Reliability Estimation for Networks with Minimal Flow Demand and Random Link Capacities

Zdravko Botev, Pierre l'Ecuyer, Bruno Tuffin

Reliability Estimation for Networks with Minimal Flow Demand and Random Link Capacities

Zdravko I. Botev, The University of New South Wales, Sydney, Australia
Pierre L'Ecuyer, Université de Montréal, Canada, and Inria–Rennes, France
Bruno Tuffin, INRIA Rennes—Bretagne Atlantique, France

Abstract: We consider a network whose links have random capacities and in which a certain target amount of flow must be carried from some source nodes to some destination nodes. Each destination node has a fixed demand that must be satisfied and each source node has a given supply. We want to estimate the unreliability of the network, defined as the probability that the network cannot carry the required amount of flow to meet the demand at all destination nodes. When this unreliability is very small, which is our main interest in this paper, standard Monte Carlo estimators become useless because failure to meet the demand is a rare event. We propose and compare two different methods to handle this situation, one based on a conditional Monte Carlo approach and the other based on generalized splitting. We find that the first is more effective when the network is highly reliable and not too large, whereas for a larger network and/or moderate reliability, the second is more effective.

Keywords: network reliability, stochastic flow network, Conditional Monte Carlo, permutation Monte Carlo, generalized splitting

1 Introduction

Network reliability estimation problems are commonplace in various application areas such as transportation, communication, and power distribution systems; see for example [18]. In many of those problems, the states of certain network components are subject to uncertainty and there is a set of conditions under which the network is operational, and one wishes to estimate the network *unreliability*, defined as the probability u that the network is in a failed state (i.e., is not operational). When u is very small, a standard (crude) Monte Carlo (MC) approach that merely generates the component states, computes the indicator function that the network is operational or not, and averages over n independent runs to estimate u, is unsatisfactory because the relative error (defined as the standard deviation of the estimator divided by the expected value u) of the MC estimator goes to infinity when $u \to 0$.

One reliability problem that has received a lot of attention is the *static network reliability* estimation problem, in which each link of the network is failed with a given probability and the network is operational when a given (specific) subset of the nodes are all connected. Effective estimation methods have been developed for this problem when u is small; see [3, 5, 14, 15, 18, 25] and the references therein. The model considered in this paper is more general. Instead of having only a binary state (up or down), each link has a random capacity that can take many possible values, there is a fixed demand that must be satisfied at certain nodes (called the destination nodes), a fixed supply is available at some other nodes (the source nodes), and the network is operational when it can carry the flow to satisfy all the demands. As a special case, there can be a single source node and a single destination node, with a fixed demand, and the network is operational when the maximum flow that can be sent from the source to the destination reaches the demand. We will describe our methods in this particular setting to simplify the notation, but the methods apply to the general setting as well. The case of links with binary states is a special case. The several methods developed for this special case do not readily apply to the network flow setting considered here, but we show how two of the best available methods for the binary case, *permutation Monte Carlo* (PMC) and *generalized splitting* (GS), can be adapted to this problem. The adaptation is not straightforward.

The PMC method [14, 18, 25] constructs an artificial continuous-time Markov chain (CTMC) defined as follows. Each capacity is assumed to have a discrete distribution over a finite set of possible values. This can approximate a continuous distribution if needed. We assume that all links start at their minimal capacity, and the capacity of one link may increase each time the CTMC has a jump. The CTMC is constructed so that the probability that the network is failed at time 1 is equal to u. PMC generates the discrete-time Markov chain (DTMC) underlying the CTMC, i.e., only the sequence of states that are visited until the network is operational, and conditional on that sequence it computes the probability that the network is failed at time 1, as an estimator of u. This conditional probability can be computed by exploiting the property that the failure time has a phase-type conditional distribution, whose cumulative distribution function (cdf) and density can be expressed in terms of matrix exponentials. We show how to adapt and apply the PMC principle to our problem. The CTMC construction is quite different than for the binary case. We also prove, under certain conditions, that the resulting PMC estimator has bounded relative error (BRE) when $u \to 0$ for a given network.

GS [2] is a rare-event estimation method where the rare event is the intersection of a nested sequence of events and its probability is the product of conditional probabilities. Each conditional probability is estimated thanks to resampling strategies, making the overall estimation more accurate than a direct estimation of the rare-event probability itself. The application of GS to this problem was discussed in [6] for the situation in which the capacities have a continuous distribution, and experimental results were reported for a small example. But the GS algorithm proposed there does not work in general when the capacities have a discrete distribution. We show however that GS can be applied in the discrete case if we combine it with the same CTMC construction as for PMC. The GS algorithm does not have BRE in the asymptotic regime when $u \to 0$, but it becomes more efficient that PMC when the size of the network increases. The relative error typically increases (empirically) as $\mathscr{O}(-\log u)$.

The remainder of the paper is organized as follows. In Section 2, we formulate the network flow model considered in this paper. In Section 3, we construct a CTMC which permits one to apply PMC to this model, for the case where each capacity is distributed over a finite set. In Section 4, we explain how to apply GS to this model. We report numerical experiments in Section 5. Our experimental results agree with the fact that PMC has BRE when $u \to 0$, under appropriate conditions. It can accurately estimate extremely small values of u when the network is not too large. When the network gets larger and u is not too small, on the other hand, GS becomes more effective than PMC.

2 The model

Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be a graph with a set of nodes $\mathscr{V}$ and a set of links $\mathscr{E}$ with cardinality $m = |\mathscr{E}|$. For $i = 1, \ldots, m$, link i has a random integer-valued flow capacity X_i with discrete marginal distribution $p_i(x) = \mathbb{P}[X_i = x]$ over the set

$$\mathscr{X}_i = \{c_{i,0}, \ldots, c_{i,b_i}\}, \qquad 0 \le c_{i,0} < c_{i,1} < \cdots < c_{i,b_i} < \infty.$$

This is a standard assumption; see [6] and the references therein. Thus, the random network state $\mathbf{X} = (X_1, \ldots, X_m)$ belongs to the space $\mathscr{X} = \prod_{i=1}^m \mathscr{X}_i$ and has joint pdf $p(\mathbf{x}) = \mathbb{P}(\mathbf{X} = \mathbf{x})$, for $\mathbf{x} \in \mathscr{X}$. We also make the standard independence assumption (see [1, 7, 12]) that $p(\mathbf{x}) = \prod_{i=1}^m \mathbb{P}[X_i = x_i]$ and that the nodes do not fail.

To keep the notation and the exposition simple, in the remainder of the paper we describe the model and the methods under the assumption that there is a single source and a single destination. The generalization to multiple sources and destinations is straightforward, as explained below. The fixed demand level at the destination is $d_{\text{net}} > 0$ and the maximum flow that can be carried from the source to the destination is a random variable $\Psi(\mathbf{X})$, which is a function of the link capacities. The well-known max-flow min-cut theorem says that the maximum value of a flow from a source to a destination is equal to the minimum capacity of a cut in the network. Efficient algorithms are available to compute $\Psi(\mathbf{X})$; for example the Ford-Fulkerson algorithm.

We are interested in estimating the unreliability of the flow network, defined here as

$$u = \mathbb{P}[\Psi(\mathbf{X}) < d_{\text{net}}] = \sum_{\{\mathbf{x} \in \mathscr{X} : \Psi(\mathbf{x}) < d_{\text{net}}\}} p(\mathbf{x});$$

that is, the probability that the maximum flow $\Psi(\mathbf{X})$ fails to meet the demand. This problem was considered in [16], for example. In the particular case where $\mathscr{X}_i = \{0, 1\}$ for each i and $d_{\text{net}} = 1$, we have an instance of the static network reliability problem mentioned in the introduction, with the source and destination as the selected set of nodes to be connected. To generalize to multiple sources and destinations, we would assume a fixed demand d_i at each destination node i, a fixed supply s_i at each source node i, and the event $\{\Psi(\mathbf{X}) < d_{\text{net}}\}$ would be replaced by the event that the network does not have sufficient capacity to send flow to satisfy all the demands from the available supplies.

For small networks, it is possible to compute and store most of the minimal cutsets or pathsets and use them to obtain exact or approximate values for u; see [23, 31] for example. But for large networks, no polynomial-time algorithm is known for computing u exactly [13], and one must rely on approximations or on estimation via Monte Carlo. Of particular interest is the situation in which the network is highly reliable, i.e., u is a very small rare-event probability, because crude Monte Carlo then becomes ineffective.

Several Monte Carlo variance-reduction methods have been proposed for network reliability estimation in rare-event situations; see, e.g., [3, 5, 9, 14, 18, 27, 24, 30] and the references given there. Most of these methods are for the special case of independent links with binary states and nodes that never fail. Some have been extended to links with three possible states [17, 19, 20], but this remains restrictive. We now describe how two of the most efficient methods, PMC and GS, can be adapted to our model.

3 Reformulating the model as a CTMC and applying PMC

We now show how to construct an artificial CTMC for this static model, which will permit us to apply PMC as described in the introduction. This CTMC construction differs from that used in [5, 18].

3.1 Constructing the CTMC

For each i, let $\mathbb{P}(X_i = c_{i,k}) = p_i(c_{i,k}) = r_{i,k} > 0$ for $k = 0,\dots,b_i$. Define independent exponential random variables $Y_{i,1},\dots,Y_{i,b_i}$ with rates $\lambda_{i,1},\dots,\lambda_{i,b_i}$, respectively, where the $\lambda_{i,k}$ still have to be chosen. Suppose that the capacity of link i is $c_{i,0}$ from time $T_{i,0} = 0$ to time $T_{i,1} = \min(Y_{i,1},\dots,Y_{i,b_i})$ (exclusive), after that it is $c_{i,1}$ from time $T_{i,1}$ to time $T_{i,2} = \min(Y_{i,2},\dots,Y_{i,b_i})$, it is $c_{i,2}$ from time $T_{i,2}$ to time $T_{i,3} = \min(Y_{i,3},\dots,Y_{i,b_i})$, and so on, and finally it is c_{i,b_i} from time T_{i,b_i} to $T_{i,b_i+1} = \infty$. Under this process, the capacity of link i at time $\gamma \geq 0$ is given by

$$\begin{aligned} X_i(\gamma) &= c_{i,k} \quad \text{for } T_{i,k} \leq \gamma < T_{i,k+1} \text{ and } 0 \leq k \leq b_i & (1)\\ &= \max_k \{c_{i,k} : T_{i,k} \leq \gamma\}. & (2)\end{aligned}$$

The times $T_{i,1},\dots,T_{i,b_i}$ are not necessarily all distinct; often, many of them are equal, so that the number of jumps at which the capacity changes can be much smaller than b_i. For example, if $T_{i,1} = Y_{i,b_i}$, then we have $T_{i,1} = T_{i,2} = \cdots = T_{i,b_i}$. As another example, if $b_i = 3$ and $Y_{i,2} < Y_{i,1} < Y_{i,3}$, then $0 < T_{i,1} = T_{i,2} < T_{i,3}$ and the capacity of link i jumps from $c_{i,0}$ to $c_{i,2}$ at time $T_{i,2} = Y_{i,2}$ and jumps again from $c_{i,2}$ to $c_{i,3}$ at time $T_{i,3} = Y_{i,3}$. In general, the process $\{X_i(\gamma),\ \gamma \geq 0\}$ has an upward jump at each of the *distinct* jump times $T_{i,k}$.

To show that this process is a CTMC, suppose that we are at time $\gamma \geq 0$ and $X_i(\gamma) = c_{i,k}$. Then we know that $Y_{i,k} \leq \gamma$ and that $Y_{i,\ell} > \gamma$ for all $\ell > k$. The $Y_{i,\ell}$ for $\ell < k$ can be anything, but they have no influence on the process trajectory after time γ. This means that the current state $X_i(\gamma)$ contains all the relevant information that needs to be known at time γ to generate the future of the process.

The capacity $X_i(\gamma)$ of link i at time $\gamma \geq 0$ satisfies

$$\mathbb{P}[X_i(\gamma) \leq c_{i,k}] = \mathbb{P}[\min(Y_{i,k+1},\dots,Y_{i,b_i}) > \gamma] = \exp[-\gamma(\lambda_{i,k+1} + \cdots + \lambda_{i,b_i})].$$

If we select the $\lambda_{i,k}$'s so that the last expression equals $r_{i,0} + \cdots + r_{i,k}$ for each k when $\gamma = 1$, then $X_i(1)$ has the exact same distribution as X_i, the capacity of link i in the original static model. This is equivalent to having

$$\lambda_{i,k+1} + \cdots + \lambda_{i,b_i} = -\ln(r_{i,0} + \cdots + r_{i,k}).$$

To achieve this, it suffices to put

$$\lambda_{i,b_i} \;=\; -\ln(r_{i,0} + \cdots + r_{i,b_i-1}) \;=\; -\ln(1 - r_{i,b_i}) \tag{3}$$

and then for $k = b_i - 1, \ldots, 1$ (in descending succession):

$$\lambda_{i,k} \;=\; -\ln(r_{i,0} + \cdots + r_{i,k-1}) - \lambda_{i,k+1} - \cdots - \lambda_{i,b_i}. \tag{4}$$

Note that (4) can be rewritten as $\lambda_{i,k} = -\ln(r_{i,0} + \cdots + r_{i,k-1}) + \ln(r_{i,0} + \cdots + r_{i,k})$, which can never be negative. We have proved the following.

Proposition 1. *If we select $\lambda_{i,b_i}, \lambda_{i,b_i-1}, \ldots, \lambda_{i,1}$ according to (3) and (4) and the process $X_i(\cdot)$ as in (2), then $\lambda_{i,k} \geq 0$ for each k, $\{X_i(\gamma), \gamma \geq 0\}$ is a CTMC process, and $X_i(1)$ has exactly the same discrete distribution as the capacity of link i in the original model: $\mathbb{P}[X_i(1) = c_{i,k}] = r_{i,k}$ for $k = 0, \ldots, b_i$. As a result, $\mathbf{X}(1) = (X_1(1), \ldots, X_m(1))$ has the same distribution as $\mathbf{X}$ and one has $u = \mathbb{P}[\Psi(\mathbf{X}(1)) < d_{\text{net}}]$.*

3.2 Applying PMC

Under the assumption that all links are independent, a simple way of applying PMC to this model is as follows. Generate all the $Y_{i,k}$'s independently with their rates $\lambda_{i,k}$, put them in a large vector

$$\mathbf{Y} = (Y_{1,1}, \ldots, Y_{1,b_1}, \ldots, Y_{m,1}, \ldots, Y_{m,b_m})$$

of size $\kappa = b_1 + \cdots + b_m$, and sort this vector in increasing order to obtain

$$Y_{\pi(1)} \leq Y_{\pi(2)} \leq \cdots \leq Y_{\pi(\kappa)},$$

where $\pi(j) = (i,k)$ if $Y_{i,k}$ is in position j in the sorted vector, so that $\pi = (\pi(1), \ldots, \pi(\kappa))$ can be seen as the permutation of the pairs (i,k) that corresponds to the sort. This permutation gives an ordering of the κ pairs (i,k). When scanning those pairs in the given order, each pair (i,k) corresponds to a potential capacity increase for link i. The capacity increases if and only if no pair (i,k') for $k' > k$ has occurred before. Conditional on π, one can add those pairs in the given order and update the capacities accordingly, until the maximum flow in the network reaches d_{net}. Suppose this occurs when adding the pair $(i,k) = \pi(C)$ for some integer $C > 0$. Let $T_C = Y_{\pi(C)}$. The (unbiased) conditional (PMC) estimator of u is then

$$\mathbb{P}[\Psi(\mathbf{X}(1)) < d_{\text{net}} \mid \pi] = \mathbb{P}[T_C > 1 \mid \pi] = \mathbb{P}[T_C > 1 \mid \pi(1), \ldots, \pi(C)] = \mathbb{P}[A_1 + \cdots + A_C > 1],$$

where $A_1 = Y_{\pi(1)}$ is an exponential random variable with rate $\Lambda_1 = \sum_{i=1}^{m} \sum_{k=1}^{b_i} \lambda_{i,k}$, each $A_j = Y_{\pi(j)} - Y_{\pi(j-1)}$ is an exponential random variable with rate $\Lambda_j = \Lambda_{j-1} - \lambda_{\pi(j-1)}$ for $j = 2, \ldots, C$, and these A_j's are independent. Given π and C, $T_C = A_1 + \cdots + A_C$ is the sum of C independent exponential random variables with rates $\Lambda_1, \ldots, \Lambda_C$, which has a phase-type distribution, whose complementary cdf is given by

$$1 - F(\gamma \mid \pi) = \mathbb{P}[T_C > \gamma \mid \pi] = \mathbf{e}_1^{\mathsf{t}} \exp(\mathbf{Q}\gamma)\mathbf{1} \tag{5}$$

where $\mathbf{e}_1^{\mathsf{t}} = (1, 0, \ldots, 0)$, $\mathbf{1} = (1, \ldots, 1)^{\mathsf{t}}$ (the $^{\mathsf{t}}$ means "transposed"), and

$$\mathbf{Q} = \begin{pmatrix} -\Lambda_1 & \Lambda_1 & 0 & \cdots & 0 \\ 0 & -\Lambda_2 & \Lambda_2 & 0 & \vdots \\ 0 & 0 & \ddots & \ddots & 0 \\ \vdots & 0 & 0 & -\Lambda_{C-1} & \Lambda_{C-1} \\ 0 & \cdots & 0 & 0 & -\Lambda_C \end{pmatrix}.$$

Reliable and fast computation of (5) is discussed in [3, 5].

To compute the critical number C at which the flow reaches the demand, we must be able to update efficiently the maximum flow in the network each time we increase the capacity of one link. We do this as explained in Section 4 of [6]. We refer to this algorithm as the *incremental maximum flow algorithm*.

To estimate u by PMC, for a fixed threshold d_{net}, we simulate n independent realizations $W_1, \ldots, W_n$ of

$$W = W(\pi) = \mathbb{P}[T_C > 1 \mid \pi] \tag{6}$$

and take the average $\bar{W}_n = (1/n)\sum_{i=1}^{n} W_i$. Compared with the crude Monte Carlo estimator that would take the indicator $I = \mathbb{I}[\Psi(\mathbf{X}(1)) < d_{\text{net}}]$ in place of W, it is always true that $\text{Var}[W] < \text{Var}[I]$, because $W = \mathbb{E}[I \mid \pi]$.

The estimators discussed so far are for a single (fixed) demand d_{net}. With PMC, it is also possible to estimate $u = u(d_{\text{net}})$ as a function of the demand d_{net}, over some interval, using the same simulations for all demands. To do this, for any given permutation π, we can compute $C = C(d_{\text{net}})$ as a function of the demand over the interval of interest. This would be a step function, often with just a few jumps. Then we compute W for all values of C that are visited over this interval. This provides an estimator $W(d_{\text{net}})$ of $u(d_{\text{net}})$ as a function of d_{net}. By averaging the n realizations of this estimator, we obtain a functional estimator of $u(d_{\text{net}})$ over the interval of interest.

3.3 Improved PMC

The PMC strategy described earlier can be improved by removing some useless jumps. First, whenever $c_{i,k} \geq d_{\text{net}}$ for $k < b_i$, we can immediately remove all the jumps $(i, k+1), \ldots, (i, b_i)$, because when the capacity of a link has reached d_{net}, it is useless to increase it further. Capacity levels larger than d_{net} can in fact be all reset to d_{net} right away in the model, the probability of d_{net} in the new model being taken as the the probability of values larger than d_{net} in the initial model. For simplicity, when the demand is fixed, we assume in our algorithm that this has been done already, so that $c_{i,b_i} \leq d_{\text{net}}$ for all i, and then there is no need to remove those useless capacity levels.

Second, the jump times $Y_{i,k}$ that do not change the capacity of link i can also be removed. That is, whenever $\pi(j) = (i,k)$ and the capacity of link i has already reached a value $c_{i,k'} > c_{i,k}$, i.e., $(i,k') = \pi(j')$ for some $j' < j$, then there is no need to consider the pair (i,k) when it is encountered in the permutation, so we can remove the corresponding jump.

Let $\tilde{\pi}$ be the permutation obtained after removing all those pairs (i,k) from the sorted vector, and $\tilde{C}$ the corresponding value of C in this reduced permutation. As soon as the max flow reaches d_{net}, we have found $\tilde{C}$. When we encounter $\tilde{\pi}(j) = (i,k)$ and the previous capacity of link i was $c_{i,k'} < c_{i,k}$, the capacity of link i jumps to $c_{i,k}$ and we must decrease Λ_j by $\tilde{\lambda}_{i,k} = \lambda_{i,k'+1} + \cdots + \lambda_{i,k}$, because the jumps that correspond to $(i,k'+1),\ldots,(i,k)$ can now be removed from consideration.

Algorithm 1 : PMC algorithm for multi-state flow network

```
 1: for $i = 1$ to $m$ do
 2:     draw $Y_{i,1},\ldots,Y_{i,b_i}$ with the appropriate rates $\lambda_{i,k}$
 3:     let $\mathscr{L}_i = \{(i,1),\ldots,(i,b_i)\}$
 4:     min $\leftarrow (i,b_i)$ and $\tilde{\lambda}_{i,b_i} \leftarrow \lambda_{i,b_i}$
 5:     for $k = b_i - 1$ to $1$ do
 6:         if $Y_{i,k} > Y_{\text{min}}$ then
 7:             remove $(i,k)$ from the list $\mathscr{L}_i$
 8:             $\tilde{\lambda}_{\text{min}} \leftarrow \tilde{\lambda}_{\text{min}} + \lambda_{i,k}$
 9:             $S_{i,k} \leftarrow 0$     // this jump is deactivated
10:         else
11:             min $\leftarrow (i,k)$     // this pair $(i,k)$ is retained
12:             $\tilde{\lambda}_{\text{min}} \leftarrow \lambda_{i,k}$
13:             $S_{i,k} \leftarrow 1$     // this jump is activated
14: merge the sorted lists $\mathscr{L}_1,\ldots,\mathscr{L}_m$ into a single list sorted by increasing order, $Y_{\tilde{\pi}(1)},\ldots,Y_{\tilde{\pi}(\tilde{\kappa})}$
15: $\Lambda_1 \leftarrow \lambda_{1,1} + \cdots + \lambda_{1,b_1} + \cdots + \lambda_{m,b_m}$
16: $j \leftarrow 0$
17: $\mathbf{X} \leftarrow (c_{1,0},\ldots,c_{m,0})$
18: while maximum flow $\Psi(\mathbf{X}) < d_{\text{net}}$ do
19:     $j \leftarrow j + 1$
20:     if $S_{\tilde{\pi}(j)} = 1$ then
21:         $(i,k) \leftarrow \tilde{\pi}(j)$     // this jump has not been removed or executed
22:         $\Lambda_{j+1} \leftarrow \Lambda_j - \tilde{\lambda}_{i,k}$
23:         $S_{i,k} \leftarrow 0$
24:         $X_i \leftarrow c_{i,k}$     // increase capacity of $i$-th link
25:         Filter()     // do nothing (default), or FilterSingle, FilterAll, etc.
26: $\tilde{C} \leftarrow j$     // the critical jump number
27: return $W \leftarrow \mathbb{P}\left[A_1 + \cdots + A_{k-1} > 1 \mid \tilde{\pi}, \tilde{C}\right]$.
```

Algorithm 1 describes this reduced version of PMC in a more formal way. It returns one realization of W. Indentation delimits the scope of the loops. In the first **for** loop, for each link i, the algorithm generates the exponential random variables $Y_{i,k}$ and then immediately eliminates those that correspond to (useless) jump times at which the capacity of the link does not change. This preliminary filtering is very easy and efficient to apply and may eliminate a significant fraction of the jumps, especially for links that have many capacity levels. The remaining jumps are sorted in a single list (for all links) and each one receives a Boolean tag $S_{\tilde{\pi}(j)}$, initialized to 1, which means that this jump is currently scheduled to occur.

Then these jumps are "executed" in chronological order, by increasing the corresponding capacities, until the critical jump number $\tilde{C}$ is found. After that, W can be computed. The Boolean variables $S_{\tilde{\pi}(j)}$ are used in the optional Filter() subroutine, which can be used to try to eliminate further useless jumps after a jump is executed and the corresponding capacity is increased (this is discussed in Section 3.4). Algorithm 1 would be invoked n times, independently, and u would be estimated by the average of the n realizations of W.

Other variants of the algorithm can be considered and some might be more efficient, but this is not completely clear. For example, instead of generating all the variables $Y_{i,k}$ at the beginning, one may think of generating the permutation π directly without generating those $Y_{i,k}$, as was done in [18] for the binary case. This appears complicated and we did not implement it.

3.4 Removing jumps having no impact on maximum flow

In Algorithm 1, in the case where Filter() does nothing, all pairs (i,k) for which the capacity of link i increases are retained in $\tilde{\pi}$ and the corresponding jumps are executed. But it sometimes occurs that increasing the capacity of link i to $c_{i,k}$ (or more) is useless because it can no longer have an impact on the event that the maximum flow exceeds the demand or not. In this case, one can cancel (deactivate) all the future jumps related to the capacity of link i. In our implementation, these future jumps are canceled by setting their Boolean variables $S_{i,k}$ to 0. Increasing the capacity of link i is useless in particular if it is already possible to send d_{net} units of flow between the two nodes connected by link i. This obviously happens if the capacity of link i is already of d_{net}, which is trivial to verify, but under our assumption (made at the beginning of Section 3.3) that a link has no capacity level above d_{net}, this cannot happen, and our algorithm ignores this possibility.

Increasing the capacity of link i is also useless when d_{net} units of flow can be sent in total, either directly on link i or indirectly via other links. This is generally harder (more costly) to verify. To detect it, one can run a max-flow algorithm to compute how much flow can be sent between these two nodes. This can be done each time the capacity of a link is increased.

Algorithm 2 does this only for the link i whose capacity has just been increased, at each step j. Since the link i generally changes at every step j, we have a different max-flow problem (for a different pair of nodes) at each step. For this reason, in our implementation we recompute the max-flow from scratch at each step j. Of course, this brings significant overhead.

Algorithm 3 is even more ambitious: it computes the max-flow between nodes for all pairs of nodes. Then for each link i for which the current max-flow between the corresponding two nodes meets the demand, it cancels all future jumps associated with that link. Doing this at each step j might be too costly, so in our implementation the user selects a positive integer ν and does it only at every ν steps, i.e., when j is a multiple of ν. We compute the max-flow for all pairs of nodes using the algorithm of [22], which is a simplified variant of the Gomory-Hu method [21]. This algorithm computes the all-pairs max-flow by applying $|\mathscr{V}|-1$ times a max-flow algorithm for one pair of nodes, which is generally more efficient than applying a max-flow algorithm for each link. Algorithm 3 also recomputes the maximum flows from scratch each time it is called, rather than reusing computations from the previous time and just updating the max flows. (In fact, the $|\mathscr{V}|-1$ pairs of nodes for which the max-flow is computed in the algorithm change from one call to the next, and we are not aware of an effective incremental algorithm that would reuse and just update the previous computations.)

Algorithm 2 : FilterSingle

```
$f \leftarrow$ compute maximum flow between terminal nodes of link $i$
if $f \geq d_{\text{net}}$ then
    for $k = 1$ to $b_i$ do
        if $S_{i,k} = 1$ then
            $S_{i,k} \leftarrow 0$
            $\Lambda_{j+1} \leftarrow \Lambda_{j+1} - \tilde{\lambda}_{i,k}$
```

Algorithm 3 : FilterAll

```
if $j$ mod $\nu = 0$ then
    $\{F_{v,w}\} \leftarrow$ max flow between all pairs of nodes $(v,w)$, computed via Gusfield's algorithm
    for all $i = (v,w) \in \mathscr{E}$ do
        if $F_{v,w} \geq d_{\text{net}}$ then
            for $k = 1$ to $b_i$ do
                if $S_{i,k} = 1$ then
                    $S_{i,k} \leftarrow 0$
                    $\Lambda_{j+1} \leftarrow \Lambda_{j+1} - \tilde{\lambda}_{i,k}$
```

3.5 Bounded relative error for PMC

For a single run, the crude MC estimator $I = \mathbb{I}[\Psi(\mathbf{X}(1)) < d_{\text{net}}]$ of u, which is a Bernoulli random variable with mean u, has variance $u(1-u)$, so its relative error (RE) is $\text{RE}[I] = \sqrt{u(1-u)}/u \approx u^{-1/2} \to \infty$ when $u \to 0$. With n runs, the variance is divided by n and the RE by $\sqrt{n}$. When u is very small, we may need an excessively large n to obtain a sufficiently small RE. With PMC, the RE is sometimes much better behaved than with MC. In this section, we obtain conditions under which the PMC estimator W has bounded relative error (BRE), i.e., $\text{RE}[W]$ remains bounded when $u \to 0$. The proofs have some similarity with those in [5].

Suppose the probabilities $r_{i,k} = r_{i,k}(\varepsilon)$ in our model depend on some parameter ε in a way that $u = u(\varepsilon) \to 0$ when $\varepsilon \to 0$. In what follows, the quantities in the model are assumed implicitly to depend on ε. A non-negative quantity that may depend on ε is $\mathscr{O}(1)$ if it remains bounded when $\varepsilon \to 0$. It is $\Theta(1)$ if it is bounded and also bounded away from 0, when $\varepsilon \to 0$.

In our setting, the vector $\mathbf{Y}$ and the permutation π have finite length κ and C is bounded by κ. The number of possible permutations is therefore finite. Let $p(\pi)$ be the probability of permutation π.

Proposition 2. *(i) If $p(\pi) = \Theta(1)$ for all π, then the PMC estimator has BRE.*
(ii) This holds in particular if $\lambda_{i,k_i}/\lambda_{j,k_j} = \Theta(1)$ for all i, j, k_i, k_j (we then say that the rates are balanced).

Proof. (i) Note that $u \geq \mathbb{P}[T_C > 1 \mid \pi] p(\pi) = W(\pi)p(\pi)$ for any π. If $p(\pi) = \Theta(1)$, then $W(\pi)/u = \mathscr{O}(1)$ and $\max_\pi W(\pi)/u = \mathscr{O}(1)$. Therefore $\mathbb{E}[W^2/u^2] = \mathscr{O}(1)$, which implies BRE.

(ii) Note that $p(\pi) = \prod_{j=1}^{\kappa} \lambda_{\pi(j)}/\Lambda_{\pi(j)}$. Under the given assumption, $\lambda_{\pi(j)}/\Lambda_{\pi(j)} = \Theta(1)$ for all j, which implies that $p(\pi) = \Theta(1)$ for all π. $\square$

As a concrete illustration of an asymptotic regime in which the $r_{i,k}$ depend on ε, we define a regime similar to one that has been widely used for highly reliable Markovian systems [26, 28, 29]. Suppose that link i is operating at capacity $c_{i,k}$. with probability

$$r_{i,k} = a_{i,k}\varepsilon^{d_{i,k}}$$

for some constants $a_{i,k} > 0$ and $d_{i,k} > 0$ independent of ε, for all $k \in \{0, \ldots, b_i - 1\}$ (that is, not at full capacity). This implies that $r_{i,b_i} = 1 - \sum_{j=0}^{b_i-1} r_{i,k} = \Theta(1)$ for all i. That is, the event that any link is not at full capacity is a rare event. This implies that failure to meet the demand is a rare event, and therefore $\mathrm{RE}[I] \to \infty$ when $\varepsilon \to 0$. More specifically, any state vector $\mathbf{x} = (c_{1,k_1}, \ldots, c_{m,k_m})$ for which $\Psi(\mathbf{x}) < d_{\mathrm{net}}$ has probability $\mathbb{P}(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{m} r_{i,k_i} = \varepsilon^{d(\mathbf{x})}(1 + o(1))$ for some $d(\mathbf{x}) > 0$. Let $d_{\min} = \min\{d(\mathbf{x}) : \Psi(\mathbf{x}) < d_{\mathrm{net}}\}$. Then

$$u = \sum_{\{\mathbf{x}:\Psi(\mathbf{x})<d_{\mathrm{net}}\}} \mathbb{P}(\mathbf{X} = \mathbf{x}) = \Theta(\varepsilon^{d_{\min}}).$$

On the other hand, we have

Proposition 3. *In the setting just defined, the PMC estimator has BRE.*

Proof. Recall that for $1 \leq i \leq m$, $\lambda_{i,b_i} = -\ln(\sum_{k=0}^{b_i-1} r_{i,k}) = \Theta(\ln(\varepsilon))$. Moreover, for all $k < b_i$, $\lambda_{i,k} = \ln(\sum_{\ell=0}^{k} r_{i,\ell}) - \ln(\sum_{\ell=0}^{k-1} r_{i,\ell}) = \Theta(\ln(\varepsilon))$. The conditions of Proposition 2 (ii) are then verified, hence the result. $\square$

The results of this section apply to the improved PMC variants as well; the proofs are easily adapted.

4 A generalized splitting algorithm

Botev et al. [6] have explained how to adapt the GS algorithm proposed and studied in [2, 3] to the stochastic flow problem considered here, but for the situation where the capacities have a continuous distribution. The aim of the algorithm is to obtain a sample of realizations of $\mathbf{X}$ which is approximately a sample from the distribution of $\mathbf{X}$ conditional on $\Psi(\mathbf{X}) < d_{\text{net}}$. The estimator is then given by the realized sample size (which is random) divided by its largest possible value. The algorithm uses intermediate demand levels $d_{\text{net}} = d_\tau < \cdots < d_1 < d_0$, where d_0 is the maximal possible flow, achieved when each link i is at its maximal capacity c_{i,b_i}. These levels and their number τ are fixed a priori and chosen so that $\mathbb{P}[\Psi(\mathbf{X}) < d_t \mid \Psi(\mathbf{X}) < d_{t-1}] \approx 1/s$ for $t = 1, \ldots, \tau - 1$, and at most $1/s$ for $t = \tau$, where s is a small integer also fixed (usually and in all our experiments in this paper, $s = 2$). The levels are estimated by pilot runs, as explained in [2, 3]. The algorithm starts by sampling $\mathbf{X}$ from its original distribution. If $\Psi(\mathbf{X}) < d_1$, it resamples each coordinate of $\mathbf{X}$ conditional on $\Psi(\mathbf{X}) < d_1$, via Gibbs sampling, repeats this s times, and keeps the states $\mathbf{X}$ for which $\Psi(\mathbf{X}) < d_2$ (their number is in $\{0, \ldots, s\}$). At each level $t = 3, \ldots, \tau$, this type of resampling is applied to each state that has been retained at the previous step (for which $\Psi(\mathbf{X}) < d_{t-1}$), by resampling that state twice from its distribution conditional on $\Psi(\mathbf{X}) < d_{t-1}$, and retaining the states for which $\Psi(\mathbf{X}) < d_t$. At the last level, we count the number N of chains for which $\Psi(\mathbf{X}) < d_\tau$, and return $W = N/s^{\tau-1}$ as an estimator of u. This is repeated n times independently, to produce n independent realizations of W, say $W_1, \ldots, W_n$, whose average $\bar{W}_n$ is an unbiased estimator of u. This estimator *does not* have BRE, because the RE increases with the number of levels; the RE is typically (roughly) proportional to $-\log u$ (see [4] for a proof of this result in an idealized setting). It can also handle large networks.

In general, this GS algorithm is not directly applicable when the capacities have discrete distributions, because then $\Psi(\mathbf{X})$ also has a discrete distribution and it may happen that this distribution is too coarse (e.g., all the probability mass is on just a few possible values). Then it may be impossible to select levels d_t for which $\mathbb{P}[\Psi(\mathbf{X}) < d_t \mid \Psi(\mathbf{X}) < d_{t-1}] \approx 1/s$ for $t = 1, \ldots, \tau - 1$.

It is nevertheless possible to apply GS in that case by constructing the vector $\mathbf{Y}$ as for PMC in the previous section, and resampling this vector instead of $\mathbf{X}$. Recall that $\mathbf{Y}$ is a vector of κ independent exponential random variables. The GS algorithm will operate similarly as the one described above, except that now the levels $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_\tau = 1$ are on T_C, and the resampling at each step t is for $\mathbf{Y}$ and is conditional on $T_C > \gamma_{t-1}$. This is valid because $\{\Psi(\mathbf{X}(\gamma_t)) < d_{\text{net}}\} = \{T_C > \gamma_t\}$. The corresponding GS procedure operates in the same way as the GS algorithm with anti-shocks in [5]. We first generate $\mathbf{Y}$ from its original distribution. Then at each level t, we take each state (realization or modification of $\mathbf{Y}$) that has been retained at the previous step (for which $T_C > \gamma_{t-1}$), we resample all its coordinates s times (i.e., for s Gibbs sampling steps, where each step starts from the result of the previous step) from its distribution conditional on $T_C > \gamma_{t-1}$, to obtain two new states, and we retain the states for which $T_C > \gamma_t$. At the last level, we count the number N of chains for which $T_C > \gamma_\tau = 1$, and return $W = N/s^{\tau-1}$ as an estimator of u. The resampling of $\mathbf{Y}$ conditional on $T_C > \gamma_{t-1}$ via Gibbs sampling can be done in a similar way as in [3, 5]. We first select a permutation π of the κ coordinates of the vector $\mathbf{Y}$. Then for $j = 1, \ldots, \kappa$, we resample $Y_{\pi(j)}$ as follows: If $\pi(j) = (i,k)$, the current capacity $X_i(\gamma_{t-1})$ of link i is less than

$c_{i,k}$ (or equivalently $\min(Y_{i,k},\ldots,Y_{i,b_i}) > \gamma_{t-1}$), and by changing the current capacity of link i to $c_{i,k}$ (or equivalently changing $Y_{i,k}$ to 0) we would have $T_C < \gamma_{t-1}$ (the maximum flow would meet the demand), then we resample $Y_{i,k}$ from its exponential density truncated to (γ_{t-1},∞). Otherwise we resample $Y_{i,k}$ from its original exponential density. To sample from the truncated density, it suffices to generate $Y_{i,k}$ from the original density and add γ_{t-1}.

5 Numerical Examples

In this section, we provide some numerical examples that compare the PMC and GS algorithms, and show how they behave when $u \to 0$. In these examples, we parameterize the models by ε in a way that $u = u(\varepsilon) \to 0$ when $\varepsilon \to 0$, exactly as in Section 3.5, in the asymptotic regime when the probability that links are not operating at full capacity is getting close to zero. For all variants of PMC, we used formula (2) in [5] with high precision arithmetic to compute (5).

5.1 Experimental setting

We used the same experimental protocol as in [5], comparing four methods. Method PMC refers to Algorithm 1 without any filtering step. PMC-Single and PMC-All refer to PMC combined with the filtering as in Algorithms 2 and 3 respectively. Method GS refers to generalized splitting, implemented as described in Section 4. The splitting levels were determined via the adaptive Algorithm 3 of [3], with $n_0 = 500$ and $s = 2$. The levels were estimated using a single run of the adapative algorithm, and these same levels were used for every independent replication of the GS algorithm.

For each example and method, we report the unreliability estimate $\bar{W}_n$, its empirical *relative error* $\mathrm{RE}[\bar{W}_n] = S_n/(\sqrt{n}\bar{W}_n)$ where S_n^2 is the empirical variance, and the *work-normalized relative variance* (WNRV) of $\bar{W}_n$, defined as $\mathrm{WNRV}[\bar{W}_n] = T \times \mathrm{RE}^2[\bar{W}_n]$, where T is the total CPU time (in seconds) for the n runs of the algorithm. One must keep in mind that T and the WNRV depend on the software and hardware used for the computations. The experiments were run on Intel Xeon E5-2680 CPUs, on a linux cluster. The sample size for every algorithm was $n = 5 \times 10^4$.

For each example we use the following model. Each link i has the capacity levels $\{0,1,\ldots,b_i\}$, i.e., $c_{i,k} = k$ for $k = 0,\ldots,b_i$. We take $r_{i,k} = \mathbb{P}(X_i = c_{i,k}) = \rho^{b_i-k-1}\varepsilon$ for $k < b_i$ and $r_{i,b_i} = 1 - \sum_{k=0}^{b_i-1}\rho^{b_i-k-1}\varepsilon$, where ρ, ε and $\{b_i\}$ are model parameters.

5.2 A 4×4 lattice graph

Our first example uses the 4×4 lattice graph, which has 16 nodes and 24 links. The flow has to be sent from one corner to the opposite corner. We take $b_i = 8$, $\rho = 0.6$ and $d_{\mathrm{net}} = 10$, and let ε range from 10^{-4} to 10^{-13}.

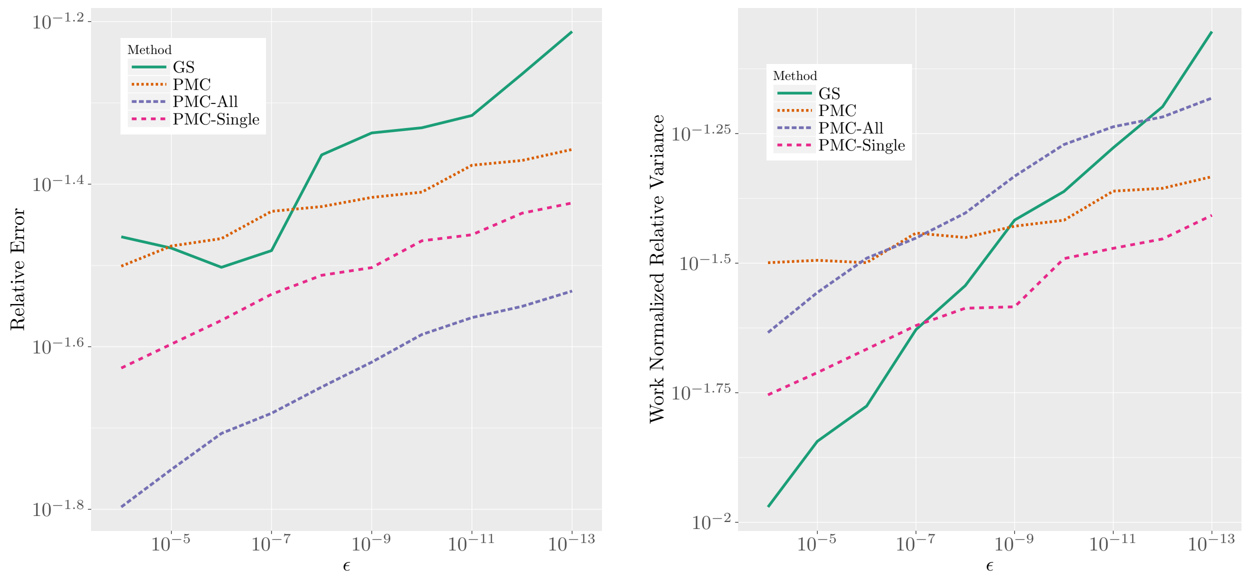


Figure 1: RE (left) and WNRV (right) for four methods, for the 4×4 lattice graph.

Table 1 reports the values of $\bar{W}_n$, RE$[\bar{W}_n]$ and WNRV$[\bar{W}_n]$, for methods GS and PMC, for some values of ε. Figure 1 shows plots of RE$[\bar{W}_n]$ and WNRV$[\bar{W}_n]$ for all four methods. We see that for this small example, PMC-All always has the smallest RE, followed by PMC-Single. These REs increase very slowly when ε decreases, for the values we have tried. They should eventually stabilize when $\varepsilon \to 0$. In terms of work-normalized relative variance, i.e., when taking the computing time into account, GS is the most effective method when ε is not too small, but when ε gets smaller, GS requires a larger simulation effort while the effort required by PMC variants remains approximately stable, so these PMC methods eventually catch up, in agreement with our asymptotic results. Among them, PMC-Single has the smallest WNRV.

	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-5}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-8}$
$\bar{W}_n$ for PMC	2.99×10^{-5}	2.98×10^{-6}	2.99×10^{-7}	2.99×10^{-8}	2.99×10^{-9}
RE$[\bar{W}_n]$ for PMC	3.16×10^{-2}	3.34×10^{-2}	3.41×10^{-2}	3.69×10^{-2}	3.74×10^{-2}
WNRV$[\bar{W}_n]$ for PMC	3.17×10^{-2}	3.20×10^{-2}	3.17×10^{-2}	3.62×10^{-2}	3.54×10^{-2}
$\bar{W}_n$ for GS	2.98×10^{-5}	2.99×10^{-6}	2.99×10^{-7}	2.98×10^{-8}	2.99×10^{-9}
RE$[\bar{W}_n]$ for GS	3.43×10^{-2}	3.32×10^{-2}	3.15×10^{-2}	3.30×10^{-2}	4.33×10^{-2}
WNRV$[\bar{W}_n]$ for GS	1.07×10^{-2}	1.43×10^{-2}	1.68×10^{-2}	2.35×10^{-2}	2.86×10^{-2}

Table 1: Estimation of u, RE, and WNRV for some values of ε, for the 4×4 lattice example

5.3 6×6 lattice graph

Figure 2 shows plots of RE$[\bar{W}_n]$ and WNRV$[\bar{W}_n]$ for a 6×6 lattice graph, with 36 nodes and 60 links. Again, the flow has to be sent from one corner to the opposite corner. Here, for all values of

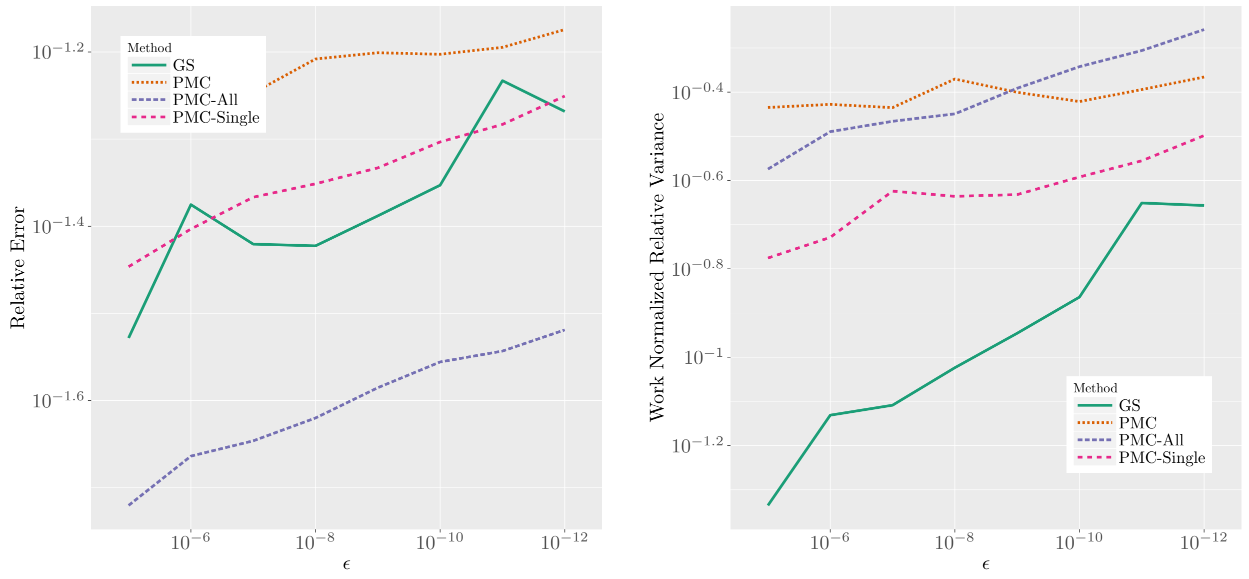


Figure 2: RE (left) and WNRV (right) for four methods, for the 6×6 lattice graph.

ε considered, PMC-All has the smallest RE while GS wins in terms of WNRV.

5.4 A dodecahedron network

In this example we use the well-known dodecahedron network (Figure 3), with 20 nodes and 30 links, often used as a standard benchmark in network reliability estimation [3, 8, 10, 11, 30]. Here we took $\rho = 0.7$, $b_i = 4$ and $d_{\text{net}} = 5$. Note that when ε is very small, most of the failures will occur because there is not enough capacity in the three links connected to node 1 (links 1, 2, 3), or not enough capacity in the three links connected to node 20 (links 28, 29, 30). These are the two bottleneck cuts.

Table 2 reports the values of $\bar{W}_n$, RE$[\bar{W}_n]$, and WNRV$[\bar{W}_n]$, for the GS and PMC methods, for different values of ε. We see that the estimates $\bar{W}_n$ agree very well across the two methods. Figure 4 shows RE$[\bar{W}_n]$ and WNRV$[\bar{W}_n]$ as functions of ε, for all four methods. We see that PMC-All has by far the smallest RE for all ε, and it also wins in terms of WNRV, except for $\varepsilon > 10^{-5}$ where GS wins. The latter case is approximately when $u \geq 7 \times 10^{-11}$, which is already pretty small. When ε decreases, the WNRV increases for GS in part because the RE increases, but also because the computing time increases. The figure shows what happens when ε gets very small.

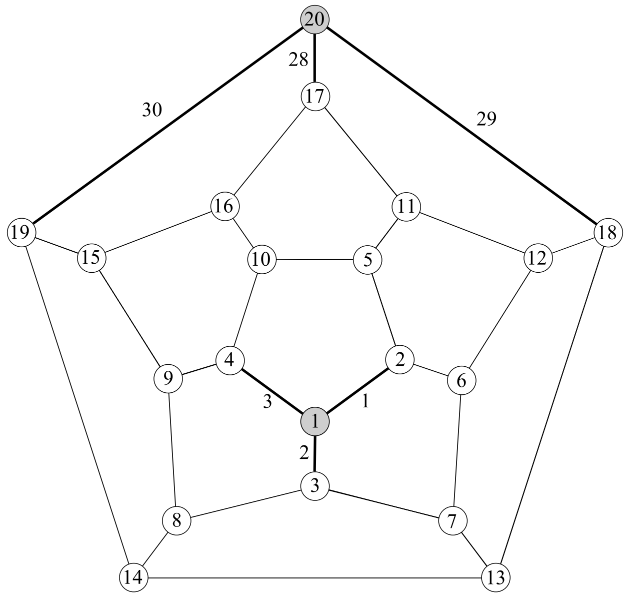
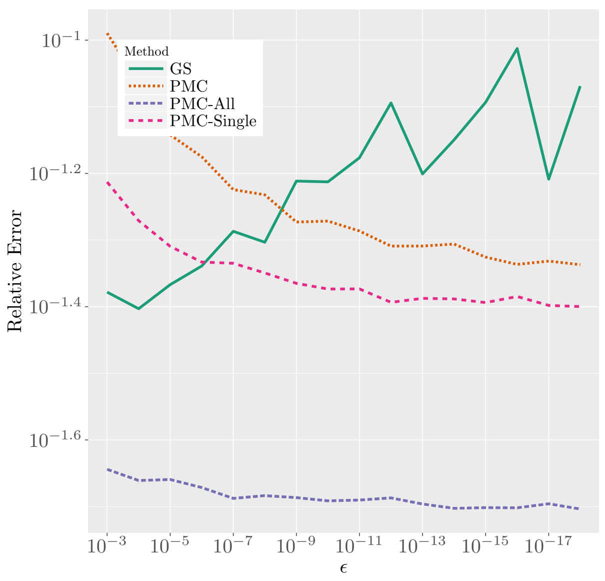


Figure 3: A dodecahedron graph with 20 nodes and 30 links (figure taken from [3]).



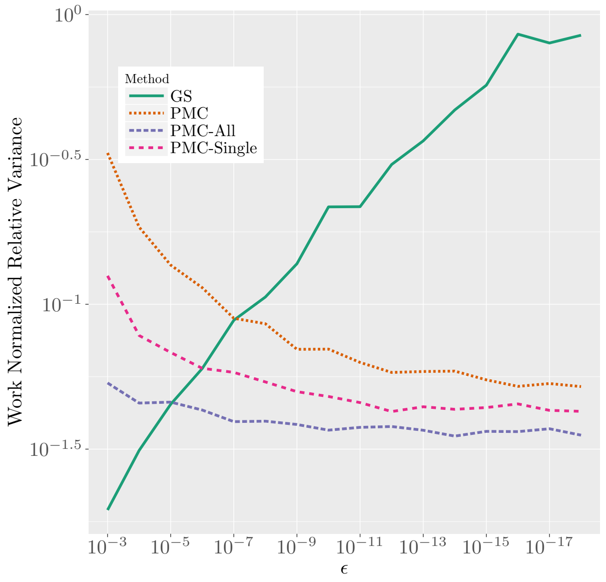


Figure 4: RE (left) and WNRV (right) for four methods, for the dodecahedron example

	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-5}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-8}$
$\bar{W}_n$ for PMC	7.08×10^{-9}	7.08×10^{-11}	7.06×10^{-13}	7.06×10^{-15}	7.05×10^{-17}
RE[$\bar{W}_n$] for PMC	8.63×10^{-2}	7.21×10^{-2}	6.68×10^{-2}	5.97×10^{-2}	5.86×10^{-2}
WNRV[$\bar{W}_n$] for PMC	1.85×10^{-1}	1.37×10^{-1}	1.14×10^{-1}	8.95×10^{-2}	8.57×10^{-2}
$\bar{W}_n$ for GS	7.07×10^{-9}	7.06×10^{-11}	7.07×10^{-13}	7.07×10^{-15}	7.05×10^{-17}
RE[$\bar{W}_n$] for GS	3.95×10^{-2}	4.30×10^{-2}	4.58×10^{-2}	5.17×10^{-2}	4.97×10^{-2}
WNRV[$\bar{W}_n$] for GS	3.13×10^{-2}	4.52×10^{-2}	6.00×10^{-2}	8.81×10^{-2}	1.06×10^{-1}

Table 2: Estimation of u, RE, and WNRV for some values of ε, for the dodecahedron example

Acknowledgments

This work has been supported by the Australian Research Council under DE140100993 Grant to Z. I. Botev, an NSERC-Canada Discovery Grant, a Canada Research Chair, and an Inria International Chair to P. L'Ecuyer. P. L'Ecuyer acknowledges the support of the Faculty of Science Visiting Researcher Award at UNSW. We are grateful to Rohan Shah, who performed the numerical experiments.

References

[1] C. Alexopoulos and G.S. Fishman, Capacity expansion in stochastic flow networks, Probability in Eng Informational Sciences 6 (1992), 99–118.

[2] Z.I. Botev and D.P. Kroese, Efficient Monte Carlo simulation via the generalized splitting method, Stat Comput 22 (2012), 1–16.

[3] Z.I. Botev, P. L'Ecuyer, G. Rubino, R. Simard, and B. Tuffin, Static network reliability estimation via generalized splitting, INFORMS J Comput 25 (2013), 56–71.

[4] Z.I. Botev, P. L'Ecuyer, and B. Tuffin, Dependent failures in highly-reliable static networks, Proc 2012 Winter Simulation Conference, IEEE Press, 2012, pp. 430–441.

[5] Z.I. Botev, P. L'Ecuyer, and B. Tuffin, Static network reliability estimation under the Marshall-Olkin copula, ACM Trans Modeling Comput Simulation 26 (2016), Article 14.

[6] Z.I. Botev, S. Vaisman, R.Y. Rubinstein, and P. L'Ecuyer, Reliability of stochastic flow networks with continuous link capacities, Proc 2014 Winter Simulation Conference, IEEE Press, 2014, pp. 543–552.

[7] S. Bulteau and M.E. Khadiri, A new importance sampling Monte Carlo method for a flow network reliability problem, Naval Res Logist 49 (2002), 204–228.

[8] H. Cancela and M. El Khadiri, A recursive variance-reduction algorithm for estimating communication-network reliability, IEEE Trans Reliab 44 (1995), 595–602.

[9] H. Cancela and M. El Khadiri, On the RVR simulation algorithm for network reliability evaluation, IEEE Trans Reliab 52 (2003), 207–212.

[10] H. Cancela, M. El Khadiri, and G. Rubino, “Rare event analysis by Monte Carlo techniques in static models," Rare event simulation using Monte Carlo methods, G. Rubino and B. Tuffin (Editors), Wiley, 2009, pp. 145–170, Chapter 7.

[11] H. Cancela, P. L'Ecuyer, M. Lee, G. Rubino, and B. Tuffin, “Analysis and improvements of path-based methods for Monte Carlo reliability evaluation of static models," Simulation methods for reliability and availability of complex systems, J. Faulin, A.A. Juan, S. Martorell, and E. Ramirez-Marquez (Editors), Springer Verlag, 2009, pp. 65–84.

[12] Y.C. Chou and P.T. Lin, An efficient and robust design optimisation of multi-state flow network for multiple commodities using generalised reliability evaluation algorithm and edge reduction method, Int J Syst Sci ahead-of-print (2014), 1–14.

[13] C.J. Colbourn, The combinatorics of network reliability, Oxford University Press, New York, 1987.

[14] T. Elperin, I.B. Gertsbakh, and M. Lomonosov, Estimation of network reliability using graph evolution models, IEEE Trans Reliab 40 (1991), 572–581.

[15] G.S. Fishman, A Monte Carlo sampling plan for estimating network reliability, Oper Res 34 (1986), 581–594.

[16] G.S. Fishman, Monte Carlo: Concepts, algorithms, and applications, Springer Series in Operations Research, Springer-Verlag, New York, NY, 1996.

[17] I.B. Gertsbakh, R. Rubinstein, Y. Shpungin, and R. Vaisman, Permutational methods for performance analysis of stochastic flow networks, Probability in Eng Informational Sciences 28 (2014), 21–38.

[18] I.B. Gertsbakh and Y. Shpungin, Models of network reliability, CRC Press, Boca Raton, FL, 2010.

[19] I.B. Gertsbakh, Y. Shpungin, and R. Vaisman, Network reliability Monte Carlo with nodes subject to failure, Int J Performability Eng 10 (2014), 163–172.

[20] I.B. Gertsbakh, Y. Shpungin, and R. Vaisman, Ternary networks: Reliability and Monte Carlo, Springer, 2014.

[21] R.E. Gomory and Z.C. Hu, Multi-terminal network flows, SIAM J Appl Math 9 (1961), 551–570.

[22] D. Gusfield, Very simple methods for all pairs network flow analysis, SIAM J Computi 19 (1990), 143–155.

[23] C.C. Jane and Y.W. Laih, Computing multi-state two-terminal reliability through critical arc states that interrupt demand, IEEE Trans Reliab 59 (2010), 338–345.

[24] P. L'Ecuyer, G. Rubino, S. Saggadi, and B. Tuffin, Approximate zero-variance importance sampling for static network reliability estimation, IEEE Trans Reliabili 8 (2011), 590–604.

[25] M. Lomonosov and Y. Shpungin, Combinatorics and reliability Monte Carlo, Random Structures Algorithms 14 (1999), 329–343.

[26] M.K. Nakayama, General conditions for bounded relative error in simulations of highly reliable Markovian systems, Advances in Appl Probability 28 (1996), 687–727.

[27] J.E. Ramirez-Marquez and D.W. Coit, A Monte-Carlo simulation approach for approximating multi-state two-terminal reliability, Reliab Eng & System Saf 87 (2005), 253–264.

[28] G. Rubino and B. Tuffin, "Markovian models for dependability analysis," Rare event simulation using Monte Carlo methods, G. Rubino and B. Tuffin (Editors), Wiley, 2009, pp. 125–144, Chapter 6.

[29] P. Shahabuddin, Importance sampling for the simulation of highly reliable Markovian systems, Manage Sci 40 (1994), 333–352.

[30] B. Tuffin, S. Saggadi, and P. L'Ecuyer, An adaptive zero-variance importance sampling approximation for static network dependability evaluation, Comput Oper Res 45 (May 2014), 51–59.

[31] M.J. Zuo, Z. Tian, and H.Z. Huang, An efficient method for reliability evaluation of multistate networks given all minimal path vectors, IIE transactions 39 (2007), 811–817.