Splitting Sequential Monte Carlo for Efficient Unreliability Estimation of Highly Reliable Networks

Radislav Vaisman[a], Dirk P. Kroese[a], Ilya B. Gertsbakh[b]

[a] *School of Mathematics and Physics, The University of Queensland, Australia*
[b] *Department of Mathematics, Ben-Gurion University , Beer-Sheva, 84105, Israel*

Abstract

Assessing the reliability of complex technological systems such as communication networks, transportation grids, and bridge networks is a difficult task. From a mathematical point of view, the problem of estimating network reliability belongs to the #P complexity class. As a consequence, no analytical solution for solving this problem in a reasonable time is known to exist and one has to rely on different approximation techniques. In this paper we focus on a well-known Sequential Monte Carlo algorithm — Lomonosov's turnip method. Despite the fact that this method was shown to be efficient under some mild conditions, it is known to be inadequate for stable estimation of network reliability in a rare-event setting. To overcome this obstacle, we suggest a quite general combination of Sequential Monte Carlo and multilevel splitting. The proposed method is shown to bring a significant variance reduction as compared to the turnip algorithm, is easy to implement and parallelize, and has a proven performance guarantee for some network typologies.

Keywords: Terminal Network Reliability, Permutation Monte Carlo, Multilevel Splitting, Rare Events

Email addresses: r.vaisman@uq.edu.au (Radislav Vaisman), kroese@maths.uq.edu.au (Dirk P. Kroese), elyager@bezeqint.net (Ilya B. Gertsbakh)

1. Introduction

Nowadays it is hard to underestimate the importance of networks in our life and, as a consequence, a natural question of their reliability arises [1, 2, 3, 4, 5]. We adopt the following definition of the network reliability problem [6]. Let $G = G(V, E, W)$ be an undirected graph, where V and E are the vertex and edge sets, respectively, and $W \subseteq V$ is a set of "terminal" nodes. We assume that the vertices never fail, but that the edges are subject to failure. In particular, every $e \in E$ has a corresponding failure probability $0 \leqslant q_e \leqslant 1$. An edge can be in an *up* or *down* state with probabilities p_e and $q_e = 1 - p_e$, respectively. Under the above framework we wish to assess the network unreliability, defined as the probability that the terminal set W is disconnected [7].

The exact solution to this network reliability problem is hard to obtain in reasonable computation time, since this problem belongs to the #P complexity class [8]. This complexity class, introduced by Valiant [9], consists of the set of counting problems that are associated with a decision problem in NP (non–deterministic polynomial time). For example, #SAT is the problem of counting the number of feasible solutions to a satisfiability formula (SAT).

For some #P-complete problems there are known efficient approximations. For example, Karp and Lubby [10] introduced a fully polynomial randomized approximation scheme (FPRAS) for counting the solutions of disjunctive normal form satisfiability formulas. To the best of our knowledge, there exists no FPRAS for estimating general terminal network reliability. That is, for this particular problem, an efficient randomized approximation algorithm is not known to exist. However, various approximation techniques were proposed [7, 11, 12, 13, 14, 15]. For more recent advances, see the work of Shafieezadeh and Ellingwood [11], the multilevel splitting algorithms of Botev et al. [16, 17] and Walter [18], and the SIS method of L'Ecuyer et al. [19].

In this paper we focus on Lomonosov's turnip (LT) algorithm [13]. This method is an improvement of the Permutation Monte Carlo (PMC) scheme which was shown to be efficient under some mild conditions. We give a brief introduction to PMC and LT in Section 2. Despite the fact that PMC and LT are designed to deal with quite hard network instances, it was shown in [16] that these methods can be very inefficient in a rare-event setting. To overcome the rare-event complication, Botev et al. [16] formulated the network reliability problem as a static rare-event probability estimation problem

and employed the Generalized Splitting Algorithm (GS) [6, Chapter 14].

The multilevel splitting framework was first used by Kahn and Harris [20] to estimate rare-event probabilities. The main idea is to partition the state space in such a way that the problem becomes one of estimating conditional probabilities that are not rare. The GS algorithm of Botev and Kroese [21] generalizes this to a method able to evaluate a wide range of rare-event estimation problems. For a survey of the general methodology we refer to [22, Chapter 4] and [23, 24].

Inspired by successful implementation of Botev et al. [16], we put the LT method into a Sequential Monte Carlo (SMC) framework combined with multilevel splitting [21, 23, 24]. The resulting algorithm introduces a significant variance reduction as compared to the basic LT method and has a proven performance guarantee for some networks. Namely, we prove that our method is an FPRAS for special families of graphs. See Section 3 for details.

The rest the paper is organized as follows. In Section 2 we give a brief introduction to the PMC and LT algorithms and show a simple family of networks for which LT's performance is inefficient. In Section 3 we put LT into a quite general SMC framework combined with multilevel splitting. We show that the resulting algorithm can be used to deliver highly reliable estimators and provide an explanation for its efficiency. In Section 4 we present various numerical examples to demonstrate the advantage of the proposed method. Finally, in Section 5 we summarize our findings and discuss possible directions for future research.

2. Permutation Monte Carlo

Below we describe the PMC algorithm of Michael Lomonosov, also called the network evolution process. This method was designed to estimate the reliability of networks with independent components having different failure probabilities. For detailed explanations, see [13] and [14, Chapter 9]. In this paper we assume that the vertices are absolutely reliable but the edges are subject to failure.

Recall that our setting is as follows. We have a network $G = G(V, E, W)$ where V is the node set, E is the edge set, and $W \subseteq V$ is the terminal set. The edges states are binary, that is, edge e can be in the *up* or *down* state with probabilities p_e and $q_e = 1 - p_e$, respectively. The network *UP* state is defined as the presence of connectivity of all terminal nodes.

The basic idea of PMC is to associate with each edge $e \in E$ an exponentially distributed random "birth time" $\mathcal{T}(e)$ with parameter $\lambda(e)$ having the following property for an arbitrary chosen time value τ:

$$\mathbb{P}(\mathcal{T}(e) \leqslant \tau) = 1 - \mathrm{e}^{-\lambda(e)\tau} = p_e, \quad e \in E. \tag{1}$$

Let us assume that all the edges are in the *down* state at time zero. Then, an edge e is born at time $\mathcal{T}(e)$, that is, at the time $\mathcal{T}(e)$ it enters the *up* state and stays there "forever". The probability that e will be "alive" at time τ is $\mathbb{P}\left(\mathcal{T}(e) \leqslant \tau\right) = p_e$. The value of τ can be arbitrary, so for simplicity we put $\tau = 1$ and note that from (1) it follows that $\lambda(e) = -\ln q_e$.

We proceed with the following crucial observation. If we take a "snapshot" of the state of all edges at time instant $\tau = 1$, we will see the network in the state which is stochastically equivalent to the static picture in which edge e is *up* or *down* with probability p_e or q_e, respectively.

Suppose that $|E| = n$ and consider the ordering (permutation) of the edges $\boldsymbol{\pi} = (e_1, \ldots, e_n)$, according to their birth times sorted in increasing order. Since the birth times are exponentially distributed, it holds that

$$\mathbb{P}(\boldsymbol{\Pi} = \boldsymbol{\pi}) = \prod_{t=1}^{n} \frac{\lambda(e_t)}{\Lambda(E_{t-1})}, \tag{2}$$

where $E_t = E \setminus \{e_1, \ldots, e_t\}$ for $1 \leqslant t \leqslant n-1$, and $\Lambda(E_t) = \sum_{e \in E_t} \lambda(e)$ [13, 25].

The first index $1 \leqslant \mathsf{a}(\boldsymbol{\pi}) \leqslant n$ of the edge permutation $\boldsymbol{\pi}$ is called an anchor of $\boldsymbol{\pi}$, if the sub-graph of G defined by $G(V, (e_1, \ldots, e_{\mathsf{a}(\boldsymbol{\pi})}), W)$ is in the UP state; that is,

$$\mathsf{a}(\boldsymbol{\pi}) = \min\left\{t \ : \ G\left(V, (e_1, \ldots, e_t), W\right) \text{ is } UP\right\}.$$

Let $\xi_1 + \cdots + \xi_t$ be the birth time of edge e_t in $\boldsymbol{\pi}$ for $1 \leqslant t \leqslant n$. Then, given the edge permutation $\boldsymbol{\Pi} = \boldsymbol{\pi}$, the probability that the network is in the UP state is given by

$$\mathbb{P}\left(\sum_{t=1}^{\mathsf{a}(\boldsymbol{\pi})} \xi_t \leqslant 1 \,\middle|\, \boldsymbol{\Pi} = \boldsymbol{\pi}\right) = \mathrm{Conv}_{1 \leqslant t \leqslant \mathsf{a}(\boldsymbol{\pi})}\left\{1 - \mathrm{e}^{-\Lambda(E_t)}\right\},$$

where Conv stands for exponential convolution. The latter can be computed in closed form using the Hypoexponential cumulative distribution function

(CDF). In particular, if $\xi_1, \ldots, \xi_m$ are independent and exponentially distributed random variables with parameters $\Lambda_1, \ldots, \Lambda_m$, then the (complementary) CDF of their sum can be obtained via a matrix exponential

$$\mathbb{P}\left(\sum_{t=1}^{m} \xi_i > \tau\right) = \mathbf{e}_1 \mathrm{e}^{D\tau}\mathbf{1} = \mathbf{e}_1 \sum_{k=0}^{\infty} \frac{D^k \tau^k}{k!}\mathbf{1}, \tag{3}$$

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

$$D = \begin{pmatrix} -\Lambda_1 & \Lambda_1 & 0 & \cdots & 0 \\ 0 & -\Lambda_2 & \Lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -\Lambda_{m-1} & \Lambda_{m-1} \\ 0 & \cdots & 0 & 0 & \Lambda_m \end{pmatrix}$$

is an $m \times m$ matrix, see [6, 26] and [14, Appendix B]. Moreover, the network *DOWN* and *UP* probabilities denoted by $\bar{r}$ and r, respectively, can be expressed as

$$\bar{r} = \sum_{\boldsymbol{\pi}} \mathbb{P}(\boldsymbol{\Pi} = \boldsymbol{\pi}) \cdot \mathbb{P}\left(\sum_{t=1}^{\mathsf{a}(\boldsymbol{\pi})} \xi_t > 1 \,\middle|\, \boldsymbol{\Pi} = \boldsymbol{\pi}\right), \tag{4}$$

and

$$r = \sum_{\boldsymbol{\pi}} \mathbb{P}(\boldsymbol{\Pi} = \boldsymbol{\pi}) \cdot \mathbb{P}\left(\sum_{t=1}^{\mathsf{a}(\boldsymbol{\pi})} \xi_t \leqslant 1 \,\middle|\, \boldsymbol{\Pi} = \boldsymbol{\pi}\right),$$

respectively, where the summation is over all permutations $\boldsymbol{\pi}$.

Since the network unreliability and reliability in (4) is expressed as an expectation, it can be estimated without bias as the sample average of conditional probabilities, $\mathbb{P}(\xi_1 + \xi_2 + ... + \xi_{\mathsf{a}(\boldsymbol{\Pi})} > 1 \mid \boldsymbol{\Pi})$ over an independent sample of trajectories $\{\boldsymbol{\Pi}^{(1)}, \boldsymbol{\Pi}^{(2)}, ..., \boldsymbol{\Pi}^{(N)}\}$. This procedure is summarized in Algorithm 2.1.

Algorithm 2.1 (PMC Algorithm For Unreliability Estimation). *Given a network $G = G(V, E, W)$, edge failure probabilities (q_e, $e \in E$), and sample size N, execute the following steps.*

1. ***(Initialization)*** *Set $S \leftarrow 0$. For each edge $e \in E$, set $\lambda(e) \leftarrow -\ln(q_e)$ and $k \leftarrow 0$.*

2. ***(Permutation Generation)*** *Set* $k \leftarrow k+1$ *and sample* $\mathbf{\Pi}^{(k)} = \left(e_1^{(k)}, \ldots, e_n^{(k)}\right)$ *using* (2).

3. ***(Find the Anchor)*** *Calculate*

$$\mathsf{a}\left(\mathbf{\Pi}^{(k)}\right) = \min\left\{t \,:\, G\left(V, \left(e_1^{(k)}, \ldots, e_t^{(k)}\right), W\right) \text{ is } UP\right\}.$$

4. ***(Calculation of Convolution)*** *Use* (3) *to calculate*

$$R^{(k)} \leftarrow 1 - \mathrm{Conv}_{1 \leqslant t \leqslant \mathsf{a}\left(\mathbf{\Pi}^{(k)}\right)}\left\{1 - e^{-\Lambda\left(E_t^{(k)}\right)}\right\},$$

where $E_t^{(k)} = E \setminus \left\{e_1^{(k)}, \ldots, e_t^{(k)}\right\}$ *for* $1 \leqslant t \leqslant \mathbf{\Pi}^{(k)}$, *and* $\Lambda\left(E_t^{(k)}\right) = \sum_{e \in E_t^{(k)}} \lambda(e)$, *and set* $S \leftarrow S + R^{(k)}$.

5. ***(Stopping Condition)*** *If* $k = N$, *return* S/N *as unbiased estimator of* $\bar{r}$; *otherwise, go to Step* 2.

Let us consider an instructive example of the edges' births as a process developing in time.

Example 2.1 (Network with 5 nodes and 7 edges). Figure 1 presents a network with 5 nodes and 7 edges. We define this network *UP* state when all the nodes are connected to each other.

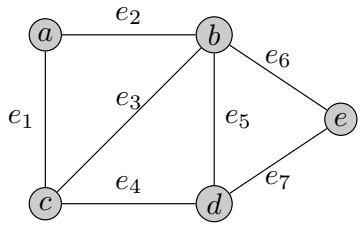
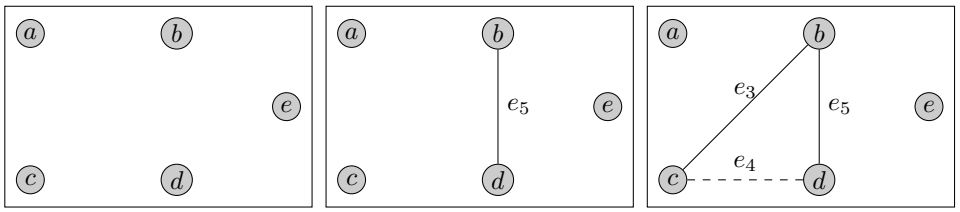


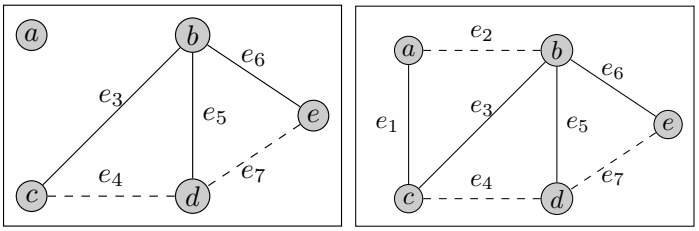
Figure 1: Network with 5 nodes and 7 edges.

The birth process starts at time $t = 0$ from state denoted as σ_0; note that no edges are born yet. Suppose that edges are born in the following order, $\boldsymbol{\pi} = (e_5, e_3, e_6, e_1, e_4, e_2, e_7)$. After the births of e_5, e_3, e_6, and e_1, the network enters the state σ_1, σ_2, σ_3, and σ_4, respectively, see Figure 2. Note that after the birth of e_1, the network enters the *UP* state; that is, $\mathsf{a}(\boldsymbol{\pi}) = 4$.

We are ready now to state the main idea of Lomonosov, namely, Algorithm 2.1 can be complemented by a so-called closure (merging) operation.



(a) The σ_0 state. (b) The σ_1 state. (c) The σ_2 state.



(d) The σ_3 state. (e) The σ_4 (UP) state.

Figure 2: The evolution process.

2.1. Turnip: Exploiting the Closure Operation

The closure of a subset $E' \subseteq E$ consists of E' and all edges of G whose vertices lie in the same component of the spanning subgraph $G(V, E')$ [13]. A subset E' is closed if it coincides with its closure. The closure operation is essentially an elimination of edges that do not change the already born connected component during the evolution process.

Let us explain the concept of closure by Example 2.1. Consider the state σ_2 on Figure 2. Edges e_3 and e_5 have already been born. These edges connect three nodes: b, c, and d. On each stage of the birth process, the edges are born and their nodes create connected components, i.e., connected sets of nodes. So, nodes b, c and d belong to one component. Note that the e_4 edge that connects nodes c and d of this component, can be merged with the already born edges (e_3 and e_5) without changing the component $\{b, c, d\}$. That is, we can conclude that the edge e_4 is not relevant and can be excluded from the further evolution process. After merging this edge, the remaining edges to consider are only e_1, e_2, e_6, and e_7. Suppose that the next edge born is e_6, see state σ_3. Now the component has nodes b, c, d, e, and the e_7 edge joins the nodes belonging to the already existing component. So, the e_7 edge

can be added to the already born edges. After merging e_7, we are left with only two non born edges, e_1 and e_2. The birth of any of them signifies the entrance of the network to the *UP* state.

To implement the merging process, all we need to do after each birth of an edge, is to look for those edges whose nodes belong to the already existing component. These edges are joined to this component and excluded from further considerations as irrelevant. This combination of merging and the evolution process causes the reliability estimator to become less variable and is called the LT algorithm [13].

Algorithm 2.2 (Lomonosov's Turnip). The LT algorithm differs from the PMC Algorithm 2.1 only at Step 3. Recall that at each iteration $t = 1, \ldots, N$, we are given an edge permutation $\mathbf{\Pi}_t$. All we need to do now is to eliminate the redundant edges using the closure operation and find the corresponding anchor. All other steps remain the same.

It was established in [13] that the estimator delivered by Algorithm 2.2 is unbiased and its relative error (RE) (see Appendix Appendix A for formal definition of the RE), is uniformly bounded with respect to the $\lambda(e)$ values. For additional details see [14, Chapter 4]. However, this algorithm can exhibit a very bad performance due to rare-event estimation problems [16]. Consider a Monte Carlo estimator

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} Z^{(k)},$$

where $\{Z^{(k)}\}$ are independent copies of Z, with $\mathbb{E}(Z) = \ell$. A crucial efficiency parameter of this estimator is the coefficient of variation (CV). The CV is defined by $\mathrm{CV} = \sqrt{\mathrm{Var}\,(Z)}/\mathbb{E}\,(Z)$.

A Monte Carlo algorithm is called efficient if the CV is bounded by a polynomial in input size [10]. From the practical point of view, CV controls the number of samples (N) that are required to get a certain RE. See Appendix Appendix A for additional explanations. We next consider an example for which the LT Algorithm 2.2 is inefficient.

Example 2.2 (The "bad" example). Consider a simple network $\mathfrak{S}(n)$ having $n + 2$ nodes and $2n + 1$ edges presented in Figure 3. The terminal set consists of two vertices, u and v. It is not very hard to see that for this particular network topology the closure operation has no effect, since during

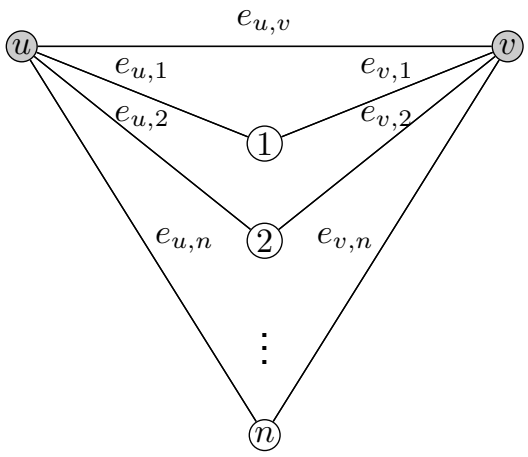


Figure 3: A simple network $\mathfrak{S}(n)$ with $n+2$ nodes, $2n+1$ edges, and $W = \{u, v\}$.

the edge birth process no edge can be merged and thus the LT algorithm turns into the regular PMC.

Suppose, for example, that each edge fails with same probability $q = 1-p$. Then, the u – v network unreliability $\bar{r}$ is given by $(1-p)(1-p^2)^n$. We next consider the distribution of the anchor for this particular network structure. Let $T = \mathsf{a}(\mathbf{\Pi})$ be the random variable that stands for the anchor. The LT algorithm returns $T = 1$ at Step 3 if the edge between u and v is the first one that enters the *up* state; that is, if $e_{u,v}$ is born first. We proceed with the analysis of the anchor distribution.

It is easy to see that $\mathbb{P}(T = 1) = 1/(2n+1)$. Consider now the probability that the birth process stops after the birth of the t-th edge ($T = t$), given that it did not stop before. It is not very hard to verify that

$$\mathbb{P}(T = t \mid T > t-1) = \frac{t}{2n+2-t}, \quad t = 1, \ldots, n+1.$$

From the above equation we obtain

$$\begin{aligned} \mathbb{P}(T = t) &= \mathbb{P}(T = t \mid T > t-1) \prod_{j=1}^{t-1} (1 - \mathbb{P}(T = j \mid T > j-1)) \qquad (5) \\ &= \frac{t}{2n+2-t} \prod_{j=1}^{t-1} \left(1 - \frac{j}{2n+2-j}\right), \quad t = 1, \ldots, n+1. \end{aligned}$$

The expression for $\mathbb{P}(T = t)$ in (5) allows us to analyze the performance of LT for our simple graph model. In particular, we consider the conditional probability

$$\mathbb{P}(\xi_1 + \cdots + \xi_T > 1 \mid T = t), \tag{6}$$

where $\xi_1, \xi_2, \ldots$ are independent, $\xi_t \sim \mathsf{Exp}(\Lambda(E_t))$ for $t = 1, \ldots, T$, where $\lambda(e) = -\ln q$ for all $e \in E$. Note that since $\Lambda(E_t) = (2n+2-t)\ln q$, the conditional probability (6) only depends on t, so, with a slight abuse of notation, we write this conditional probability as $\mathrm{Conv}(t)$. From the unbiasedness of the LT algorithm it holds that

$$\mathbb{E}(\mathrm{Conv}(T)) = \sum_{t=1}^{n+1} \mathrm{Conv}(t) \cdot \mathbb{P}(T = t) = \bar{r} = (1-p)(1-p^2)^n,$$

where $\bar{r}$ is the network unreliability. The second moment of $\mathrm{Conv}(T)$, that is, $\sum_{t=1}^{n+1} \mathrm{Conv}(t)^2 \cdot \mathbb{P}(T = t)$, does not reduce to a simple expression, but can be readily calculated.

We evaluated the values of $\mathrm{Conv}(t)$ and $\mathbb{P}(T = t)$ for $t = 1, \ldots, 51$. The graph in Figure 4 shows the CV for simple $\mathfrak{S}(n)$ graphs of different sizes ($1 \leqslant n \leqslant 50$), and different edge *down* probabilities q, namely, $q = 10^{-1}$ and $q = 10^{-3}$. Note that the CV shows a clear exponential growth as a function of n.

For example, the CV for $p = 0.9$ and $n = 50$ is about 10^6. So, in order to obtain a modest RE of say 10%, the required sample size N in Algorithm 2.2 should satisfy $\mathrm{CV}/\sqrt{N} = 0.1 \Rightarrow N \approx 10^{14}$. Such sample size is clearly unmanageable from the computation time point of view thus making the LT algorithm impractical.

To get a more intuitive understanding about the reason of LT's inefficiency, consider the unreliability estimation for the simple graph $\mathfrak{S}(50)$ and $p = 0.9$. The graph unreliability is given by

$$\bar{r} = (1-p)(1-p^2)^n = (1-0.9)(1-0.9^2)^{50} \approx 8.66 \times 10^{-38}.$$

However, when running the LT algorithm, we usually get an estimate of order 10^{-43} which is a clear underestimation. We next explain this phenomenon which happens because of the rare-event involvement. Figure 5 shows the convolution $\mathrm{Conv}(t)$ and the probability $\mathbb{P}(T = t)$ as a function of anchor $1 \leqslant t \leqslant 51$.

From Figure 5 we can see that the long trajectories contribute the most mass to the estimator; but, these long trajectories appear with very small

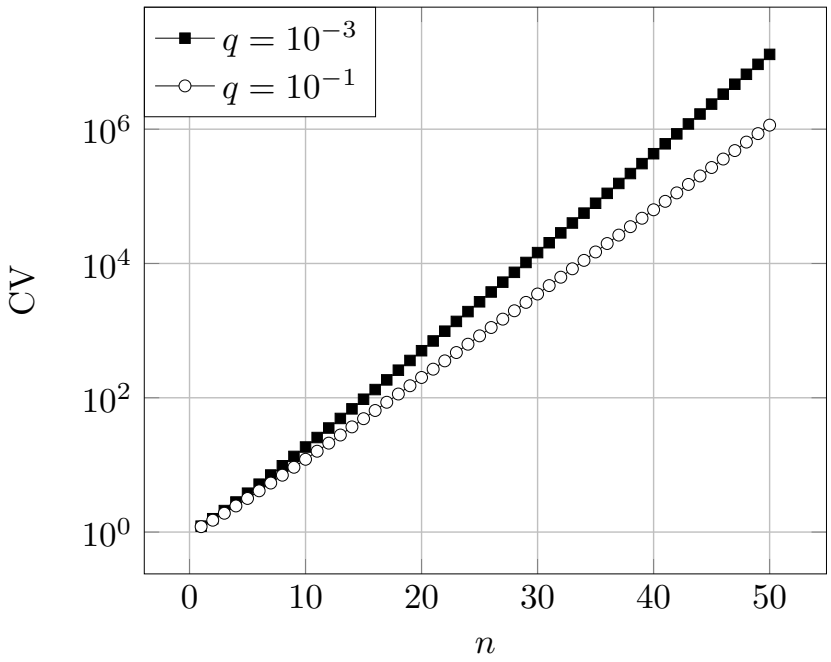


Figure 4: Logarithmically scaled CV of LT as a function of n for $\mathfrak{S}(n)$ networks.

probabilities. For example, we found that the average trajectory length is about 11.69. However, the trajectories that contribute most to the estimator are of length greater than 40. These trajectories are generated with a probability less than 10^{-6}. This issue can be clearly observed in the upper plot of Figure 5 by noting that the intersection of the horizontal line (which represents the true unreliability value), and the convolution curve, occurs near $t = 40$. Long trajectories are generated with very small probabilities as can be verified from Figure 5 (lower plot) and thus the resulting estimator tends to underestimate the true value of interest.

To overcome the problem presented in the above example, we propose to combine the LT algorithm with the splitting method, which was proved to be very useful when working under the rare-event setting. To do so, we first give a short introduction to a general SMC algorithm and show how it can be combined with the splitting framework.

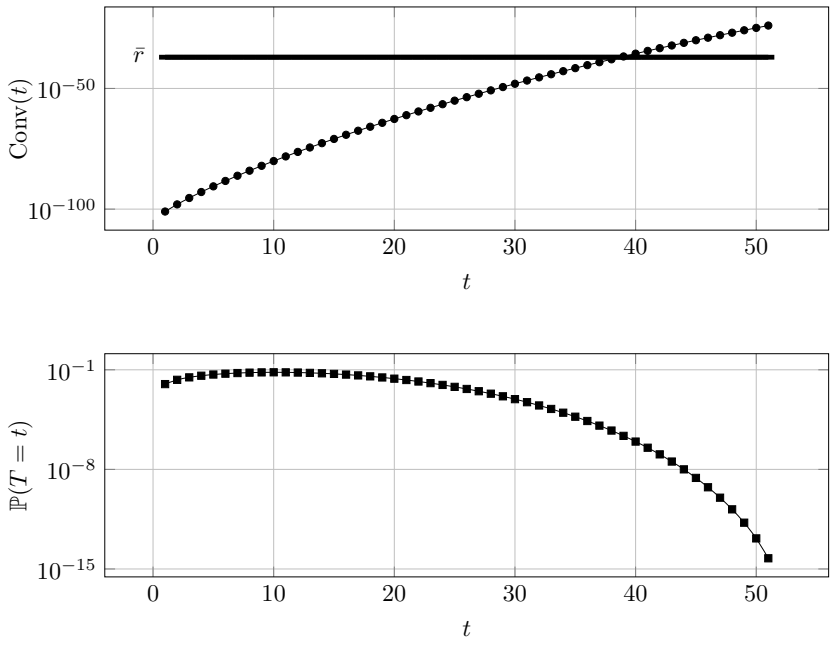


Figure 5: Conv(t) and $\mathbb{P}(T = t)$ as a function of t for $\mathfrak{S}(50)$ and $p = 0.9$. The true unreliability of 8.66×10^{-38} is given by the horizontal line in the upper plot.

3. The Splitting Sequential Monte Carlo

We start by examining a quite generic setting. Consider a random variable (vector) $\mathbf{X}$ taking values in a set $\mathcal{X}$. A general objective of Monte Carlo simulation is to calculate $\ell = \mathbb{E}_f\left(H\left(\mathbf{X}\right)\right)$, where $H : \mathcal{X} \to \mathbb{R}$ is a real-valued function. The Crude Monte Carlo (CMC) estimator of ℓ is given by

$$\widehat{\ell} = \frac{1}{N}\sum_{k=1}^{N} H\left(\mathbf{X}^{(k)}\right),$$

where $\mathbf{X}^{(k)}$ for $k = 1, \ldots, N$, are independent copies of a random variable $\mathbf{X}$ generated from $f(\mathbf{x})$.

In this paper we consider the SMC framework [27]. Suppose that the vector $\mathbf{X} \in \mathcal{X}$ is decomposable and that it can be of different length $T \in \{1, \ldots, n\}$, where T is a stopping time of $\mathbf{X}$'s generation process. Thus, $\mathbf{X}$ can be written as

$$\mathbf{X} = (X_1, X_2, \ldots, X_T),$$

where for each $t = 1, \ldots, T$, X_t can be multidimensional. We assume that $\mathbf{X}$ can be constructed sequentially such that its probability density function (PDF) $f(\mathbf{x})$ constitutes a product of conditional PDFs:

$$f(\mathbf{x}) = f_1(x_1) f_2(x_2 \mid x_1) \cdots f_t(x_t \mid x_1, \ldots, x_{t-1}), \quad \text{when } |\mathbf{x}| = t,\ t = 1, \ldots, n,$$

where $|\mathbf{x}|$ is the length of $\mathbf{x}$.

This setting frequently occurs in practice. For example, consider a coin that is tossed repeatedly until the first "success" (1) appears or until n tosses have been made. The sample space is equal to

$$\mathcal{X} = \{(1),\ \ (0,1),\quad (0,0,1), \ldots, (\underbrace{0, \ldots, 0,}_{n-1 \text{ times}} 1),\ \ (\underbrace{0, \ldots, 0}_{n \text{ times}})\},$$

that is, the samples have different lengths: $t = 1, 2, 3, \ldots, n$. Let $\mathcal{X}_t = \{\mathbf{x} \in \mathcal{X} \ : \ |\mathbf{x}| = t\}$ be the set of all samples of length $t = 1, 2, \ldots, n$. Then, the sets $\mathcal{X}_1, \ldots, \mathcal{X}_n$ define a partition of $\mathcal{X}$; that is

$$\mathcal{X} = \bigcup_{t=1}^{n} \mathcal{X}_t, \quad \mathcal{X}_{t_1} \cap \mathcal{X}_{t_2} = \emptyset \quad \text{for } 1 \leqslant t_1 < t_2 \leqslant n.$$

Since we are working under the SMC framework, the generation of $\mathbf{X} = (X_1, \ldots, X_T) \in \mathcal{X}_T$, is sequential in the following sense. We start from "empty" $\mathbf{X} = ()$. Then X_1 is sampled from $f_1(x_1)$ and at each step $t \geqslant 2$, we sample X_t from $f_t(x_t \mid x_1, \ldots, x_{t-1})$ until the stopping time T that is determined from the generated X_t's. This procedure terminates at time $1 \leqslant T \leqslant n$ if $\mathbf{X} \in \mathcal{X}_T$. The above process is summarized in Algorithm 3.1.

Algorithm 3.1 (Crude Sequential Monte Carlo (CSMC)). *Given the density $f(\mathbf{x}) = f_1(x_1) f_2(x_2 \mid x_1) \cdots f_t(x_t \mid x_1, \ldots, x_{t-1})$, $\mathbf{x} \in \mathcal{X}_t$, $t = 1, \ldots, n$, and $H : \mathcal{X} \to \mathbb{R}$, output Z — an unbiased estimator of $\mathbb{E}_f(H(\mathbf{X}))$.*

1. ***(Initialization)*** *Set $t \leftarrow 0$ and $\mathbf{X} \leftarrow ()$.*
2. ***(Simulate and update)*** *Set $t \leftarrow t+1$, sample $X_t \sim f_t(x_t \mid X_1, \ldots, X_{t-1})$, and set $\mathbf{X} \leftarrow (X_1, \ldots, X_{t-1}, X_t)$.*
3. ***(Stopping Condition)*** *If $T = t$ (the stopping condition which can be determined from $\mathbf{X} = (X_1, \ldots, X_t)$), output $Z \leftarrow H(\mathbf{X})$; otherwise, go to Step 2.*

Remark 3.1 (LT Algorithm 2.2 under the SMC Framework). To see that the PMC and the LT Algorithms 2.1 and 2.2 are aligned with the SMC framework described above, let $\mathbf{X} = (\Pi_1, \ldots, \Pi_T)$, with $T = \mathsf{a}(\Pi)$, and

$$H(\mathbf{x}) = 1 - \mathrm{Conv}_{1 \leqslant t \leqslant T} \left\{ 1 - \mathrm{e}^{-\Lambda(E_i)} \right\}.$$

Moreover, $f(\mathbf{x})$ is distributed according to (2), which is of the product form

$$f(\mathbf{x}) = \prod_{j=1}^{t} f_j, \quad 1 \leqslant t \leqslant n,$$

where f_j is defined by $f_j(\Pi_j = e_j \mid \Pi_1 = e_1, \ldots, \Pi_{j-1} = e_{j-1}) = \frac{\lambda(e_j)}{\Lambda(E_{j-1})}$.

For the forthcoming discussion, it will be convenient to define an event

$$\{\text{the SMC generation process did not stop at steps } 1, \ldots, t\} \coloneqq \{T > t\}.$$

The $\mathbf{X}$ generation stochastic process can be visualized using Figure 6. A random walk starts from the root of the tree $\{T > 0\}$ and ends at one of tree leaves $\{T = 1\}, \ldots, \{T = n\}$.

We next proceed with a brief introduction to multilevel splitting [6, 22, 24] and show how the latter can be combined with the general CSMC Algorithm 3.1 to solve the rare-event problem in Example 2.2. The splitting idea is straightforward. Instead of running a single sampling process, one launches a few processes in parallel. This simple modification is very beneficial as we show in the illustrative Example 3.1.

Example 3.1 (Splitting Example). Consider the SMC process tree in Figure 6 and suppose for simplicity that for $t = 1, \ldots, n-1$,

$$\mathbb{P}(T = t \mid T > t - 1) = 1 - \mathbb{P}(T > t \mid T > t - 1) = 1/2,$$

and $\mathbb{P}(T = n \mid T > n - 1) = 1$. We start a single walk from the tree root which ends at some leaf $\{T = t\}$ for $t = 1, \ldots, n$. It is not very hard to see that the probability of reaching the last leaf in the tree is equal to

$$\mathbb{P}(T = n) = 2^{-(n-1)}.$$

Note that we already encountered the similar setting in Example 2.2 were we had to deal with long trajectories that are generated with very small

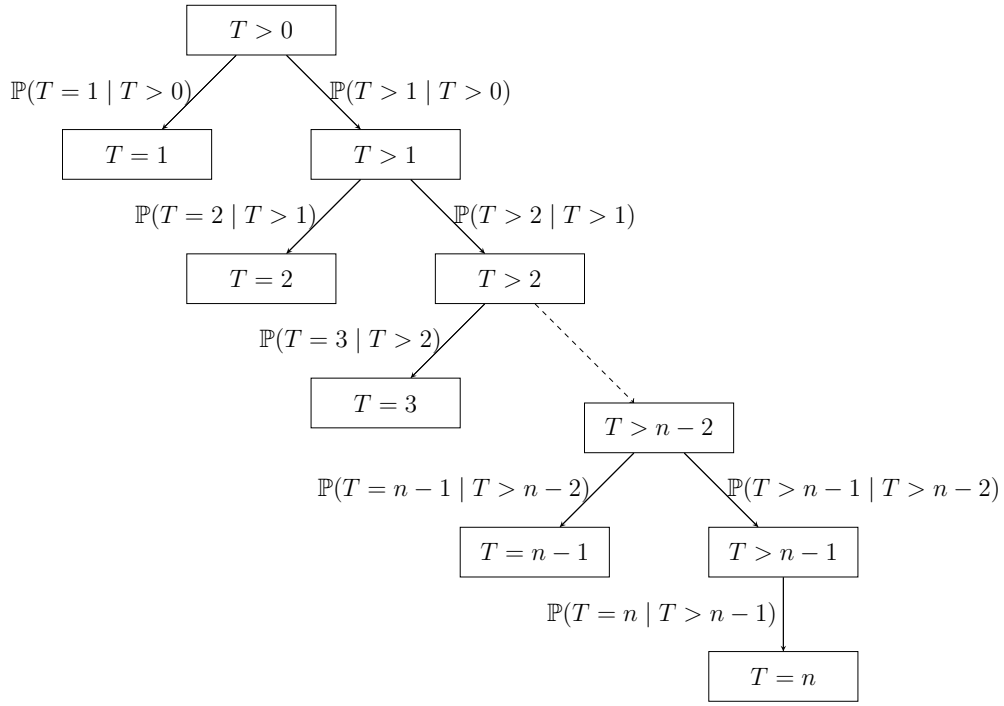


Figure 6: SMC process.

probabilities. Consequently, we are interested in a different sampling process that overcomes the tiny $2^{-(n-1)}$ probability and hence the rare-event setting. In other words, we would like to increase the probability of sampling low tree levels.

Let us define some budget $B \in \mathbb{N} \setminus \{0\}$ and launch B parallel walks (trajectories), from the tree root. At each step $t = 1, \ldots, n-1$, we detect the trajectories that "finished" their execution, that is, they are at node $\{T = t\}$. We are interested, at each level t, to keep B trajectories "alive", so, we duplicate some of the paths at the $\{T > t\}$ node and continue with B trajectories, abandoning the "finished" ones. The latter is called a trajectory splitting. This simple mechanism allows the process to reach $\{T = n\}$ with reasonably high probability while using a relatively small budget B which can be logarithmic in the tree height. The mathematics is as follows. Note that the probability that this splitting process reaches the level t given that

it reached level $t-1$ is

$$\mathbb{P}(T = t \mid T > t-1) = 1 - 1/2^B,$$

and as a consequence, the probability to reach $\{T = n\}$ is equal to

$$\mathbb{P}\left(\text{The splitting reaches the } \{T = n\} \text{ leaf}\right) = \left(1 - 1/2^B\right)^{n-1}.$$

We conclude the above discussion with a crucial observation: the choice of $B = \lceil \log_2(n-1) \rceil$ results in

$$\mathbb{P}\left(\text{The process reaches } \{T = n\}\right) \geqslant (1 - 1/2^{\log_2(n-1)})^{n-1} \to \mathrm{e}^{-1} \quad \text{as } n \to \infty.$$

Example 3.1 implies that the splitting idea can be used to improve the sampling of rare (long) trajectories under the SMC framework. We next proceed with a general algorithm that combines the splitting mechanism and the CSMC Algorithm 3.1. The Splitting Sequential Monte Carlo (SSMC) algorithm is summarized in Algorithm 3.2.

Algorithm 3.2 (Splitting Sequential Monte Carlo (SSMC)). *Given the density* $f(\mathbf{x}) = f_1(x_1) f_2(x_2 \mid x_1) \cdots f_t(x_t \mid x_1, \ldots, x_{t-1})$, *for* $\mathbf{x} \in \mathcal{X}_t$, $1 \leqslant t \leqslant n$, $H : \mathcal{X} \to \mathbb{R}$, *and a budget* $B \in \mathbb{N} \setminus \{0\}$, *output* C *— an unbiased estimator of* $\mathbb{E}_f\left(H\left(\mathbf{X}\right)\right)$.

1. ***(Initialization)*** *Set* $t \leftarrow 0$, $P_t \leftarrow 1$ *— an estimator of* $\mathbb{P}\left(T > t\right)$, $C \leftarrow 0$, *and define*

$$\mathcal{W}^{(t)} = \left\{\mathbf{X}_1^{(t)}, \ldots, \mathbf{X}_B^{(t)}\right\},$$

   where $\mathbf{X}_j^{(t)} \leftarrow ()$ *for* $j = 1, \ldots, B$, *which we call the "working" set, because it contains unfinished trajectories.*

2. ***(Simulate and Update)*** *Set* $t \to t{+}1$. *For each* $\mathbf{X} = (X_1, \ldots, X_{t-1}) \in \mathcal{W}^{(t-1)}$, *sample*

$$X_t \sim f_t(x_t \mid X_1, \ldots, X_{t-1}),$$

   and update: $\mathbf{X} \leftarrow (X_1, \ldots, X_t)$. *Update the "finished" and "working" sets:*

$$\mathcal{F}^{(t)} \leftarrow \left\{\mathbf{X} \in \mathcal{W}^{(t-1)} \,:\, \mathbf{X} \in \mathcal{X}_t\right\}, \quad B_t \leftarrow \left|\mathcal{F}^{(t)}\right|,$$
$$\mathcal{W}^{(t)} \leftarrow \left\{\mathbf{X} \in \mathcal{W}^{(t-1)} \,:\, \mathbf{X} \notin \mathcal{X}_t\right\}, \quad B_t' \leftarrow \left|\mathcal{W}^{(t)}\right|.$$

   If $B_t = 0$, *go to Step* 2; *otherwise, set*

$$C_t \leftarrow P_{t-1} \frac{1}{B} \sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X}), \quad C \leftarrow C + C_t, \quad P_t \leftarrow P_{t-1} \frac{B_t'}{B}.$$

3. ***(Stop Condition)*** *If $B'_t = 0$, output C as an estimator of $\mathbb{E}_f(H(\mathbf{X}))$.*
4. ***(Splitting)*** *Insert K_j copies of each $\mathbf{X}_j \in \mathcal{W}^{(t)}$ into $\mathcal{W}^{(t)}$, where K_j satisfies*

$$K_j = \lfloor B/B'_t \rfloor + L_j,$$

and $L_j \sim \mathsf{Ber}(0.5)$ conditional on $\sum_{s=1}^{B'_t} L_s = B \bmod B'_t$. Go to Step 2.

The usage of SSMC Algorithm 3.2 for network reliability estimation is as follows.

Algorithm 3.3 (The Split-Turnip (ST)). *Given a network $G = G(V, E, W)$, edge failure probabilities (q_e, $e \in E$), a budget $B \in \mathbb{N} \setminus \{0\}$, and a sample size N, execute the following steps.*

1. ***(Initialization)*** *Define the sample space $\mathcal{X}$, its partition $\mathcal{X}_1, \ldots, \mathcal{X}_n$, the function $H(\mathbf{x})$ and the PDF $f(\mathbf{x})$ according to Remark 3.1. Set $k \leftarrow 0$ and $S \leftarrow 0$.*
2. ***(Apply SSMC Algorithm)*** *Set $k \leftarrow k + 1$ and apply Algorithm 3.2 with parameters f, H and B, to obtain an estimator $C^{(k)}$ of the unreliability and set $S \leftarrow S + C^{(k)}$.*
3. ***(Stop Condition)*** *If $k = N$, return S/N as unbiased estimator of $\bar{r}$; otherwise, go to step 2.*

Remark 3.2 (Computational Complexity). Suppose that the complexity of calculating the exponential convolution is given by $\mathcal{O}(\mathrm{Cnv})$. Note that the LT method requires $\mathcal{O}(|E|)$ time to produce a random permutation and find its anchor, so, its overall complexity is $\mathcal{O}(N|E|\mathrm{Cnv})$. On the other hand, the ST method calculates the convolution at most $B|E|$ times during a single run. Consequently, the ST algorithm complexity is equal to $\mathcal{O}(NB|E|\mathrm{Cnv})$, that is, ST is more expensive than LT in the order of the budget B.

We next proceed with the analysis of the SSMC Algorithm 3.2.

3.1. The Analysis

Theorem 3.1 (Unbiased estimator). *The SSMC Algorithm 3.2 outputs an unbiased estimator; that is, it holds that*

$$\mathbb{E}(C) = \mathbb{E}_f(H(\mathbf{X})).$$

Proof: To start with, recall that $\mathcal{X}_1, \ldots, \mathcal{X}_n$ is a partition of the entire sample space $\mathcal{X}$, so, from the law of total expectation we arrive at

$$\mathbb{E}_f\left(H\left(\mathbf{X}\right)\right) = \sum_{t=1}^{n} \mathbb{E}_f\left(H\left(\mathbf{X}\right) \mid T = t\right) \mathbb{P}(T = t).$$

With the above equation in mind, it will be enough to prove that for all $t = 1, \ldots, n$, it holds that

$$\mathbb{E}\left(C_t\right) = \mathbb{E}\left(P_{t-1} \frac{1}{B} \sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X})\right) = \mathbb{E}_f\left(H\left(\mathbf{X}\right) \mid T = t\right) \mathbb{P}(T = t).$$

To prove this, we will need the following.

1. Note that the samples in the $\mathcal{F}^{(t)}$ set are clearly dependent, but have the same distribution. Hence, it holds that

$$\mathbb{E}\left(\sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X})\right) \underbrace{=}_{\left|\mathcal{F}^{(t)}\right| = B_t} B_t\, \mathbb{E}_f\left(H(\mathbf{X}) \mid T = t\right).$$

2. From the well-known result of unbiasedness of basic multilevel splitting estimator [6, 21], it holds that

$$\mathbb{E}\left(P_{t-1} \frac{B_t}{B}\right) = \mathbb{E}\left(\frac{B_t}{B} \prod_{j=1}^{t-1} \frac{\left|\mathcal{W}^{(j)}\right|}{B}\right) = \mathbb{E}\left(\frac{B_t}{B} \prod_{j=1}^{t-1} \frac{B'_j}{B}\right) = \mathbb{P}(T = t). \quad (7)$$

We conclude the proof by noting that

$$\mathbb{E}\left(C_t\right) = \mathbb{E}\left(P_{t-1} \frac{1}{B} \sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X})\right) = \mathbb{E}\left[\frac{1}{B} \mathbb{E}\left(P_{t-1} \sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X}) \,\middle|\, B_1, \ldots, B_t\right)\right]$$
$$\underbrace{=}_{P_{t-1} = \prod_{j=1}^{t-1} \frac{B_j}{B}} \mathbb{E}\left[P_{t-1} \frac{1}{B} \mathbb{E}\left(\sum_{\mathbf{X} \in \mathcal{F}^{(t)}} H(\mathbf{X}) \,\middle|\, B_1, \ldots, B_t\right)\right]$$

$$\underbrace{=}_{(7)} \mathbb{E}\left[P_{t-1}\frac{B_t}{B}\mathbb{E}_f\left(H(\mathbf{X}) \mid T=t\right)\right] = \mathbb{E}_f\left(H(\mathbf{X}) \mid T=t\right)\mathbb{E}\left[P_{t-1}\frac{B_t}{B}\right]$$
$$\underbrace{=}_{(7)} \mathbb{E}_f\left(H\left(\mathbf{X}\right) \mid T=t\right)\mathbb{P}(T=t).$$

Despite that it is generally hard to analyse the efficiency of SSMC for a given problem in terms of RE, Theorem 3.2 provides performance guaranties under some simplified assumptions. However, it is important to note that Theorem 3.1 holds for general SMC procedures which can be presented in the form of CSMC Algorithm 3.1.

Theorem 3.2 (Efficiency of SSMC Algorithm 3.2). *Suppose that the following holds for all $t = 1, \ldots, n$.*

1. *For $t = 1, \ldots, n$, $f_t(x_t \mid x_1, \ldots, x_{t-1}) = p_t$ (constant) for all $\mathbf{x} = (x_1, \ldots, x_{t-1}) \in \mathcal{X}_{t-1}$, and $p_t = \mathcal{O}(1/\mathcal{P}_n)$, where $\mathcal{P}_n$ is a polynomial in n.*
2. *$H(\mathbf{x}) = H_t$ (constant) for all $\mathbf{x} \in \mathcal{X}_t$, $t = 1, \ldots, n$.*

Then, under above assumptions, the SSMC Algorithm 3.2 is efficient [10], that is, it holds that $\mathrm{CV} = \sqrt{\mathrm{Var}\left(C\right)}/\mathbb{E}\left(C\right)$ *is upper-bounded by a polynomial in n.*

Proof: The analysis is by obtaining the lower and the upper bounds for the first and the second moments of C, respectively.

1. **First moment**. Since $H(\mathbf{x}) = H_t$ for $\mathbf{x} \in \mathcal{X}_t$, it holds that

$$\mathbb{E}_f(H(\mathbf{X}) \mid T=t) = H_t, \quad t = 1, \ldots, n.$$

Combining this with Theorem 3.1 yields

$$\mathbb{E}(C) = \sum_{t=1}^{n} \mathbb{E}_f(H(\mathbf{X}) \mid T=t)\mathbb{P}(T=t) = \sum_{t=1}^{n} H_t(1-p_t)\prod_{j=1}^{t-1} p_j,$$

hence,

$$[\mathbb{E}(C)]^2 = \left(\sum_{t=1}^{n} H_t(1-p_t)\prod_{j=1}^{t-1} p_j\right)^2 \geqslant \sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1} p_j^2. \qquad (8)$$

2. **Second moment**. Since the entrance states $\left\{X_1^{(t)}, \ldots, X_B^{(t)}\right\}$ are independent for $t = 1, \ldots, n$, the random variables B_t and B_t' are binomially distributed according to $\mathsf{Bin}(B, p_t)$ and $\mathsf{Bin}(B, 1-p_t)$, respectively. Hence, the second moments of B_t/B and B_t'/B are given by

$$\mathbb{E}\left(B_t/B\right)^2 = \left(\frac{p_t(1-p_t)}{B} + p_t^2\right), \tag{9}$$

and

$$\mathbb{E}\left(B_t'/B\right)^2 = \left(\frac{p_t(1-p_t)}{B} + (1-p_t)^2\right), \tag{10}$$

respectively. Let

$$p_{\min} = \min_{1 \leqslant t \leqslant n} \{p_t\},$$

and note that

$$\begin{aligned}
\mathbb{E}\left(C^2\right) &= \mathbb{E}\left(\sum_{t=1}^{n} H_t(1-P_t)\prod_{j=1}^{t-1} P_j\right)^2 \underbrace{\leqslant}_{\text{Jensen inequality [25]}} \qquad (11)\\
&\leqslant n\sum_{t=1}^{n} H_t^2\,\mathbb{E}\left((1-P_t)\prod_{j=1}^{t-1} P_t\right)^2 = \sum_{t=1}^{n} H_t^2\,\mathbb{E}\left(\frac{B_t}{B}\prod_{j=1}^{t-1}\frac{B_j}{B}\right)^2\\
&\underbrace{=}_{(9,\ 10)} n\sum_{t=1}^{n} H_t^2\left(\frac{p_t(1-p_t)}{B} + (1-p_t)^2\right)\prod_{j=1}^{t-1}\left(\frac{p_j(1-p_j)}{B} + p_j^2\right)\\
&= n\sum_{t=1}^{n} H_t^2(1-p_t)^2\left(1 + \frac{p_t}{B(1-p_t)}\right)\prod_{j=1}^{t-1}\left(p_j^2\left(1 + \frac{1-p_j}{Bp_j}\right)\right)\\
&\leqslant n\left(1+\frac{1}{B}\right)\sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1} p_j^2\prod_{j=1}^{t-1}\left(1+\frac{1}{Bp_{\min}}\right)\\
&\leqslant 2n\left(1+\frac{1}{Bp_{\min}}\right)^n\sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1} p_j^2\\
&\underbrace{\leqslant}_{B \geqslant \lceil n/p_{\min}\rceil} 2ne\sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1} p_j^2,
\end{aligned}$$

where the last inequality follows from the well-known identity:

$$(1+1/n)^n \leqslant \mathrm{e}, \quad n > 0.$$

Note that $B = \lceil n/p_{\min} \rceil$ is a polynomial in n. We complete the proof of the theorem by combining (8) and (11) and arriving at

$$\mathrm{CV} \leqslant \frac{2n\mathrm{e}\sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1}p_j^2}{\sum_{t=1}^{n} H_t^2(1-p_t)^2\prod_{j=1}^{t-1}p_j^2} = 2n\mathrm{e}.$$

We already saw that both PMC Algorithm 2.1 and the LT Algorithm 2.2 can be viewed as CSMC Algorithm 3.1. To see the merit of using SSMC for network reliability estimation, consider an immediate efficiency result for $\mathfrak{S}(n)$ networks which is presented in Corollary 3.1.

Corollary 3.1 (Efficiency of SSMC for $\mathfrak{S}(n)$ networks). *The PMC Algorithm 2.1 combined with SSMC Algorithm 3.2, is an FPRAS for networks $\mathfrak{S}(n)$, $n > 0$.*

Proof: The proof is an immediate consequence of Theorem 3.2. Recall the Example 2.2 and note that

$$\begin{aligned}\mathbb{P}(\mathsf{a}(\mathbf{\Pi}) = t \mid \mathsf{a}(\mathbf{\Pi}) > t-1) &= \mathbb{P}(T = t \mid T > t-1)\\ &= \frac{t}{2n+2-t} = \mathcal{O}\left(\frac{1}{2n+1}\right),\end{aligned}$$

for each $t = 1, \ldots, n+1$. Moreover, since for the $\mathfrak{S}(n)$ network $p_e = p$ for all $e \in E$, the function $H(\mathbf{x}) = H_t = \mathrm{Conv}(t)$ is constant in $\mathcal{X}_t$. We conclude that the first and the second conditions of Theorem 3.2 holds, thus completing the proof.

It is important to note that ST is an FPRAS for any family of graphs that satisfies the conditions of Theorem 3.2. Although these families are not "very interesting" from a practical point of view, our numerical results indicate that ST introduces an excellent performance for quite general graph typologies. We next proceed with demonstrative numerical examples.

4. Numerical Results

In this section we introduce some typical example cases in order to demonstrate the efficacy of the proposed ST method. In the first test case we verify numerically the theoretical result of Corollary 3.1 using the $\mathfrak{S}(50)$ network. For the second model we take the dodecahedron graph with 20 vertices and 30 edges. This graph is widely considered to be a good benchmark for network reliability. Finally, in our third example, we consider a bigger model of a size for which simulation is typically required. In particular, similar to [19], we consider three merged copies of the dodecahedron graph.

We performed many experiments with the LT and ST algorithms discussed above. In particular, all the tests were executed on a desktop quad-core 3.4Ghz processor. To report our findings, we developed a software package named `RelSplit`. This software and some example models are freely available for download at `http://www.smp.uq.edu.au/people/RadislavVaisman/#software`. The results should be interpreted as follows.

- $\overline{R}$ is the estimator of network unreliability.
- $\widehat{\mathrm{RE}}$ is an estimated relative error.
- The relative experimental error (REE) is given by $\mathrm{REE} = \left|\overline{R} - \bar{r}\right| \Big/ \bar{r}$, where $\bar{r}$ is the exact network unreliability.
- B and N are the budget and the sample size parameters, respectively. The budget B is used in ST algorithm while the sample size N stands for the number of independent repetitions to perform prior to averaging and delivering the final result $\overline{R}$.

To ensure a fair comparison, we set the sample size of LT to be equal to the sample size of ST multiplied by the budget B, see Remark 3.2. Next, we proceed with the models.

4.1. Model 1 — the graph $\mathfrak{S}(n)$

We consider the performance of LT and ST on the $\mathfrak{S}(50)$ network with $p = 0.9$. For the ST algorithm, we set $B = 1000$ and $N = 100$. Consequentially, we use $N = 10^5$ sample size for the LT algorithm. Table 4.1 summarizes the average performance of LT and ST for the $\mathfrak{S}(50)$ network using the above parameters. The bad performance of LT is not very surprising, since we know that for $\mathfrak{S}(50)$, the CV is of order 10^6.

Algorithm	$\overline{R}$	$\widehat{\mathrm{RE}}$	REE
LT	1.93×10^{-41}	76.5%	99.7%
ST	8.67×10^{-38}	2.48%	2.41%

Table 1: The performance obtained for the $\mathfrak{S}(50)$ network with $p = 0.9$ using the LT and the ST algorithms. The true unreliability is 8.66×10^{-38}, see Example 2.2.

4.2. Model 2 — the dodecahedron graph

In this model we consider the dodecahedron graph with 20 vertices and 30 edges, see Figure 7.

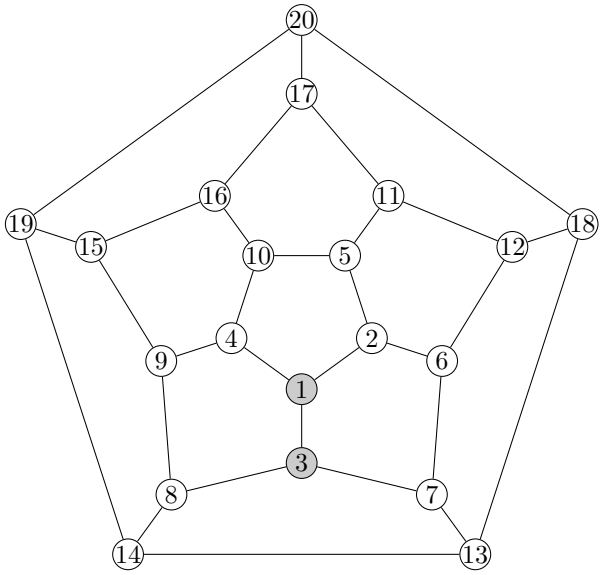


Figure 7: The dodecahedron graph with 20 vertices, 30 edges, and $W = \{1, 3\}$.

Both LT and GS were reported to deliver an excellent results in Botev et al. [16]. Indeed, this network is relatively small and in case of all edges having the same failure probability, the rare-event phenomenon does not apply. However, we show next that even for such a small network it is possible to assign the failure probabilities in a "bad" way. Our experiment is as follows. We set the terminal nodes to be $W = \{1, 3\}$. All the edge

failure probabilities are equal to 0.5 except of the following component. The edges $(1,2),(2,6),(6,7),(3,7)$ and $(1,3)$ failure probabilities are all set to be $q = 10^{-j}$, for $j = 1,\ldots,15$.

In particular, for very small values of q, we expect that the vertex component $\{1,2,3,6,7\}$ will be born at early stages of the evolution process with high probability, and thus, a rare event is created since long trajectories will appear more rarely. For the ST algorithm, we set $B = 10$ and $N = 6 \cdot 10^4$. The sample size of LT is $N = 6 \cdot 10^5$. Table 4.2 summarizes the average performance of LT and ST for the dodecahedron network using the above parameters.

	LT		ST	
q	$\overline{R}$	$\widehat{\mathrm{RE}}$	$\overline{R}$	$\widehat{\mathrm{RE}}$
10^{-15}	6.35×10^{-31}	58.1%	3.24×10^{-30}	5.04%
10^{-14}	8.80×10^{-29}	56.9%	3.25×10^{-28}	4.77%
10^{-13}	1.49×10^{-26}	55.3%	3.25×10^{-26}	4.18%
10^{-12}	6.85×10^{-25}	50.7%	3.24×10^{-24}	3.78%
10^{-11}	1.39×10^{-22}	50.7%	3.23×10^{-22}	3.45%
10^{-10}	1.01×10^{-20}	49.9%	3.22×10^{-20}	3.00%
10^{-9}	4.94×10^{-18}	48.2%	3.24×10^{-18}	2.81%
10^{-8}	2.14×10^{-16}	47.8%	3.23×10^{-16}	2.50%
10^{-7}	2.50×10^{-14}	43.8%	3.23×10^{-14}	2.22%
10^{-6}	4.30×10^{-12}	42.5%	3.24×10^{-12}	1.96%
10^{-5}	3.46×10^{-10}	32.8%	3.24×10^{-10}	1.66%
10^{-4}	3.18×10^{-8}	17.9%	3.24×10^{-8}	1.36%
10^{-3}	3.24×10^{-6}	6.68%	3.24×10^{-6}	1.23%
10^{-2}	3.20×10^{-4}	1.85%	3.20×10^{-4}	0.75%
10^{-1}	2.82×10^{-2}	0.43%	2.82×10^{-2}	0.42%

Table 2: A summary of average performance obtained for the dodecahedron network using the LT and ST algorithms.

Table 4.2 clearly shows the superiority of ST for this model. In particular, the ST methods shows better RE as compared to LT. Moreover, a crucial observation is that the LT algorithm provides an order of magnitude underestimation. We next continue with the larger merged dodecahedron model.

4.3. Model 3 — Series of three dodecahedrons

We consider three merged copies of the dodecahedron graph from the previous model. These graphs are connected in series as in [19, Example 10]. Let the nodes numbered 1 and 20 be the source and the destination of each dodecahedron, respectively. We define the terminal set of the merged graph to be the source of the first dodecahedron and the destination of the third dodecahedron, respectively. To connect the dodecahedron copies we do the following. The destination of the first (respectively second) copy is the source of the second (respectively third) [19]. The resulting merged graph has 90 edges.

Next, we set the failure probability of every edge to be 0.4 and run LT to obtain the unreliability. Using $N = 10^6$ sample size, we estimated the network unreliability to be approximately equal to 0.812 with confidence interval of width 0.03%. Next, we add a new edge e between the merged graph terminal nodes (u and v), see Figure 8, and consider the performance of LT and ST for this network and for different failure probabilities q of e.

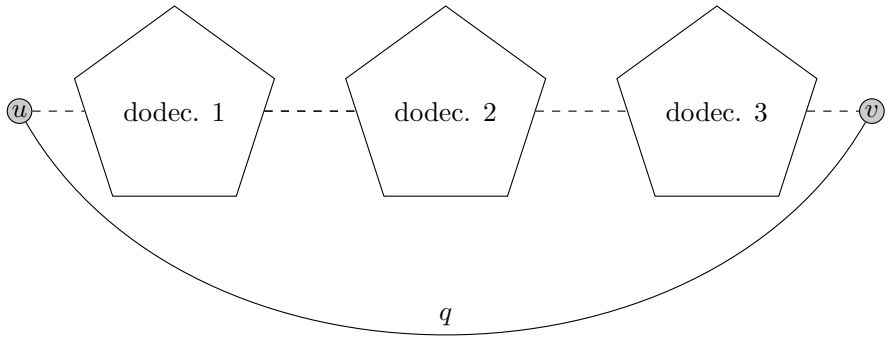


Figure 8: The merged dodecahedron graph with $W = \{u, v\}$.

In particular, we consider $q = 10^{-1}, 10^{-2}, \ldots, 10^{-15}$. For each such failure rate q, the unreliability of the system is equal to q times the unreliability of the series system of three dodecahedrons, so approximately $0.812 \cdot q$. For the ST method we set $B = 30$ and $N = 2000$. For the LT algorithm we set $N = 6 \cdot 10^4$ sample size, respectively. Figure 9 summarizes the algorithm's expected REEs (with respect to the "true" unreliability $0.812 \cdot q$) and REs for different values of q. We can observe from Figure 9 that the LT algorithm becomes very inaccurate for $q \leqslant 10^{-9}$.

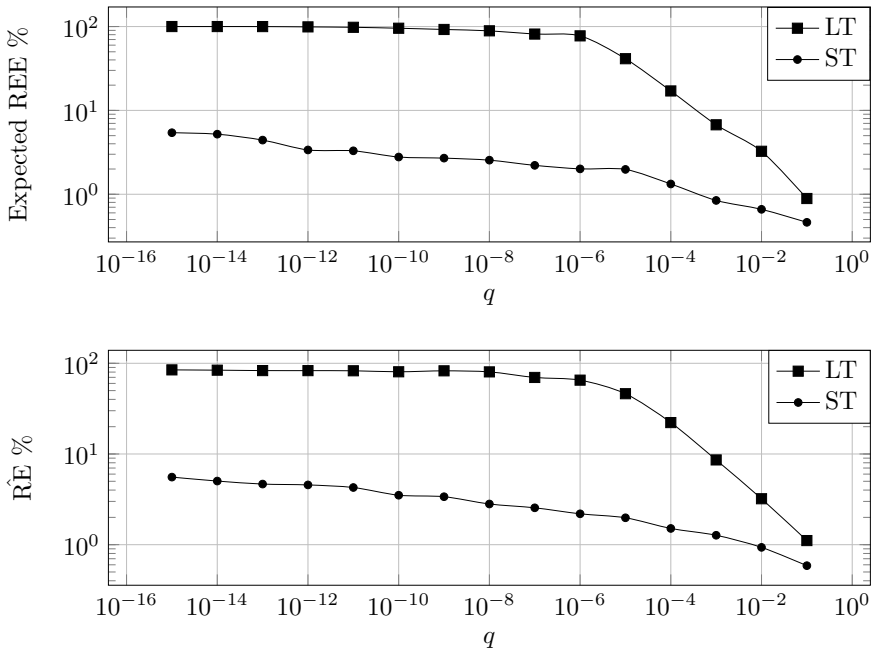


Figure 9: Expected REE % and $\widehat{\text{RE}}$ % of LT and ST as a function of q.

5. Concluding Remarks

In this paper we developed a general scheme that combines Sequential Monte Carlo and multilevel splitting. In particular, we used our method to improve the performance of Lomonosov's turnip by developing the Split-Turnip algorithm. We showed that the Split-Turnip method is efficient in the sense of existence of theoretically provable performance guarantees for specific networks and demonstrated numerically that its performance is generally outperforms the turnip. Of interest for future research is to apply our method to different problems which are currently been solved under the Sequential Monte Carlo framework. Additionally, it will be interesting to rigorously analyze these problems in the spirit of Theorem 3.2.

Appendix A. Monte Carlo and Rare-Events

Consider a set $\mathcal{X}$ and a subset $\mathcal{X}^* \subseteq \mathcal{X}$. Suppose that the probability $\ell = |\mathcal{X}^*|/|\mathcal{X}|$ is to be estimated. Further suppose that one can sample uniformly at random from $\mathcal{X}$. The Crude Monte Carlo (CMC) procedure

for the estimation of ℓ is defined as follows. Generate N independent and uniformly distributed samples, $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$ from $\mathcal{X}$ and let $Z^{(k)}$ (for $k = 1, \ldots, N$) be (independent) Bernoulli random variable such that

$$Z^{(k)} = \mathbb{1}_{\{\mathbf{X}^{(k)} \in \mathcal{X}^*\}} = \begin{cases} 1 & \text{if } \mathbf{X}^{(k)} \in \mathcal{X}^* \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbb{1}$ is indicator random variable. The CMC estimator $\widehat{\ell}$ is given by:

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} Z^{(k)}.$$

It is not hard to see that $\mathbb{E}\left(Z^{(k)}\right) = \ell$ and $\mathrm{Var}\left(Z^{(k)}\right) = \sigma^2 = \ell(1-\ell)$ for all $k = 1, \ldots, N$. Consequentially, it holds that the estimator is unbiased; that is,

$$\mathbb{E}\left(\widehat{\ell}\right) = \mathbb{E}\left(\frac{1}{N} \sum_{k=1}^{N} Z^{(k)}\right) = \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}\left(Z^{(k)}\right) = \ell. \tag{A.1}$$

Moreover, the variance of the estimator is given by

$$\mathrm{Var}\left(\widehat{\ell}\right) = \mathrm{Var}\left(\frac{1}{N} \sum_{k=1}^{N} Z^{(k)}\right) = \frac{1}{N^2} \sum_{i=1}^{N} \mathrm{Var}\left(Z^{(k)}\right) = \frac{\ell(1-\ell)}{N}. \tag{A.2}$$

With (A.1) and (A.2), we can measure the accuracy of the CMC estimator. Note that according to central limit theorem, $\widehat{\ell}$ is approximately $\mathsf{N}\left(\ell, \sigma^2/N\right)$ distributed, where N stands for normal distribution. Let z_γ be the γ quantile of the $\mathsf{N}\left(0,1\right)$ distribution; that is, $\Phi(z_\gamma) = \gamma$, where Φ denotes the standard normal cumulative distribution function. Then,

$$\mathbb{P}\left(\ell \in \widehat{\ell} \pm z_{1-\gamma/2} \frac{\sigma}{\sqrt{N}}\right) \approx 1 - \gamma,$$

holds, and $\widehat{\ell} \pm z_{1-\gamma/2} \frac{\sigma}{\sqrt{N}}$ defines a confidence interval for the point estimate $\widehat{\ell}$. The width of this interval is given by

$$w_a = 2 z_{1-\gamma/2} \frac{\sigma}{\sqrt{N}}.$$

When dealing with very small values, say $\ell = 10^{-11}$, an absolute width of this interval, for example, a reasonable 5% ($w_a = 0.05$) is meaningless and,

one should instead consider a relative width defined by $w_r = w_a/\widehat{\ell}$. We can write the confidence interval in terms of w_r as $\widehat{\ell}(1 \pm w_r/2)$. This brings us to standard measure of rare-event estimator efficiency called relative error (RE) of an estimator $\widehat{\ell}$ which is defined as $\sqrt{\mathrm{Var}(\widehat{\ell})}/\mathbb{E}(\widehat{\ell})$, [28]. The RE of the CMC estimator $\widehat{\ell}$ is thus

$$\mathrm{RE}\left(\widehat{\ell}\right) = \frac{\sqrt{\mathrm{Var}\left(\widehat{\ell}\right)}}{\mathbb{E}\left(\widehat{\ell}\right)} \underbrace{=}_{\text{(A.1),(A.2)}} \frac{\sqrt{\frac{\ell(1-\ell)}{N}}}{\ell}.$$

Note that the relative width is just proportional to the RE: $w_r = 2z_{1-\gamma/2}\mathrm{RE}$. Proceeding with the rare-event setting, for which it holds that $\ell \ll 1$, we see that

$$\mathrm{RE}\left(\widehat{\ell}\right) = \frac{\sqrt{\frac{\ell(1-\gamma)}{N}}}{\ell} \approx \frac{1}{\sqrt{N\ell}}. \tag{A.3}$$

The above equation imposes a serious challenge, as illustrated in the following Example Appendix A.1.

Example Appendix A.1 (Sample size under rare-event setting). Let $\ell = 10^{-11}$, and suppose that we are interested in a modest 10% RE. It will not be very hard to verify from (A.3), that the required number of experiments N is about 10^{13}. Such N is unmanageable in the sense of computation effort, and one needs to resort to variance minimization techniques.

This discussion implies that the Monte Carlo algorithm efficiency is measured via the so-called coefficient of variation (CV) which is given by $\mathrm{CV} = \frac{\sqrt{\mathrm{Var}(Z)}}{\mathbb{E}(Z)}$, since the number of independent samples N that is required for adequate estimation is proportional to the CV. In other words, the smaller the CV the better, since the RE is given by $\mathrm{CV}/\sqrt{N}$. Note that the CV in Example Appendix A.1 is approximately equal to 3.16×10^5.

Acknowledgement

This work was supported by the Australian Research Council Centre of Excellence for Mathematical & Statistical Frontiers, under grant number CE140100049.

References

[1] R. Skjong, C. G. Soares, Safety of maritime transportation, Rel. Eng. & Sys. Safety 93 (9) (2008) 1289–1291.

[2] E. OBrien, F. Schmidt, D. Hajializadeh, X.-Y. Zhou, B. Enright, C. Caprani, S. Wilson, E. Sheils, A review of probabilistic methods of assessment of load effects in bridges, Structural Safety 53 (0) (2015) 44 – 56.

[3] K. Xie, R. Billinton, Tracing the unreliability and recognizing the major unreliability contribution of network components, Rel. Eng. & Sys. Safety 94 (5) (2009) 927–931.

[4] U. J. Na, M. Shinozuka, Simulation-based seismic loss estimation of seaport transportation system, Rel. Eng. & Sys. Safety 94 (3) (2009) 722–731.

[5] E. Zio, Reliability engineering: Old problems and new challenges, Rel. Eng. & Sys. Safety 94 (2) (2009) 125–141.

[6] D. P. Kroese, T. Taimre, Z. I. Botev, Handbook of Monte Carlo methods, John Wiley & Sons, New York, 2011.

[7] I. B. Gertsbakh, Y. Shpungin, Network Reliability and Resilience, Springer Briefs in Electrical and Computer Engineering, Springer, New York, 2011.

[8] D. R. Karger, A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem, in: Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing, STOC '95, ACM, New York, NY, USA, 1995, pp. 11–17.

[9] L. G. Valiant, The complexity of enumeration and reliability problems, SIAM Journal on Computing 8 (3) (1979) 410–421.

[10] R. M. Karp, M. Luby, Monte-Carlo algorithms for enumeration and reliability problems, in: Proceedings of the 24th Annual Symposium on Foundations of Computer Science, SFCS '83, IEEE Computer Society, Washington, DC, USA, 1983, pp. 56–64.

[11] A. Shafieezadeh, B. R. Ellingwood, Confidence intervals for reliability indices using likelihood ratio statistics, Structural Safety 38 (2012) 48 – 55.

[12] R. E. Barlow, F. Proschan, L. C. Hunter, Mathematical Theory of Reliability, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1996.

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

[14] I. B. Gertsbakh, Y. Shpungin, Models of network reliability: analysis, combinatorics, and Monte Carlo, CRC Press, New York, 2009.

[15] F. J. Samaniego, On Closure of the IFR Class Under Formation of Coherent Systems, IEEE Transactions on Reliability 34 (1985) 69–72.

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

[17] Z. I. Botev, P. L'Ecuyer, B. Tuffin, Modeling and estimating small unreliabilities for static networks with dependent components, in: Proceedings of SNA&MC 2013: Supercomputing in Nuclear Applications and Monte Carlo, 2013.

[18] C. Walter, Moving particles: A parallel optimal multilevel splitting method with application in quantiles estimation and meta-model based algorithms, Structural Safety 55 (0) (2015) 10 – 25.

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

[20] H. Kahn, T. E. Harris, Estimation of particle transmission by random sampling, National Bureau of Standards Applied Mathematics Series 12 (1951) 27–30.

[21] Z. I. Botev, D. P. Kroese, Efficient Monte Carlo simulation via the Generalized Splitting method, Statistics and Computing 22 (2012) 1–16.

[22] R. Y. Rubinstein, A. Ridder, R. Vaisman, Fast Sequential Monte Carlo Methods for Counting and Optimization, John Wiley & Sons, New York, 2013.

[23] M. J. J. Garvels, The splitting method in rare event simulation, Ph.D. thesis, Universiteit Twente, Enschede (October 2000).

[24] P. Glasserman, P. Heidelberger, P. Shahabuddin, T. Zajic, Splitting for rare event simulation: Analysis of simple cases., in: Winter Simulation Conference, 1996, pp. 302–308.

[25] W. Feller, An Introduction to Probability Theory and Its Applications, Vol. 1 of Wiley mathematical statistics series, John Wiley & Sons, New York, 1968.

[26] S. Ross, Introduction to probability models, 9th Edition, Academic Press, New York, 2007.

[27] J. S. Liu, Monte Carlo strategies in scientific computing, Springer, New York, Berlin, Heidelberg, 2008.

[28] R. Y. Rubinstein, D. P. Kroese, Simulation and the Monte Carlo Method, 2nd Edition, John Wiley & Sons, New York, 2008.