Static Network Reliability Estimation Via Generalized Splitting

Zdravko I. Botev, Pierre L'Ecuyer
DIRO, Université de Montreal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7, CANADA,
{botev@iro.umontreal.ca, lecuyer@iro.umontreal.ca}

Gerardo Rubino
INRIA Rennes Bretagne Atlantique, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, FRANCE,
gerardo.rubino@inria.fr

Richard Simard
DIRO, Université de Montreal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7, CANADA,
simardr@iro.umontreal.ca

Bruno Tuffin
INRIA Rennes Bretagne Atlantique, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, FRANCE,
bruno.tuffin@inria.fr

We propose a novel simulation-based method that exploits a generalized splitting (GS) algorithm to estimate the reliability of a graph (or network), defined here as the probability that a given set of nodes are connected, when each link of the graph is failed with a given (small) probability. For large graphs, in general, computing the exact reliability is an intractable problem and estimating it by standard Monte Carlo methods poses serious difficulties, because the *unreliability* (one minus the reliability) is often a rare-event probability. We show that the proposed GS algorithm can accurately estimate extremely small unreliabilities and we exhibit large examples where it performs much better than existing approaches. It is also flexible enough to dispense with the frequently made assumption of independent edge failures.

Key words: Network Reliability, Generalized Splitting, Graph Connectivity, Gibbs sampling, Permutation Monte Carlo, Rare-Event Simulation, Conditional Monte Carlo.
History: Submitted October 24, 2011

---

1. Introduction

We consider the problem of estimating the *static reliability* of a graph (or network), defined as the probability that a given set of nodes are connected by operating links, where each link of the graph is in the operational state with a given probability, called the reliability of the link. Equivalently, we estimate the *unreliability*, defined as the complementary probability.

This problem is encountered in a wide range of applications such as telecommunications, transportation, energy supply systems, and social networking, among others (Cancela et al., 2009a; Gertsbakh and Shpungin, 2010).

If the graph has m links, then the configuration of all links (operating or failed) can be given by an m-dimensional binary vector with 2^m possible values, and the reliability can be computed (in principle) by computing and adding the probabilities of all configurations in which the graph is operational (that is, the selected nodes are connected). But as m increases, running through the 2^m configurations quickly becomes impractical. In fact, the exact calculation of network reliability (or even just counting the number of operational configurations) is a #P-complete problem (Ball and Provan, 1982; Colbourn, 1987). Other exact methods, for example based on cuts, paths, and various reductions, and the methods that approximate and bound the network reliability (Barlow and Marshall, 1964; Barlow and Proschan, 1975; Burtin and Pittel, 1972; Esary et al., 1967) are impractical in general for large graphs and their applicability depends heavily on the topology of the network.

All effective algorithms for estimating the reliability of large networks are based on some form of Monte Carlo sampling combined with variance reduction ideas. The standard (or crude) Monte Carlo method consists of simulating n independent copies of the binary vector that represents the states of the m links and counting how many of the corresponding graphs are operational. This number divided by n is an unbiased estimator of the reliability. However, in applications, the unreliability is often very small (e.g., smaller than 10^{-10}), in which case the situation where the selected nodes are not connected is a rare event, so crude Monte Carlo is very ineffective and variance reduction techniques are essential. Several variance reduction methods have been proposed for this problem. Among the most prominent ones, we find conditional Monte Carlo approaches (Cancela and El Khadiri, 2003; Cancela et al., 2009b; Elperin et al., 1991; Gertsbakh and Shpungin, 2010; Lomonosov and Shpungin, 1999), methods that change the sampling distribution (usually by concentrating the sampling in subsets of the sample space where there is more uncertainty, using paths and cuts to determine those subsets) (Fishman, 1986; Manzi et al., 2001; Zenklusen and Laumanns, 2011), approximate zero-variance importance sampling (L'Ecuyer et al., 2011), and combinations of those (Cancela et al., 2010). There are many more. Cancela et al. (2009a) survey and compare the best available methods so far for large highly-reliable networks. They conclude that one of the best performing methods is that of Hui et al. (2005), which combines a cross-entropy technique with the merge process (also called turnip method) described in

Gertsbakh and Shpungin (2010). On the other hand, Hui et al. (2005) observe that the cross-entropy technique only brings a limited improvement over the turnip alone (at best a factor 1.5 of variance reduction in their examples), and requires additional work. For this reason, in this paper, the performance comparison of our method with others is focused on the turnip, which furthermore uses a similar dynamic representation of the problem. Note that other methods can be more efficient for certain particular examples, depending on the network topology, its size, and its reliability. No method is a universal winner. Some methods provide estimators having (provably) bounded relative error when the link unreliabilities converge to 0 while the network topology and size remain fixed. The method in L'Ecuyer et al. (2011) even has vanishing relative error, which means that the relative error converges to 0 when the unreliabilities converge to 0, under some conditions. A method having one of those properties will eventually win over a method that does not have it, when the unreliabilities become small enough, but not necessarily if the component unreliabilities are not very small (the network can still have a very small unreliability if it is large and can fail in many ways). All the efficient methods mentioned above assume that the links fail independently. The important situation where the link failures are dependent has received little attention so far. Our proposal is able to deal with this situation.

The new method proposed here is an adaptation of the *generalized splitting* (GS) algorithm introduced by Botev and Kroese (2010), which is itself a modification of the classical multilevel splitting methodology for rare-event simulation (L'Ecuyer et al., 2009). The general idea of GS is to define a discrete-time Markov chain whose state (at any given step) represents a realization of the random variables involved in the simulation. This chain evolves by resampling these random variables (at each step) under a conditional distribution that pushes the chain toward a state that corresponds to the rare event of interest. In our context, this rare event occurs when the network is in a non-operating state, and the simulation (generating the link states) normally involves generating a vector of m binary random variables. However, GS does not work very well with such a binary vector as a state of the Markov chain, because we do not have a simple way of measuring how close each binary vector is to the rare event. To make GS work effectively, we use an artificial state that contains more information than the binary vector, based on a scheme borrowed from Elperin et al. (1991) and Lomonosov (1974, 1994). The idea is to transform the static model into a dynamic one by assuming that all the links start in the failed state at time 0 and that each link is "repaired" after a random time, whose distribution is selected so that the probability of repair

at time 1 is the reliability of the link. Thus, the reliability of the graph is the probability that it is operational at time 1, and the crude Monte Carlo algorithm can be reformulated as follows: Generate the vector of repair times and check if the graph is operational at time 1, repeat this n times independently, and estimate the reliability by the proportion of those graphs that are operational. So far, this is just equivalent to crude Monte Carlo, but if we take the vector $\mathbf{Y}$ of repair times as the state of the system (we call it the *latent state* of the graph) then we can compute at what time the graph becomes operational and use this number to measure how close we are to the rare event. Our GS implementation exploits this feature. Note that the coordinates of $\mathbf{Y}$ do not have to be independent, so it is possible (at least in principle) to handle dependence between the component failures.

For any latent state $\mathbf{Y}$, let $S(\mathbf{Y})$ denote the first time when the graph becomes operational with this $\mathbf{Y}$, and call it the *score* or *measure of importance* of $\mathbf{Y}$. Ideally, we would like to sample $\mathbf{Y}$ (repeatedly) from its distribution conditional on $S(\mathbf{Y}) > 1$ (i.e., the graph being not yet operational at time 1). To achieve this with GS, we first partition the interval $[0, 1]$ by fixed levels $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_\tau = 1$. At stage t of the algorithm, for $t \geq 1$, we have a collection of latent states $\mathbf{Y}$ (not necessarily independent) from their distribution conditional on $S(\mathbf{Y}) > \gamma_{t-1}$. For each of them, we construct a Markov chain starting in that latent state, and which evolves by resampling some of the link repair times under their distribution conditional on the score remaining larger than γ_{t-1}. While running all those chains for a given number of steps, we collect all the visited latent states $\mathbf{Y}$ whose score is above γ_t, and use them for the next stage. By discarding the latent states with score below γ_t and starting new chains from those with score above γ_t, and repeating this at each stage, we favor the latent states with larger repair times and eventually obtain a sufficiently large number of states with score above 1. Based on the number of steps of the Markov chains at each stage, and the proportion of latent states that reach the final level, we obtain an unbiased estimator of $\mathbb{P}[S(\mathbf{Y}) > 1]$, which is the graph unreliability.

The remainder is organized as follows. In Section 2, we define the static reliability problem considered in this paper. In Section 3, we give a formulation with latent variables and discuss alternative choices of repair time distributions. In Section 4, we state our GS algorithm, and detail an implementation based on Gibbs sampling for the case of independent links. In Section 5 we provide an adaptive pilot algorithm to estimate good choices of the levels. In Section 6, we give the details of an appropriate data structure to represent the graph and its connectivity structure for the current state $\mathbf{Y}$ of the Markov chain, for an effective GS

implementation. Numerical experiments reported in Section 7 show that the GS method can outperform the best other available methods on some large examples. In Section 8, we discuss the case of dependent links, where the dependence is captured via a copula, and we show that, unlike other methods, GS also applies in that context.

2. The static graph reliability problem

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of vertices (or nodes) $\mathcal{V}$ and a set of edges (or links) $\mathcal{E}$. Each edge can be either operational or failed. The *configuration* of the system can be denoted by $\mathbf{x} = (x_1, \ldots, x_m)$ where m is the number of edges, $x_i = 1$ if edge i is operational, and $x_i = 0$ if edge i is failed. Let $\mathcal{G}(\mathbf{x})$ denote the subgraph of $\mathcal{G}$ that contains only the edges i for which $x_i = 1$.

A subset of nodes $\mathcal{V}_0 \subset \mathcal{V}$ is selected a priori and the system (or graph) is said to be *operational* in the configuration $\mathbf{x}$ if all nodes in $\mathcal{V}_0$ are connected to each other by at least one path in $\mathcal{G}(\mathbf{x})$. Let $\Psi(\mathbf{x}) = 1$ when the system is operational and $\Psi(\mathbf{x}) = 0$ otherwise. This function Ψ is referred to as the *structure function* of the graph (Barlow and Proschan, 1975).

Suppose that edge i is operational with probability r_i, so the system's configuration is a random vector $\mathbf{X} = (X_1, \ldots, X_m)$ where $\mathbb{P}(X_i = 1) = r_i = 1 - \mathbb{P}(X_i = 0)$. It is typically assumed in the literature that the X_i's are independent. Here, we consider both the independent and dependent cases. In the dependent case, $\mathbf{X}$ has a multivariate Bernoulli distribution, which may be specified via a copula, as explained in Section 8. The goal is to estimate $r = \mathbb{P}(\Psi(\mathbf{X}) = 1)$, the reliability of the system.

An important special case is the *two-terminal network reliability* problem, where $\mathcal{V}_0$ contains only two nodes, $\mathcal{V}_0 = \{v_0, v_1\}$, and $\Psi(\mathbf{x}) = 1$ if and only if there is a path between v_0 and v_1. Another special case of interest is the *all-terminal network reliability* problem, where $\mathcal{V}_0 = \mathcal{V}$, so $\Psi(\mathbf{x}) = 1$ if and only if all nodes are connected.

Standard Monte Carlo estimates r by generating n independent realizations of $\mathbf{X}$, and taking the average of the n replicates of $\Psi(\mathbf{X})$ as an estimator of r, or equivalently taking the average of the n replicates of $1 - \Psi(\mathbf{X})$ as an estimator of the unreliability $u = 1 - r$. The square relative error (the relative variance) of this estimator of u is $r/(nu)$, which increases to infinity when $u \to 0$. For highly-reliable systems, u is very small, so we have a rare-event situation, and n must be very large to get a meaningful estimator. For example, if $u = 10^{-10}$,

we need $n > 10^{12}$ to obtain a relative error below 10%. This is much too inefficient and better sampling strategies are needed.

3. A time-dependent formulation with latent variables

Inspired by the graph evolution method in Elperin et al. (1991), we turn the static system into a fictive dynamic model where all edges are initially failed, and edge i becomes operational (repaired) at some random time Y_i, where Y_i has a continuous cumulative distribution function (cdf) F_i for which $F_i(0) = 0$ and $F_i(1) = r_i$. This vector $\mathbf{Y} = (Y_1, \ldots, Y_m)$ is the latent state of the system.

We start with the situation where the X_i's are assumed independent (the dependent case is discussed in Section 8). Then, the Y_i are also taken as independent and they are easy to generate by inverting F_i. Note that $\mathbb{P}[Y_i \le \gamma] = F_i(\gamma)$ and $\mathbb{P}[Y_i \le 1] = F_i(1) = r_i$. We define $X_i(\gamma) = \mathbb{I}(Y_i \le \gamma)$ for all $\gamma \ge 0$, where $\mathbb{I}$ is the indicator function, and $X_i = X_i(1)$. The random variable $X_i(\gamma)$ gives the status of link i at time γ. The resulting random variables $\mathbf{X} = (X_1, \ldots, X_m)$ and $\Psi(\mathbf{X})$ have exactly the same distributions as for the original static model. The interpretation is that the components (or edges) become operational one by one at random times and we are interested in the reliability at time 1. We also define

$$S(\mathbf{Y}) = \inf\{\gamma \ge 0 : \Psi(\mathbf{X}(\gamma)) = 1\},$$

the first time when the graph becomes operational. The function S acts as an *importance function* in the splitting method. The figure of merit $S(\mathbf{Y})$ can be interpreted as a measure of closeness of $\mathbf{Y}$ to a non-operational state (the higher the value, the closer we are, and $S(\mathbf{Y}) > 1$ means that the graph is already non-operational). We will explain in Section 6 how to compute $S(\mathbf{Y})$ efficiently, for any $\mathbf{Y}$, and update it when we change $\mathbf{Y}$.

This reformulation with the latent variables Y_i allows us to consider the reliability at any time $\gamma > 0$: the configuration is $\mathbf{X}(\gamma)$ and the graph is operational if and only if $\Psi(\mathbf{X}(\gamma)) = 1$. Once the latent variables Y_i are generated, we can compute from them an estimator of the reliability for any $\gamma > 0$. Note that the "time" γ and the Y_i's here have nothing to do with simulation time; they are just a convenient abstraction useful for the GS algorithm.

In Elperin et al. (1991, 1992); Gertsbakh and Shpungin (2010); Lomonosov (1974, 1994), Lomonosov and Shpungin (1999), each F_i is taken as an exponential distribution with rate $\lambda_i = -\ln(1 - r_i)$ (or mean $1/\lambda_i$). Then $\mathbb{P}[Y_i \le \gamma] = 1 - e^{-\lambda_i \gamma} = 1 - (1 - r_i)^\gamma$ and

$\mathbb{P}[Y_i \le 1] = r_i$. To generate Y_i by inversion in this case, we generate $U_i \sim U(0,1)$ and return $Y_i = \ln(1-U_i)/\ln(1-r_i)$. A simpler choice is the uniform distribution over the interval $[0, 1/r_i]$, for which $\mathbb{P}[Y_i \le \gamma] = r_i\gamma$. In this case, Y_i is generated via $Y_i = U_i/r_i$. Another possible choice is the normal distribution with mean 0 and standard deviation $\sigma = 1/\Phi^{-1}(r_i)$, for which $\mathbb{P}[Y_i \le \gamma] = \mathbb{P}[Y_i/\sigma \le \gamma/\sigma] = \Phi(\gamma/\sigma)$, where Φ is the standard normal cdf. Here, we can generate Y_i via $Y_i = \Phi^{-1}(U_i)/\Phi^{-1}(r_i)$. This one can be used for the modeling of dependent links via a normal copula, as we will see later. Note that in this case, the repair times Y_i are generated over $(-\infty, \infty)$, and we must take $\gamma_0 = -\infty$ instead of $\gamma_0 = 0$.

Changing the distributions F_i, for example from exponential to uniform, does not change the distribution of the graph state at time $\gamma = 1$, but it can change the distribution of the repair permutation (the order in which the links are repaired) when the r_i are not all equal. To illustrate this, consider two links, with reliabilities $r_1 = 0.99$ and $r_2 = 0.5$. Suppose that their repair times are generated by inversion from $U_1 = 0.693$ and $U_2 = 0.25$, respectively. If the Y_i are taken as exponential, this gives $Y_1 \approx 0.2564$ and $Y_2 \approx 0.4150$, so $Y_1 < Y_2$, whereas if they are taken as uniform, this gives $Y_1 = 0.7$ and $Y_2 = 0.50$, so $Y_1 > Y_2$. The performance of the GS method may thus depend on the choice of distribution F_i, because the chains that are selected for splitting at intermediate levels may differ. On the other hand, if $r_1 = r_2$ and if we assume that the same type of distribution F_i is used for all components (we see no reason for doing otherwise), then $Y_1 < Y_2$ if and only if $U_1 < U_2$ for all choices of F_i, which means that the order of repairs is the same for all continuous repair-time distributions, including the exponential and the uniform. This implies that if all the links have the same reliability and we use the same F_i for all i, all choices of repair distributions are equivalent.

4. A generalized splitting algorithm

We adapt to the present setting the GS algorithm introduced by Botev and Kroese (2010). This provides an efficient way to sample sufficiently many latent states $\mathbf{Y}$ in the region where $S(\mathbf{Y}) > 1$ and obtain from that an unbiased estimator of the unreliability $u = \mathbb{P}[S(\mathbf{Y}) > 1]$. This algorithm is a modification of the classical multilevel splitting method used for rare-event simulation (Ermakov and Melas, 1995; Garvels et al., 2002; Glasserman et al., 1999; Kahn and Harris, 1951; L'Ecuyer et al., 2006, 2007, 2009). In the GS algorithm, we first choose an integer $s \ge 2$, called the *splitting factor*, then we select an integer $\tau > 0$ and real

numbers $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_\tau = 1$ for which

$$\rho_t \stackrel{\text{def}}{=} \mathbb{P}[\Psi(\mathbf{X}(\gamma_t)) = 0 \mid \Psi(\mathbf{X}(\gamma_{t-1})) = 0] = \mathbb{P}[S(\mathbf{Y}) > \gamma_t \mid S(\mathbf{Y}) > \gamma_{t-1}] \approx 1/s \qquad (1)$$

for $t = 1, \ldots, \tau$ (except for ρ_τ, which can be larger than $1/s$). These γ_t represent the levels of the splitting algorithm and τ is the number of levels. Good values can be estimated by an (independent) adaptive pilot algorithm, as explained in Section 5. In Botev and Kroese (2010), the authors typically use $s = 10$, so $\rho_t \approx 1/10$, but this choice is arbitrary and has no particular justification. Our empirical experiments with various choices of s (see Section 7) suggest than $s = 2$ typically gives the most efficient estimator (i.e., whose product of variance and expected computing time is smallest). The splitting factor of $e^2 \approx 7.39$ has been proved optimal for classical multilevel splitting under simplifying assumptions, but this does not apply to GS, which operates quite differently.

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

$$f_t(\mathbf{y}) \stackrel{\text{def}}{=} f(\mathbf{y}) \frac{\mathbb{I}[S(\mathbf{y}) > \gamma_t]}{\mathbb{P}[S(\mathbf{Y}) > \gamma_t]}, \qquad (2)$$

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 $\mathbf{Y}_{t,j}$ conditional on the current state $\mathbf{Y}_{t,j-1}$, is denoted by $\kappa_t(\cdot \mid \mathbf{Y}_{t,j-1})$. There are many possibilities for how to construct κ_t; two of them, based respectively on Gibbs sampling and on hit-and-run, will be examined later. Note that at the first level, generating $\mathbf{Y}$ conditional on $S(\mathbf{Y}) > \gamma_0$ is the same as generating it unconditionally, so the density f_0 is the same as f, and if a generated state $\mathbf{Y}$ satisfies $S(\mathbf{Y}) > \gamma_1$, then its distribution is obviously the distribution of $\mathbf{Y}$ conditional on $S(\mathbf{Y}) > \gamma_1$, so it has density f_1. 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 1 states this procedure with a single starting chain. It computes an unbiased estimator W of the unreliability u and returns its value. In the algorithm, $\mathcal{X}_t$ denotes a set of latent states $\mathbf{Y}$ that have reached the level γ_t. Indentation delimits the scope of the **if**, **else**, and **for** statements. This algorithm will be invoked n times, independently, to obtain n realizations $W_1, \ldots, W_n$ of W, and one can estimate u by the average $\bar{W}_n = (W_1 + \cdots + W_n)/n$

Algorithm 1 : The GS algorithm; returns W, an unbiased estimate of u

Require: $s, \tau, \gamma_1, \ldots, \gamma_\tau$

```
Generate a repair-time vector Y from its unconditional density f.
if S(Y) > γ_1 then
    X_1 ← {Y}
else
    return W ← 0
for t = 2 to τ do
    X_t ← ∅      // set of states that have reached the level γ_t
    for all Y_0 ∈ X_{t−1} do
        for j = 1 to s do
            sample Y_j from the density κ_{t−1}(· | Y_{j−1})
            if S(Y_j) > γ_t then
                add Y_j to X_t
return W ← |X_τ|/s^{τ−1} as an unbiased unreliability estimate.
```

and the variance $\sigma^2 = \mathrm{Var}[W]$ by the empirical variance $S_n^2 = \sum_{i=1}^n (W_i - \bar{W}_n)^2/(n-1)$. Proposition 1 states that these are unbiased estimates. From this, a confidence interval for u can be computed in the usual way; for example $[\bar{W}_n - 1.96\, S_n/\sqrt{n},\ \bar{W}_n + 1.96\, S_n/\sqrt{n}]$ for a 95% confidence interval based on the normal distribution.

Proposition 1 *The empirical mean $\bar{W}_n$ and variance S_n^2 defined above are unbiased estimators of u and σ^2, respectively.*

Proof. The result follows directly from Proposition 3.1 of Botev and Kroese (2010). Note that $\mathbb{E}[S_n^2] = \sigma^2$ follows from the fact that the W_j are independent. □

To sample $\mathbf{Y}_j$ from the density $\kappa_{t-1}(\cdot \mid \mathbf{Y}_{j-1})$ in the inner loop of the algorithm, in the independent case, one possibility (which we adopt here) is to use Gibbs sampling as follows. We first select a permutation of the m coordinate indexes $\{1, \ldots, m\}$. For example, it could be just the identity permutation (the natural order), which gives the systematic scan Gibbs sampler, or a new random permutation at each step, which gives the *random scan* Gibbs sampler (Rubinstein and Kroese, 2007). Then we visit the coordinates one by one, in the order specified by the permutation. When visiting coordinate Y_i (say) of the current vector $\mathbf{Y}$, we erase and forget the current value of Y_i and resample it from its distribution conditional on $S(\mathbf{Y}) > \gamma_{t-1}$, given the other coordinates of $\mathbf{Y}$, so 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, then the modified $\mathbf{Y}$ still has density f_{t-1}.

It turns out that there are just two possibilities for the conditional distribution of Y_i when we resample it. If we currently have $Y_i > \gamma_{t-1}$ and if by adding link i to the current configuration (changing Y_i to a value smaller than γ_{t-1}) we would have $S(\mathbf{Y}) \leq \gamma_{t-1}$, i.e., if adding link i makes the graph operational, then we must resample Y_i from its distribution conditional on $Y_i > \gamma_{t-1}$. Otherwise, we resample it from its original distribution, which is allowed because link i alone has no influence on the graph status given the current configuration of the other links. This Gibbs sampler is stated in Algorithm 2. In this algorithm, $(Y_1, \ldots, Y_{i-1}, 0, Y_{i+1}, \ldots, Y_m)$ represents the current vector $\mathbf{Y}$ but with component i forced to be repaired at time 0, so the condition $S(Y_1, \ldots, Y_{i-1}, 0, Y_{i+1}, \ldots, Y_m) \leq \gamma_{t-1}$ means that the graph becomes operational at level γ_{t-1} when component i is forced to be operational.

Algorithm 2 : Gibbs sampling for the transition density κ_{t-1} in the independent case

Require: $\mathbf{Y} = (Y_1, \ldots, Y_m)$ for which $S(\mathbf{Y}) > \gamma_{t-1}$ and a permutation π of $\{1, \ldots, m\}$.
  for $k = 1$ **to** m **do**
    $i \leftarrow \pi(k)$
    if $S(Y_1, \ldots, Y_{i-1}, 0, Y_{i+1}, \ldots, Y_m) \leq \gamma_{t-1}$ **then**
      resample Y_i from its density truncated to (γ_{t-1}, ∞)
    else
      resample Y_i from its original density
  return $\mathbf{Y}$ as the resampled vector.

For the situation where the links are independent, when the Y_i are uniform over $[0, 1/r_i]$, their distribution conditional on $Y_i > \gamma$ is uniform over $(\gamma, 1/r_i]$. If Y_i is exponential with rate λ_i, then the conditional distribution is that of an exponential with rate λ_i, shifted to the right by γ, so we can just generate the exponential as usual and add γ. For the normal distribution, we can generate Y_i conditional on $Y_i > \gamma$ by generating U uniform over $(\Phi(\sigma\gamma), 1)$ and returning $Y_i = \sigma\Phi^{-1}(U)$.

5. Pilot algorithm to estimate the levels

We now explain the adaptive pilot algorithm that we use to estimate the appropriate levels γ_t for the GS algorithm. Given an integer $s \geq 2$, we want to estimate the levels γ_t for which (1) is satisfied. To do this, we first select a large integer n_0, multiple of s. We generate n_0 independent states $\mathbf{Y}$ from density $f_0 = f$, and select γ_1 so that exactly n_0/s of them have $S(\mathbf{Y}) > \gamma_1$. Then at each step t, for $t = 2, 3, \ldots$, we run a Markov chain with transition kernel density κ_{t-1} for s steps from each of those n_0/s states $\mathbf{Y}$ for which $S(\mathbf{Y}) > \gamma_{t-1}$. This

gives another n_0 states and we select γ_t so that exactly n_0/s of them have $S(\mathbf{Y}) > \gamma_t$. This is done until $\gamma_t \geq 1$ for some t. Then τ is set to this t and we put $\gamma_\tau = 1$. This is stated in Algorithm 3.

Algorithm 3 Adaptive pilot algorithm to estimate the appropriate levels.

Require: an integer $s \geq 2$ and a large integer n_0 multiple of s.

```
q ← n_0 − n_0/s
X_1 ← ∅
for i = 1 to n_0 do
    generate a repair-time vector Y from density f and add it to X_1
sort the elements of X_1 by increasing order of S(Y), say Y_(1), ..., Y_(n_0)
γ_1 ← [S(Y_(q)) + S(Y_(q+1))]/2
t ← 1
while γ_t < 1 do
    t ← t + 1
    X_{t−1} ← {Y_(q), ..., Y_(n_0)}      // retain only the best n_0/s elements from X_{t−1}
    X_t ← ∅
    for all Y_0 ∈ X_{t−1} do
        for j = 1 to s do
            sample Y_j from the density κ_{t−1}(· | Y_{j−1}) and add it to X_t
    sort the elements of X_t by increasing order of S(Y), say Y_(1), ..., Y_(n_0)
    γ_t ← [S(Y_(q)) + S(Y_(q+1))]/2
τ ← t and γ_τ ← 1
return τ, γ_1, ..., γ_τ
```

In this algorithm, $\mathcal{X}_t$ denotes a set of latent states $\mathbf{Y}$ for which $S(\mathbf{Y}) > \gamma_{t-1}$. When this set contains n_0 elements, we sort it to retain the n_0/s states having the largest value of $S(\mathbf{Y})$, and we remove the other states from this set. The level γ_t is placed heuristically midway between the (n_0/s)th and the $(n_0/s+1)$th largest values of $S(\mathbf{Y})$.

6. Data structures for an efficient implementation

An efficient implementation of the GS algorithm requires an adequate representation of the graph so that the appropriate conditional distribution for resampling a link can always be identified rapidly, and this representation must be easy to update (quickly) after a link has been resampled. In our implementation, we use the following representation.

The basic structure of the graph is represented by a data structure (an object) that contains the set of nodes and the set of edges, both stored in an array. Each node has a list of adjacent edges and corresponding neighbors. Each edge i connects a pair of nodes j

and k (we write $i = (j, k)$) and has parameter r_i. Other fixed parameters of the links, for example the means $1/\lambda_i$ for the exponential distributions, can be memorized in this structure. This structure is stored in a single object that does not change during the execution of the algorithm.

During the execution of the GS algorithm, we keep track of the current vector $\mathbf{Y} = (Y_1, \ldots, Y_m)$ for each instance of the Markov chain. From this vector $\mathbf{Y}$, we immediately obtain $\mathbf{X}(\gamma)$, and we can eventually compute $\Psi(\mathbf{X}(\gamma))$, for any level γ. But we want a data structure that permits us to see immediately from what conditional distribution each Y_i 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_i is changed. The information in this data structure (or object) 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, at the current level γ_{t-1}, we maintain the set of connected components of the graph when the configuration is $\mathbf{X}(\gamma_{t-1})$ by following the principles described in Zaroliagis (2002). Any two vertices in the same connected component are connected by a path formed by links i for which $X_i(\gamma_{t-1}) = 1$, and there is no such link if the two vertices are in different connected components. Thus, the connected components form a partition of the set $\mathcal{V}$ of vertices. In the terminology of dynamic connectivity algorithms, each component is connected by (at least) some *spanning tree*, and the set of components form a *forest* of spanning trees. For each vertex, we memorize the component (or tree) to which it currently belongs.

When a given Y_i is modified, if it was larger than the current level γ_{t-1} and it becomes smaller, then link i is added to the configuration. If this link connects two nodes j and k in the same component, there is nothing to do, otherwise the two components are merged into a single one. This is straightforward. If Y_i was smaller than the current level γ_{t-1} and it becomes larger, then link i must be removed from the configuration. If $i = (j, k)$, then we must check if there is still a path between j and k 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. We do it via a *breadth-first search* (BFS) algorithm that starts from j and tries to connect with k, using only the links that are up and connect nodes that belong to the (previous) tree that contained j and k. Our implementation is adapted from the C++ graph package LEDA (Zaroliagis, 2002; Italiano, 2009). The state

representation as a forest of spanning trees reduces the number of calls to the (expensive) BFS algorithm, because any link that does not belong to a spanning tree can be removed without invoking BFS.

When the level is increased, say from γ_{t-1} to γ_t, we take all the links i for which $\gamma_{t-1} < Y_i \leq \gamma_t$, set $X_i = 1$ for these links (they are added to the current configuration), and we merge the connected components that these added links may connect.

In the special case where $\mathcal{V}_0 = \{v_0, v_1\}$ (only two nodes), we resample conditional on $Y_i > \gamma$ if and only if v_0 and v_1 are in two different components (trees) and link i connects these same two components. This is straightforward to check from the data structure.

In the GS and pilot algorithms, we must also check, at each step of the chain, if the next level has been reached or not. That is, if we are simulating at level γ_{t-1}, after each step of the Markov chain we must check if $\Psi(\mathbf{X}(\gamma_t)) = 0$, i.e., if the nodes of $\mathcal{V}_0$ are still not connected at the next level γ_t. If $\Psi(\mathbf{X}(\gamma_t)) = 0$, then we make a copy of the chain and insert it in the set $\mathcal{X}_t$, so that it can be used as one of the starting points at the next stage. To compute $\Psi(\mathbf{X}(\gamma_t))$, we temporarily add the links i for which $\gamma_{t-1} < Y_i \leq \gamma_t$, and check if it connects the nodes in $\mathcal{V}_0$.

7. Numerical examples

In this section, we report numerical experiments with the GS algorithm. The reported results are a representative selection from a larger set of experiments that we performed. Unless indicated otherwise, the link unreliabilities q_i are all equal to the same constant q, the splitting factor is $s = 2$ (in our experiments this was always the best choice), the levels were estimated from the adaptive Algorithm 3 with $n_0 = 10^4$, and we used uniform lifetime distributions. For each graph, we tried several values of q. GS was applied with $n = 10^6$ in most cases, but sometimes with $n = 10^7$ or 10^8 when deemed necessary to make the results more meaningful. In the tables, we report the number τ of levels specified by Algorithm 3, 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 CPU time T (in seconds) required by the n runs of the GS algorithm, and the *work-normalized relative variance* (WNRV) of $\bar{W}_n$, defined as $\mathrm{WNRV}[\bar{W}_n] = T \times \mathrm{RE}^2[\bar{W}_n]$. The latter measure is approximately independent of n, when n is large. It is appropriate for comparing unbiased estimators by taking into account both the variance and the computing time. On the other hand, the computing time

may depend strongly on the computing platform and the quality of the implementation. For this reason, the WNRV is not always a definitive measure and it is also relevant to look at the relative variance $S_n^2/(\bar{W}_n)^2$.

We also compare the performance of GS with the permutation Monte Carlo of Elperin et al. (1991), and its improvement, the turnip method described in Gertsbakh and Shpungin (2010), which is one of the most effective algorithms that we know.

7.1 The dodecahedron graph

We start with the dodecahedron graph illustrated in Figure 1, and often used as a benchmark for network reliability estimation methods (Cancela et al., 2009a,b). This graph has 20 nodes and 30 links. The system is operational when nodes 1 and 20 (shaded) are connected.

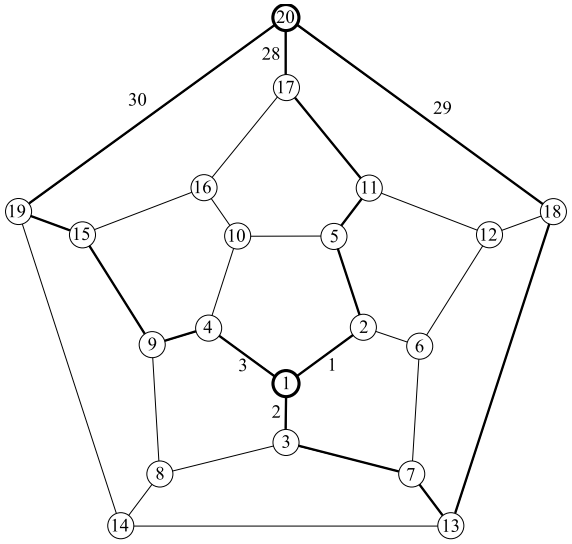

Figure 1: A dodecahedron graph with 20 nodes and 30 links. The sets of links {1, 2, 3} and {28, 29, 30} form two important disjoint cuts: Among the minimal cuts that separate nodes 1 and 20, they are the two cuts with the smallest number of links.

In Table 1, we compare some values of the split factor s for $q = 10^{-3}$, for which the graph unreliability is approximately 2×10^{-9}, and $n = 10^7$. When s increases, the number of levels decreases (as expected). We also find that the CPU time decreases, but the variance increases at a faster rate, and the WNRV increases as a result. Thus, $s = 2$ seems to be the optimal value. We made similar experiments with other examples and observed the same type of behavior. An intuitive explanation is that with a smaller s, the computing effort is readjusted more frequently toward the chains that have already passed the next threshold.

Table 1: GS for the dodecahedron: $n = 10^7$, $q = 10^{-3}$, $\mathcal{V}_0 = \{1, 20\}$

split factor	2	5	10	20
τ	29	13	9	7
$\bar{W}_n$	2.004e-9	2.017e-9	2.019e-9	1.988e-9
$S_n^2/(\bar{W}_n)^2$	60.2	265	829	2241
RE[$\bar{W}_n$]	0.00245	0.00515	0.00911	0.015
T (sec)	2408	969	638	519
WNRV[$\bar{W}_n$]	0.0145	0.0257	0.0529	0.116

Table 2: GS for the dodecahedron: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

q	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
τ	9	19	29	39	49	59
$\bar{W}_n$	0.002877	2.054e-6	2.022e-9	2.01e-12	1.987e-15	1.969e-18
$S_n^2/(\bar{W}_n)^2$	16.211	38.388	59.101	79.179	98.37	124.49
RE[$\bar{W}_n$]	0.0040	0.0062	0.0077	0.0089	0.0099	0.0112
T (sec)	93	167	224	278	334	376
WNRV[$\bar{W}_n$]	0.0015	0.0064	0.0132	0.0221	0.0329	0.0469

Table 2 reports simulation results with $n = 10^6$, for link unreliabilities q ranging from 10^{-1} to 10^{-6}, giving unreliabilities u from 3×10^{-3} to 2×10^{-18}, approximately. Both the relative error and the CPU time T increase very slowly when q decreases. The increase in T is essentially due to the fact that there are more levels when the unreliability gets smaller. The slow increase in RE is apparently due to an accumulation of noise (variance) from level to level. Thus, GS does not appear to have bounded relative error. However, the relative error remains quite reasonable (around 1.1%) even for the smallest graph unreliability of 2×10^{-18}. With exponential instead of uniform lifetime distributions, the CPU times increase by about 35% for $q = 10^{-1}$, up to 70% for $q = 10^{-9}$.

Table 3: GS for the dodecahedron: $n = 10^6$, $\mathcal{V}_0 = \{1, 4, 7, 10, 13, 16, 20\}$

q	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
τ	7	18	28	37	48	57
$\bar{W}_n$	0.009047	7.188e-6	6.997e-9	7.024e-12	6.853e-15	7.128e-18
$S_n^2/(\bar{W}_n)^2$	12.675	33.849	54.683	74.732	91.04	110.81
RE[$\bar{W}_n$]	0.0036	0.0058	0.0074	0.0086	0.0095	0.0105
T (sec)	75	154	212	260	331	379
WNRV[$\bar{W}_n$]	0.00095	0.0052	0.0116	0.0195	0.0302	0.0420

Table 4: GS for the dodecahedron with random q_i's: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

Uniform	τ	18	25	27	20	23	21
	$\bar{W}_n$	5.636e-6	4.289e-8	1.209e-8	1.363e-6	2.019e-7	6.258e-7
	$S_n^2/(\bar{W}_n)^2$	50.172	68.585	910.06	47.466	57.644	58.975
	RE[$\bar{W}_n$]	0.0071	0.0083	0.0302	0.0069	0.0076	0.0077
	T (sec)	164	200	226	177	188	176
	WNRV[$\bar{W}_n$]	0.0082	0.0137	0.2060	0.0084	0.0108	0.0104
Exponential	τ	18	25	27	20	23	21
	$\bar{W}_n$	5.659e-6	4.296e-8	1.213e-8	1.362e-6	2.005e-7	6.256e-7
	$S_n^2/(\bar{W}_n)^2$	38.521	60.355	55.67	42.514	53.284	43.481
	RE[$\bar{W}_n$]	0.0062	0.0078	0.0075	0.0065	0.0073	0.0066
	T (sec)	215	283	302	240	269	241
	WNRV[$\bar{W}_n$]	0.0083	0.0171	0.0168	0.0102	0.0143	0.0105

To illustrate the fact that GS works well for an arbitrary set of terminal nodes, we report in Table 3 our results for a larger set $\mathcal{V}_0$, selected (somewhat randomly) as $\mathcal{V}_0 = \{1, 4, 7, 10, 13, 16, 20\}$. The graph is operational when all these nodes are connected, so the unreliability is larger than for $\mathcal{V}_0 = \{1, 20\}$. The relative errors and the CPU times are very similar to those for $\mathcal{V}_0 = \{1, 20\}$.

We performed empirical investigations on the impact of the choice of distribution for the repair times Y_i's when the q_i's are different. Overall, although the CPU times are slightly smaller with the uniform, our experiments suggest that the exponential distribution is a more robust choice in general, in the sense that it seems to provide more reliable mean and variance estimators. As a representative illustration, Table 4 reports the results of an experiment where the 30 link unreliabilities q_i were generated randomly as $q_i = 10^{-e_i}$, with the e_i independent and uniformly distributed over the interval $[1, 4]$. After generating the q_i's, we applied the pilot algorithm with $n_0 = 10^4$ to determine the levels γ_t, then we applied GS with $n = 10^6$. This was done with the repair times generated first from the uniform distribution (upper half of the table), then from the exponential distribution (lower half), based on exactly the same uniform random numbers for each run (common random numbers). This experiment was replicated six times, independently, and the six columns of the table give the results of those six independent replications, which correspond to six totally different choices of the link unreliabilities. We see that the CPU time T is always larger with the exponential case, but the empirical variance is always smaller, by more than an order of magnitude in one case (the third column in the table, for which the unreliability

is much smaller and the relative variance with the uniform is much larger than for the other columns). The WNRV is marginally smaller with the uniform case (but not significantly smaller), except for the third replication. In other examples that we tried, the WNRV was a bit smaller for the exponential. The variance was always smaller for the exponential.

Recall that at the final iteration of GS ($t = \tau$), each state $\mathbf{Y}$ for which $S(\mathbf{Y}) > 1$, that is for which $\mathbf{Y} \in \mathcal{X}_\tau$, has been generated from the conditional density f_τ of $\mathbf{Y}$ given that the system is failed at time 1 (although these states are not independent and there is also a dependence between their values and their number). Note that if the exact conditional density f_τ was used in an importance sampling scheme here, this would give zero variance (L'Ecuyer and Tuffin, 2008; L'Ecuyer et al., 2011). The empirical distribution of the states $\mathbf{Y} \in \mathcal{X}_\tau$ collected at the end of the n runs of the GS algorithm provides an approximation of this conditional distribution. To get an idea of its behavior for the dodecahedron example, we computed the average of each coordinate of $\mathbf{Y}$ over all states $\mathbf{Y} \in \mathcal{X}_\tau$, for $q = 10^{-6}$, for the case of exponential repair times. Note that the original (unconditional) mean repair time in this case is $1/\lambda_i = -1/\ln q = 1/(6 \ln 10) \approx 0.07238$. Denoting $\hat{\mu}_i$ the empirical average for coordinate i (that is, the average repair time for edge i), we found $\hat{\mu}_1 = 0.5714$, $\hat{\mu}_2 = 0.5714$, $\hat{\mu}_3 = 0.5712$, $\hat{\mu}_{28} = 0.5733$, $\hat{\mu}_{29} = 0.5734$, $\hat{\mu}_{30} = 0.5736$, and all other $\hat{\mu}_i$'s were in the interval $[0.0721,\ 0.0727]$. This shows that conditional on the system's failure, the edges in the two edge sets (minimal cuts) $\{1, 2, 3\}$ and $\{28, 29, 30\}$ have much larger repair times, or much smaller reliabilities, than unconditionally, and the averages are practically unchanged for the other edges. Those two sets are *minimal cuts* in the graph, meaning that if none of the links in one of those cuts is operational, the system cannot be operational (nodes 1 and 20 are necessarily disconnected), and this property no longer holds if we remove one link from the cut. They are also the minimal cuts with maximal probability, that is, the probability that all links of the cut are down is maximal, because they have the smallest number of links and all links have the same failure probability q. In this case, the probability that a given cut with ℓ links is down is simply q^ℓ, so this probability is larger for these two cuts compared with that of the other cuts by at least a factor of $1/q$, which is 10^6 in this numerical example. We can also observe that typically, when all the edges in $\{1, 2, 3\}$ are down, the edges in $\{28, 29, 30\}$ are not, and vice-versa. This shows up via a strong negative dependence between the minimum repair time $R_1 = \min(Y_1, Y_2, Y_3)$ of edges $\{1, 2, 3\}$ and the minimum $R_2 = \min(Y_{28}, Y_{29}, Y_{30})$ for edges $\{28, 29, 30\}$. Figure 2 gives a scatter plot of the pairs (R_1, R_2) for the 2092 states $\mathbf{Y}$ that reached the last level in the first 4000 runs of GS.

In all of these pairs, either R_1 or R_2 is slightly larger than 1, but never both simultaneously. This means that at time 1, either node 1 is totally isolated, or node 20 is totally isolated, but not both at the same time. We observed the same behavior for all 10^6 runs of GS. In total, there were 573366 states $\mathbf{Y}$ that reached the last level, and for all the corresponding pairs (R_1, R_2), either R_1 or R_2 was larger than 1, but never both. Note that the event that both R_1 and R_2 are smaller than 1 while $S(\mathbf{Y}) > 1$ because nodes 1 and 20 are disconnected in a different way has a positive probability, but this probability is very small here for $q = 10^{-6}$, because the two three-link minimal cuts have a failure probability at least one million times larger than any other cut. With larger values of q, we have observed realizations where $R_1 < 1$ and $R_2 < 1$, and their frequency increased when q increased. For example, for $q = 10^{-2}$, there were 16258 such realizations out of 537488 chains reaching the last level for $n = 10^6$.

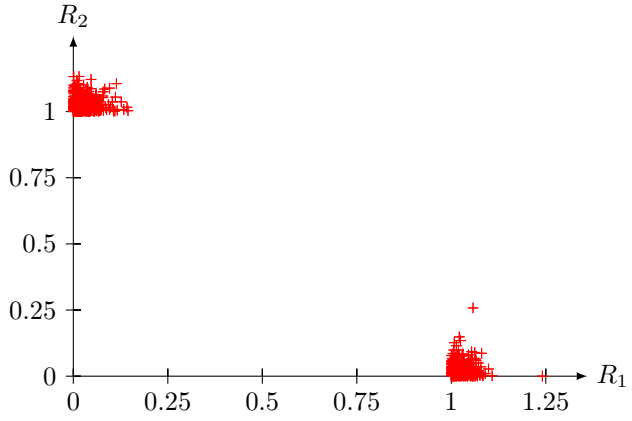

Figure 2: Scatter plot of 2092 observed pairs (R_1, R_2) for GS applied to the dodecahedron example with $q = 10^{-6}$.

For this example, the probability that one of these two cuts is down is $2q^3 - q^6$, which is very close to the value of $\bar{W}_n$ given in Table 2 when q is small. The fact that failure occurs mostly by the failure of one of these two cuts can be exploited in many ways to provide more effective estimators than for GS. For example, Fishman's sampling plan (Fishman, 1986; Manzi et al., 2001) and the importance sampling scheme of L'Ecuyer et al. (2011) exploit this type of structural information and can be more effective than GS for this example when q is small. However, an important property of GS is its ability to identify automatically a nearly optimal way of sampling the repair times, without exploiting this type of prior knowledge on important cuts or paths in the network. With GS, there is no need to select

cuts and paths at all.

We also performed experiments with a larger graph constructed as in L'Ecuyer et al. (2011) by juxtaposing three copies of the dodecahedron of Figure 1 in parallel, merging the terminal nodes 1 of the three copies as a single node, and similarly for the three terminal nodes 20. The resulting system has 56 nodes and 90 links. It is operational when the (merged) nodes 1 and 20 are connected. We found that even for $q = 10^{-6}$, which gives an unreliability near 8×10^{-54} and 176 levels for GS, the relative error with one million runs remained around 2%. Thus, GS was able to learn adaptively the joint repair times distribution conditional on network failure for this larger three-dodecahedron network as well, without exploiting specific knowledge of its structure.

7.2 A lattice graph

An $m_1 \times m_2$ lattice graph has m_1 rows that contain m_2 nodes each, arranged in a rectangular lattice. Each node is connected to its neighbors on the left, right, above, and below, when they exist. As an illustration, a 5×8 lattice graph is displayed in Figure 3. We performed experiments with GS for square lattice graphs of several sizes, from 10×10 to 50×50. The terminal nodes were the two nodes at opposite corners, shaded in the figure. The results for the 50×50 graph are shown in Table 5. Even with this very large number of nodes and links, GS manages to estimate the reliability to reasonably good accuracy. Note that here, there is a very large number of cuts with maximal probability.

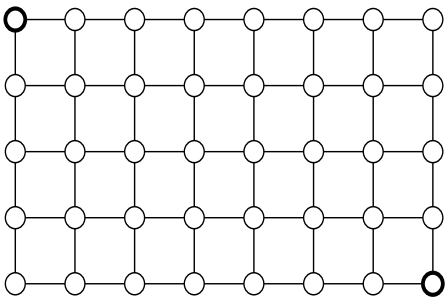

Figure 3: A 5×8 lattice graph, with 40 nodes and 67 edges. The terminal nodes are shaded.

Table 5: GS results for a 50×50 lattice graph, with 2500 nodes and 4900 edges, for $n = 10^4$ and $V_0 = \{1, 2500\}$

q	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
τ	13	19	26	33	39
$\bar{W}_n$	2.148e-4	2.085e-6	2.179e-8	2.156e-10	1.932e-12
$S_n^2/(\bar{W}_n)^2$	21.7	36.5	46.0	61.7	82.6
RE$[\bar{W}_n]$	0.0466	0.0604	0.0678	0.0785	0.0909
T (sec)	19818	19283	18413	17967	17851
WNRV$[\bar{W}_n]$	42.9	70.4	84.7	111	148

7.3 Comparison with the Permutation Monte Carlo and Turnip Methods

7.3.1 Permutation Monte Carlo

As mentioned in the introduction, one of the most effective Monte Carlo-based methods for static network reliability estimation is the turnip algorithm, described in Gertsbakh and Shpungin (2010) and Lomonosov and Shpungin (1999), based on ideas from Elperin et al. (1991, 1992); Lomonosov (1974), and Lomonosov (1994). We will therefore compare GS with this method, which also uses a time-dependent formulation with latent variables. The conceptual principle of the method is to generate repair times $Y_1, \ldots, Y_m$ for the links, exactly as in Section 3, with F_i taken as the exponential distribution with rate $\lambda_i = -\ln(1 - r_i)$, sort these repair times by increasing order to obtain the order statistics $Y_{(1)}, \ldots, Y_{(m)}$, then erase the repair times and retain only the order in which the links are repaired. This gives a permutation $\pi = (\pi(1), \ldots, \pi(m))$ of the links, where $Y_{(j)} = Y_{\pi(j)}$ for each j. One can then compute numerically (using an exact formula) the probability that the graph is operational at time 1, conditional on the permutation π. This conditional probability is the *permutation Monte Carlo* (PMC) estimator. To compute it, one can add the links to the graph one by one, in the specified order, until it becomes operational, say at step C. This integer random variable C is called the *critical number*. At step j of this procedure, before adding the jth link $\pi(j)$, we know that the time $A_j = Y_{\pi(j)} - Y_{\pi(j-1)}$ until this link is added (the next repair time) is an exponential with rate Λ_j equal to the sum of repair rates of all links not yet repaired. We have $\Lambda_1 = \lambda_1 + \cdots + \lambda_m$, and $\Lambda_{j+1} = \Lambda_j - \lambda_{\pi(j)}$ for $j \geq 1$.

Conditional on π and on $C = c$ (an integer), the repair time of the graph is then $A_1 + \cdots + A_c$, a sum of c independent exponential random variables with known rates $\Lambda_1, \ldots, \Lambda_c$. This sum has an *hypoexponential* (or generalized Erlang) distribution, a special case of a

phase-type distribution, and whose (complementary) cumulative distribution function (cdf) can be written using a matrix exponential:

$$\mathbb{P}\left[A_1 + \cdots + A_c > t\right] = \mathbf{e}_1 e^{\mathbf{D}t}\mathbf{1} = \mathbf{e}_1 \sum_{k=0}^{\infty} \frac{\mathbf{D}^k\, t^k}{k!}\mathbf{1}, \tag{3}$$

where $\mathbf{e}_1 = (1, 0, \ldots, 0)$ is an $1 \times c$ vector, $\mathbf{1}$ is a $c \times 1$ column vector of ones, and

$$\mathbf{D} = \begin{pmatrix} -\Lambda_1 & \Lambda_1 & 0 & \ldots & 0 \\ 0 & -\Lambda_2 & \Lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & -\Lambda_{c-1} & \Lambda_{c-1} \\ 0 & \ldots & 0 & 0 & -\Lambda_c \end{pmatrix}$$

is a $c \times c$ matrix (Neuts, 1981). The formula (3) can be developed into

$$\mathbb{P}\left[A_1 + \cdots + A_c > t\right] = \sum_{j=1}^{c} e^{-\Lambda_j t} \prod_{k=1,\, k\neq j}^{c} \frac{\Lambda_k}{\Lambda_k - \Lambda_j}, \tag{4}$$

which can be computed in $\mathcal{O}(c^2)$ time (Ross, 2007, page 299) and (Gertsbakh and Shpungin, 2010, Appendix B). The PMC method computes $\mathbb{P}\left[A_1 + \cdots + A_c > t \mid \pi\right]$ via (4) or (3) and uses it as an estimator of u.

7.3.2 Numerical stability problems

The expression (4) is usually faster to compute than the matrix exponential, but it becomes numerically unstable when c gets large and/or the λ_i are too small (which happens when the link unreliabilities q_i are not sufficiently small), because it is an alternating sum with relatively large terms of similar size. When the λ_i are small, many of the factors $\Lambda_{j+1} - \Lambda_j = \lambda_{\pi(j)}$ in (4) are small, and the effect of this is amplified when c is large. This gives rise to large terms of opposite sign in the sum and the formula becomes unstable. For example, with the computations done in standard 64-bit floating-point arithmetic, if $\lambda_i = 0.1$ for all i, the formula (4) breaks down already for $c \approx 15$, while if $\lambda_i = 3$ for all i, it is usable up to $c \approx 300$. Formula (4) is still usable with the PMC method in the case of the 15×15 square graph with 225 nodes and 420 edges when the link unreliabilities are $q_i = 0.01$ ($\lambda_i = 4.6$), for which $\mathbb{E}[C] \approx 276$; however it breaks down for the 6×6 square graph (with 36 nodes and 60 edges) when $q_i = 0.5$ ($\lambda_i \approx 0.7$), for which $\mathbb{E}[C] \approx 38$. Gertsbakh and Shpungin (2010, Chapter 7) partially recognize the problem: they say that (4) becomes unusable when C exceeds about 30 or 40 (they do not mention the λ_i's), and suggest using Monte Carlo to estimate the

convolution conditional on π in that case. This makes the method much less efficient. In fact, the instability depends on both the λ_i's and on the generated permutation π.

A naive evaluation of the matrix exponential, for example via the sum in (3), is also unreliable and unstable in much the same way as formula (4). We experimented with standard procedures for computing the matrix exponential in software libraries and were able to handle more cases than with (4), but soon ran into instabilities as well. These problems were solved when we switched to the recently proposed scaling and squaring algorithm with Padé approximating polynomials (Higham, 2009). We found this algorithm to be stable and accurate in all of our numerical experiments. On the other hand, this algorithm is significantly slower than the direct use of (4), as we will illustrate in our examples.

7.3.3 The turnip method

An improvement on PMC is the so-called *turnip* method described in Gertsbakh and Shpungin (2010); its difference from PMC is that each time a link is added (repaired), all the links not yet repaired and that connect two nodes that are already connected (that is, the links that become redundant) are immediately removed from consideration. Thus, the subgraph formed by the links repaired so far never contains a cycle. The estimator is also computed via (4), but with Λ_j defined as the sum of repair rates of the links that are still under consideration immediately before step j. Thus, when redundant links are removed from consideration, their repair rates are subtracted from the current Λ_j.

Algorithm 4 : The turnip method

$\Lambda_1 \leftarrow \lambda_1 + \cdots + \lambda_m$
$\mathcal{L} \leftarrow \{1, \ldots, m\}$ // set of links still under consideration
$\mathbf{x} = (x_1, \ldots, x_m) \leftarrow (0, \ldots, 0)$ // current configuration
$k \leftarrow 1$
while the nodes in $\mathcal{V}_0$ are not all connected **do**
    draw link I at random from $\mathcal{L}$, with $\mathbb{P}[I = i] = \lambda_i / \Lambda_k$
    $x_I \leftarrow 1$ and $k \leftarrow k + 1$
    $\Lambda_k \leftarrow \Lambda_{k-1} - \lambda_I$ and remove I from $\mathcal{L}$
    for all $i \in \mathcal{L}$ that connect two nodes already connected in $\mathcal{G}(\mathbf{x})$ **do**
        $\Lambda_k \leftarrow \Lambda_k - \lambda_i$ and remove i from $\mathcal{L}$
$C \leftarrow k$
return the unreliability estimate W computed from (4) or (3), with $t = 1$.

For the turnip method, the naive formula (4) is usable for a larger number of cases than for the PMC method, because the difference $\Lambda_{j+1} - \Lambda_j$ in the denominator is often a sum of

many λ_k's and thus is larger than for the corresponding PMC case. For example, the turnip with the naive formula (4) yields stable numerical results for the 6×6 square graph when $q_i = 0.5$, unlike the PMC method. It also works fine for the 20×20 square graph with 400 nodes and 760 edges with $q_i = 0.01$, for which $\mathbb{E}[C] \approx 383$.

In our experiments, the turnip always performed better than PMC, and was comparatively much better when the link unreliabilities q_i were small and/or when the number of links was large. For example, we tried complete graphs (with a link between each pair of nodes). For a (relatively small) complete graph with 7 nodes and 21 edges, with $q = 10^{-3}$, PMC took approximately 100 times more CPU time to run and its variance was about 6700 times larger than for the turnip. PMC was more efficient than GS only for very small graphs, such as complete graphs with 6 nodes or less and $k \times k$ square lattice graphs with $k \leq 4$.

For either variant, PMC or turnip, it is possible to generate the permutation π directly without generating the repair times, as follows (Gertsbakh and Shpungin, 2010). The first link in the permutation is selected as link j with probability $\lambda_j/(\lambda_1 + \cdots + \lambda_m)$. At step k, the kth link in the permutation is selected among the links still in consideration, with probability λ_j/Λ_k for link j, where Λ_k is the sum of repair rates of the links that remain in consideration. For turnip, this gives Algorithm 4. As for GS, this algorithm will be invoked n times, independently, and the empirical mean $\bar{W}_n$ and variance S_n^2 of the n realizations $W_1, \ldots, W_n$ of W will be used to estimate the unreliability u and the variance of W, and eventually compute a confidence interval.

This algorithm has been proved to provide bounded relative error when all the link unreliabilities q_i converge to 0 together, for a fixed graph (Gertsbakh and Shpungin, 2010). This can be observed empirically in Table 6, which reports the results of $n = 10^6$ independent runs of Algorithm 4 for the dodecahedron graph, using formula (4). The average value of C was approximately 15.6 for all values of q. We see that the relative error is approximately independent of q when q is small. We also find that the turnip is more efficient than GS (smaller RE and smaller WNRV) for this example. With the PMC method (not shown in the table), the variance was roughly 100 times larger than with the turnip for $10^{-6} \leq q \leq 10^{-2}$, and the WNRV was similar to that of GS.

7.3.4 Limitation of the turnip method

A limitation of the turnip is that for large highly-reliable graphs, there may be a relatively small set $\mathcal{P}^*$ of permutations π for which $\mathbb{P}[S(\mathbf{Y}) > 1] \approx \mathbb{P}[S(\mathbf{Y}) > 1 \text{ and } \pi \in \mathcal{P}^*]$ and

Table 6: Turnip for the dodecahedron: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

q	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
$\bar{W}_n$	2.881e-3	2.065e-6	2.006e-9	1.992e-12	1.999e-15	2.005e-18
$S_n^2/(\bar{W}_n)^2$	9.1348	17.736	18.738	19.002	18.924	18.842
RE$[\bar{W}_n]$	0.00302	0.00421	0.00433	0.00436	0.00435	0.00434
T (sec)	15.6	15.5	15.5	15.5	15.5	15.5
WNRV$[\bar{W}_n]$	0.000142	0.000275	0.000290	0.000294	0.000293	0.000292

$\mathbb{P}[\pi \in \mathcal{P}^*]$ is extremely small. Then, getting a permutation π that contributes significantly to the estimator becomes a rare event, and we are back into the trap of rare event simulation.

For the example with three dodecahedrons in parallel, using (4) to compute the conditional probabilities, the turnip gave empirical relative variances $S_n^2/(\bar{W}_n)^2$ larger than for GS by factors ranging roughly from 100 to 300. This indicates that the turnip has difficulty finding the important permutations, whereas GS finds them by some form of multilevel learning. The WNRV was also larger for the turnip, but by a smaller factor (from 2 to 10), because the CPU time per run was smaller. For the turnip, both the relative error and the WNRV are essentially independent of q for q small enough. If we use the stable matrix exponential formula instead of (4) to compute the conditional probabilities, the CPU time (and the WNRV) increase approximately by a factor between 70 and 90 with our implementation, for $10^{-5} \leq q \leq 10^{-1}$. With the PMC method, the variance is about 100 times larger than for the turnip for $q = 0.1$, while for $q \leq 10^{-2}$, the variance is so large that the estimator becomes practically useless (and estimating the variance would be much too time-consuming).

Figure 4 provides more insight on the probabilistic behavior (the distribution) of the GS and turnip estimators for the example of three dodecahedrons in parallel, with $q = 10^{-2}$. It shows three histograms, with a logarithmic horizontal scale (in base 10), and a vertical scale expressed in percentages of the observations visible in the histograms. The histogram represents the nonzero values of W for GS. There were 15482 nonzero values of W out of $n = 10^6$ (that is 1.55%), and their average was $5.7 \times 10^{-16} \approx 10^{-15.25}$, while $\bar{W}_n \approx 8.8 \times 10^{-18} \approx 10^{-17.06}$. The histogram on the left represents the $n = 10^8$ values of W for the turnip, truncated to the values larger than 10^{-25}. The truncated part represents approximately 99.5% of the realizations and its total contribution to the average $\bar{W}_n \approx 8.8 \times 10^{-18}$ is negligible (less than 10^{-27}). It goes down to values smaller than 10^{-150}, the bins with largest counts (in log scale for W) are around 10^{-63}, and the largest values are

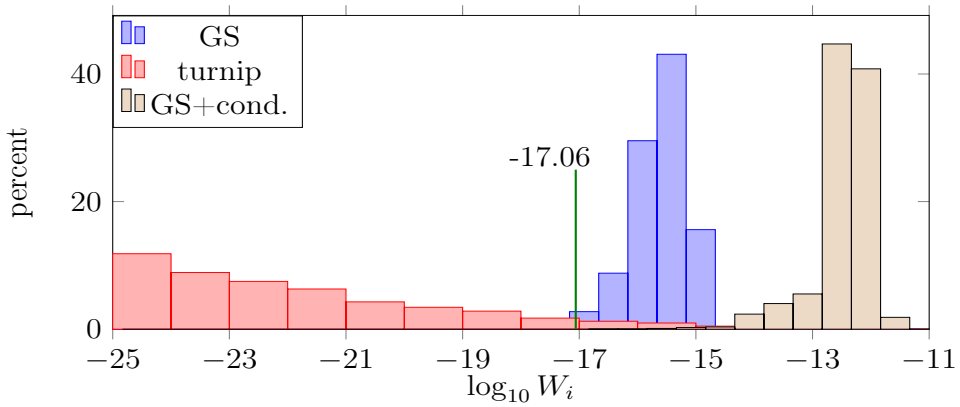

Figure 4: Histograms of the realizations of $\log_{10}(W)$ for GS with $n = 10^6$ (middle, in blue), turnip with $n = 10^8$ (left, in red), and for the conditional probabilities of failure for the permutations π obtained at the last level of GS (right, in brown), for the example of three dodecahedrons in parallel, with $q = 10^{-2}$. The countings are all expressed in percentages of the visible parts. The exact average to be estimated here is $\mu \approx 8.8 \times 10^{-18} \approx 10^{-17.06}$.

between 10^{-12} and 10^{-11} (there are 12 values out of 10^8 in that bin). The histogram looks somewhat like a symmetric bell curve with maximum at around 10^{-63}. The turnip clearly provides a histogram with much larger variability than GS. For the third histogram, on the right, we examined the 15482 GS runs for which W was nonzero, and for each of them, we collected all the states $\mathbf{Y}$ for which $S(\mathbf{Y}) > 1$ at the end of the algorithm. There were 631078 such states. For each of them, we found the corresponding permutation π, then forgot about $\mathbf{Y}$ and computed $\mathbb{P}[S(\mathbf{Y}) > 1 \mid \pi]$ as in the turnip algorithm. Then we made a histogram of these conditional probabilities. Note that the average of these conditional probabilities is *not* a meaningful estimator of the probability μ of interest (it is highly biased, with an average around 10^{-13}). The purpose of this third histogram is to show that the permutations π found by GS at the last level are typically the important permutations, that correspond to significant conditional probabilities $\mathbb{P}[S(\mathbf{Y}) > 1 \mid \pi]$, whereas the turnip has difficulties finding these important permutations.

In summary, the types of situations that favor GS over the turnip appear to be mostly when the graph is large (the typical situation when simulation is required) and the q_i are not so small, while the turnip should be preferred in the opposite situation.

Another difficulty with the turnip is that the stable computation of the matrix exponential sometimes becomes too slow. For example, for the 20×20 square graph, the turnip with the stable matrix exponential method is approximately between 1600 times and 2000 times

slower than the naive method for $10^{-5} \leq q \leq 10^{-1}$. For the 40×40 lattice graph (1600 nodes and 3120 edges), one run of the stable turnip algorithm takes about 17.5 minutes on our desktop computer. An experiment with $n = 10^6$ runs would take about 33 years. GS was able to solve these examples easily.

8. The case of dependent links

In the dependent case, the distribution of $\mathbf{Y}$ is taken as a multivariate distribution with density f, for which $\mathbf{X}$ has the correct target multivariate Bernoulli distribution. The most general way of specifying a dependence structure for the vector $\mathbf{Y}$ (and indirectly for the vector $\mathbf{X}$) is via a copula (Joe, 1997; Nelsen, 1999). A copula is simply a multivariate cdf whose one-dimensional marginal distributions are all uniform over $(0, 1)$. To generate $\mathbf{Y} = (Y_1, \ldots, Y_m)$, we can generate $\mathbf{U} = (U_1, \ldots, U_m)$ from the copula, and put $Y_i = F_i^{-1}(U_i)$ for all i, where F_i is the selected cdf for Y_i, for which $F_i(1) = r_i$ and $F_i(\gamma_0) = 0$. The multivariate distribution of $\mathbf{Y}$ determines that of $\mathbf{X}$ in a unique way.

A simple example is a *normal (or Gaussian) copula*, which can be specified via a correlation matrix $\mathbf{R}$. To generate $\mathbf{U}$ from the copula, we generate a vector $\mathbf{Z} = (Z_1, \ldots, Z_m)$ from the multinormal distribution with mean $\mathbf{0}$ and covariance matrix $\mathbf{R}$, and we put $U_i = \Phi(Z_i)$ for each i. Each Z_i has the standard normal distribution, and we can obtain a correlation matrix of $\mathbf{U}$ with prespecified elements $r_{i,j}^{(\mathrm{u})}$ by taking a matrix $\mathbf{R}$ with elements $r_{i,j} = 2\sin(r_{i,j}^{(\mathrm{u})}\pi/6)$ (Kruskal, 1958).

A more flexible copula model, still easy to handle, is the *multivariate t copula*, defined as follows. A random vector $\mathbf{Y}$ (in general) has a *multivariate (Student) t* distribution with ν degrees of freedom, mean vector $\boldsymbol{\mu}$, and positive-definite scale matrix $\boldsymbol{\Sigma}$, if its density is given by

$$f(\mathbf{y}) = \frac{\Gamma\left((\nu+m)/2\right)}{(\pi\nu)^{m/2}\,\Gamma\left(\nu/2\right)\sqrt{\det(\Sigma)}}\left(1 + \frac{1}{\nu}\,(\mathbf{y}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)^{-(\nu+m)/2}.$$

We denote this by $\mathbf{Y} = (Y_1, \ldots, Y_m) \sim \mathbf{t}_\nu(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The covariance matrix of $\mathbf{Y}$ is $\frac{\nu}{\nu-2}\Sigma$ for $\nu > 2$.

We now assume that $\mathbf{Y} = (Y_1, \ldots, Y_m) \sim \mathbf{t}_\nu(\mathbf{0}, \Sigma)$ where the diagonal elements $\sigma_{i,i}$ of $\boldsymbol{\Sigma}$ are selected so that

$$\sigma_{i,i}^2 = \sigma_i^2 = \frac{1}{(F^{-1}(q_i;\nu))^2} \quad \text{for } i = 1, \ldots, m\,,$$

where $F^{-1}(\cdot;\nu)$ is the inverse cdf of the univariate Student t distribution with ν degrees of freedom, $t_\nu(0,1)$. With this choice of diagonal elements, it follows that $\mathbb{P}(Y_i > 1) = q_i = 1 - r_i$.

Here we take each F_i as the Student distribution with ν degrees of freedom. We could also take F_i as another distribution (uniform, exponential, etc.). To do this, we could first transform each Y_i to a $U(0,1)$ random variable U_i by applying the appropriate Student cdf, and then transforming U_i to a new random variable Y_i with cdf F_i by applying the inverse cdf: $U_i \leftarrow F(Y_i/\sigma_i;\nu)$ and $Y_i \leftarrow F_i^{-1}(U_i)$. In this case, in GS, $\gamma_0 = -\infty$.

In contrast to the independent case, where the transition density $\kappa_t(\cdot \mid \mathbf{Y}_{j-1})$ is defined via the Gibbs sampler, in our experiments we found it computationally simpler and more efficient to define the transition density $\kappa_t(\cdot \mid \mathbf{Y}_{j-1})$ of the Markov chain via a *hit-and-run* sampling scheme (Smith, 1984; Chen et al., 2000), as follows.

Suppose the current state of the Markov chain is $\mathbf{Y}_{t,j-1}$. The hit-and-run algorithm first generates a random direction of movement in the m-dimensional space by generating a random point $\mathbf{d}$ on the surface of the m-dimensional unit sphere. For this, we generate a vector $\mathbf{Z} = (Z_1, \ldots, Z_m)$ of m independent standard normal random variables, $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)$, and normalize its Euclidean length to 1. Then we generate the length Λ of the proposed move in that direction from the density

$$f_\Lambda(\lambda) = \frac{f(\mathbf{Y}_{t,j-1} + \lambda\,\mathbf{d})}{\int_{-\infty}^{\infty} f(\mathbf{Y}_{t,j-1} + \theta\,\mathbf{d})\,\mathrm{d}\theta}\,, \tag{5}$$

so the proposed new point has a density proportional to f in the given direction. The chain moves to the new proposed state $\mathbf{Y} = \mathbf{Y}_{t,j-1} + \Lambda\,\mathbf{d}$ if and only if $S(\mathbf{Y}) > \gamma_t$, otherwise it remains in the same state. That is,

$$\mathbf{Y}_{t,j} = \begin{cases} \mathbf{Y} & \text{if } S(\mathbf{Y}) > \gamma_t \\ \mathbf{Y}_{t,j-1} & \text{otherwise.} \end{cases} \tag{6}$$

The transition density thus defined is that of a Markov chain whose stationary density is the conditional density f_t in (2). Therefore, Proposition 1 applies and the GS algorithm provides an unbiased estimator. The hit-and-run sampler is summarized in Algorithm 5. In this algorithm, the parameter b is a positive integer that can be taken as 1, but a higher value of b reduces the dependence between the input state $\mathbf{Y}_{t,j-1}$ and the output state $\mathbf{Y}_{t,j}$.

For the case where $\mathbf{Y}$ has a $\mathbf{t}_\nu(\mathbf{0}, \mathbf{\Sigma})$ density, one can show that the density (5) becomes

Algorithm 5 : Transition density $\kappa_t(\mathbf{y}_j \mid \mathbf{y}_{j-1})$ defined via hit-and-run sampler

Require: a vector $\mathbf{Y}_{t,j-1}$ of repair times such that $S(\mathbf{Y}_{t,j-1}) > \gamma_t$ and a positive integer b

$\mathbf{Y}_0 \leftarrow \mathbf{Y}_{t,j-1}$

for $i = 1$ **to** b **do**

generate $\mathbf{d}$ uniformly over the m-dimensional unit sphere:

$$\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I) \quad \text{and} \quad \mathbf{d} = \left(\frac{Z_1}{\|\mathbf{Z}\|}, \ldots, \frac{Z_m}{\|\mathbf{Z}\|} \right)$$

generate a random scale factor Λ from the density (5)

if $S(\mathbf{Y}_{i-1} + \Lambda\, \mathbf{d}) > \gamma_t$ **then**

$\mathbf{Y}_i \leftarrow \mathbf{Y}_{i-1} + \Lambda\, \mathbf{d}$

else

$\mathbf{Y}_i \leftarrow \mathbf{Y}_{i-1}$

return $\mathbf{Y}_{t,j} \leftarrow \mathbf{Y}_b$.

that of the Student distribution (here $\mathbf{y} = \mathbf{Y}_{t,j-1}$)

$$\mathbf{t}_\nu \left(-\frac{\mathbf{y}'\Sigma^{-1}\mathbf{d}}{\mathbf{d}'\Sigma^{-1}\mathbf{d}},\ \frac{\nu + \mathbf{y}'\Sigma^{-1}\mathbf{y}}{\nu(\mathbf{d}'\Sigma^{-1}\mathbf{d})} - \frac{(\mathbf{y}'\Sigma^{-1}\mathbf{d})^2}{\nu(\mathbf{d}'\Sigma^{-1}\mathbf{d})^2} \right) . \tag{7}$$

To see this, note that $f_\Lambda(\lambda)$ in (5) is proportional to

$$\left(1 + \frac{(\mathbf{y} + \lambda\mathbf{d})'\Sigma^{-1}(\mathbf{y} + \lambda\mathbf{d})}{\nu}\right)^{-(\nu+m)/2} = \left(1 + \frac{(\lambda + \frac{\mathbf{y}'\Sigma^{-1}\mathbf{d}}{\mathbf{d}'\Sigma^{-1}\mathbf{d}})^2 - (\frac{\mathbf{y}'\Sigma^{-1}\mathbf{d}}{\mathbf{d}'\Sigma^{-1}\mathbf{d}})^2 + \frac{\mathbf{y}'\Sigma^{-1}\mathbf{y}}{\mathbf{d}'\Sigma\mathbf{d}}}{\nu/(\mathbf{d}'\Sigma^{-1}\mathbf{d})}\right)^{-(\nu+m)/2},$$

which after rearrangement and normalization yields the distribution (7). To speed up the generation of Λ in this case, the Cholesky factor of Σ can be precomputed and memorized once and for all to facilitate the computation of $\Sigma^{-1}\mathbf{d}$ and $\Sigma^{-1}\mathbf{x}_{i-1}$ via backward and forward substitution.

For a purely diagonal covariance matrix $\Sigma = \sigma^2 I$, the distribution of Λ simplifies to

$$\Lambda \sim \mathbf{t}_\nu(-\mathbf{x}'\mathbf{d}, \sigma^2).$$

For a numerical illustration, suppose the unreliability of each edge in the dodecahedron graph of Figure 1 is $q_i = 1 - r_i = 10^{-6}$, and that the links fail in such a way that all repair times Y_i are positively correlated. We take the covariance matrix $\Sigma = \sigma^2\widetilde{\Sigma}$, where $\sigma = -1/F^{-1}(1 - r; \nu)$ and $\widetilde{\Sigma}$ is a correlation matrix whose non-diagonal elements are all equal to 0.5.

We used the GS algorithm with $n = 30000$ and a splitting factor of $s = 2$, for different values of the number of degrees of freedom ν. For the pilot runs we took $n_0 = 10^3$. Table 7

shows the numerical results. It is striking to see how much the unreliability increases when the number of degrees of freedom ν become smaller. For $\nu = 30$, system's failure is no longer a rare event. The last row with $\nu = \infty$ corresponds to a Gaussian copula model and is the most reliable. If the edges were to fail independently (that is, $\tilde{\Sigma} = I$), then the unreliability (in the case with $\nu = \infty$) would be $u \approx 10^{-18}$.

Table 7: The unreliablity of the dodecahedron network for different values of the tail-parameter ν. The last row, with $\nu = \infty$, corresponds to a Gaussian copula model.

ν	$\bar{W}_n$	RE$[\bar{W}_n]$
30	1.21×10^{-2}	0.024
50	7.07×10^{-5}	0.028
100	3.10×10^{-7}	0.030
200	1.55×10^{-8}	0.036
1000	1.23×10^{-9}	0.037
∞	6.74×10^{-10}	0.028

9. Conclusion and Further Work

We have introduced a new rare-event simulation technique for the static reliability estimation of a graph when links are subject to random failures. Our method is an adaptation of a generalized multilevel splitting method applied to a transformation of the static model into a dynamic model for which the definition of intermediate levels is easier and more effective. We have also described and experimented an effective implementation of the method, using an appropriate data structure to manage the connected components efficiently. We compared our GS method with the turnip algorithm, which is recognized in the literature as one of the most efficient for this reliability problem. We found GS to be much more efficient than the turnip in situations where the graph has a large number of links whose unreliabilities are not very small. In those situations, the graph can fail in a large number of ways. The GS method also applies to the important case of graphs with dependent link failures, something barely addressed in the literature.

Our future research activity on this topic includes the development of a method that combines GS with the turnip, and/or with importance sampling, and perhaps with other variance reduction techniques. We also plan to study specific models of dependent link failures, for instance batch failures and cascading failures (Iyer et al., 2009).

Acknowledgments

This work has been supported by an NSERC-Canada Discovery Grant and a Canada Research Chair to the second author, the EuroNF Network of Excellence to the third and fifth author, and INRIA's associated team MOCQUASIN to all authors. We benefited from the computing facilities of the Réseau québécois de calcul haute performance (RQCHP).

References

Ball, M. O., J. S. Provan. 1982. Bounds on the reliability polynomial for shellable independence systems. *SIAM Journal on Algebraic and Discrete Methods* **3** 166–181.

Barlow, R., A. Marshall. 1964. Bounds for distribution with monotone hazard rate, I and II. *Annals of Mathematical Statistics* **35** 1234–1274.

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

Botev, Z. I., D. P. Kroese. 2010. Efficient Monte Carlo simulation via the generalized splitting method. *Statistics and Computing* URL `http://dx.doi.org/10.1007/s11222-010-9201-4`. To appear.

Burtin, Y., B. Pittel. 1972. Asymptotic estimates of the reliability of a complex system. *Engineering Cybernetics* **10** 445–451.

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

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

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

Cancela, H., P. L'Ecuyer, G. Rubino, B. Tuffin. 2010. Combination of conditional Monte Carlo and approximate zero-variance importance sampling for network reliability estimation. B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, E. Yücesan, eds., *Proceedings of the 2010 Winter Simulation Conference*. 1263–1274.

Chen, M.-H., Q. M. Shao, J. G. Ibrahim. 2000. *Monte Carlo Methods in Bayesian Computations*. Springer-Verlag, New York.

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

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

Elperin, T., I. B. Gertsbakh, M. Lomonosov. 1992. An evolution model for Monte Carlo estimation of equilibrium network renewal parameters. *Probability in the Engineering and Informational Sciences* **6** 457–469.

Ermakov, S. M., V. B. Melas. 1995. *Design and Analysis of Simulation Experiments*. Kluwer Academic, Dordrecht, The Netherlands.

Esary, J. D., F. Proschan, D. W. Walkup. 1967. Association of random variables, with applications. *Annals of Mathematical Statistics* **38** 1466–1473.

Fishman, G. S. 1986. A Monte Carlo sampling plan for estimating network reliability. *Operations Research* **34** 581–594.

Garvels, M. J. J., D. P. Kroese, J.-K. C. W. Van Ommeren. 2002. On the importance function in splitting simulation. *European Transactions on Telecommunications* **13** 363–371.

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

Glasserman, P., P. Heidelberger, P. Shahabuddin, T. Zajic. 1999. Multilevel splitting for estimating rare event probabilities. *Operations Research* **47** 585–600.

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

Hui, K.-P., N. Bean, M. Kraetzl, D. Kroese. 2005. The cross-entropy method for network reliability estimation. *Annals of Operations Research* **134** 101–118.

Italiano, G. F. 2009. LEDA extension package: Dynamic graph algorithms. `http://www.mpi-inf.mpg.de/LEDA/friends/dyngraph.html`.

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

Joe, H. 1997. *Multivariate Models and Dependence Concepts*. Chapman and Hall, London.

Kahn, H., T. E. Harris. 1951. Estimation of particle transmission by random sampling. *National Bureau of Standards Applied Mathematical Series* **12** 27–30.

Kruskal, W. 1958. Ordinal measures of association. *Journal of the Americal Statistical Association* **53** 814–861.

L'Ecuyer, P., V. Demers, B. Tuffin. 2006. Splitting for rare-event simulation. *Proceedings of the 2006 Winter Simulation Conference*. IEEE Press, 137–148.

L'Ecuyer, P., V. Demers, B. Tuffin. 2007. Rare-events, splitting, and quasi-Monte Carlo. *ACM Transactions on Modeling and Computer Simulation* **17** Article 9.

L'Ecuyer, P., F. LeGland, P. Lezaud, B. Tuffin. 2009. Splitting techniques. G. Rubino, B. Tuffin, eds., *Rare Event Simulation Using Monte Carlo Methods*. Wiley, 39–62. Chapter 3.

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

L'Ecuyer, P., B. Tuffin. 2008. Approximate zero-variance simulation. *Proceedings of the 2008 Winter Simulation Conference*. IEEE Press, 170–181.

Lomonosov, M. 1974. Bernoulli scheme with closure. *Problems of Information Transmission (USSR)* **10** 73–81.

Lomonosov, M. 1994. On Monte Carlo estimates in network reliability. *Probability in the Engineering and Informational Sciences* **8** 245–264.

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

Manzi, E., M. Labbé, G. Latouche, F. Maffioli. 2001. Fishman's sampling plan for computing network reliability. *IEEE Transactions on Reliability* **50** 41–46.

Nelsen, R. B. 1999. *An Introduction to Copulas*, *Lecture Notes in Statistics*, vol. 139. Springer-Verlag, New York, NY.

Neuts, M. F. 1981. *Matrix-Geometric Solutions in Stochastic Models*. John Hopkins, University Press, Baltimore.

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

Rubinstein, R., D. P. Kroese. 2007. *Simulation and the Monte Carlo Method*. 2nd ed. John Wiley & Sons, New York.

Smith, R. L. 1984. Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research* **32** 1296–1308.

Zaroliagis, C. 2002. Implementations and experimental studies of dynamic graph algorithms. R. Fleischer, B. M. E. Moret, E. M. Schmidt, eds., *Experimental Algorithms*, *Lecture Notes in Computer Science*, vol. 2547, chap. 11. Springer, Berlin, Heidelberg, 229–278.

Zenklusen, R., M. Laumanns. 2011. High-confidence estimation of small s-t reliabilities in acyclic networks. *Networks* **57** 376–388.