider systems with multistate (instead of just binary) components (Natvig, 2011).

The rest of the paper is organized as follows. In Section 2, we define the model and problem, and explain how the MO copula introduces dependence via latent variables that represent shock times. We state a relationship between the failure rates in the MO model and the reliabilities of subsets of components. In Section 3, we adapt the PMC algorithm to our setting. We consider different variants, one where we generate and sort all shock times, one where we only generate the (partial) permutation directly, one where we scan the shocks in reverse order to reconstruct the network, and one where we generate anti-shocks instead of shocks. We examine numerical issues that occur when computing the conditional expectation for PMC, and we provide a very effective formula for the special case where the shock rates are all equal. In Section 4, we show how to adapt the turnip method to our case. Again, we give different variants, with shocks and with anti-shocks, and we summarize the different PMC and turnip variants and their combinations. In Section 5, we give sufficient conditions under which the PMC and turnip estimators can be proved to have

BRE, and also necessary conditions, both in the context where $u \to 0$ for a fixed network. Interestingly, the conditions are weaker with anti-shocks than with shocks. In Section 6, we adapt the GS method to our setting, both with shocks and with anti-shocks. In Section 7, we discuss some data structures used for an efficient implementation of these algorithms. Such techniques are required when the graph gets large. In Section 8, we summarize our numerical experiments with various examples. This is followed by a conclusion in Section 9. The two conference papers of Botev et al. (2012) and Botev et al. (2014) gave a preliminary sketch of some of the ideas developed here.

2 Problem formulation and MO copula model

We consider a graph with set of nodes $\mathcal{V}$ and a set of m links that connect m distinct pairs of nodes. Associated with each link i is a Bernoulli random variable X_i denoting whether the link is operational ($X_i = 1$) or failed ($X_i = 0$), with $\mathbb{P}(X_i = 0) = u_i$, the unreliability of link i, for $i = 1, \ldots, m$. The random vector $\mathbf{X} = (X_1, \ldots, X_m)$ represents the configuration (or state) of the network. Typically, the coordinates of $\mathbf{X}$ are assumed to be independent, but here we relax this assumption. A subset of nodes $\mathcal{V}_0 \subset \mathcal{V}$ is selected a priori and the network is said to be *operational* if all nodes in $\mathcal{V}_0$ are connected to each other by at least one tree of operational links. We define the *structure function* Φ of the graph by $\Phi(\mathbf{x}) = 1$ when the network is operational in configuration $\mathbf{x}$, and $\Phi(\mathbf{x}) = 0$ otherwise. The *unreliability* u of the network is the probability that it is *not* operational:

$$u = \mathbb{P}(\Phi(\mathbf{X}) = 0).$$

The static network reliability problem consists in estimating u.

The most general way of modeling the distribution of $\mathbf{X}$ for a static network is to assign a probability $p(\mathbf{x}) \geq 0$ to each of the 2^m configurations $\mathbf{x} \in \{0,1\}^m$ of the system, so that these probabilities sum to 1. In this paper, the multivariate Bernoulli distribution of $\mathbf{X}$ is defined by an MO copula model which can be almost as general, as we shall see below (it can also have 2^m degrees of freedom). The MO copula is defined in terms of a vector $\mathbf{Y}$ of independent exponential latent variables that represent shock times. For any subset $\mathbf{s}$ of components (or links), a shock that provokes the joint failure of all components of $\mathbf{s}$ occurs at an exponential time with rate $\lambda_{\mathbf{s}}$. Let $\mathcal{L} = \{\mathbf{s} : \lambda_{\mathbf{s}} > 0\}$ and $\kappa = |\mathcal{L}|$. We will index the elements of $\mathcal{L}$ by j and number them from 1 to κ. For practical implementations, we shall assume that κ is not too large, so that we can easily store and visit all elements of $\mathcal{L}$. We denote the jth subset by $\mathbf{s}(j)$, its corresponding shock rate by $\lambda_j = \lambda_{\mathbf{s}(j)}$, and the random exponential shock time by Y_j. The vector $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$ is the latent state of the system. Component i fails at time $\tilde{Y}_i = \min\{Y_j : i \in \mathbf{s}(j)\}$, which is an exponential with rate $\tilde{\lambda}_i = \sum_{\{j : i \in \mathbf{s}(j)\}} \lambda_j$. We denote its state at any time $\gamma \geq 0$ by $X_i(\gamma) = \mathbb{I}[\tilde{Y}_i > \gamma]$, and let $\mathbf{X}(\gamma) = (X_1(\gamma), \ldots, X_m(\gamma))$. The time at which the network fails is

$$\tilde{S}(\mathbf{Y}) = \inf\{\gamma \geq 0 : \Phi(\mathbf{X}(\gamma)) = 0\}.$$

By definition, the MO copula is the multivariate distribution of $\mathbf{U} = (U_1, \ldots, U_m)$, where $U_i = 1 - \exp[-\tilde{\lambda}_i \tilde{Y}_i]$ is uniform over $(0,1)$ for each i.

We put $X_i = X_i(1)$ and $\mathbf{X} = \mathbf{X}(1)$, so that the operational links in the static network are those that are still alive at time 1, and the static network is operational if and only if $\tilde{S}(\mathbf{Y}) > 1$. In this model, $\mathbb{P}[X_i(\gamma) = 1] = \mathbb{P}[\tilde{Y}_i > \gamma] = \mathbb{P}[U_i > 1 - \exp[-\tilde{\lambda}_i \gamma]] = \exp[-\tilde{\lambda}_i \gamma]$, and if we want the reliability of component i to be $\mathbb{P}[X_i = 1] = r_i$, we must have $\tilde{\lambda}_i = -\ln r_i$. But these r_i or $\tilde{\lambda}_i$ are not sufficient to specify the model, because they do not specify the dependence.

We can write a system of linear relationships between the survival probabilities of subsets $\mathbf{r}$ of components and the nonzero rates $\lambda_{\mathbf{s}}$. For each subset $\mathbf{r}$, let $q_{\mathbf{r}}$ be the probability that all components in $\mathbf{r}$ are up in the network (i.e., survive up to time 1). Then we have

$$g_{\mathbf{r}} \overset{\text{def}}{=} -\ln q_{\mathbf{r}} = \sum_{\mathbf{s}} \delta_{\mathbf{s},\mathbf{r}} \lambda_{\mathbf{s}} \qquad \text{for all } \mathbf{r},$$

where $\delta_{\mathbf{s},\mathbf{r}} = \mathbb{I}[\mathbf{s} \cap \mathbf{r} \neq \emptyset]$. If we know (or decide to specify) the $q_{\mathbf{r}}$'s (or equivalently the $g_{\mathbf{r}}$'s), in principle we can compute the corresponding rates $\lambda_{\mathbf{s}}$ by solving this system of 2^m equations in 2^m unknown. An explicit formula for the solution is given in Lemma 4.1 of Sun et al. (2011):

$$\lambda_{\mathbf{s}} = \sum_{\mathbf{r}\subseteq\mathbf{s}} (-1)^{|\mathbf{s}|-|\mathbf{r}|+1} g_{\bar{\mathbf{r}}}, \tag{1}$$

where $\bar{\mathbf{r}}$ is the complement of $\mathbf{r}$. If all the rates $\lambda_{\mathbf{s}}$ given by this formula are non-negative, then they provide an MO representation that corresponds to the given probabilities $q_{\mathbf{r}}$. Conversely, if such a representation exists, then the $\lambda_{\mathbf{s}}$ must satisfy (1) and be non-negative.

Of course, when m is large, solving the full linear system (1) is impractical, because its size increases exponentially in m. One must then restrict a priori the set of nonzero $\lambda_{\mathbf{s}}$, and estimate them in some way (e.g., via least squares). Another approach could be to let many $\lambda_{\mathbf{s}}$'s be nonzero, but to parameterize them with a small number of parameters, and estimate the parameters. Our algorithms in this paper are designed for the former case, where the number of nonzero $\lambda_{\mathbf{s}}$'s is limited.

For any given pair of links (i,k), let $\tilde{\lambda}_{i,k} = \sum_{\{j:i,k\in\mathbf{s}(j)\}} \lambda_j$, the total rate of shocks that affect both i and k simultaneously. We have $\mathrm{Cov}[X_i, X_k] = \mathbb{P}[X_i = X_k = 1] - \mathbb{P}[X_i = 1]\mathbb{P}[X_k = 1] = \mathbb{P}[\tilde{Y}_i > 1, \tilde{Y}_k > 1] - \mathbb{P}[\tilde{Y}_i > 1]\mathbb{P}[\tilde{Y}_k > 1] = \exp[-\tilde{\lambda}_i - \tilde{\lambda}_k + \tilde{\lambda}_{i,k}] - \exp[-\tilde{\lambda}_i - \tilde{\lambda}_k] \geq 0$. That is, the MO copula cannot give negative covariances between the X_i's. For example, a two-component system where each component is down with positive probability, but the two cannot be down at the same time cannot be modeled by the MO copula, because the covariance is negative. This type of constraint applies more generally. For any subset $\mathbf{s}$ of components, and any partition of $\mathbf{s}$, the probability that all components in $\mathbf{s}$ are down must be at least as large as the product of the failure probabilities over the subsets that form the partition. Most real-life systems should satisfy this condition. Negative dependence between failures is rarely realistic. In this sense, the MO copula permits one to specify a very general and flexible class of distributions for $\mathbf{X}$.

One good feature of the MO model is that it can cover cascading failures. For example, suppose that the subset $\mathbf{s}_1$ of components fail together at rate λ_1 and that such a failure also triggers immediate failure of subset $\mathbf{s}_2$ with probability p. We can model this simply by assigning failure rate $(1-p)\lambda_1$ to subset $\mathbf{s}_1$ and $p\lambda_1$ to subset $\mathbf{s}_1 \cup \mathbf{s}_2$. This generalizes easily to more general cascading.

The crude Monte Carlo method estimates the unreliability $u = \mathbb{P}[\tilde{S}(\mathbf{Y}) \leq 1]$ as follows. Generate $\mathbf{Y}$, sort its coordinates by increasing order, and remove in this order the components (links) affected by these shocks until the network fails. The time of the last considered shock, at which the network fails, is $\tilde{S}(\mathbf{Y})$. Repeat this n times, independently, and estimate u by the average of the n replicates of $\mathbb{I}[\tilde{S}(\mathbf{Y}) < 1]$. It is well-known that when u is very small, this performs very poorly because the indicator is nonzero extremely rarely (Asmussen and Glynn, 2007; Rubino and Tuffin, 2009). In what follows, we propose viable alternative methods that perform much better for small u.

3 Adapting PMC to the MO model

3.1 PMC with shocks

To apply the PMC method with the MO model, we can generate the vector $\mathbf{Y}$ of shock times Y_j, sort them by increasing order to get the order statistics $Y_{(1)}, \ldots, Y_{(\kappa)}$, and retain only the order in which the shocks occur, i.e., the permutation $\pi = (\pi(1), \ldots, \pi(\kappa))$ such that $Y_{(j)} = Y_{\pi(j)}$ for each j. Conditional on this permutation π, we then compute numerically the probability that the graph is failed at time 1. This is the PMC estimator. To compute this probability, we can add the shocks j one by one in their order of occurrence and remove the links $i \in \mathbf{s}(j)$ affected by these shocks, until the network fails. The number of shocks required to put the system down is a random variable C_{s} called the *critical shock number*.

At step k of this procedure, before adding the kth shock $\pi(k)$, we know that the time $A_k = Y_{\pi(k)} - Y_{\pi(k-1)}$ until this next shock occurs is an exponential with rate Λ_k equal to the sum of rates of all the shocks that did not occur yet. These Λ_k obey $\Lambda_1 = \lambda_1 + \cdots + \lambda_\kappa$, and $\Lambda_{j+1} = \Lambda_j - \lambda_{\pi(j)}$ for $j \geq 1$. Conditional on π and on $C_{\mathrm{s}} = c$, the failure time is $A_1 + \cdots + A_c$, a sum of c independent exponential random variables with rates $\Lambda_1, \ldots, \Lambda_c$. This sum has an *hypoexponential* distribution, whose cumulative distribution function (cdf)

$\mathbb{P}[A_1 + \cdots + A_c \le \gamma \mid \pi]$, which is the PMC estimator of u based on one simulation run, can be written as a matrix exponential and can be computed as we explain below (see also Botev et al. (2013)).

This PMC procedure is stated in Algorithm 1. In this algorithm (and all others stated in this paper), indentation delimits the scope of the **if**, **else**, and **for** statements. This algorithm computes an unbiased estimator U of the unreliability u and returns its value. It will be invoked n times, independently, to obtain n realizations $U_1, \ldots, U_n$ of U, and one can estimate u by the average $\bar{U}_n = (U_1 + \cdots + U_n)/n$ and the variance $\sigma^2 = \mathrm{Var}[U]$ by the empirical variance $S_n^2 = \sum_{i=1}^n (U_i - \bar{U}_n)^2/(n-1)$. This can be used to compute a confidence interval on u.

ALGORITHM 1: A PMC algorithm with shocks

```
$\Lambda_1 \leftarrow \lambda_1 + \cdots + \lambda_\kappa$
$\mathbf{x} = (x_1, \ldots, x_m) \leftarrow (1, \ldots, 1)$      // all links are operational
$k \leftarrow 1$
draw the $\kappa$ shock times and sort them in increasing order
this gives the ordered list $\pi = (\pi(1), \ldots, \pi(\kappa))$
while the nodes in $\mathcal{V}_0$ are all connected do
    $j \leftarrow \pi(k)$
    for all $i \in \mathbf{s}(j)$ do
        if $x_i = 1$ then
            $x_i \leftarrow 0$, remove link $i$ from graph
    $k \leftarrow k + 1$
    $\Lambda_k \leftarrow \Lambda_{k-1} - \lambda_j$
$C_s \leftarrow k - 1$      // shock number at which $\mathcal{V}_0$ is disconnected
return $U \leftarrow \mathbb{P}[A_1 + \cdots + A_{C_s} \le 1 \mid \pi]$, an unbiased estimate of $u$ computed using $C_s, \Lambda_1, \ldots, \Lambda_{C_s}$.
```

3.2 Computing the hypoexponential cdf

The hypoexponential *complementary cdf* can be written explicitly as

$$\mathbb{P}[A_1 + \cdots + A_c > \gamma] = \sum_{j=1}^{c} e^{-\Lambda_j \gamma} p_j \tag{2}$$

where

$$p_j = \prod_{k=1,\, k \ne j}^{c} \frac{\Lambda_k}{\Lambda_k - \Lambda_j},$$

and (2) can be computed in $\mathcal{O}(c^2)$ time; see Ross (2007, page 299) and Gertsbakh and Shpungin (2010, Appendix B). However, this formula is numerically unstable when c is large or if the shock rates λ_j's are too small. What goes wrong is that the products p_j in (2) are very large and of comparable sizes, and have alternating signs $(-1)^{j-1}$. When the λ_j are small, the Λ_j and the exponential factors that multiply those products are close to each other and near 1, so we have a sum of very large alternating terms while the sum itself is between 0 and 1; a situation that leads to a loss of precision and numerical errors. Note that the sizes of the products p_j themselves do not depend on the sizes of the λ_j's. For example, if we multiply all λ_j's by the same constant, this multiplies the Λ_j's by the same constant and changes nothing in the products p_j.

The hypoexponential complementary cdf (2) can in fact be written as a matrix exponential (see Botev et al. (2013)) and a more stable and accurate algorithm to compute this matrix exponential is given in Higham (2009). However, that algorithm involves multiplication of $c \times c$ matrices, which has $\mathcal{O}(c^3)$ time complexity in our implementation, so it is much slower and becomes impractical when c is too large. As an illustration, with our implementation, in one example, the matrix exponential algorithm was about 15 times slower than using (2) for $c = 10$ and about 825,000 times slower for $c = 1000$.

Another numerical problem comes from the fact that (2) actually gives $1 - U$ in Algorithm 1. When U is very small (which is typical), the representation error on $1 - U$ in double precision arithmetic (with 52-bit mantissa) is around 10^{-15}, which means that we have limited accuracy on U. In fact, we have (roughly) no accuracy at all on the returned U when $U < 10^{-15}$. To derive a more direct formula for U, note that

$$\sum_{j=1}^{c} p_j = \mathbb{P}\left[A_1 + \cdots + A_c > 0\right] = 1,$$

so we can rewrite

$$\mathbb{P}\left[A_1 + \cdots + A_c \leq \gamma\right] \quad = \quad 1 - \sum_{j=1}^{c} e^{-\Lambda_j \gamma} p_j \quad = \quad \sum_{j=1}^{c} (1 - e^{-\Lambda_j \gamma}) p_j \tag{3}$$

which permits one to compute U directly, by using predefined functions that can compute $1 - e^{-\Lambda_j \gamma}$ accurately. For example, when $\Lambda_j \gamma$ is very small, $e^{-\Lambda_j \gamma}$ is very close to 1, so if we do the subtraction from 1 explicitly we may lose all accuracy, but one can write $1 - e^{-\Lambda_j \gamma} \approx \Lambda_j \gamma - (\Lambda_j \gamma)^2/2 + \cdots$, and this series converges very fast (so it permits one a very accurate evaluation) when $\Lambda_j \gamma$ is very small. When c gets too large, however, (3) also becomes unstable in the same way as (2). Our GS method does not have this type of limitation.

3.3 Generating the permutation directly

The permutation π can also be generated directly, without generating and sorting the shock times Y_j, and only for the first C_s shocks, as follows. At step k, the kth shock is selected among the shocks still in consideration, with probability λ_j / Λ_k for shock j, where Λ_k is the sum of occurrence rates of the shocks still under consideration, for $k \geq 1$. This avoids the sorting, which takes $\mathcal{O}(\kappa \log \kappa)$ time.

However, unless the λ_j's are all equal (we will come back to this special case in Section 3.6), updating the probabilities λ_j / Λ_k and selecting the next shock according to those probabilities at each step involves overhead when these probabilities are different. In fact, computing and updating all these probabilities in general would take $\mathcal{O}(\kappa)$ time at each step, and therefore $\mathcal{O}(C_s \kappa)$ time overall, which could be much slower than generating and sorting the shock times. To avoid recomputing the probabilities at each step, a different approach to generate the permutations is to compute a table of the probabilities λ_j / Λ_k and the corresponding cdf before doing the n simulation runs, and re-use this same table at all steps and for all runs to generate the sequence of shocks. When a shock occurs, we mark it in the table as already selected, and if it is selected again later we just skip it and generate another one, as in an acceptance-rejection method. After each run, we remove all the marks to reset the table for the next run. This works fine and is typically more efficient that generating and sorting all shock times when $C_s / \kappa \ll 1$. But it can become very slow when κ is large and C_s / κ is near 1, because of the high rejection probabilities when k gets large. The PMC procedure with direct generation of π is stated in Algorithm 2. In our experiments, Algorithm 2 was typically around 10 to 30% faster than Algorithm 1. Nevertheless, for the empirical results reported here with the other PMC and turnip methods that follow, we simply generated and sorted the shocks.

ALGORITHM 2: PMC algorithm without generating the shock times

```
$\Lambda_1 \leftarrow \lambda_1 + \cdots + \lambda_\kappa$
$\mathcal{L}_1 \leftarrow \{1, \ldots, \kappa\}$      // shocks still under consideration
$\mathbf{x} = (x_1, \ldots, x_m) \leftarrow (1, \ldots, 1)$      // all links are operational
$k \leftarrow 1$
while the nodes in $\mathcal{V}_0$ are all connected do
    draw $J$ from $\mathcal{L}_1$, with $\mathbb{P}[J = j] = \lambda_j / \Lambda_k$
    $\pi(k) \leftarrow J$
    for all $i \in \mathbf{s}(j)$ do
        if $x_i = 1$ then
            $x_i \leftarrow 0$, remove link $i$ from graph
    $k \leftarrow k + 1$
    $\Lambda_k \leftarrow \Lambda_{k-1} - \lambda_J$ and remove $J$ from $\mathcal{L}_1$
$C_s \leftarrow k - 1$
return $U \leftarrow \mathbb{P}[A_1 + \cdots + A_{C_s} \leq 1 \mid \pi]$, an unbiased estimate of $u$ computed using $C_s, \Lambda_1, \ldots, \Lambda_{C_s}$.
```

3.4 Scanning the shocks in reverse order

To compute C_s given the (full) permutation π, instead of adding shocks until the system fails, one can assume that *all* the shocks have already occurred, and we remove them one by one in their reverse order of occurrence, until the system is repaired. If there are c_i shocks with positive rates that can affect component i, then component i will be repaired when those c_i shocks have been removed. We assume that $c_i > 0$ for each i. In the implementation, for each component i, we maintain a counter d_i that starts at c_i and decreases by 1 each time a shock that affects component i is removed. That is, when a shock j is removed, for all components $i \in \mathbf{s}(j)$, we subtract 1 from d_i. Whenever d_i becomes 0, link i gets *repaired*, so we add it to the current configuration and if it connects two different connected components of the network (that are not already connected), we merge those two connected components, exactly as in Botev et al. (2013). As soon as the system becomes operational, we know the critical shock number C_s defined in Section 3, and we can compute the same unreliability estimator as in Algorithm 1, based on the first C_s shocks. This C_s is the number of the last shock that had been removed before the system became operational. This gives Algorithm 3. Note that to implement this, we need to generate all the shocks. The difference between Algorithms 1 and 3 is that the latter reconstructs the network by removing the shocks one by one, whereas the former destroys the networks by adding the shocks one by one. Both work on the same ordered list of shocks and yield exactly the same estimator. Only the computing time differs. Generally speaking, adding the shocks in their order of occurrence is faster when C_s is much smaller than C_a, otherwise starting with all the shocks and removing them one by one (adding anti-shocks) is usually faster, because updating the data structures that represent the graph is faster when adding links than when removing links.

ALGORITHM 3: A reverse PMC algorithm

```
draw the $\kappa$ shock times and sort them in increasing order
$\mathcal{L}_1 \leftarrow \{\pi(1), \dots, \pi(\kappa)\}$
   // list of shocks, by increasing order of occurrence
for $i = 1$ to $m$ do
    $d_i \leftarrow c_i$ and $x_i \leftarrow 0$    // configuration of links
$k \leftarrow \kappa$
while the nodes in $\mathcal{V}_0$ are not all connected do
    $j \leftarrow \pi(k)$
    for all $i \in \mathbf{s}(j)$ do
        $d_i \leftarrow d_i - 1$
        if $d_i = 0$ then
            $x_i \leftarrow 1$, and if link $i$ joints two connected components, merge them
    $k \leftarrow k - 1$
$C_s \leftarrow k + 1$    // shock number at which $\mathcal{V}_0$ is disconnected
$\Lambda_1 \leftarrow \lambda_1 + \cdots + \lambda_\kappa$
for $k = 1$ to $C_s - 1$ do
    $\Lambda_{k+1} \leftarrow \Lambda_k - \lambda_{\pi(k)}$
return $U \leftarrow \mathbb{P}[A_1 + \cdots + A_{C_s} \le 1 \mid \pi]$, estimator of $u$.
```

3.5 PMC with anti-shocks generated directly

It is also possible to define exponential anti-shock times, and use them to define a different PMC estimator, based on a sum of anti-shock times. We start in a state where all the shocks have occurred. Each anti-shock removes the corresponding shock. If anti-shock j occurs at time R_j which is an exponential of rate μ_j, we must have $1 - e^{-\mu_j} = \mathbb{P}[R_j \le 1] = \mathbb{P}[Y_j > 1] = e^{-\lambda_j}$, and therefore

$$\mu_j = -\ln(1 - e^{-\lambda_j}).$$

This gives the correct probability for the shock to have occurred after time 1. Thus, we can generate the anti-shocks times R_j as exponential with those rates μ_j and sort them in increasing order. This sorting corresponds to a permutation $\pi' = (\pi'(1), \dots, \pi'(\kappa))$ as for the shocks. This π' has actually the same distribution as the reverse of the permutation π for the shocks. Now, we add the anti-shocks in this order until all nodes in $\mathcal{V}_0$ are connected, say when the C_ath anti-shock occurs. Before adding the kth anti-shock $\pi'(k)$, the time $A'_k = R_{\pi'(k)} - R_{\pi'(k-1)}$ until this anti-shock occurs is exponential with rate Λ'_k defined recursively via $\Lambda'_1 = \mu_1 + \cdots + \mu_\kappa$ and $\Lambda'_{j+1} = \Lambda'_j - \mu_{\pi'(j)}$ for $j \geq 1$. Conditional on π' and on $C_\mathrm{a} = c$, the failure time is $A'_1 + \cdots + A'_c$, a sum of c independent exponential random variables with rates $\Lambda'_1, \ldots, \Lambda'_c$. Finally, we have the unbiased unreliability estimator $U' = \mathbb{P}[A'_1 + \cdots + A'_{C_\mathrm{a}} > 1 \mid \pi']$, which can be computed as explained earlier, but with the Λ_j replaced by Λ'_j in the formulas and in the definition of p_j.

If the permutation π for the shocks has critical shock number C_s, π' is the reverse permutation and it has critical anti-shock number C_a, then $C_\mathrm{s} + C_\mathrm{a} = \kappa + 1$. This means that we can generate either π or π' and compute at the same time either C_s or C_a from one of these permutations, by adding shocks or by adding anti-shocks, and then use either $U = \mathbb{P}[A_1 + \cdots + A_{C_\mathrm{s}} \leq 1 \mid \pi]$ or $U' = \mathbb{P}[A'_1 + \cdots + A'_{C_\mathrm{a}} > 1 \mid \pi']$ as an estimator. One advantage of the latter when u is small is that it is expressed directly in terms of the tail probability given in (2), which is often much more stable than (3), used in Algorithm 1, when u is very small. Algorithm 4 gives one version of this, with generation and sorting of anti-shock times, and U' as an estimator.

ALGORITHM 4: A PMC algorithm with exponential anti-shocks

```
$\Lambda'_1 \leftarrow \mu_1 + \cdots + \mu_\kappa$
for $i = 1$ to $m$ do
    $d_i \leftarrow c_i$ and $x_i \leftarrow 0$    // configuration of links
draw the $\kappa$ anti-shock times $R_j$ and sort them in increasing order to obtain the permutation
$\pi' = (\pi'(1), \ldots, \pi'(\kappa))$
$k \leftarrow 1$
while the nodes in $\mathcal{V}_0$ are not all connected do
    $j \leftarrow \pi'(k)$
    for all $i \in \mathbf{s}(j)$ do
        $d_i \leftarrow d_i - 1$
        if $d_i = 0$ then
            $x_i \leftarrow 1$, and if link $i$ joints two connected components, merge them
    $\Lambda'_{k+1} \leftarrow \Lambda'_k - \mu_{\pi'(k)}$
    $k \leftarrow k + 1$
$C_\mathrm{a} \leftarrow k - 1$    // anti-shock at which $\mathcal{V}_0$ is connected
return $U' \leftarrow \mathbb{P}\left[A'_1 + \cdots + A'_{C_\mathrm{a}} > 1 \mid \pi'\right]$.
```

3.6 When the λ_j's are all equal

In the special case where all the shock rates λ_j are equal, faster PMC implementations are available. When generating the permutation π directly, at each step the next shock must be drawn uniformly among the shocks that did not occur yet. Thus, generating the sequence of shocks up to the critical one amounts to generating the first C_s elements of a random permutation of $\{1, \ldots, \kappa\}$ objects. This can be done very efficiently (Knuth, 1998, Section 3.4.2): Put the numbers $1, \ldots, \kappa$ in a table of size κ, then at each step k, draw J in $\{k, \ldots, \kappa\}$ at random, and exchange the table entries in positions J and k. The number at position k is the shock selected at step k. There is no need to update probabilities or to use rejection.

Moreover, when all λ_j are equal, say to λ, in the case of PMC with shocks, we have $\Lambda_j = \lambda \times (\kappa + 1 - j)$. In this case, one can rewrite (3) as

$$\begin{aligned}
\mathbb{P}\left[A_1 + \cdots + A_c \leq \gamma\right] &= \sum_{j=1}^{c} (1 - e^{-(\kappa+1-j)\lambda\gamma}) \prod_{k=1, k\neq j}^{c} \frac{\kappa + 1 - k}{j - k} \\
&= \kappa \binom{\kappa - 1}{c - 1} \int_{e^{-\lambda\gamma}}^{1} t^{\kappa - c} (1 - t)^{c-1} \mathrm{d}t \qquad (4) \\
&= 1 - I_{e^{-\lambda\gamma}}(\kappa - c + 1, c), \qquad (5)
\end{aligned}$$

where the equality in (4) is a direct consequence of Formula (6.6.4) in Abramowitz and Stegun (1970) and the last expression contains the cdf of the beta distribution (or regularized incomplete beta function), defined by

$$I_x(\alpha,\beta) = \frac{1}{B(\alpha,\beta)} \int_0^x t^{\alpha-1}(1-t)^{\beta-1}\mathrm{d}t$$

for $\alpha, \beta > 0$ and $x \in [0,1]$, where $B(\alpha,\beta)$ is the beta function. The equality (4) can also be verified directly by expanding the $(1-t)^{c-1}$ in the integrand using the Binomial Theorem to get a polynomial in powers of t, and integrating those powers of t from $e^{-\lambda}$ to 1 to obtain the expression on the previous line. To evaluate (5) accurately when $e^{-\lambda\gamma}$ is close to 1, one can use the identity

$$1 - I_{e^{-\lambda\gamma}}(\kappa - c + 1, c) = I_{1-e^{-\lambda\gamma}}(c, \kappa - c + 1).$$

For PMC with anti-shocks, still with all shock rates equal to λ, we can use a similar reformulation of (2) with λ replaced by $\mu = -\ln(1-e^{-\lambda})$. This gives

$$\mathbb{P}\left[A'_1 + \cdots + A'_c > \gamma\right] = \sum_{j=1}^{c} e^{-(\kappa+1-j)\mu\gamma} \prod_{k=1,k\neq j}^{c} \frac{\kappa+1-k}{j-k} = I_{e^{-\mu\gamma}}(\kappa - c + 1, c). \tag{6}$$

Formulas (5) and (6) do not apply for the turnip method defined in the next section, but only for PMC. Aside from being faster to compute, they have two additional advantages: (i) for PMC, where the direct formula (2) suffers from numerical instabilities, (5) can be used directly instead of the slow matrix exponential, and (ii) these formulas can compute cdf values much smaller than 10^{-16}, because they do not rely on the subtraction $F(x) = 1 - \bar{F}(x)$. They only require an accurate computation of the beta cdf.

Because of this important speedup, it could be worthwhile to construct the model in the first place under the constraint that all shock rates must be equal. To make some of the shocks much more probable than others, one can simply duplicate them. That is, we may have $n_{\mathbf{s}} \geq 0$ different shocks that affect the same subset $\mathbf{s}$, all with rate λ. This is equivalent to having $\lambda_{\mathbf{s}} = n_{\mathbf{s}}\lambda$. This puts restrictions on the shock rates, as they must now be all integer multiple of the same constant λ, but on the other hand, this may allow a much faster estimation of u.

4 Adapting the turnip method

The idea of the *turnip* method (Gertsbakh and Shpungin, 2010), adapted to our setting, is as follows: While we add the shocks [or anti-shocks] one by one in increasing order of their occurrence, we remove from consideration at each step all the shocks [or anti-shocks] that did not yet occur and can no longer contribute to system failure [or repair].

4.1 Turnip with shocks

When shocks are added, any shock that takes down only links that are already failed can be removed from consideration. Also, if a shock takes down some links that are not yet failed, but none of these links belongs to a path that connects two nodes of $\mathcal{V}_0$ in the current configuration of the network, then this shock can be removed from consideration. Removing these useless shocks may speed up things considerably in some situations, as was observed in Gertsbakh and Shpungin (2010) and Botev et al. (2013) for the case of independent links, but it also entails additional overhead for the maintenance of data structures and for the computations to identify the shocks that can be removed from further consideration. This overhead can be more important in the MO model with shocks than with independent links.

Note that when we add the shocks until the system fails, all nodes in $\mathcal{V}_0$ are connected, and therefore belong to the same connected component of the graph. Then, any link that connects two nodes that are not in this connected component can be removed from consideration, because it can no longer connect nodes in $\mathcal{V}_0$. We can actually put those links to the failed state right away, and keep only the links that connect nodes that are in the same connected component as $\mathcal{V}_0$. In this way, there will always be a single connected component under consideration, until the network fails. Whenever a shock takes down only links that are already failed, we remove it from consideration. This is our adaptation of the turnip using shocks, for this

ALGORITHM 5: An adapted turnip algorithm

```
$\Lambda_1 \leftarrow \lambda_1 + \cdots + \lambda_\kappa$
$\mathcal{L}_1 \leftarrow \{1, \ldots, \kappa\}$      // shocks still under consideration
$\mathbf{x} = (x_1, \ldots, x_m) \leftarrow (1, \ldots, 1)$      // all links are operational
$k \leftarrow 1$
while the nodes in $\mathcal{V}_0$ are all connected do
    draw $J$ from $\mathcal{L}_1$, with $\mathbb{P}[J = j] = \lambda_j / \Lambda_k$
    $\pi(k) \leftarrow J$
    for all $i \in \mathbf{s}(j)$ do
        $x_i \leftarrow 0$, remove link $i$, update the connected components if needed, and if the two nodes that were
        connected by $i$ are no longer in the same connected component, remove all the links in the component
        which is no longer connected with $\mathcal{V}_0$
    $k \leftarrow k + 1$
    $\Lambda_k \leftarrow \Lambda_{k-1} - \lambda_J$ and remove $J$ from $\mathcal{L}_1$
    for all shock $j \in \mathcal{L}_1$ that affects only failed links do
        $\Lambda_k \leftarrow \Lambda_k - \lambda_j$ and remove $j$ from $\mathcal{L}_1$
            // shock $j$ is discarded and $k$ is unchanged
            // this step distinguishes turnip and PMC
$C_{\rm s} \leftarrow k - 1$
return  $U \leftarrow \mathbb{P}[A_1 + \cdots + A_{C_{\rm s}} \leq 1 \mid \pi]$, an unbiased estimate of $u$ computed using $C_{\rm s}, \Lambda_1, \ldots, \Lambda_{C_{\rm s}}$.
```

MO case. It is given in Algorithm 5, in a version where the permutation is generated directly. One can also generate the shock times and sort them to generate the permutation.

Removing the useless shocks here can improve things in two ways: it can reduce the work if we save more by handling fewer shocks than the additional overhead, and it can reduce the variance of the estimator. Indeed, when shocks are removed from consideration at the last step of the while loop, k is not increased, so these discarded shocks are not considered when computing the estimator U. Therefore, the $C_{\rm s}$ returned by Algorithm 5 can be much smaller than the one returned by PMC, and the resulting estimator is not the same and can have much smaller variance. In our experiments, we often observed a variance reduction but an increase in work (because of the important overhead).

4.2 Turnip with anti-shocks

In the case where we assume that all the shocks have occurred and we add anti-shocks until the system is repaired, we can also remove the anti-shocks that can no longer contribute. This is done in Algorithm 6, which returns an estimator U' based on the $C_{\rm a}$ non-discarded anti-shocks. In this version, all the anti-shock times are first generated and sorted in increasing order. The ordered list of anti-shocks $\mathcal{L}_1$ can be maintained in an array and the removed entries just marked instead of physically removed. When scanning this ordered list at the end of the algorithm, we scan the unmarked entries in the array. Links are eliminated from the network when we find that they can no longer connect anything new, and future anti-shocks that can only take down failed links are discarded. The counter k counts the number of non-discarded anti-shocks that have occurred, while k' also includes those that were discarded. The estimator U' is based on the times R_j between the non-discarded anti-shocks.

4.3 Summary of PMC and turnip versions

We summarize the different versions of PMC and turnip that we have introduced. They distinguish themselves by four main binary decisions in the definition of the algorithm.

(a) We can generate all shock or anti-shock times, then sort them in increasing order to determine the permutation, or we can generate the permutation directly without generating the shock times.

(b) To determine the critical shock or anti-shock number $C_{\rm s}$ or $C_{\rm a}$, we can use a destruction process that adds the shocks one by one, or use a construction process that removes the shocks one by one (reverse process).

(c) We can compute the critical numbers by considering all shocks or anti-shocks (this is PMC) or eliminate along the way the shocks or anti-shocks that are found to be useless (this is the turnip).

(d) We can define the estimator as a conditional probability for a sum of C_s exponential times between shocks, or a sum of C_a exponential times between anti-shocks (we denote the latter by "anti").

Note that when using the reverse process in (b), we need to generate the entire permutation. Overall, there are 16 possible combinations. This gives 16 different variants of the algorithm. Algorithms 1 to 6 are only six examples out of those 16. Which combination is best is problem-dependent. For (a), in our experiments, generating the permutation directly was always a bit faster, but this may not always be true, in particular when the critical number is close to κ. For (b), the destruction process (adding shocks) is usually faster when $C_\mathrm{s} \ll C_\mathrm{a}$, and removing shocks is faster when $C_\mathrm{s} \gg C_\mathrm{a}$. For (c), we generally expect turnip to be faster, but it depends on the overhead required to maintain the appropriate data structures to identify the useless shocks or anti-shocks. What is best in (d) is similar to (b): roughly, adding the shocks is generally better (smaller variance) when $C_\mathrm{s} < C_\mathrm{a}$, and vice-versa. However, when u is very small, the formula based on the C_a anti-shocks times is sometimes the only one that is numerically stable and usable as an estimator. Our numerical examples will illustrate all of this.

ALGORITHM 6: A turnip algorithm with anti-shocks generated at exponential times

```
$\Lambda'_1 \leftarrow \mu_1 + \cdots + \mu_\kappa$
draw the $\kappa$ anti-shock times, with rates $\mu_j$, and sort them in increasing order
$\mathcal{L}_1 \leftarrow \{\pi(1), \ldots, \pi(\kappa)\}$   // list of anti-shocks in increasing order of occurrence, all unmarked
$\mathcal{F} \leftarrow \{1, \ldots, m\}$   // set of failed links
for $i = 1$ to $m$ do
    $d_i \leftarrow c_i$ and $x_i \leftarrow 0$   // current configuration of links
$k \leftarrow 1$ and $k' \leftarrow 1$
$\mathcal{F}_0 \leftarrow \emptyset$   // a list of links that can be discarded
while the nodes in $\mathcal{V}_0$ are not all connected do
    for each $i' \in \mathcal{F}_0$ do
        for each unmarked $j' \in \mathcal{L}_1$ that contain $i'$ do
            if shock $j'$ affects no link in $\mathcal{F}$ then
                    // shock $j'$ can be discarded
                mark $j'$ in $\mathcal{L}_1$
                $\Lambda'_k \leftarrow \Lambda'_k - \mu_{j'}$
    while $\pi(k')$ is marked do
        $k' \leftarrow k' + 1$
    $j \leftarrow \pi(k')$ and mark $j$
      // in what follows, we remove shock $j$
    $\Lambda'_{k+1} \leftarrow \Lambda'_k - \mu_j$
    $\mathcal{F}_0 \leftarrow \emptyset$   // a local list of links that become operational or can be discarded after anti-shock $j$
    for all $i \in \mathbf{s}(j)$ do
        $d_i \leftarrow d_i - 1$
        if $d_i = 0$ then
            $x_i \leftarrow 1$, remove $i$ from $\mathcal{F}$, and add $i$ to $\mathcal{F}_0$   // link $i$ gets repaired
            if link $i$ joints two connected components, merge them, remove from $\mathcal{F}$ all the links $i' \in \mathcal{F}$ that
            connect these two previous components, and add them to $\mathcal{F}_0$
    $k \leftarrow k + 1$ and $k' \leftarrow k' + 1$
$C_\mathrm{a} \leftarrow k - 1$   // critical anti-shock number
return $U' \leftarrow \mathbb{P}\left[A'_1 + \cdots + A'_{C_\mathrm{a}} > 1 \mid \pi\right]$.
```

5 Bounded relative error for PMC and turnip

We now derive conditions under which the PMC and turnip methods provide estimators having BRE when $u \to 0$. For this, we parameterize all the rates λ_j by a single rarity parameter $\epsilon > 0$, so that $\lambda_j = \lambda_j(\epsilon)$ is non-increasing in ϵ and the corresponding $u = u(\epsilon) \to 0$ when $\epsilon \to 0$, and we study the behavior when $\epsilon \to 0$. Recall that for two non-negative functions f and g, we say that $f(\epsilon) = \mathcal{O}(g(\epsilon))$ if there is a constant K such that $f(\epsilon) \le K g(\epsilon)$ for all $\epsilon > 0$, and $f(\epsilon) = \Theta(g(\epsilon))$ when both $f(\epsilon) = \mathcal{O}(g(\epsilon))$ and $g(\epsilon) = \mathcal{O}(f(\epsilon))$.

We start with the PCM method with shocks. The estimator

$$U = Z(\pi) = \mathbb{P}[A_1 + \cdots + A_{C_s} \le 1 \mid \pi] \tag{7}$$

(for a single realization) is a function of the (random) permutation π. If $p(\pi)$ denotes the probability of π, then

$$u = u(\epsilon) = \mathbb{E}[Z(\pi)] = \sum_{\pi} Z(\pi)p(\pi) \tag{8}$$

and

$$\mathrm{RE}^2[Z(\pi)] = \frac{\mathbb{E}[Z^2(\pi)] - u^2}{u^2} = \frac{1}{u^2}\sum_{\pi} Z^2(\pi)p(\pi) - 1. \tag{9}$$

Here, both $Z(\pi)$ and $p(\pi)$ are functions of ϵ, although we omit to write it explicitly to simplify the notation. Also, we only need to consider the partial permutations $\tilde{\pi} = (\pi(1), \ldots, \pi(C_s))$ rather than the full permutations $\pi = (\pi(1), \ldots, \pi(\kappa))$, in the sense that all the full permutations that yield the same $\tilde{\pi}$ can be considered as the same π in the sums (7), (9), and elsewhere, and these sums can be defined over $\tilde{\pi}$ instead of π. This means that in our development below, π could be replaced by $\tilde{\pi}$, and similarly with π' for the anti-shocks.

We have BRE if and only if $\mathrm{RE}^2[Z(\pi)] = \mathcal{O}(1)$, which occurs if and only if $Z^2(\pi)p(\pi)/u^2 = \mathcal{O}(1)$ for all π, because the total number of permutations π is finite. This holds for both PMC and turnip.

Theorem 5.1 *If $p(\pi) = \Theta(1)$ for all π, i.e., if each $p(\pi)$ remains bounded away from 0 when $\epsilon \to 0$, then the PMC and turnip estimators with shocks have BRE.*

Proof. For all π, we know that $Z(\pi)p(\pi) \le u$, so if $p(\pi) = \Theta(1)$, then $Z^2(\pi)p(\pi)/u^2 = \mathcal{O}(Z^2(\pi)p^2(\pi)/u^2) \le 1$. □

Corollary 5.2 *If $\lambda_j/\lambda_k = \Theta(1)$ for all $j \ne k$, then PMC and turnip with shocks give BRE.*

Proof. Under the given assumption, we have $\lambda_j/\Lambda_k = \Theta(1)$ for all j and k in Algorithm 2, which implies that $p(\pi) = \Theta(1)$ for all π. □

The conditions in Theorem 5.1 are only sufficient. We now give necessary and sufficient conditions in the setting where the λ_j's are polynomial functions of ϵ, say $\lambda_j = \Theta(\epsilon^{a_j})$ for some $a_j > 0$, for each link j. In that case, it is easily seen that for any given π, $p(\pi)$ and $Z(\pi)$ are also polynomial in ϵ, which means that for each π, there exist real numbers $m_1(\pi) \ge 0$ and $m_2(\pi) \ge 0$ such that $p(\pi) = \Theta(\epsilon^{m_1(\pi)})$ and $Z(\pi) = \Theta(\epsilon^{m_2(\pi)})$. There is also an $r \ge 0$ such that $u = \Theta(\epsilon^r)$ and $r = \min_\pi(m_1(\pi) + m_2(\pi))$. In this setting, $Z^2(\pi)p(\pi)/u^2 = \epsilon^{m_1(\pi)+2m_2(\pi)-2r} = \mathcal{O}(1)$ if and only if $m_1(\pi) + 2m_2(\pi) \ge 2r$. We have proved:

Theorem 5.3 *If for each π we have $p(\pi) = \Theta(\epsilon^{m_1(\pi)})$ and $Z(\pi) = \Theta(\epsilon^{m_2(\pi)})$, then the PMC and turnip estimators with shocks have BRE if and only if $m_1(\pi) + 2m_2(\pi) \ge 2r$ for all π.*

Note that $m_1(\pi) + m_2(\pi) \ge r$ always hold, so if $m_1(\pi) = 0$ then $m_2(\pi) \ge r$ and the condition of the theorem is satisfied. When this holds for all π, this is the situation of Theorem 5.1. If $m_1(\pi) + m_2(\pi) = r$, so the contribution of π to u does not vanish asymptotically when $\epsilon \to 0$, then $m_1(\pi) = 0$ (i.e., $p(\pi) = \Theta(1)$) is necessary for BRE to hold. The interpretation of what happens otherwise is that $p(\pi) \to 0$ and $Z(\pi)/u = \Theta(1/p(\pi)) \to \infty$ when $\epsilon \to 0$, so when ϵ is very small, $Z(\pi)$ has an enormous value but π is practically never sampled. We can have $m_1(\pi) > 0$ only when $m_1(\pi) + m_2(\pi) > r$. Note that the conditions in this theorem depend not only on the rates λ_j, but also on the topology of the network, via the $Z(\pi)$'s.

The conditions of Corollary 5.2 are equivalent to the BRE conditions established in Gertsbakh and Shpungin (2010) and Lomonosov and Shpungin (1999) for the case of independent links (exactly one shock per link). Note that these are only sufficient conditions. Below, we will give an example where the conditions are not satisfied and BRE does not hold. It corresponds to a situation where there is a permutation π that has a significant ($\Theta(u)$) contribution to u in (8), but this permutation becomes too rare in the sense that

$p(\pi) \to 0$ when $\epsilon \to 0$, so the contribution from this permutation is lost with very large probability when ϵ is too small.

For PMC or turnip with anti-shocks, we also have (8) and (9) with π' in place of π, and Theorem 5.1 becomes

Theorem 5.4 *If $p(\pi') = \Theta(1)$ for all π', then the PMC and turnip estimators with anti-shocks have BRE.*

We also have:

Corollary 5.5 *If $-\ln \lambda_j/(-\ln \lambda_k) = \Theta(1)$ for all $j \neq k$, then PMC and turnip with anti-shocks give BRE.*

Proof. Recall that $\mu_j = -\ln(1-e^{-\lambda_j}) = -\ln(\lambda_j - \mathcal{O}(\lambda_j^2))$. Then, the given assumption implies that $\mu_j/\Lambda'_k = \Theta(1)$ for all j and k in Section 3.5. As a result, for each permutation π', $p(\pi') = \prod_{k=1}^{\kappa} \mu_{\pi'(k)}/\Lambda'_k = \Theta(1)$ and Theorem 5.4 applies. □

The sufficient BRE condition in Corollary 5.5 is weaker than that in Corollary 5.2. Note that these conditions are not necessary, so we may have BRE even when the conditions fail. We now provide an example where BRE holds with the anti-shocks but not with the shocks, showing that BRE indeed holds more generally with the anti-shocks.

Example 5.6 We consider a graph with 4 nodes and 4 links, as shown in Figure 1, with shocks only on the links, one shock per link, and $\mathcal{V}_0 = \{1, 4\}$. We have $\lambda_j = \epsilon^2$ for $j = 1, 2$ and $\lambda_j = \epsilon$ for $j = 3, 4$, for some small $\epsilon > 0$. Recall that when λ_j is small, the probability that shock j occurs before time 1 is $1 - e^{-\lambda_j} \approx \lambda_j$. Thus, the probability that the graph is disconnected because only shock 1 occurs is approximately ϵ^2, the probability that it is disconnected because only shocks 3 and 4 occurs is also approximately ϵ^2, and all other possibilities have probability $\mathcal{O}(\epsilon^3)$. Therefore, $u \approx 2\epsilon^2$, and $u/(2\epsilon^2) \to 1$ when $\epsilon \to 0$.

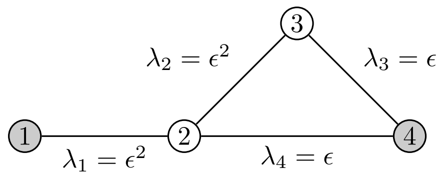

Figure 1: A small graph with 4 nodes and 4 links.

Here, $\lambda_3/\lambda_1 = 1/\epsilon \to \infty$ when $\epsilon \to 0$, so the conditions of Corollary 5.2 do not hold. However, $-\ln \lambda_3/(-\ln \lambda_1) = -\ln \epsilon/(-2 \ln \epsilon) = 1/2$ and this quantity is either 1/2 or 1 or 2 for the other pairs (i, j). Therefore, Corollary 5.5 applies.

Table 1: Simulation results with $n = 10^6$ for the small graph

	PMC with shocks		PMC with anti-shocks	
ϵ	$\bar{W}_n$	$\mathrm{RE}^2[\bar{W}_n]$	$\bar{W}_n$	$\mathrm{RE}^2[\bar{W}_n]$
10^{-1}	1.998e-2	3.87	1.978e-2	1.112
10^{-2}	2.007e-4	48.75	1.997e-4	1.640
10^{-4}	1.960e-8	4996.67	1.999e-8	1.708
10^{-6}	1.000e-12	1.000e-6	2.001e-12	1.706
10^{-8}	1.000e-16	0.000	1.998e-16	1.709
10^{-10}	1.000e-20	0.000	2.003e-20	1.705

Table 1 shows the estimated unreliability $\bar{W}_n$ and its estimated relative variance $\mathrm{RE}^2[\bar{W}_n] = S_n^2/(n\bar{W}_n^2)$, based on $n = 10^6$ independent replications, for PMC with shocks and PMC with anti-shocks, with some small values of ϵ. For PMC with anti-shocks, $\bar{W}_n$ is always close to $u \approx 2\epsilon^2$ and the RE remains stable when $\epsilon \to 0$, as expected. For PMC with shocks, when ϵ is small, the estimator $\bar{W}_n$ is erratic and eventually becomes close to $\epsilon^2 \approx u/2$ instead of u, while the estimated RE is 0 most of the time. What happens is the probability that shocks 3 and 4 are the first two shocks (in any order) in the permutation π converges to 1, and the probability that shock 1 comes before these two converges to 0. When shocks 3 and 4 come first, the probability of failure is $Z(\pi) \approx \epsilon^2$. And if this occurs in all realizations, then $U = \epsilon^2$ in all cases and the empirical variance is $S_n^2 = 0$. In other words, a graph failure coming from shock 1 is never observed, and this causes an apparent bias, because this part of the contribution to u is missing. In reality, there is no bias and the true RE is very large. In the rare cases where failure comes from shock 1 for one or more realizations, the empirical RE would be very large. We can see this for $\epsilon = 10^{-2}$ and 10^{-4}.

6 Adapting GS to the MO model

6.1 GS with shocks

We now adapt the GS algorithm proposed in Botev and Kroese (2012) and Botev et al. (2013) for independent links to the MO copula setting. This algorithm provides an unbiased estimator with low relative error for the unreliability u. It addresses the rare-event issue by forcing the sampling of more realizations of the vector $\mathbf{Y}$ of shock times in the region where the system is failed at time 1. To make our formulation compatible with the GS algorithm given in (Botev et al., 2013), where we want to select trajectories so that they go *above* given thresholds, we take $S(\mathbf{Y}) = 1/\tilde{S}(\mathbf{Y})$ as the *importance function* in the splitting method. When network failure is a rare event, most realizations of $S(\mathbf{Y})$ will be small (much closer to 0 than to 1) under the original sampling distribution, and we want to apply GS to get more realizations for which $S(\mathbf{Y}) > 1$. For this, we choose an integer $s \geq 2$ called the *splitting factor*, an integer $\tau > 0$, and thresholds $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_\tau = 1$, such that

$$\rho_t = \mathbb{P}[S(\mathbf{Y}) > \gamma_t \mid S(\mathbf{Y}) > \gamma_{t-1}] \approx 1/s$$

for $t = 1, \ldots, \tau$ (except for ρ_τ, which can be larger than $1/s$), just like in Botev et al. (2013), and we apply GS in the same way. Botev et al. (2013) recommend $s = 2$ and give an adaptive pilot algorithm to estimate good values of the splitting levels γ_t.

For each level γ_t, we construct a Markov chain $\{\mathbf{Y}_{t,\ell},\ \ell \geq 0\}$ with a stationary density equal to the density of $\mathbf{Y}$ conditional on $S(\mathbf{Y}) > \gamma_t$, given by

$$f_t(\mathbf{y}) \stackrel{\text{def}}{=} f(\mathbf{y}) \frac{\mathbb{I}[S(\mathbf{y}) > \gamma_t]}{\mathbb{P}[S(\mathbf{Y}) > \gamma_t]}, \tag{10}$$

where $f \equiv f_0$ is the unconditional density of $\mathbf{Y}$. The transition kernel density of this Markov chain, which is the density of the next state $\mathbf{Y}_{t,\ell}$ conditional on the current state $\mathbf{Y}_{t,\ell-1}$, is denoted by $\kappa_t(\cdot \mid \mathbf{Y}_{t,\ell-1})$. One possibility for the construction of κ_t is via Gibbs sampling, as explained later.

At the t-th stage, if a Markov chain starts from a state having density f_{t-1} and evolves according to the kernel $\kappa_{t-1}(\cdot \mid \mathbf{Y}_{t-1,j-1})$, then each visited state also has density f_{t-1}, which is a stationary density for the Markov chain with kernel κ_{t-1}. In particular, the chain will never again go below the level γ_{t-1} that we have already reached.

Algorithm 7 states this procedure with a single starting chain. It returns an unbiased estimator U of u. In the algorithm, $\mathcal{X}_t$ denotes a set of latent states $\mathbf{Y}$ that have reached the level γ_t. This algorithm will be invoked n times, independently, and the empirical mean $\bar{U}_n$ and variance S_n^2 of the n realizations $U_1, \ldots, U_n$ of U can be used to estimate the unreliability u and the variance of U. Proposition 1 of Botev et al. (2013) states that these are both unbiased estimators. They can be used to compute a confidence interval on u.

To sample $\mathbf{Y}_\ell$ from the density $\kappa_{t-1}(\cdot \mid \mathbf{Y}_{\ell-1})$ in Algorithm 7, we use Gibbs sampling as follows. We first select a permutation of the κ coordinate indexes $\{1, \ldots, \kappa\}$ (it can be just the identity permutation). Then we visit the κ coordinates in the order specified by the permutation. When we visit coordinate Y_j of the current $\mathbf{Y}$, for some j, we erase the current value of Y_j and we resample it from its distribution conditional

ALGORITHM 7: A GS algorithm based on shocks; returns U, an unbiased estimate of u

```
Require: $s, \tau, \gamma_1, \ldots, \gamma_\tau$
  Generate a vector $\mathbf{Y}$ of shock times from its unconditional density $f$.
  if $S(\mathbf{Y}) > \gamma_1$ then
      $\mathcal{X}_1 \leftarrow \{\mathbf{Y}\}$
  else
      return $U \leftarrow 0$
  for $t = 2$ to $\tau$ do
      $\mathcal{X}_t \leftarrow \emptyset$      // states that have reached level $\gamma_t$
      for all $\mathbf{Y}_0 \in \mathcal{X}_{t-1}$ do
          for $\ell = 1$ to $s$ do
              sample $\mathbf{Y}_\ell$ from the density $\kappa_{t-1}(\cdot \mid \mathbf{Y}_{\ell-1})$
              if $S(\mathbf{Y}_\ell) > \gamma_t$ [$\equiv \{\tilde{S}(\mathbf{Y}_\ell) < 1/\gamma_t\}$] then
                  add $\mathbf{Y}_\ell$ to $\mathcal{X}_t$
  return $U \leftarrow |\mathcal{X}_\tau|/s^{\tau-1}$, an unbiased estimate of $u$.
```

on $S(\mathbf{Y}) > \gamma_{t-1}$, given the other coordinates of $\mathbf{Y}$. Then the chain will never again go below the level γ_{t-1} that we have already reached. If $\mathbf{Y}$ has density f_{t-1} and we resample any of its coordinates as just described, the modified $\mathbf{Y}$ still has density f_{t-1}.

In fact, there are just two possibilities for the conditional distribution of Y_j when we resample it. Suppose we currently have $Y_j < 1/\gamma_{t-1}$. If by changing Y_j to a value larger than $1/\gamma_{t-1}$ we would have $S(\mathbf{Y}) \leq \gamma_{t-1}$ (that is, removing the shock j would make the system operational at time $1/\gamma_{j-1}$), then we must resample Y_j from its distribution conditional on $Y_i < 1/\gamma_{t-1}$. Otherwise, we resample it from its original distribution, because shock j alone has no influence on the failure time of the system, given the occurence times of the other shocks. This Gibbs sampler is stated in Algorithm 8. In this algorithm, $(Y_1, \ldots, Y_{j-1}, \infty, Y_{j+1}, \ldots, Y_\kappa)$ represents the current vector $\mathbf{Y}$ but where shock j never occurs. Thus, the condition $S(Y_1, \ldots, Y_{i-1}, \infty, Y_{i+1}, \ldots, Y_\kappa) \leq \gamma_{t-1}$ means that the graph becomes operational at level γ_{t-1} (or time $1/\gamma_{t-1}$) when shock j is removed.

ALGORITHM 8: Gibbs sampling for the transition density κ_{t-1}

```
Require: $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$ for which $S(\mathbf{Y}) > \gamma_{t-1}$ and a permutation $\pi$ of $\{1, \ldots, \kappa\}$.
  for $k = 1$ to $\kappa$ do
      $j \leftarrow \pi(k)$
      if $S(Y_1, \ldots, Y_{j-1}, \infty, Y_{j+1}, \ldots, Y_\kappa) \leq \gamma_{t-1}$ then
          resample $Y_j$ from its density truncated to $(0, 1/\gamma_{t-1})$
      else
          resample $Y_j$ from its original density
  return $\mathbf{Y}$ as the resampled vector.
```

If Y_j is exponential with rate λ_j, its distribution conditional on $Y_j < 1/\gamma$ can be generated by inversion by generating U uniformly over $(0,\, 1-\exp[-\lambda_j/\gamma])$ and returning $Y_j = -\ln(1-U)/\lambda_j$ (a truncated exponential). To see this, note that we want $-\ln(1-U)/\lambda_j < 1/\gamma$, i.e., $U < 1 - \exp[-\lambda_j/\gamma]$.

Checking the condition that $S(Y_1, \ldots, Y_{j-1}, \infty, Y_{j+1}, \ldots, Y_\kappa) \leq \gamma_{t-1}$ in the Gibbs sampling algorithm could be a bit tricky when a single component can be affected by more than one type of shock. To do that, one must first identify the set G of all components $i \in \mathbf{s}(j)$ that did not already fail due to another shock before time $1/\gamma_{t-1}$, and then check if the system is still failed at time $1/\gamma_{t-1}$ if all components in G are put in the operational state. To facilitate the identification of G, we maintain a table that gives the number d_i of shocks that affect component i and that have already occurred, for each i. When that number exceeds 1 for component i, we know that component i is still failed even if we remove a shock that affects it.

6.2 GS with anti-shocks

In the dual approach, we generate and maintain a vector of anti-shock times $\mathbf{R} = (R_1, \dots, R_\kappa)$ instead of a vector $\mathbf{Y}$ of shock times. The resulting algorithm is very similar to the GS algorithm in Botev et al. (2013) and in Algorithm 7. In this algorithm, $\mathbf{R}$ is the vector of anti-shock times and $S(\mathbf{R})$ is the time at which the system gets repaired. To resample from the conditional density $\kappa_{t-1}(\cdot \mid \mathbf{R}_{\ell-1})$, we use Gibbs sampling as follows. At each step, we resample one coordinate R_j, as in Algorithm 8. If the anti-shock j would repair the network in the current configuration, we resample R_j from its exponential density truncated to (γ_{t-1}, ∞), otherwise we resample it from its original exponential density. Thus, when resampling the time of an anti-shock j, we must first check what links this anti-shock would immediately repair, and see if adding those links would make $\mathcal{V}_0$ connected. The corresponding Gibbs sampler is stated in Algorithm 10 and the splitting procedure is in Algorithm 9, for a single starting chain. In our experiments, we did not find much difference in performance between these two versions of GS (shocks vs anti-shocks).

ALGORITHM 9: A GS algorithm based on anti-shocks; returns U, an unbiased estimate of u

```
Require: $s, \tau, \gamma_1, \dots, \gamma_\tau$
  Generate a vector $\mathbf{R}$ of anti-shock times from its unconditional density.
  if $S(\mathbf{R}) > \gamma_1$ then
      $\mathcal{X}_1 \leftarrow \{\mathbf{R}\}$
  else
      return $U \leftarrow 0$
  for $t = 2$ to $\tau$ do
      $\mathcal{X}_t \leftarrow \emptyset$      // states that have reached level $\gamma_t$
      for all $\mathbf{R}_0 \in \mathcal{X}_{t-1}$ do
          for $\ell = 1$ to $s$ do
              sample $\mathbf{R}_\ell$ from the density $\kappa_{t-1}(\cdot \mid \mathbf{R}_{\ell-1})$
              if $S(\mathbf{R}_\ell) > \gamma_t$ then
                  add $\mathbf{R}_\ell$ to $\mathcal{X}_t$
  return $U \leftarrow |\mathcal{X}_\tau|/s^{\tau-1}$, an unbiased estimate of $u$.
```

ALGORITHM 10: Gibbs sampling for anti-shocks density $\kappa_{t-1}(\cdot \mid \mathbf{R})$

```
Require: $\mathbf{R} = (R_1, \dots, R_\kappa)$ for which $S(\mathbf{R}) > \gamma_{t-1}$ and a permutation $\pi$ of $\{1, \dots, \kappa\}$.
  for $k = 1$ to $\kappa$ do
      $j \leftarrow \pi(k)$
      if $S(R_1, \dots, R_{j-1}, 0, R_{j+1}, \dots, R_\kappa) \le \gamma_{t-1}$ then
          resample $R_j$ from its density truncated to $(\gamma_{t-1}, \infty)$
      else
          resample $R_j$ from its original density
  return $\mathbf{R}$ as the resampled vector.
```

7 Data structures for an efficient implementation

For an efficient implementation of the PMC, turnip, and GS algorithms, we need a representation of the graph that permit us to identify rapidly which conditional distribution should be used to resample a shock or an anti-shock. The representation must also be quick and easy to update after a shock or anti-shock has been resampled. We have used a slight modification of the representation in Botev et al. (2013), which we summarize here (this description is largely borrowed from Botev et al. (2013)).

The graph is represented by a data structure that contains the set of nodes and the set of links, each one in an array. Each node has a list of adjacent links and corresponding neighbors. Each link $i = (k, \ell)$ connects a pair of nodes k and ℓ. The fixed (permanent) parameters of the links are memorized in this structure. We also need a table of size κ that lists the subsets $\mathbf{s}(j)$ for $j = 1, \dots, \kappa$ and the means $1/\lambda_j$ for the exponential distributions for the associated failure rates. These structures are stored in single objects that do not change during the execution of the algorithm.

Some of the graph characteristics change during execution and differ across the different instances of the Markov chain. For those, we must keep one copy for each instance of the chain. This is the case for $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$. From this $\mathbf{Y}$, we easily obtain $\mathbf{X}(\gamma)$ and we can compute $\Phi(\mathbf{X}(\gamma))$ for any level γ. We also want a data structure that permits us to see immediately from what conditional distribution each Y_j should be resampled in the conditional Gibbs sampling at the current level γ, that tells us immediately the value of $\mathbf{X}(\gamma)$, and that can be updated quickly after Y_j is changed. The information in this data structure represents the state of the Markov chain in the splitting algorithm. It changes at each step of the chain and must be cloned each time we make a new copy of the chain. Therefore, it must be kept small. In our implementation, we maintain a set of connected components at each level γ_{t-1} as described in Botev et al. (2013). We also maintain a table that gives the number d_i of shocks j that have already occurred in the current configuration $\mathbf{Y}$ and that affect link i, for each i.

When a given Y_j is modified, if $1/Y_j$ was larger than the current level γ_{t-1} and it becomes smaller, then for each $i \in \mathbf{s}(j)$, the counter d_i is decreased by 1, and if $d_i = 0$, link i is added to the configuration (becomes operational). In the latter case, if this link connects two nodes k and ℓ that are not in the same connected component, these two components are merged into a single one. This is straightforward. If $1/Y_j$ was smaller than the current level γ_{t-1} and it becomes larger, then for each $i \in \mathbf{s}(j)$, the counter d_i is increased by 1, and if $d_i = 1$ (the link was operational before adding this shock), link i is removed from the configuration (becomes failed). In the latter case, if $i = (k, \ell)$, then we must check if there is still a path between k and ℓ after removing link i; if not, then the component (or tree) that contains these nodes must be split in two. This verification is on average the most time-consuming task. It is done as explained in Botev et al. (2013).

When the level is increased from γ_{t-1} to γ_t, we take all the shocks j for which $\gamma_{t-1} < 1/Y_j \le \gamma_t$, and for each of them, for each $i \in \mathbf{s}(j)$, we decrease d_i by 1, and if $d_i = 0$, we add link i to the configuration and if this link connects two nodes that are not in the same connected component, we merge these two components into a single one.

In the GS algorithm, when simulating at level γ_{t-1}, after each step of the Markov chain we must check if $\Phi(\mathbf{X}(\gamma_t)) = 0$, i.e., if we have reached the next level γ_t. If we did, we make a copy of the chain and insert it in $\mathcal{X}_t$, to be used as one of the starting points at the next stage. To compute $\Phi(\mathbf{X}(\gamma_t))$, we temporarily add the links i that become operational when we remove the shocks j for which $\gamma_{t-1} < 1/Y_j \le \gamma_t$, and check if this connects the nodes in $\mathcal{V}_0$.

8 Numerical experiments

In this section, we compare the performance of the various algorithms introduced earlier, on some examples. In our examples, we consider one shock for each link and one shock for each node, with the exception of the source and target nodes on which there are no shocks. A shock on a node takes down all the links connected to that node. We assume that all these shocks have the same rate $\lambda_j = \lambda$. Then, all anti-shocks also have the same $\mu_j = \mu = -\ln(1 - e^{-\lambda})$. Since λ will be small, μ will be large. For each example, we tried various values of λ.

For the PMC and turnip, unless indicated otherwise, we generated all the shock times and sorted them to find the permutation. For the basic PMC and turnip, we also report results when we generate π directly, and compare. We find very little difference in computing time. In the tables, we append π to the algorithm name when π is generated directly without generating the times, we append "rev" (for "reverse") when the critical number is computed by removing the shocks, and we append "anti" when the estimator U' based on anti-shocks is used. The estimator was always computed using the faster $\mathcal{O}(c^2)$ formula (2) or (3) when it was reasonably accurate; otherwise we switched to the slower but more accurate method of Higham (2009) mentioned in Section 3.2. For PMC, we also used the formula based on the beta cdf, that holds for equal λ_j's, for comparison.

For GS, the splitting factor is always $s = 2$, and the levels and their number τ are estimated by the adaptive Algorithm 3 of Botev et al. (2013), with $n_0 = 10^4$.

The sample size n varies from $n = 10^4$ to $n = 10^6$, depending on the example. In the tables, we report the unreliability estimate $\bar{W}_n$, the empirical relative variance of W, $S_n^2/(\bar{W}_n)^2$, the *relative error* of $\bar{W}_n$, defined as $\mathrm{RE}[\bar{W}_n] = S_n/(\sqrt{n}\bar{W}_n)$, the average critical shock or anti-shock number $\bar{C}$, the CPU time T (in seconds) required by the n runs of the algorithm, and the *work-normalized relative variance* (WNRV) of $\bar{W}_n$, defined as $\mathrm{WNRV}[\bar{W}_n] = T \times \mathrm{RE}^2[\bar{W}_n]$. This WNRV is approximately independent of n when n is large. It compares unbiased estimators by taking into account both the variance and the computing time. One should keep in mind that T (and therefore the WNRV) may depend significantly on the computing platform and on the implementation. For PMC and turnip, it also depends very much on the formula used to compute the conditional probability. We put a $\bullet$ next to the CPU time when it is computed using the matrix exponential (this is slow), a $\circ$ when one of the formulas (2) or (3) was used (this is faster), and nothing when using one of the formulas (5) or (6) for the beta distribution available for PMC when the λ_j are all equal (the fastest method).

Example 8.1 We start our numerical illustrations with a dodecahedron graph with 20 nodes and 30 links, often used as a benchmark in network reliability estimation (Botev et al., 2013; Cancela and El Khadiri, 1995; Cancela et al., 2009; Cancela et al., 2009; Tuffin et al., 2014), and shown in Figure 2 (taken from Botev et al. (2013)). We have 48 different shocks in total. The network is operational when nodes 1 and 20 are connected.

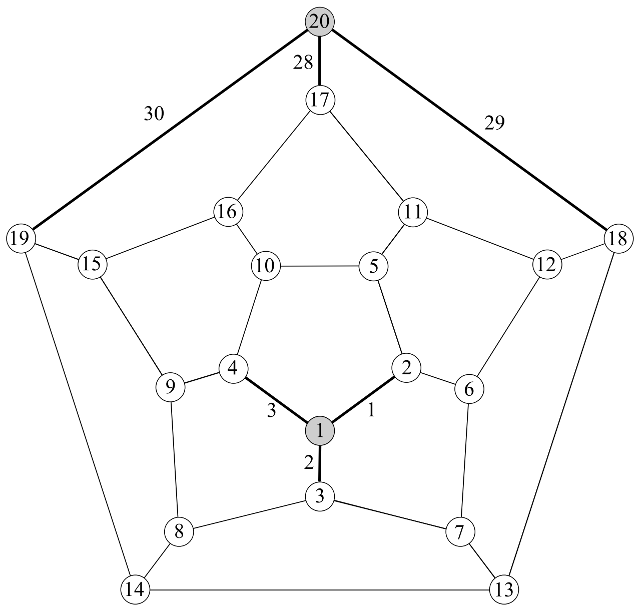

Figure 2: A dodecahedron graph with 20 nodes and 30 links.

Table 2 reports simulation results with $n = 10^6$, for $\lambda = 10^{-3}$ and 10^{-7}. We see that for the PMC and turnip methods, the RE is about the same for all variants and all values of λ, except for the turnip with anti-shocks, for which the RE is approximately halved. This agrees with the BRE property. In fact, the RE and WNRV would remain approximately the same for any smaller λ. For GS, the RE increases slightly when λ decreases, and the WNRV increases even more, because the computing time T increases significantly and is approximately proportional to the number of levels. (The number of levels for GS is around $-\log_2(u) \approx 67$ for $\lambda = 10^{-20}$). In terms of WNRV, turnip with anti-shocks wins (followed closely by PMC-rev and PMC-anti) in all cases, even though it has a larger average critical number. Its advantage over GS increases as λ decreases. GS has a smaller RE than PMC-turnip, but is much slower, and its WNRV is larger as a result. For $\lambda = 10^{-7}$, the turnip estimators can be computed only for the anti-shock versions, because of subtractive cancellation when computing estimators smaller than 10^{-16} from $F(t) = 1 - \bar{F}(t)$. The matrix exponential method of Higham (2009) computes $\bar{F}(t)$ and is not designed to compute $F(t)$ directly when it is very small.

Table 2: Dodecahedron with $n = 10^6$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE[$\bar{W}_n$]	$\bar{C}$	T(sec)	WNRV
			$\lambda = 10^{-3}$			
PMC	1.62e-8	993	0.032	12.7	35	0.035
PMC	1.62e-8	993	0.032	12.7	• 62	0.062
PMC-π	1.59e-8	1009	0.032	12.7	21	0.022
PMC-rev	1.62e-8	993	0.032	12.7	17	0.017
PMC-rev	1.62e-8	993	0.032	12.7	• 42	0.042
PMC-anti	1.60e-8	1004	0.032	36.3	17	0.018
PMC-anti	1.60e-8	1004	0.032	36.3	◦ 29	0.029
turnip	1.63e-8	894	0.030	10.7	• 72	0.064
turnip-π	1.59e-8	920	0.030	10.7	• 64	0.059
turnip-anti	1.58e-8	296	0.017	35.8	◦ 45	0.013
GS	1.59e-8	53	0.007		437	0.023
GS-anti	1.60e-8	56	0.007		425	0.024
			$\lambda = 10^{-7}$			
PMC	1.65e-20	1047	0.032	12.7	32	0.034
PMC-π	1.59e-20	1090	0.033	12.7	21	0.023
PMC-rev	1.65e-20	1047	0.032	12.7	17	0.018
PMC-anti	1.66e-20	1044	0.032	36.3	18	0.019
PMC-anti	1.66e-20	1044	0.032	36.3	◦ 29	0.030
turnip-anti	1.58e-20	311	0.018	35.8	◦ 44	0.014
GS	1.59e-20	143	0.012		982	0.140
GS-anti	1.58e-20	124	0.011		1106	0.137

We performed further experiments in which the shocks on nodes had rates 10 times larger, or 10 times smaller, than those on links, and the results were qualitatively very similar. We observed that the RE increases when the shocks on links have larger rates.

We also experimented with the case where each node has a shock, including those of $\mathcal{V}_0$. In that case, when λ is very small, a single shock on one of the two nodes of $\mathcal{V}_0$ is sufficient to take the graph down. The probability that any given shock occurs before time 1 is $1 - e^{-\lambda} \approx \lambda$, while the probability that two given shocks both occur is approximately λ^2, which is negligible with respect to λ. Therefore, in this case, failure of the graph is almost always caused by one of these two shocks on the nodes of $\mathcal{V}_0$, and the unreliability is $u \approx 2\lambda$. Apart from these two shocks, the other shocks have practically no impact. We observed this very clearly in our experiments (for all examples given in this paper) and this is the reason why we removed the shocks on the nodes of $\mathcal{V}_0$.

Example 8.2 Following L'Ecuyer et al. (2011), we construct a larger graph by putting three copies of the dodecahedron in parallel, merging the three copies of node 1 as a single node, and the three copies of node 20 as a single node. The resulting graph has 56 nodes and 90 links. Again, we take $\mathcal{V}_0 = \{1, 20\}$. We have a shock on each link and on each node not in $\mathcal{V}_0$, for a total of 144 shocks, all with rate λ. Table 3 gives simulation results for $n = 10^6$, for $\lambda = 0.1$ and $\lambda = 0.001$. GS performs well in both cases. The variance of the PMC and turnip estimators is about 100 times larger than that of GS for $\lambda = 0.1$, For $\lambda = 0.001$, the ratio is much larger, although the empirical values given in the table are very far from the exact ones: $\bar{W}_n$ underestimates u by several orders of magnitude, and S_n^2 certainly underestimates the true variance in the same way, so the estimated RE and WNRV reported in the table are meaningless. This may seem to contradict the BRE property that we have proved! The explanation is that in the case of this larger graph, when λ is small, only a tiny fraction of the 144! permutations π have a significant contribution in the sum (8), and the PMC and turnip rarely generate these important permutations. This is a rare-event situation. Moreover, this phenomenon is amplified when λ decreases, because the relative weight of the very rare most important permutations increases. Recall that we have proved BRE for an asymptotic regime where $\lambda \to 0$ while the graph topology is fixed. When λ is fixed, the BRE might actually increase very fast as a function of the size of the graph, or the number of shocks. This is what happens here.

Table 3: Three dodecahedrons in parallel, $n = 10^6$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	$\bar{C}$	T(sec)	WNRV
			$\lambda = 0.1$			
pmc	1.79e-5	3157	0.056	50	207	0.66
pmc-π	1.73e-5	2912	0.054	50	167	0.49
pmc-rev	1.79e-5	3157	0.056	50	53	0.17
pmc-anti	1.72e-5	2410	0.049	95	52	0.13
turn	1.77e-5	2572	0.051	38	● 771	1.98
turn-π	1.77e-5	2320	0.048	38	● 734	1.70
turn-anti	1.73e-5	1473	0.038	94	○ 215	0.32
GS	1.79e-5	31	0.0056		1094	0.034
GS-anti	1.78e-5	30	0.0055		1141	0.034
			$\lambda = 0.001$			
pmc	1.84e-35	2.5e5	0.50	50	210	51
pmc-π	1.38e-35	3.3e5	0.57	50	175	57
pmc-rev	1.84e-35	2.5e5	0.50	50	51	13
pmc-anti	6.10e-34	9.8e5	0.99	95	52	52
turn-anti	1.20e-29	5.7e5	0.75	94	○ 216	12
GS	4.13e-24	158	0.013		4366	0.70
GS-anti	4.06e-24	197	0.014		3552	0.70

Example 8.3 Our next set of examples is with a square lattice graph, as shown in Figure 3. Each node is connected to its neighbors on the left, right, up and down, when they exist. The set $\mathcal{V}_0$ contains two opposite corners. We report experiments with 20×20 and 40×40 lattices. The first has 400 nodes, 760 links, and 1158 different shocks. The second has 1600 nodes, 3120 links, and 4718 different shocks. With these large numbers of shocks, the number of permutation π is astronomical, so the PMC and turnip methods might not be able to generate permutations π that contribute significantly to u in (8). This would translate into gross under-estimations of u. The results, in Tables 4 and 5, confirm this.

For the 20×20 graph, GS works nicely and is clearly the most efficient method. The shock and anti-shock versions are equally good. It still works well for the 40×40 graph, except that for $\lambda = 10^{-10}$, the estimates returned by GS and GS-anti differ by more than two (empirical) standard deviations. We re-ran this case independently with $n = 10^5$ and obtained $\bar{W}_n \approx 7.7 \times 10^{-20}$ and RE$[\bar{W}_n] \approx 0.036$ for both GS and GS-anti. The PMC and turnip methods are much more noisy than GS. In the PMC case, where we can use the beta cdf to compute the conditional expectation because the λ_j's are all equal, the two versions that use the construction process (removes the shocks) are much faster than all other methods. However, their variance is much larger than that of GS. Their RE is up to 100% for the 20×20 graph. For the 40×40 graph, the RE is certainly much larger, but the PMC and turnip methods are so noisy that the results (including the RE estimators) are meaningless. These methods underestimate u by huge factors. They miss the important permutations among the 4718! different permutations of the shocks. For the PMC and turnip methods that do not use the beta cdf, most of the CPU time is to compute the conditional expectation.

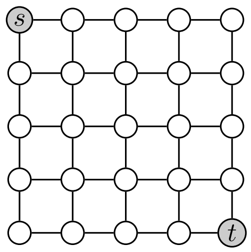

Figure 3: A 5×5 lattice graph.

Table 4: 20×20 lattice graph, $n = 10^5$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	$\bar{C}$	T(sec)	WNRV
			$\lambda = 10^{-5}$			
PMC	6.67e-10	9.9e4	1.0	202	1062	1050
PMC-π	1.34e-9	4.9e4	0.70	202	928	457
PMC-rev	6.67e-10	9.9e4	1.0	202	60	60
PMC-anti	6.73e-10	9.8e4	0.99	957	60	58
turnip	6.67e-10	9.9e4	1.0	176	• 4380	4350
turnip-π	1.34e-9	4.9e4	0.70	176	• 3868	1900
turnip-anti	9.61e-10	9.3e3	0.30	905	○ 1928	179
GS	8.46e-10	62	0.025		3655	2.3
GS-anti	7.97e-10	61	0.025		3730	2.3
			$\lambda = 10^{-10}$			
PMC	1.34e-19	5.0e4	0.71	202	1018	509
PMC-π	2.68e-19	2.5e4	0.50	202	977	244
PMC-rev	1.34e-19	5.0e4	0.71	202	60	29
PMC-anti	2.98e-34	2.5e4	0.50	957	60	15
turnip-anti	3.01e-20	3.0e4	0.55	905	○ 1694	514
GS	8.24e-20	121	0.035		4899	5.9
GS-anti	8.00e-20	114	0.034		4974	5.7

Table 5: 40×40 lattice graph, $n = 10^4$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	$\bar{C}$	T(sec)	WNRV
			$\lambda = 10^{-5}$			
PMC	6.1e-27	1.0e4	1	818	2234	2230
PMC-rev	6.1e-27	1.0e4	1	818	42	42
PMC-anti	3.4e-74	1.0e4	1	3907	43	43
turnip-anti	5.2e-35	9988	1	3680	○ 3946	3946
GS	7.98e-10	57	0.076		6183	35
GS-anti	7.88e-10	69	0.083		5980	41
			$\lambda = 10^{-10}$			
PMC	2.0e-134	1.0e4	1	812	2199	2200
PMC-rev	2.0e-134	1.0e4	1	812	48	48
PMC-anti	3.1e-104	1.0e4	1	3906	55	55
turnip-anti	1.9e-33	1.0e4	1	3679	○ 3531	3531
GS	5.0e-20	151	0.12		6034	91
GS-anti	8.9e-20	124	0.11		6688	83

Example 8.4 We now consider a complete graph with n_0 nodes, with one link for each pair of nodes, and $\mathcal{V}_0 = \{1, n_0\}$. We take $n_0 = 30$, which gives 435 links and 463 shocks, and $n_0 = 100$, which gives 4950 links and 5048 shocks. The results for $n_0 = 30$, $n = 10^5$, $\lambda = 0.5$ and 0.1, are given in Table 6. We see that GS is much more efficient than all PMC and turnip methods, which have much larger RE and give estimates that vary by up to a factor of 3 for $\lambda = 0.5$, and are off by huge factors for $\lambda = 0.1$, with $n = 10^5$. We redid some experiments with $n = 10^7$: the variance S_n^2 was similar, the REs were still quite large, and the estimates were still completely wild. The PMC methods with anti-shocks are much faster than all other methods because they use the beta cdf to compute the conditional expectation. The results for $n_0 = 100$, with $n = 10^4$ and $\lambda = 0.5$, are in Table 7. Only GS gives meaningful estimates in that case.

Example 8.5 Here we consider a case where the critical shock number C_s is much smaller than the critical anti-shock number C_a, to illustrate a situation where using the destruction process with shocks is much more efficient than using anti-shocks. We take a graph with 202 nodes and 202 links, with $\mathcal{V}_0 = \{1, 202\}$. There are two separate strings of 100 nodes and 101 links each that connect these two nodes (two series system in parallel). There is one shock per link (only), at rate $\lambda = 10^{-4}$. The averages of C_s and C_a are approximately

Table 6: Complete graph with 30 nodes, $n = 10^5$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	$\bar{C}$	T(sec)	WNRV
			$\lambda = 0.5$			
PMC	6.30e-6	8.0e4	0.897	371	140	113
PMC-π	6.77e-7	5.8e4	0.759	371	127	73
PMC-rev	6.30e-6	8.0e4	0.897	371	15	12
PMC-anti	8.17e-7	8.0e4	0.896	93	15	12
turnip	1.78e-6	2.0e4	0.45	175	• 3417	691
turnip-π	3.07e-6	1.7e4	0.408	175	• 3404	568
turnip-anti	7.38e-7	8.1e4	0.898	93	• 710	572
GS	2.09e-6	36	0.0189		460	0.16
GS-anti	2.04e-6	36	0.0191		427	0.16
			$\lambda = 0.1$			
PMC	6.62e-88	1.0e5	1	371	142	142
PMC-rev	6.62e-88	1.0e5	1	371	15	15
PMC-anti	4.90e-87	1.0e5	1	94	16	16
turnip-anti	2.82e-83	9.5e4	0.975	93	○ 40	38
GS	3.33e-22	132	0.036		2058	2.7
GS-anti	3.16e-22	152	0.039		1580	2.4

Table 7: Complete graph with 100 nodes, $n = 10^4$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	T(sec)	WNRV
		$\lambda = 0.5$			
GS	2.45e-20	109	0.11	3859	42
GS-anti	2.49e-20	128	0.11	4004	51

3 and 200. Here all the PMC and turnip variants give approximately the same RE, which is about 10^{-3} for $n = 10^6$, but the anti-shock versions are much slower. The turnip with anti-shock is 20 times slower than the turnip with shocks, even though the former uses the fast formula to compute the conditional probability whereas the shock versions use the matrix method of Higham (2009) (because the faster formula is unstable): computing the exponential of a 3×3 matrix is sufficiently fast and (much more importantly) $C_{\mathrm{s}} \ll C_{\mathrm{a}}$. Another interesting observation here is that the PMC-π is three times faster than the PMC for which we generate and sort the shock times. Since the average critical shock is 3, generating and sorting 202 shock times wastes a lot of CPU time.

9 Summary and conclusion

We introduced a static network reliability model with dependent link failures, based on a Marshall-Olkin copula, and proposed several adapted versions of the PMC, turnip, and GS methods to estimate accurately the unreliability u under this model when u is very small. Some of those algorithms add shocks, others remove them by adding anti-shocks, one by one. We proved that the PMC and turnip give estimators with BRE, under certain conditions on the shock rates, and these conditions are weaker with the anti-shocks than with the shocks. We showed that when all shocks have the same rates, the PMC estimators can be computed very quickly and accurately using the beta cdf. This suggests and motivates the construction of models in which all rates are equal, with the possibility of having more than one shock on certain subsets of components. In a numerical example of moderate size (a dodecahedron graph with different 48 shocks), the BRE property was observed very clearly with both shocks and anti-shocks. One can then easily estimate an arbitrarily small unreliability. In examples with larger graphs, with thousands of links and thousands of different shocks, the PMC and turnip eventually fail to provide meaningful estimates, because the relevant permutations become too rare. GS, on the other hand, remains viable for these large graphs, even when u is extremely small. The two versions of GS (with shocks and anti-shocks) perform equally well in the examples we tried. Our development generalizes easily to arbitrary multicomponent systems with binary states for the components and monotone increasing structure function. However, certain details in the implementation would have to be adapted to the problem at hand, in particular the data structures and methods to store and update the state of the system, to determine the time at which the system fails or gets repaired, and the range in which the new shock times should be resampled in GS. This is matter for further research.

References

Abramowitz, M. and Stegun, I.A. 1970. Handbook of Mathematical Functions. Dover, New York.

Alexopoulos, C. and Shultes, B.C. 2001. Estimating reliability measures for highly-dependable Markov systems, using balanced likelihood ratios. IEEE Transactions on reliability 50, 3, 265–280.

Asmussen, S. and Glynn, P.W. 2007. Stochastic Simulation. Springer-Verlag, New York.

Barlow, R. and Proschan, F. 1975. Statistical Theory of Reliability and Life Testing. Holt, Rinehart and Wilson, New York.

Botev, Z.I. and Kroese, D.P. 2012. Efficient Monte Carlo simulation via the generalized splitting method. Statistics and Computing 22, 1, 1–16.

Botev, Z.I., L'Ecuyer, P., Rubino, G., Simard, R., and Tuffin, B. 2013. Static network reliability estimation via generalized splitting. INFORMS Journal on Computing 25, 1, 56–71.

Botev, Z.I., L'Ecuyer, P., and Tuffin, B. 2012. Dependent failures in highly-reliable static networks. In Proceedings of the 2012 Winter Simulation Conference. IEEE Press, 430–441.

Botev, Z. I., L'Ecuyer, P., and Tuffin, B. 2013. Markov chain importance sampling with application to rare event probability estimation. Statistics and Computing 25, 1, 56–71.

Botev, Z.I., L'Ecuyer, P., and Tuffin, B. 2014. Modeling and estimating small unreliabilities for static networks with dependent components. In Proceedings of SNA& MC 2013: Supercomputing in Nuclear Applications and Monte Carlo. Paper A247. to appear.

Cancela, H. and El Khadiri, M. 1995. A recursive variance-reduction algorithm for estimating communication-network reliability. IEEE Transactions on Reliability 44, 4, 595–602.

Cancela, H. and El Khadiri, M. 2003. On the RVR simulation algorithm for network reliability evaluation. IEEE Transactions on Reliability 52, 2, 207–212.

Cancela, H., El Khadiri, M., and Rubino, G. 2009. Rare event analysis by Monte Carlo techniques in static models. In Rare Event Simulation Using Monte Carlo Methods, G. Rubino and B. Tuffin, Eds. Wiley, 145–170. Chapter 7.

Cancela, H., El Khadiri, M., Rubino, G., and Tuffin, B. 2014. Balanced and approximate zero-variance recursive estimators for the network reliability problem. ACM Transactions on Modeling and Computer Simulation *25,* 1, Article 5.

Cancela, H., L'Ecuyer, P., Lee, M., Rubino, G., and Tuffin, B. 2009. Analysis and improvements of path-based methods for Monte Carlo reliability evaluation of static models. In Simulation Methods for Reliability and Availability of Complex Systems, J. Faulin, A. A. Juan, S. Martorell, and E. Ramirez-Marquez, Eds. Springer Verlag, 65–84.

Colbourn, C.J. 1987. The Combinatorics of Network Reliability. Oxford University Press, New York.

Elperin, T., Gertsbakh, I.B., and Lomonosov, M. 1991. Estimation of network reliability using graph evolution models. IEEE Transactions on Reliability 40, 5, 572–581.

Gertsbakh, I.B. and Shpungin, Y. 2010. Models of Network Reliability. CRC Press, Boca Raton, FL.

Higham, N.J. 2009. The scaling and squaring method for the matrix exponential revisited. SIAM Review 51, 4, 747–764.

Iyer, S.M., Nakayama, M. K., and Gerbessiotis, A.V. 2009. A Markovian dependability model with cascading failures. IEEE Transactions on Computers 58, 9, 1238–1249.

Kalyoncu, H. and Sankur, B. 1992. Estimation of survivability of communication networks. Electronic Letters 28, 19, 1790–1791.

Knuth, D.E. 1998. The Art of Computer Programming, Volume 2: Seminumerical Algorithms Third Ed. Addison-Wesley, Reading, MA.

L'Ecuyer, P., Rubino, G., Saggadi, S., and Tuffin, B. 2011. Approximate zero-variance importance sampling for static network reliability estimation. IEEE Transactions on Reliability 8, 4, 590–604.

Lomonosov, M. and Shpungin, Y. 1999. Combinatorics and reliability Monte Carlo. Random Structures and Algorithms 14, 4, 329–343.

Marshall, A. and Olkin, I. 1967. A multivariate exponential distribution. Journal of the American Statistical Association 62, 30–44.

Natvig, B. 2011. Multistate Systems Reliability Theory with Applications. Wiley, Chichester, UK.

Nelsen, R.B. 2006. An Introduction to Copulas Second Ed. Springer Series in Statistics. Springer, New York, NY.

Ross, S.M. 2007. Introduction to Probability Models ninth Ed. Academic Press.

Rubino, G. and Tuffin, B., Eds. 2009. Rare Event Simulation using Monte Carlo Methods. Wiley.

Sahinoglu, M. and Rice, B. 2010. Network reliability evaluation. WIREs Computational Statistics 2, 189–211.

Sun, Y., Mendoza-Arriaga, R., and Linestsky, V. 2011. Marshall-Olkin multivariate exponential distributions, multivariate Lévy subordinators, efficient simulation, and applications to credit risk. Available at http://ssrn.com/abstract=1702087.

Tuffin, B., Saggadi, S., and L'Ecuyer, P. 2014. An adaptive zero-variance importance sampling approximation for static network dependability evaluation. Computers and Operations Research 45, 51–59.