Permutational Methods for Performance Analysis of Stochastic Flow Networks

Ilya Gertsbakh [a], Reuven Rubinstein[b] [1],
Yoseph Shpungin [c] and Radislav Vaisman [d]

[a] Department of Mathematics,
Ben Gurion University, Beer-Sheva 84105, Israel
elyager@bezeqint.net

[b] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
ierrr01@ie.technion.ac.il,

[c] Department of Software Engineering,
Shamoon College of Engineering, Beer-Sheva 84105, Israel
yosefs@sce.ac.il

[d] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
slvaisman@gmail.com

March 18, 2013

[1]Corresponding author (http://iew3.technion.ac.il/Home/ Users/ierrr01.phtml)

[1†] This research was supported by the BSF (Binational Science Foundation, grant No 2008482)

Abstract

In this paper we show how the permutation Monte Carlo method, originally developed for reliability networks, can be successfully adapted for stochastic flow networks, and in particular for estimation of the probability that the maximal flow in such a network is above some fixed level, called the *threshold*. A stochastic flow network is defined as one, where the edges are subject to random failures. A failed edge is assumed to be erased (broken) and, thus, not able to deliver any flow. We consider two models; one where the edges fail with the same failure probability and another where they fail with different failure probabilities. For each model we construct a different algorithm for estimation of the desired probability; in the former case it is based on the well known notion of the D-spectrum and in the later one - on the permutational Monte Carlo. We discuss the convergence properties of our estimators and present supportive numerical results.

Keywords. Permutation, Performance Analysis, Reliability Networks, Stochastic Flow Networks.

Contents

1 Introduction **3**

2 Max Flow with Equal Failure Probabilities **5**

2.1 D-spectrum and its Properties 5

2.2 Estimation of D-spectrum and $\mathbb{P}(DOWN; q)$ 8

2.3 Numerical Results 10

3 Max Flow with Non-Equal Edge Failure Probabilities **15**

3.1 Transformation of the Static Flow Model into a Dynamic 15

3.2 Permutational Algorithm for Estimating $\mathbb{P}(DOWN; \mathbf{q})$ . . 16

3.3 Numerical Results 19

4 Extension to Random Capacities **20**

4.1 Numerical Results 21

5 Concluding Remarks and Further Research **25**

1 Introduction

The purpose of this paper is to investigate the probabilistic properties of the maximal flow in a network with randomly failing edges. Edge failure means that it is erased (broken) and is not able to deliver any flow. Because of the randomness of these failures, the maximum flow from the source to the sink is also a random variable. We call such a network, the *stochastic flow network*.

Before proceeding let us define formally our network. The network is a triple $\mathbf{N} = (\mathbf{V}, \mathbf{E}, \mathbf{C})$, where $\mathbf{V}$ is the set of vertices (nodes), $|\mathbf{V}| = n$, $\mathbf{E}$ is the set of edges, $|\mathbf{E}| = m$, and $\mathbf{C}$ is the set of edge capacities $\mathbf{C} = (c_1; \dots; c_m)$, where c_i is an item of type $c_i = \{(a, b), w_i\}$, where w_i is the maximal flow capacity from node a to node b along the directed edge (a, b). In case that there are directed edges from a to b and from b to a, these edges get different numbers, say r and s, and $\mathbf{C}$ will contain two items of the above type: $c_r = \{(a, b), w_r\}$ and $c_s = \{(b, a), w_s\}$.

Denote by s and t the source and sink nodes of the network. Denote next by M the maximum flow when all edges are operational. Note that there exist an extensive literature with several fast polynomial time algorithms for finding the maximum flow in networks with perfect edges [1]. Unless stated otherwise we shall use the Goldberg-Rao algorithm [13] with the complexity $\mathcal{O}(min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E|\ln(\frac{|V|^2}{|E|})\ln(U))$, where U is the largest edge capacity in the network.

The main goal of this paper is to obtain the probability that the maximal flow in a stochastic flow network is below some fixed level $\Phi = \gamma M,\ (\gamma < 1)$, called the *threshold*. We say that the network is in $DOWN$ state if its maximal flow is below Φ, otherwise it is in UP state. It is important to note that if the maximal flow drops to zero, the network (s and t nodes) become disconnected. Therefore, the flow model can be viewed as a generalization of the $s-t$ connectivity in the classic reliability model [9], which is based on the *permutational Monte Carlo* (PMC) method.

There exists a vast literature on stochastic flow networks with a number of clever algorithms. For a good survey see [16] and the references therein. It is not our goal to evaluate the PMC method for flow networks versus its alternatives, but rather to show the beauty of this method, discuss its convergence properties and present supportive numerical results.

We consider the following two models:

Model 1. All edges fail independently with the same failure probability q. For this model our goal is to find the probability $\mathbb{P}(DOWN; q)$ that the network is $DOWN$ as a function of q, $0 < q \leq 1$.

Model 2. All edges fail independently, with arbitrary failure probabilities $q_1, \ldots, q_m$. For this model our goals is to find the probability $\mathbb{P}(DOWN; \boldsymbol{q})$ for fixed vector $\boldsymbol{q} = (q_1, \ldots, q_m)$.

The rest of the paper is organized as follows. In Section 2 we consider **Model 1** and derive a closed expression for the function $\mathbb{P}(DOWN; q)$. It is based on the D-spectrum, which represents the distribution of the so-called *anchor* for randomly generated permutations and which is approximated via a Monte Carlo procedure. The D-spectrum has been widely used in the literature on stochastic network reliability analysis [5, 4, 12, 7, 8, 9, 10, 11]. Numerically it is identical with the so-called *signature* introduced in [19] and later on independently by [5] under the name *Internal Distribution* (ID). Here we will also consider the main properties of the D-spectrum and its usefulness to stochastic flow networks, as well as present numerical results for a flow network.

Section 3 is devoted to **Model 2** and in particular to estimation of $\mathbb{P}(DOWN; \mathbf{q})$ for non equal components of the vector $\boldsymbol{q}$. Here we introduce a specially designed *evolution*, also called *construction or birth* process, first introduced in [5] and then widely used in network reliability analysis [9, 15].

We describe in detail the procedure of obtaining permutations for this process and construct an efficient Monte Carlo estimator for $\mathbb{P}(DOWN; \mathbf{q})$. A numerical example concludes this section.

Section 4 extends **Model 2** to the case of random capacities. We show that although the estimator of $\mathbb{P}(DOWN; \mathbf{q})$ is not as accurate as in the case of the fixed capacity vector $\boldsymbol{C}$, the algorithm derived for fixed $\boldsymbol{C}$ is also applicable here. A numerical example supporting our findings is presented as well.

Section 5 presents concluding remarks and some directions for further research.

2 Max Flow with Equal Failure Probabilities

Here we derive an analytic expression for $\mathbb{P}(DOWN, q)$ while considering the **Model 1**, that is for the one with equal failure probabilities of the edges. Our derivation is based on the notion of D-spectrum [9].

2.1 D-spectrum and its Properties

Denote network edges by $e_1, e_2, \dots, e_m$. Suppose that all edges are initially operational and thus the network is *UP*. Let $\boldsymbol{\pi} = (e_{i_1}, \dots, \mathrm{e}_{i_m})$ be a permutation of the network edges. Then the D-spectrum algorithm can be described as follows:

Algorithm 2.1 (**D-spectrum Algorithm**) Given a network and a set of terminal nodes s and t, execute the following steps.

1. Start turning the edges down (erase them) moving through the permutation from left to right, and check the state (*UP*/*DOWN*) of the network after each step.

2. Find the position of the first edge i_r when the network switches from *UP* to *DOWN*. This can be done, for example, by using the Goldberg-Rao maximum flow polynomial algorithm (oracle) [13]. The serial

number r of this edge of $\boldsymbol{\pi}$ is called the *anchor* of $\boldsymbol{\pi}$ and denoted as $r(\boldsymbol{\pi})$.

3. Consider the set of all $m!$ permutations and assign to each of them the probability $1/m!$

4. Define the event $A(i) = \{r(\boldsymbol{\pi}) = i\}$ and denote $f_i = \mathbb{P}(A(i))$. Obviously,
$$f_i = \frac{\#\text{ of permutations with } r(\boldsymbol{\pi}) = i}{m!}. \tag{1}$$
The set of $\{f_i\ , i = 1, \ldots, m\}$ defines a proper discrete density function. It is called the *density D-spectrum*, where "D" stands for "destruction".

5. Define the cumulative D-spectrum or simply *D-spectrum* as
$$F(x) = \sum_{i=1}^{x} f_i,\ \ x = 1, \ldots, m. \tag{2}$$

Note that Algorithm 2.1 can be speeded up by using a bisection procedure for turning edges down instead of the sequential one-by-one. This is implemented in our main permutational Algorithm 2.2 below.

The nice feature of the D-spectrum is that once $F(x)$ is available one can calculate directly the sought failure probability $\mathbb{P}(DOWN; \boldsymbol{q})$ (see (4) below). Indeed, denote by $\mathcal{N}(x)$ the number of network failure sets of size x. Note that each such set is a collection of x edges whose failure result in $DOWN$ state of the network. So, if the network is $DOWN$ when edges $A_x = \{e_{j_1}, \ldots, e_{j_x}\}$ are down (erased), and all other edges are operational, we say that A_x is a failure set of size x. It is readily seen that
$$\mathcal{N}(x) = F(x)\binom{m}{x}. \tag{3}$$

This statement has a simple combinatorial explanation: $F(x)$ is a fraction of all failure sets of size x among all subsets of size x taken randomly from the set of m components.

From (3) we immediately obtain our main result for **Model 1**.
$$\mathbb{P}(DOWN; q) = \sum_{x=1}^{m} \mathcal{N}(x) q^x (1-q)^{m-x}. \tag{4}$$

Indeed, (4) follows from the facts that

- Network is $DOWN$ if and only if it is in one of its failure states.
- For fixed q each failure set of size x has the probability $\rho_x = q^x(1-q)^{m-x}$.
- All failure sets of size x have the probability $\mathcal{N}(x)\rho_x$.

Example 2.1 Figure 1 represents a simple directed graph with $n = 3$ nodes denoted by s, b, t (s and t being the source and the sink), $m = 3$ edges denoted by sb, bt, st and a 3-dimensional flow capacity vector $\boldsymbol{C} = (sb, bt, st) = (1, 2, 2)$.

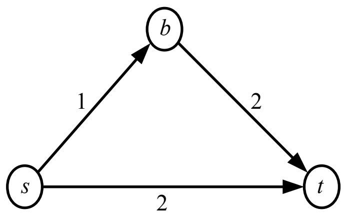

Figure 1: A network with $e_1 = (s, b), e_2 = (b, t), e_3 = (s, t)$ and capacity vector $\boldsymbol{C} = (1, 2, 2)$

It is easy to check that the maximal flow is $M = 3$. Assume that $\Phi = 2$, that is the network is $DOWN$ when the max flow drops below level 2. Let us find its D-spectrum. The total number of permutations is 3!=6. If the permutation starts with edge e_3, the anchor is $r = 1$. In permutations $(1, 3, 2)$ and $(2, 3, 1)$ $DOWN$ appears at the second step, $r = 2$. In permutations having e_3 on the third position, the flow becomes 0 at the third step. Thus $f_1 = f_2 = f_3 = 1/3$ and $F(1) = 1/3, F(2) = 2/3, F(3) = 1$. Now by (3) we obtain that $\mathcal{N}(1) = 1$, $\mathcal{N}(2) = 2$, and $\mathcal{N}(3) = 1$. Indeed, in order for the network to be in the $DOWN$ state, there is one failure set of size one containing the edge $\{e_3\}$, two failure sets of size two containing the edges $\{e_2, e_3\}$ and $\{e_1, e_3\}$ and one failure set of size 3 containing the edges $\{e_1, e_2, e_3\}$.

Simple calculations of (4) yield that $\mathbb{P}(DOWN; q) = q$.

A nice feature of (4) is that once $\mathcal{N}(x)$ is available we can calculate analytically the probability $\mathbb{P}(DOWN; q)$ simultaneously for multiple values of q.

Remark 2.1 The D-spectrum is a purely combinatorial characteristic of the network. It depends only on its topology, the edge capacity vector $\boldsymbol{C}$ and the threshold value Φ. It does *not* depend on the probabilistic mechanism which governs edge failure.

Note that instead of the destruction process one can use its dual version, the so-called *construction* one. In the latter we start the system at *DOWN* state and turn the edges from down to up one-by-one (or using bisection) until the system becomes *UP*. We shall use the construction process in Section 3.

Since network failure probability is a monotone function of its component reliability, we immediately obtain the following

Corollary Let edges fail independently, and edge i fails with probability q_i. Suppose that for all i, $q_i \in [q_{min}, q_{max}]$. Then, obviously,

$$\mathbb{P}(DOWN; q_1, \ldots, q_m) \in [\mathbb{P}(DOWN; q_{min}), \mathbb{P}(DOWN; q_{max})] \qquad (5)$$

This corollary may be useful for the case where exact information about edge failure probabilities is not available and the only statement we can be made is that the edges fail independently and that their failure probabilities lie within some known interval.

2.2 Estimation of D-spectrum and $\mathbb{P}(DOWN; q)$

For $m \leq 10$, the total number of permutations $m!$ is not too large, and the probabilities $f_i,\ i = 1, \ldots, m$ and $\mathbb{P}(DOWN; q)$ might be computed by full enumeration. For $m > 10$ we need to resort to Monte Carlo simulation. In our numerical examples below we shall show that with a sample size of $N = 10^6$ of permutations one can estimate $\mathbb{P}(DOWN : q)$ with relative error not exceeding 2% for flow networks with the number of edges $m = 200 - 300$. Note that the most time consuming part of the simulation process is to check *after edge destruction* whether or not the system switches from *UP* to *DOWN*. As mentioned this can be done by a maximum flow algorithm (oracle), and in particular by the Goldberg-Rao algorithm, which finds the location of permutation anchor in $\mathcal{O}(min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \ln(\frac{|V|^2}{|E|}) \ln(U))$ operations.

The Monte Carlo estimators of $F(x)$ and $\mathbb{P}(DOWN; q)$ is straightforward. Basically we apply Algorithm 2.1 N times. During each replication we find the anchor $r(\boldsymbol{\pi})$ of $\boldsymbol{\pi}$. In analogy to (1) we estimate the density $f(x),\ \ x = 1, \dots, m$ as follows

$$\widehat{f}(x) = \frac{\#\text{ of permutations with } r(\boldsymbol{\pi}) = x}{N}. \tag{6}$$

Note that (6) differs from (1) that $m!$ is replaced by N. Note also that $\widehat{f}(x)$ represents a histogram of $f(x)$ in (1).

The corresponding estimators of $F(x)$ and $\mathbb{P}(DOWN; q)$ (see (2) and (4)) are

$$\widehat{F}(x) = \sum_{i=1}^{x} \widehat{f}_i,\ \ x = 1, \dots, m \tag{7}$$

and

$$\widehat{\mathbb{P}}(DOWN; q) = \sum_{x=1}^{m} \widehat{F}(x) \binom{m}{x} q^x (1-q)^{m-x}, \tag{8}$$

respectively.

Below we present our main algorithm for estimating $F(x)$ and $\mathbb{P}(DOWN; q)$.

Algorithm 2.2 **(Main D-spectrum Algorithm for Estimating** $F(x)$ **and** $\mathbb{P}(DOWN; q)$**)**

Given a network and a set of terminal nodes s and t, execute the following steps.

1. Simulate a random permutation $\boldsymbol{\pi} = (\pi_1, \dots, \pi_m)$ of the edges $1, \dots, m$.
2. Set $low = 1$ and $high = m$
3. Set $b = low + \lceil \frac{high - low}{2} \rceil$
4. Consider $\pi_1, \dots, \pi_b$ and $\pi_1, \dots, \pi_{b+1}$ and use Goldberg-Rao algorithm to check if the network changed its state from *UP* to *DOWN* at index $b+1$ and b respectively.
   If so, denote by $r = r(\boldsymbol{\pi})$ the final number of the *anchor* of $\boldsymbol{\pi}$, corresponding to the non-operational network , output $r(\boldsymbol{\pi}) = b$ as

the anchor at which the network is in $DOWN$ state and go to step 5.

If the network state at b is still UP set $high = b$ else, if for both b and $b+1$ the network is in the $DOWN$ state, set $low = b$ and repeat step 3.

5. Output $r(\boldsymbol{\pi}) = b$ as the anchor at which the network is in $DOWN$ state.

6. Repeat steps 1-5 N times and deliver $\widehat{F}(x)$ and $\widehat{\mathbb{P}}(DOWN; q)$ as per (7) and (8), respectively.

2.3 Numerical Results

Below we present simulation results for the D-spectrum and $\mathbb{P}(DOWN; q)$ for the following two models (i) *dodecahedron graph* and (ii) *Erdos - Renyi* graph.

(i) *Dodecahedron graph* with $|V| = 20$, $|E| = 54$ is depicted in Figure 2. We set $s = 1$ ant $t = 10$.

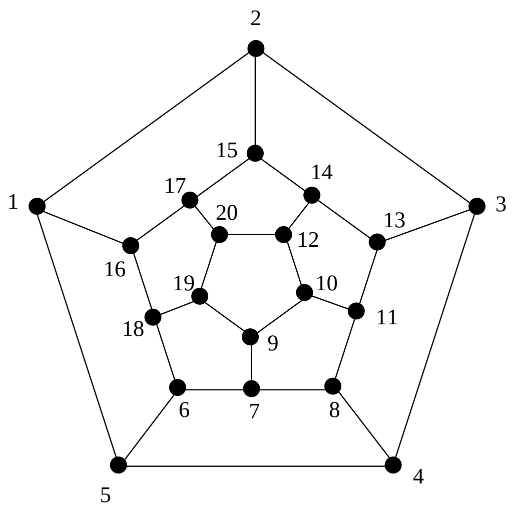

Figure 2: The dodecahedron graph.

Table 1 presents the values of the edge capacities $c_1, \ldots, c_{54}$. They where generated (for each edge independently) using a discrete uniform pdf $\mathcal{U}(5, 10)$.

Table 1: Edge capacities for the dodecahedron graph

$e=(a,b)$	$c(e)$	$e=(a,b)$	$c(a,b)$	$e(a,b)$	$c(a,b)$	$e=(a,b)$	$c(a,b)$
(1,2)	5	(18,6)	8	(12,10)	7	(19,18)	5
(1,16)	6	(18,19)	10	(9,10)	7	(7,6)	8
(1,5)	5	(6,7)	8	(3,2)	5	(8,4)	9
(2,3)	9	(4,8)	5	(15,2)	7	(14,13)	7
(2,15)	9	(13,14)	7	(17,16)	8	(11,13)	7
(16,17)	6	(13,11)	6	(18,16)	7	(12,14)	6
(16,18)	7	(14,12)	6	(6,5)	8	(19,20)	5
(5,6)	6	(20,19)	5	(4,5)	8	(12,20)	9
(5,4)	8	(20,12)	10	(4,3)	9	(9,19)	9
(3,4)	10	(19,9)	9	(13,3)	5	(8,7)	6
(3,13)	6	(7,8)	7	(17,15)	7	(9,7)	6
(15,17)	10	(7,9)	9	(14,15)	8	(11,8)	7
(15,14)	7	(8,11)	9	(20,17)	6		
(17,20)	6	(11,10)	8	(6,18)	8		

Using the Goldberg-Rao algorithm we found that the maximum flow with the perfect edges equals $M = 16$. For our simulation studies we set the threshold level $\Phi = 14$. In all our experiments below we took $N = 50,000$ samples.

Table 2 presents the D-spectrum estimator $\widehat{F}(x)$ as function of x for the dodecahedron graph with $\Phi = 14$ based on $N = 5 \cdot 10^4$ replications.

Table 2: D-spectrum estimator $\widehat{F}(x)$ for the dodecahedron graph with $\Phi = 14$

x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$
1	0.0547	15	0.8899	29	0.99999	43	1
2	0.1148	16	0.9199	30	0.99999	44	1
3	0.1780	17	0.9435	31	0.99999	45	1
4	0.2475	28	0.9606	32	1	46	1
5	0.3212	19	0.9732	33	1	47	1
6	0.3993	20	0.9822	34	1	48	1
7	0.4804	21	0.9882	35	1	49	1
8	0.5592	22	0.9924	36	1	50	1
9	0.6359	23	0.9952	37	1	51	1
10	0.7072	24	0.9969	38	1	52	1
11	0.7701	25	0.9982	39	1	53	1
12	0.8263	26	0.9990	40	1	54	1
13	0.8721	27	0.9995	41	1		
14	0.9086	28	0.9997	42	1		

It follows from Table 2 that the network fails with probability equal at least 0.7, 0.9 and 0.99, when the number x of failed edges exceeds 10, 14, and 22, respectively. It fails with probability one when x exceeds 31.

Recall that once $\widehat{F}(x)$ is available one can calculate analytically the estimator $\widehat{\mathbb{P}}(DOWN;q)$ of the true probability $\mathbb{P}(DOWN;q)$ for any q by applying (8).

Table 3 presents $\widehat{\mathbb{P}}(DOWN;q)$ for different values of q. It also present the corresponding relative error RE based on $K = 10$ independent runs of the entire algorithm. The RE was calculated as

$$RE = \frac{S}{\widetilde{\ell}}, \tag{9}$$

where

$$\widehat{\ell} = \widehat{\mathbb{P}}(DOWN;q),\ S^2 = \frac{1}{K-1}\sum_{i=1}^{K}(\widehat{\ell}_i - \widetilde{\ell})^2 \text{ and } \widetilde{\ell} = \frac{1}{\mathrm{K}}\sum_{\mathrm{i}=1}^{\mathrm{K}}\widehat{\ell}_{\mathrm{i}}.$$

Table 3: $\widehat{\mathbb{P}}(DOWN;q)$ and RE for different values of q for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN;q)$	2.99E-06	2.99E-05	2.99E-04	3.00E-03	3.05E-02	3.57E-01	5.54E-01
RE	1.84E-02	2.08E-02	1.84E-02	1.79E-02	1.41E-02	4.50E-03	2.96E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN;q)$ equals to 6.6 seconds.

It follows from Table 3 that

- In order to guarantee network reliability $1 - \widehat{\mathbb{P}}(DOWN;q)$ =0.9, the probability q of edges failure must not exceed 0.1.
- A sample $N = 5 \cdot 10^4$ guaranties relative error $\leq 1.9\%$ for all q scenarios.

(ii) *Erdos - Renyi* graph, named for Paul Erdos and Alfred Renyi. Our graph was generated according to what is called the $G(n,p)$ Erdos - Renyi random graph [6].

In the $G(n,p)$ model, (n is fixed) a graph is constructed by connecting nodes randomly. Each edge is included in the graph with probability p independent from every other edge. Equivalently, all graphs with n nodes and m edges have the same probability

$$p^m(1-p)^{\binom{n}{2}-m}.$$

A simple way to generate a random graph in $G(n,p)$ is to consider each of the possible $\binom{n}{2}$ edges in some order and then independently add each edge to the graph with probability p. Note that the expected number of edges in $G(n,p)$ is $p\binom{n}{2}$, and each vertex has expected degree $p(n-1)$. Clearly as p increases from 0 to 1, the model becomes more dense in the sense that is it is more likely that it will include graphs with more edges than less edges.

We considered an instance of Erdos-Renyi $G(n,p)$ random graph with $p = 0.1$ and $|V| = 35$ vertices. While generating it we obtained $|E| = 109$ connected edges. We set $s = 1$ and $t = 35$.

Similar to the dodecahedron graph, each capacity $c_1, \ldots, c_{109}$ was generated independently using a discrete uniform $\mathcal{U}(5,10)$ pdf. Using the Goldberg-Rao algorithm we found that the maximum flow with the perfect edges equals $M = 30$. For our simulation studies we set the threshold level $\Phi = 27$. Again, as before, we set $N = 50,000$.

Tables 4 and 5 present data similar to these of Tables 2 and 3 for the above Erdos-Renyi graph with $|V| = 35$ vertices and $|E| = 109$. Note again that each value of $\widehat{\mathbb{P}}(DOWN; q)$ and the corresponding values RE were calculated based on 10 independent runs. The results of the tables are self explanatory.

Table 4: The estimator $\widehat{F}(x)$ of the D-spectrum for the Erdos-Renyi graph with $\Phi = 27$

x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$
1	0.0354	31	0.8931	61	0.99994	91	1
2	0.0722	32	0.9074	62	0.99996	92	1
3	0.1078	33	0.9200	63	0.99996	93	1
4	0.1436	34	0.9312	64	0.99997	94	1
5	0.1786	35	0.9414	65	0.99997	95	1
6	0.2122	36	0.9504	66	0.99998	96	1
7	0.2457	37	0.9584	67	0.99998	97	1
8	0.2798	38	0.9655	68	1	98	1
9	0.3128	39	0.9716	69	1	99	1
10	0.3469	40	0.9769	70	1	100	1
11	0.3785	41	0.9814	71	1	101	1
12	0.4100	42	0.9851	72	1	102	1
13	0.4420	43	0.9884	73	1	103	1
14	0.4735	44	0.9908	74	1	104	1
15	0.5059	45	0.9928	75	1	105	1
16	0.5362	46	0.9944	76	1	106	1
17	0.5669	47	0.9955	77	1	107	1
18	0.5955	48	0.9967	78	1	108	1
19	0.6239	49	0.9974	79	1	109	1
20	0.6526	50	0.9982	80	1		
21	0.6812	51	0.9987	81	1		
22	0.7076	52	0.9990	82	1		
23	0.7319	53	0.9992	83	1		
24	0.7563	54	0.9995	84	1		
25	0.7797	55	0.9996	85	1		
26	0.8015	56	0.9997	86	1		
27	0.8223	57	0.9998	87	1		
28	0.8422	58	0.9999	88	1		
29	0.8608	59	0.9999	89	1		
30	0.8777	60	0.9999	90	1		

Table 5: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the Erdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	4.05E-06	4.05E-05	4.05E-04	4.04E-03	3.98E-02	3.74E-01	5.45E-01
RE	1.68E-02	1.68E-02	1.67E-02	1.57E-02	9.54E-03	3.36E-03	2.32E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 22.4 seconds.

All numerical experiments were performed on Intel Core $i5$ 650 3.20

GHz CPU having GB RAM.

We also estimated $\mathbb{P}(DOWN; q)$ for the Erdos-Renyi models with several hundreds edges. For example, setting $|V| = 55$ and $p = 0.1$ we generated a random graph with the number of connected edges $|E| = 313$. We found that

- $M = 28$ and we set $\Phi = 25$.
- In order to provide relative error RE ≤ 0.02 we need to take the sample size $N = 6 \cdot 10^5$

The CPU time was about 400 seconds.

3 Max Flow with Non-Equal Edge Failure Probabilities

The simplest way to estimate $\mathbb{P}(DOWN, \boldsymbol{q})$ for different failure probability vector $\boldsymbol{q} = (q_1, \ldots, q_m)$ is to simulate the state vector $\boldsymbol{X} = (X_1, \ldots, X_m)$ of edges $i_1, \ldots, i_m$ failure from $\text{Ber}(\boldsymbol{q})$ distribution with independent components and then calculate for each realization of $\boldsymbol{X} = (X_1, \ldots, X_m)$ the maximum flow in the network and the corresponding network state. This naive Monte Carlo procedure is extremely time consuming, however.

To overcome this difficulty we shall adopt here the *evolution* process of Elperin, Gertsbakh and Lomonosov [5] originally developed for reliability network estimation.

3.1 Transformation of the Static Flow Model into a Dynamic

The main idea of the evolution process is to replace a failing edge i having a Bernoulli $\text{Ber}(q_i)$ distribution by one with an exponentially distributed *birth time* τ_i with birth rate $\lambda_i = -\ln(q_i)$. More specifically, at time $t = 0$ edge i starts its birth process which terminates at some random moment τ_i. Once this edge is "born" it stays forever in up state. The probability that this event happens before time $t = 1$ is

$$\mathbb{P}(\tau_i \leq 1) = 1 - \exp(-\lambda_i \cdot 1) = 1 - \exp(\ln q_i) = 1 - q_i = p_i. \qquad (10)$$

Thus, if we take a snapshot of the state of all edges at time instant $t = 1$ we can see the static picture in the sense that edge i will be up with probability $p_i = 1 - q_i$ and down with probability q_i.

With this in mind, we can represent the edge birth process as one evolving in time in a form of a sequence of random sequential births

$\{Y_j(k)\}$, $j = 1, 2, \ldots, m$, where $Y_j(\cdot)$ is the instant of the $j - th$ birth, and $k,\ k = 1, \ldots, m$ is the number of the born edge. Then the whole birth history can be represented by the following sequence

$$0 < Y_1(k_1) < Y_2(k_2) < ... < Y_j(k_j) < ... < Y_m(k_m). \tag{11}$$

Note that here, in contrast to **Model 1**

- $\boldsymbol{\pi} = (k_1, k_2, \ldots, k_m)$ represents an *ordered* sample reflecting the order of edge birth sequence and not an arbitrary random permutation.
- It describes a *construction* process instead of a destruction one, namely starting with the network in *DOWN* state and ending in *UP* state.

Since all birth times are exponential, it is easy to generate a single birth "history", also called the sample path or trajectory.

3.2 Permutational Algorithm for Estimating $\mathbb{P}(DOWN; \mathbf{q})$

Before presenting the algorithm for estimating $\mathbb{P}(DOWN; \mathbf{q})$ we make the following observations.

1. The time to the first birth is a random variable ξ_1 distributed exponentially with parameter $\Lambda_1 = \sum_i^m \lambda_i$. It follows that
   - The first birth will occur at edge k_1 with probability λ_{k_1}/Λ_1.
   - At the instant ξ_1 all edges except the born one are not yet born, but because of the memoryless property they behave as if they all are born at that instant ξ_1. This means that the time interval ξ_2 between the first and second births, given that edge k_1 is already born, is distributed exponentially with parameter $\Lambda_2 = \Lambda_1 - \lambda_{k_1}$.
2. The second birth will occur at edge k_2 with probability λ_{k_2}/Λ_1, and so on. Clearly that given the edges $k_1, k_2, \ldots, k_s$ are already born,

the time $\xi_{k_{s+1}}$ to the next birth will be exponentially distributed with parameter Λ_{s+1}, which equals the sum of λ-s of all non born edges, and with probability λ_j/Λ_{s+1} the next birth will occur at edge j.

Based on the above we can design a simple permutational Monte Carlo algorithm for estimating $\mathbb{P}(DOWN; \mathbf{q})$.

Algorithm 3.1 (**Permutational Algorithm for Estimating** $\mathbb{P}(DOWN; \mathbf{q})$) Given a network and a set of terminal nodes s and t, execute the following steps.

1. Generate the birth times of the edges $\tau_1, \ldots, \tau_m$, with $\tau_i \sim \exp(\lambda_i)$, and rate $\lambda_i = -\ln(q_i)$.

2. Arrange these birth times in increasing order obtaining the order statistics $\tau_{(1)}, \ldots, \tau_{(m)}$, then ignore the actual birth times retaining only the order in which the edges are born. This yields a permutation $\pi = (\pi(1), \ldots, \pi(m))$ of the edge numbers under which their birth times occur.

3. Similar to Algorithm 2.2, use a combination of the Goldberg-Rao algorithm (oracle) and the bisection procedure to allocate the edge r_α whose birth brings the network *UP*. Call the sequence $(\pi_1, \ldots, \pi_\alpha) = \omega$.

4. Calculate analytically $\mathbb{P}(\xi_1 + \xi_2 + ... + \xi_\alpha \leq 1|\omega)$. Note that $\xi_1 + \xi_2 + ... + \xi_\alpha$ has a hypo-exponential distribution. Note also that the event $\sum_{i=1}^{\alpha} \xi_i \leq 1$ is equivalent to the network being *UP*, by the definition of α.

5. Repeat N times Steps 1- 4, and estimate $\mathbb{P}(DOWN; \mathbf{q})$ as

$$\widehat{\mathbb{P}}(DOWN; \mathbf{q}) = 1 - \frac{1}{N} \sum_{i=1}^{N} \mathbb{P}(\sum_{j=1}^{\alpha} \xi_j \leq 1|\omega_i). \tag{12}$$

Note that

- The complexity of Algorithm 3.1 is governed by the Goldberg-Rao oracle and is therefore

$$\mathcal{O}(N \ln(m)[min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \ln(\frac{|V|^2}{|E|}) \ln(U)]) \tag{13}$$

This follows from the fact that a single run has complexity $\mathcal{O}(\ln(m)[min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E|\ln(\frac{|V^2}{|E|})\ln(U)])$ and we perform N such runs.

- As for networks reliability the relative error of the estimator $\widehat{\mathbb{P}}(DOWN; \mathbf{q})$ is uniformly bounded with respect to the λ_i values, (see [4, 7]).
- Besides the permutation Algorithm 3.1 one could apply some other alternatives, such as the turnip [5], cross-entropy [14] and splitting [2, 3, 17].

Example 3.1 Example 2.1 continued

Consider again the network of Example 1 and assume that edge e_i fails with probability q_i. Let $\Phi = 2$, that is assume that network is *UP* if the maximum flow is either 2 or 3. Figure 3 shows the tree of all five possible trajectories of the evolution process. It starts moving from the root to the *UP* state shown by double circles.

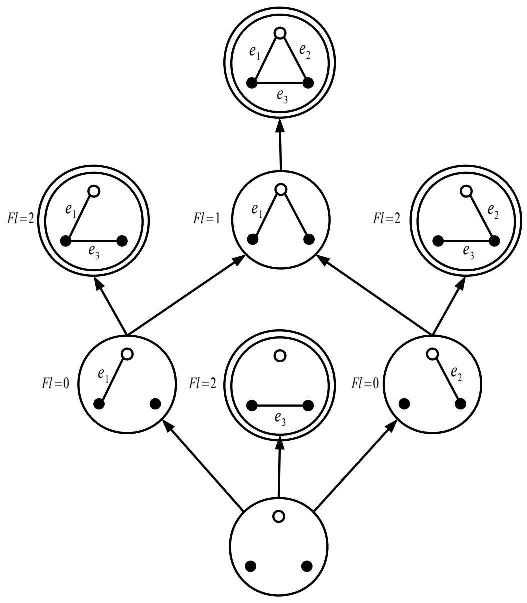

Figure 3: The evolution process for the network of Example 2.1

Let us consider one of them, namely the one corresponding to $\omega = \{1, 3\}$, meaning that the first birth occurs at edge e_1 and the second at edge e_3. This trajectory will occur with probability

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \cdot \frac{\lambda_3}{\lambda_2 + \lambda_3}.$$

The movement along this trajectory from the root to the UP state lasts $\xi = \xi_1 + \xi_2$ units of time, where $\xi_1 \sim \exp(\lambda_1+\lambda_2+\lambda_3)$ and $\xi_2 \sim \exp(\lambda_2+\lambda_3)$. Computing $\mathbb{P}(\xi \leq 1|\omega)$ is a simple task.

3.3 Numerical Results

Here we apply the permutation Algorithm 3.1 to estimate $\mathbb{P}(DOWN; \boldsymbol{q})$ for the same two models in Section 2.

Tables 6 and 7 present data similar to Tables 3 and 5 while assuming that all components of the vector $\boldsymbol{q}$ are equal. This was done purposely with view to see how the performance of the permutation Algorithm 3.1 compares with that of the D-Spectrum Algorithm 2.2.

It follows from comparison of Tables 6, 7 with Tables 3, 5 that both produce almost identical $\widehat{\mathbb{P}}(DOWN; q)$ values and they also close in terms of RE and CPU times. The main difference is that with Algorithm 2.2 we was calculated $\widehat{\mathbb{P}}(DOWN; q)$ simultaneously for different values of q using a *single* simulation, while with Algorithm 3.1 we required to calculate $\widehat{\mathbb{P}}(DOWN; q)$ separately for each q value, that is using *multiple* simulations.

Table 6: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; q)$	2.99E-06	3.02E-05	3.00E-04	3.00E-03	3.06E-02	3.57E-01	5.54E-01
RE	2.08E-02	1.94E-02	1.53E-02	2.11E-02	1.09E-02	3.13E-03	2.81E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 8.6 seconds.

Table 7: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the Edrdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; q)$	4.01E-06	4.00E-05	4.03E-04	3.99E-03	3.97E-02	3.73E-01	5.44E-01
RE	2.84E-02	1.64E-02	2.36E-02	2.74E-02	1.07E-02	3.84E-03	3.77E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 34 seconds.

Tables 8 and 9 present data similar to Tables 6 and 7, but for different values of the components of the vector $\boldsymbol{q}$. More specifically, define α_1 and α_2 to be the minimal and the maximal failure probability, respectively. Define next $\delta = (\alpha_2 - \alpha_1)/|E|$. Then, we set $q_i = \alpha_1 + i\delta$ for $i \in \{0, \ldots |E| - 1\}$. In all tables below we set $\alpha_2 = q$ and $\alpha_1 = \frac{\alpha_2}{10} = \frac{q}{10}$.

Table 8: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	3.42E-07	3.50E-06	3.50E-05	3.55E-04	3.97E-03	8.04E-02	1.52E-01
RE	1.76E-02	2.01E-02	2.16E-02	2.21E-02	2.39E-02	1.32E-02	5.23E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ was 9 seconds.

Table 9: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Edrdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	2.16E-06	2.14E-05	2.16E-04	2.13E-03	2.14E-02	2.10E-01	3.11E-01
RE	2.54E-02	1.29E-02	2.95E-02	2.49E-02	1.71E-02	5.55E-03	5.47E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ was 34.8 seconds. The results are self explanatory.

We also applied Algorithm 3.1 for Erdos-Renyi models with the size of several hundreds edges. As for Algorithm 2.2 we

- Considered the model with $p = 0.1$, $|V| = 55$ and $|E| = 313$.
- In order to keep the RE ≤ 0.05 we required to increase the sample size from $N = 5 \cdot 10^4$ to $N = 10^5$.

The CPU time was about 9 minutes for each q value.

4 Extension to Random Capacities

Algorithm 3.1 can be readily modified for the case where not only the edges are subject of failure, but the capacity vector is random as well, that is when the capacity c_k of each edge $k, k = 1, ..., m$ is a random variable independent of edge failure Ber $(\boldsymbol{q})$ distribution. We assume that the capacity random vector is unbiased with respect to the true vector $\mathbf{C}$ and has independent components with some known distribution $\mathcal{F} = (\mathcal{F}_1, \dots, \mathcal{F}_m)$. Denote the joint distribution of $\mathcal{F}_k$ and $\text{Ber}(1-q_k)$ by $\mathcal{R}_k$, that is $\mathcal{R}_k = \mathcal{F}_k \cdot \text{Ber}(1-q_k)$, $k = 1, \dots, m$ and call it the modified capacity distribution. In this case, only Step 1 of Algorithm 3.1 should be modified as

Generate the birth times $\tau_1, \dots, \tau_m$ *of the edges* $\tau_i \sim \exp(\lambda_i)$, *and put* $\lambda_i = -\ln(q_i)$. *Generate a capacity random vector* $\boldsymbol{Z} = (Z_1, \dots, Z_m)$ *according to the modified capacity distribution* $\mathcal{R} = (\mathcal{R}_1, \dots, \mathcal{R}_m)$, while the rest of Algorithm 3.1 remains the same. Note that as soon the edge birth times and the edge capacities were generated at step 1 all the remaining steps do not change and the Goldberg-Rao algorithm is applied on those capacities.

Clearly the variability of such a noise estimator $\mathbb{P}(DOWN; \boldsymbol{q})$ increases with the variability of $\boldsymbol{Z}$. Our numerical results below support this.

4.1 Numerical Results

We model each component Z_k of the random capacity vector $\boldsymbol{Z} = (Z_1, \dots, Z_m)$ by setting $Z_k = \zeta_k c_k(1 + \varepsilon\eta)$, where $\zeta_k \sim \text{Ber}(1-q_k)$ and η is a random variable with $\mathbb{E}\eta = 0$ and $\text{Var}\eta = \sigma^2$. Note that for $\varepsilon = 0$ (deterministic capacities) we have $Z_k = c_k\zeta_k$ and for perfect edges, Z_k reduces to $Z_k = c_k$. We assume for concreteness that $\eta \sim \mathcal{U}(-0.5, 0.5)$.

Before proceeding with numerical results lets us discuss the choice of the threshold level Φ for random capacities. Consider, for example the dodecahedron graph. Recall that with deterministic capacities the maximal flow was $M = 16$ and we selected $\Phi = 14$ for our simulation studies. Observe also that in the case of random capacities (even with perfect edges) the maximal flow is a random variable, which will fluctuate from replication to replication. Denote by $\boldsymbol{M} = (\mathcal{M}_1, \dots, \mathcal{M}_N)$ the maximal flow random vector corresponding to a simulation of length N. Clearly, that the variability of the components of $\boldsymbol{M} = (\mathcal{M}_1, \dots, \mathcal{M}_N)$ increases in ε and it is quite easy to chose an ε (perhaps large enough) such that many the components of $\boldsymbol{M} = (\mathcal{M}_1, \dots, \mathcal{M}_N)$ will be below some fixed threshold Φ and in particular below our earlier one, $\Phi = 14$. It follows from above that setting, say $\Phi = 14$ for large ε is meaningless.

To resolve this difficulty we propose the following adaptive algorithm based on what is called *elite* sampling [18] for finding a meaningful Φ.

Algorithm 4.1 (**Adaptive Algorithm for Fining Φ in a Flow Network with Random Capacities)**

Given a network and a set of terminal nodes s and t, execute the following steps.

1. Generate a sample of random capacity vectors $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N$. Calculate the corresponding maximum flow values $\mathcal{M}_1, \ldots, \mathcal{M}_N$ using the Goldberg-Rao algorithm.

2. Order them from the smallest to the largest. Denote the ordered values by $\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(N)}$. Write $\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(N)}$ as

$$\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(e)}; \mathcal{M}_{(e+1)}, \ldots, \mathcal{M}_{(a)}; \mathcal{M}_{(a+1)}, \ldots, \mathcal{M}_{(N)}. \tag{14}$$

   Here

   - $\mathcal{M}_{(a)} \leq \mathcal{M}_a \leq \mathcal{M}_{(a+1)}$, $\mathcal{M}_{(a)} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{M}_i$ is the sample average of $\mathcal{M}_1, \ldots, \mathcal{M}_N$.
   - $\mathcal{M}_{(e)}$ corresponds to the sample $(1-\rho)$-quantile of the sequence (14), namely $\mathcal{M}_{(e)} = \mathcal{M}_{N^e}$ and $N^e = \lceil (1-\rho)N \rceil$. In other words $\mathcal{M}_{(e)}$ corresponding to $\rho\%$ of the largest sample value in the sequence (14) starting from the left.

3. Set $\Phi = \mathcal{M}_{(e)}$ and call it the *adaptive* threshold. Note that the "distance" $|\mathcal{M}_a(\varepsilon) - \mathcal{M}_{(e)}(\varepsilon)|$ between $\mathcal{M}_a$ and $\mathcal{M}_{(e)}$ increases in ε.

If not stated otherwise we assume that the sample quantile $\rho = 0.05$.

Tables 10 and 11 present $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the dodecahedron and Erdos- Renyi graphs with $\rho = 0.05$.

Table 10: $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the dodecahedron graph with $\rho = 0.05$

ε	0.5	1	2	3	4	5	6
$\mathcal{M}_{(e)}$	15.516	15.033	14.063	13.099	12.119	11.145	10.168
$\lvert\mathcal{M}_a - \mathcal{M}_{(e)}\rvert$	0.484	0.967	1.939	2.901	3.866	4.794	5.682

Table 11: $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the Erdos-Renyi graph with $\rho = 0.05$

ε	0.5	1	2	3	4	5	6
$\mathcal{M}_{(e)}$	29.440	28.880	27.762	26.645	25.518	24.390	23.231
$\|\mathcal{M}_a - \mathcal{M}_{(e)}\|$	0.560	1.120	2.238	3.356	4.466	5.560	6.648

It follows from Tables 10 and 11 that

1. For $\varepsilon \leq 2$ one can use our earlier (deterministic) threshold values $\Phi = 14$ and $\Phi = 27$ for the dodecahedron and Erdos-Renyi graphs, respectively and still be within the 5% of elites.
2. For $\varepsilon > 2$ one should use the appropriate threshold values given in Tables 10 and 11. For example, for $\varepsilon = 3$ the thresholds are $\Phi = 13.099$ and $\Phi = 26,645$ for the dodecahedron and the Erdos-Renyi graphs, respectively and for $\varepsilon = 6$ they are $\Phi = 10.146$ and $\Phi = 23.231$, respectively.

We proceed below with the following 3 scenarios of ε: (i) $\varepsilon = 0.5$, (ii) $\varepsilon = 3$ and (iii) $\varepsilon = 6$. Note that (i) corresponds to the above case 1 ($\varepsilon \leq 2$), while (ii) and (iii) corresponds to the case 2 ($\varepsilon > 2$).

In all three experiments we set the sample size $N = 5 \cdot 10^4$.

Experiment (i), $\varepsilon = 0.5$. It follows from Tables 10 and 11 that in this case we can still use the original thresholds $\Phi = 14$ and $\Phi = 27$ chosen for the deterministic capacities for the dodecahedron and Erdos-Renyi graphs, respectively. Based on that Tables 12 and 13 present data similar to Tables 8 and 9 for $\varepsilon = 0.5$, with $\Phi = 14$ and $\Phi = 27$, respectively.

Table 12: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 0.5$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	9.76E-07	9.77E-06	9.98E-05	9.98E-04	1.02E-02	1.27E-01	2.11E-01
RE	2.48E-02	2.41E-02	2.51E-02	2.16E-02	2.02E-02	8.29E-03	5.53E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 6.53 seconds.

Table 13: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 0.5$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	2.45E-06	2.42E-05	2.45E-04	2.46E-03	2.44E-02	2.35E-01	3.43E-01
RE	1.77E-02	2.31E-02	2.34E-02	2.38E-02	2.64E-02	4.52E-03	3.84E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 26.1 seconds It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained RE $\leq 2.5\%$ in both cases.

Experiment (ii), $\varepsilon = 3$. The corresponding thresholds (see Tables 10 and 11) are $\Phi = 13.099$ and $\Phi = 26,645$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Based on that Tables 14 and 15 present data similar to Tables 12 and 13 for $\varepsilon = 3$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Table 14: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 3$ and $\Phi = 13.099$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	8.61E-07	8.58E-06	8.55E-05	8.64E-04	8.58E-03	7.87E-02	1.12E-01
RE	2.33E-02	2.59E-02	3.64E-02	2.12E-02	2.08E-02	1.68E-02	8.21E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 2.64 seconds.

Table 15: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 3$ and $\Phi = 26,645$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	1.02E-06	1.02E-05	1.01E-04	1.01E-03	1.01E-02	9.84E-02	1.43E-01
RE	2.33E-02	3.22E-02	4.09E-02	4.30E-02	1.59E-02	8.16E-03	9.98E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 26.1 seconds It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained the RE $\leq 2.5\%$ in both cases.

Experiment (iii), $\varepsilon = 6$. The corresponding thresholds (see Tables 10 and 11) are $\Phi = 10.168$ and $\Phi = 23.231$ for the dodecahedron and the

Erdos-Renyi graphs, respectively. Based on that Tables 16 and 17 present data similar to Tables 14 and 15 for $\varepsilon = 6$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Table 16: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 6$ and $\Phi = 10.168$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	6.44E-07	6.46E-06	6.51E-05	6.37E-04	6.31E-03	5.98E-02	8.55E-02
RE	3.58E-02	3.33E-02	2.15E-02	3.23E-02	2.31E-02	9.60E-03	1.38E-02

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 1.77 seconds.

Table 17: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 6$ and $\Phi = 23.231$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	8.82E-07	9.01E-06	9.01E-05	9.10E-04	9.02E-03	8.89E-02	1.31E-01
RE	4.00E-02	2.68E-02	3.13E-02	2.42E-02	2.65E-02	1.46E-02	1.07E-02

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 9.2 seconds. It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained the RE $\leq 4.0\%$ in both cases.

We also estimated $\mathbb{P}(DOWN; \boldsymbol{q})$ for the Erdos-Renyi models with several hundreds edges with different ε values. In particular we considered our previous model with $p = 0.1$, $|V| = 55$ and $|E| = 313$. We found that

- For $\varepsilon = 3$ one should set $\Phi = 24.634$ in order to insure $\rho = 0.05$ (5% elites).
- In order for the relative error RE ≤ 0.05 (with $\Phi = 24.634$), we need to increase the sample size from $N = 5 \cdot 10^4$ to $N = 5 \cdot 10^5$.

The CPU time for this scenario was about 25 minutes.

5 Concluding Remarks and Further Research

We show how the permutation Monte Carlo method originally developed for reliability networks [5] can be successfully adapted for stochastic flow networks, and in particular for estimation of the probability that the maximal flow in such a network is above some fixed level, called the *threshold*. A stochastic flow network is defined as one, where the edges are subject to random failures. A failed edge is assumed to be erased (broken) and, thus, not able to deliver any flow. We consider two models; one where the edges fail with the same failure probability and another where they fail with different failure probabilities. For each model we construct a different algorithm for estimation of the desired probability; in the former case it is based on the well known notion of the D-spectrum and in the later one - on the permutational Monte Carlo. We discuss the convergence properties of our estimators and present supportive numerical results.

One of our directions of further research will be in obtaining efficient Monte Carlo estimators based on network edge *important measures* (see [12, 9]) and their applications to stochastic flow networks with a particular emphasis on the optimal network design under some budget constraints.

Another direction will be an extension of our permutational technique to a wider class of functionals associated with network edges. The crucial property for the applicability of that methodology for computing the probability that the maximal $s-t$ flow M exceeds the threshold level Φ is the monotonicity of the maximal $s-t$ in the process of edge destruction (construction). This means that the maximal $s-t$ flow in the destruction process,can only *decrease*, or equivalently, the "birth" of new edges can lead only to a *increase* of the maximal $s-t$ flow and similarly for the construction one. But this "edge monotonicity" property holds true not only for the network flow. Consider, for example, the total weight $W_{\max}$ of the maximal spanning tree of the network. $W_{\max}$ *decreases* if the edges are sequentially eliminated. Similar monotone behavior exhibit the minimal distance between any pair of fixed nodes in the network, the total number of $s-t$ paths in it, and many other of its important combinatorial characteristics.

ACKNOWLEDGMENT

We are thoroughly grateful to anonymous Reviewer for his/her valuable constructive remarks and suggestions.

References

[1] K. Ravindra Ahuja, L. Thomas Magnanti, and B. James Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] Z. I. Botev and D. P. Kroese. Efficient monte carlo simulation via the generalized splitting method. *Statistics and Computing*, 22:1–16, 2012.

[3] Z. I. Botev, P. L'Ecuyer, G. Rubino, R. Simard, and B. Tuffin. Static network reliability estimation via generalized splitting. *INFORMS Journal on Computing*, page doi:10.1287/ijoc.1110.0493, 2012.

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

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

[6] P. Erdos and A. Rnyi. On random graphs. i. *Publicationes Mathematicae*, 6:290–297, 1959.

[7] I. Gertsbakh and Y. Shpungin. Combinatorial approaches to monte carlo estimation of network lifetime distribution. *Applied Stochastic Models in Business and Industry*, 20:49–57, 2004.

[8] I. Gertsbakh and Y. Shpungin. Network reliability importance measures: combinatorics and monte carlo based computations. *WSEAS Trans Comp*, 4:21–23, 2007.

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

[10] I. Gertsbakh and Y. Shpungin. *Network reliability design: combinatorial and Monte Carlo approach.* Proceedings of the International Conference on Modeling and Simulation, pages 88-94, Canada, 2010.

[11] I. Gertsbakh and Y. Shpungin. *Network Reliability and Resilience* -. Springer, Berlin, Heidelberg, 1st edition. edition, 2011.

[12] I. Gertsbakh and Y. Shpungin. *Spectral Approach to Reliability Evaluation of Flow Networks.* Proceedings of the European Modeling and Simulation Symposium, Vienna, pages 68-73, 2012.

[13] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[14] D. P. Kroese, K.-P. Hui, and S. Nariai. Network reliability optimization via the cross-entropy method. *IEEE Transactions on Reliability*, 56(2):275–287, 2007.

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

[16] Y. Lin. Reliability of a stochastic-flow network with unreliable branches & nodes, under budget constraints. *IEEE Transactions on Reliability*, 53(3):381–387, 2004.

[17] R. Rubinstein, R. Vaisman, and Z. Botev. Hanging edges for fast reliability estimation. Technical report, Technion, Haifa, Israel, 2012.

[18] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method, Second Edition.* John Wiley and Sons, New York, 2007.

[19] F. J. Samaniego. On closure of the ifr under formation of coherent systems. *IEEE Trans Reliab*, 34:69–72, 1985.