Static Network Reliability Estimation Under the Marshall-Olkin Copula

Pierre L'Ecuyer
Université de Montréal, Canada, and Inria–Rennes, France

joint work with

Zdravko Botev, New South Wales University, Australia
Richard Simard, Université de Montréal, Canada
Bruno Tuffin, Inria–Rennes, France

Statistics Seminar Series, UNSW, Sydney, February 2014

A static network reliability problem

A system has m components, in state 0 (failed) or 1 (operating).
System state: $\mathbf{X} = (X_1, \ldots, X_m)^t$.
Structure function: $\Phi : \{0,1\}^m \to \{0,1\}$, assumed monotone.
System is operational iff $\Phi(\mathbf{X}) = 1$.
Unreliability: $u = \mathbb{P}[\Phi(\mathbf{X}) = 0]$.

A static network reliability problem

A system has m components, in state 0 (failed) or 1 (operating).
System state: $\mathbf{X} = (X_1, \ldots, X_m)^t$.
Structure function: $\Phi : \{0,1\}^m \to \{0,1\}$, assumed monotone.
System is operational iff $\Phi(\mathbf{X}) = 1$.
Unreliability: $u = \mathbb{P}[\Phi(\mathbf{X}) = 0]$.

If we know $p(\mathbf{x}) = \mathbb{P}[\mathbf{X} = \mathbf{x}]$ for all $\mathbf{x} \in \{0,1\}^m$, in theory we can compute

$$u = \sum_{\mathbf{x} \in \mathcal{D} = \{\mathbf{X}:\Phi(\mathbf{X})=0\}} p(\mathbf{x}).$$

But the cost of enumerating $\mathcal{D}$ is generally exponential in m.

The X_i's may be dependent.

Monte Carlo (MC): Generate n i.i.d. realizations of $\mathbf{X}$, say $\mathbf{X}_1, \ldots, \mathbf{X}_n$, compute $W_i = \Phi(\mathbf{X}_i)$ for each i, and estimate u by
$\bar{W}_n = (W_1 + \cdots + W_n)/n \sim \text{Binomial}(n, u)/n \approx \text{Poisson}(nu)/n$.
Can also estimate $\text{Var}[\bar{W}_n]$ and compute a confidence interval on u.

Monte Carlo (MC): Generate n i.i.d. realizations of $\mathbf{X}$, say $\mathbf{X}_1, \ldots, \mathbf{X}_n$, compute $W_i = \Phi(\mathbf{X}_i)$ for each i, and estimate u by
$\bar{W}_n = (W_1 + \cdots + W_n)/n \sim \text{Binomial}(n, u)/n \approx \text{Poisson}(nu)/n$.
Can also estimate $\text{Var}[\bar{W}_n]$ and compute a confidence interval on u.

When u is very small (failure is a rare event), direct MC fails.
Ex: if $u = 10^{-10}$, system fails once per 10 billion runs on average.

Monte Carlo (MC): Generate n i.i.d. realizations of $\mathbf{X}$, say $\mathbf{X}_1, \ldots, \mathbf{X}_n$, compute $W_i = \Phi(\mathbf{X}_i)$ for each i, and estimate u by
$\bar{W}_n = (W_1 + \cdots + W_n)/n \sim \text{Binomial}(n, u)/n \approx \text{Poisson}(nu)/n$.
Can also estimate $\text{Var}[\bar{W}_n]$ and compute a confidence interval on u.

When u is very small (failure is a rare event), direct MC fails.
Ex: if $u = 10^{-10}$, system fails once per 10 billion runs on average.

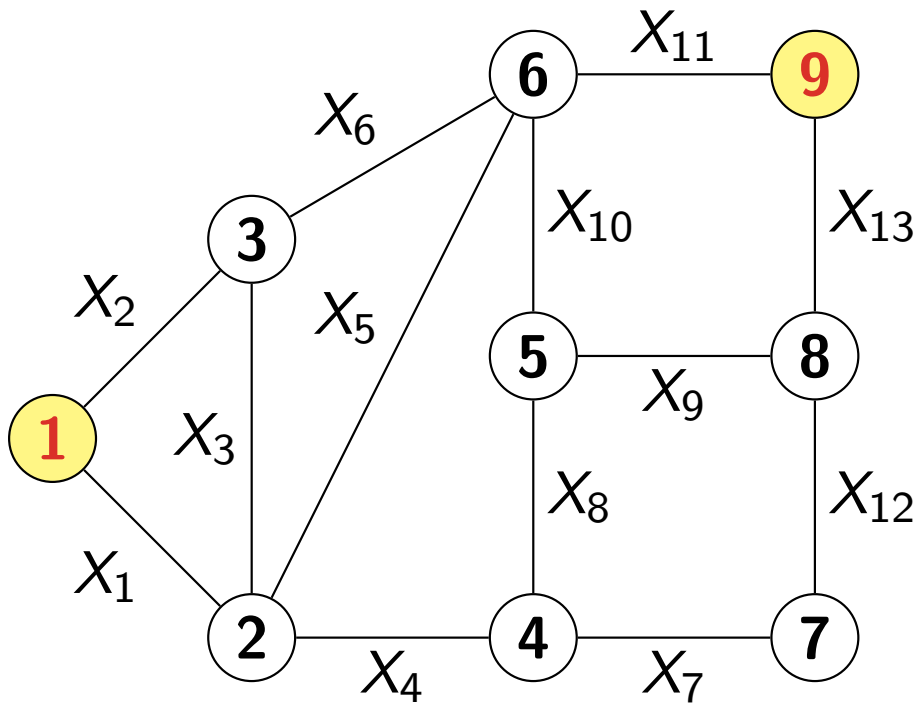
Relative error

$$\text{RE}[\bar{W}_n] \stackrel{\text{def}}{=} \frac{\sqrt{\text{MSE}[\bar{W}_n]}}{u} \stackrel{\text{here}}{=} \frac{\sqrt{1-u}}{\sqrt{nu}} \to \infty \quad \text{when } u \to 0.$$

For example, if $u \approx 10^{-10}$, we need $n \approx 10^{12}$ to have $\text{RE}[\bar{W}_n] \leq 10\%$.

We would like bounded RE (or almost) when $u \to 0$.

Although our methods apply much more generally, we focus here on the case where Φ is defined by a graph. Link i "works" iff $X_i = 1$.
The system is operational iff all the nodes in a given set $\mathcal{V}_0$ are connected.



Given $\mathbf{X}$, $\Phi(\mathbf{X})$ is easy to evaluate by graph algorithms (e.g., minimal spanning tree). Challenge: How to sample $\mathbf{X}$ effectively.

We propose methods based on (a) **conditional MC** and (b) **splitting**.

Conditional MC with auxiliary variables

[Elperin, Gertsbach, Lomonosov 1974, 1991, 1992, etc.]
Special case: the X_i's are independent with $\mathbb{P}[X_i = 0] = u_i$.

Conceptually, suppose each link i is initially failed and gets repaired at time $Y_i \sim \text{Expon}(\mu_i)$ where $\mu_i = -\ln(u_i)$. Then $\mathbb{P}[Y_i > 1] = \mathbb{P}[X_i = 0] = u_i$.
Let $\mathbf{Y} = (Y_1, \ldots, Y_m)$ and π the permutation s.t. $Y_{\pi(1)} < \cdots < Y_{\pi(m)}$.

Conditional on π, we can forget the Y_i's, add the (non-redundant) links one by one until the graph is operational, say at step C.
Data structure: forest of spanning trees. Adding a link may merge two trees.

Conditional MC with auxiliary variables

[Elperin, Gertsbach, Lomonosov 1974, 1991, 1992, etc.]
Special case: the X_i's are independent with $\mathbb{P}[X_i = 0] = u_i$.

Conceptually, suppose each link i is initially failed and gets repaired at time $Y_i \sim \text{Expon}(\mu_i)$ where $\mu_i = -\ln(u_i)$. Then $\mathbb{P}[Y_i > 1] = \mathbb{P}[X_i = 0] = u_i$.
Let $\mathbf{Y} = (Y_1, \ldots, Y_m)$ and π the permutation s.t. $Y_{\pi(1)} < \cdots < Y_{\pi(m)}$.

Conditional on π, we can forget the Y_i's, add the (non-redundant) links one by one until the graph is operational, say at step C.
Data structure: forest of spanning trees. Adding a link may merge two trees.

Permutation Monte Carlo (PMC) estimator: conditional probability that the total time for these repairs is larger than 1:

$$\mathbb{P}\left[A_1 + \cdots + A_c > 1 \mid \pi,\, C = c\right].$$

At step j, the time A_j to next repair is exponential with rate Λ_j, the sum of repair rates of all links not yet repaired. Sum is an hypoexponential.
Theorem [Gertsback and Shpungin 2010]. Gives BRE when the $u_i \to 0$.

Conditional MC with auxiliary variables

[Elperin, Gertsbach, Lomonosov 1974, 1991, 1992, etc.]
Special case: the X_i's are independent with $\mathbb{P}[X_i = 0] = u_i$.

Conceptually, suppose each link i is initially failed and gets repaired at time $Y_i \sim \text{Expon}(\mu_i)$ where $\mu_i = -\ln(u_i)$. Then $\mathbb{P}[Y_i > 1] = \mathbb{P}[X_i = 0] = u_i$.
Let $\mathbf{Y} = (Y_1, \ldots, Y_m)$ and π the permutation s.t. $Y_{\pi(1)} < \cdots < Y_{\pi(m)}$.

Conditional on π, we can forget the Y_i's, add the (non-redundant) links one by one until the graph is operational, say at step C.
Data structure: forest of spanning trees. Adding a link may merge two trees.

Permutation Monte Carlo (PMC) estimator: conditional probability that the total time for these repairs is larger than 1:

$$\mathbb{P}\left[A_1 + \cdots + A_c > 1 \mid \pi,\, C = c\right].$$

At step j, the time A_j to next repair is exponential with rate Λ_j, the sum of repair rates of all links not yet repaired. Sum is an hypoexponential.
Theorem [Gertsback and Shpungin 2010]. Gives BRE when the $u_i \to 0$.

Improvement: **turnip**; at each step, discard redundant unrepaired links.

We have

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

This formula becomes unstable when c is large and/or the Λ_j are small. The product terms are very large and have alternate signs $(-1)^{j-1}$.

Higham (2009) propose a stable method for matrix exponential. More reliable, but significantly slower.

We have

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

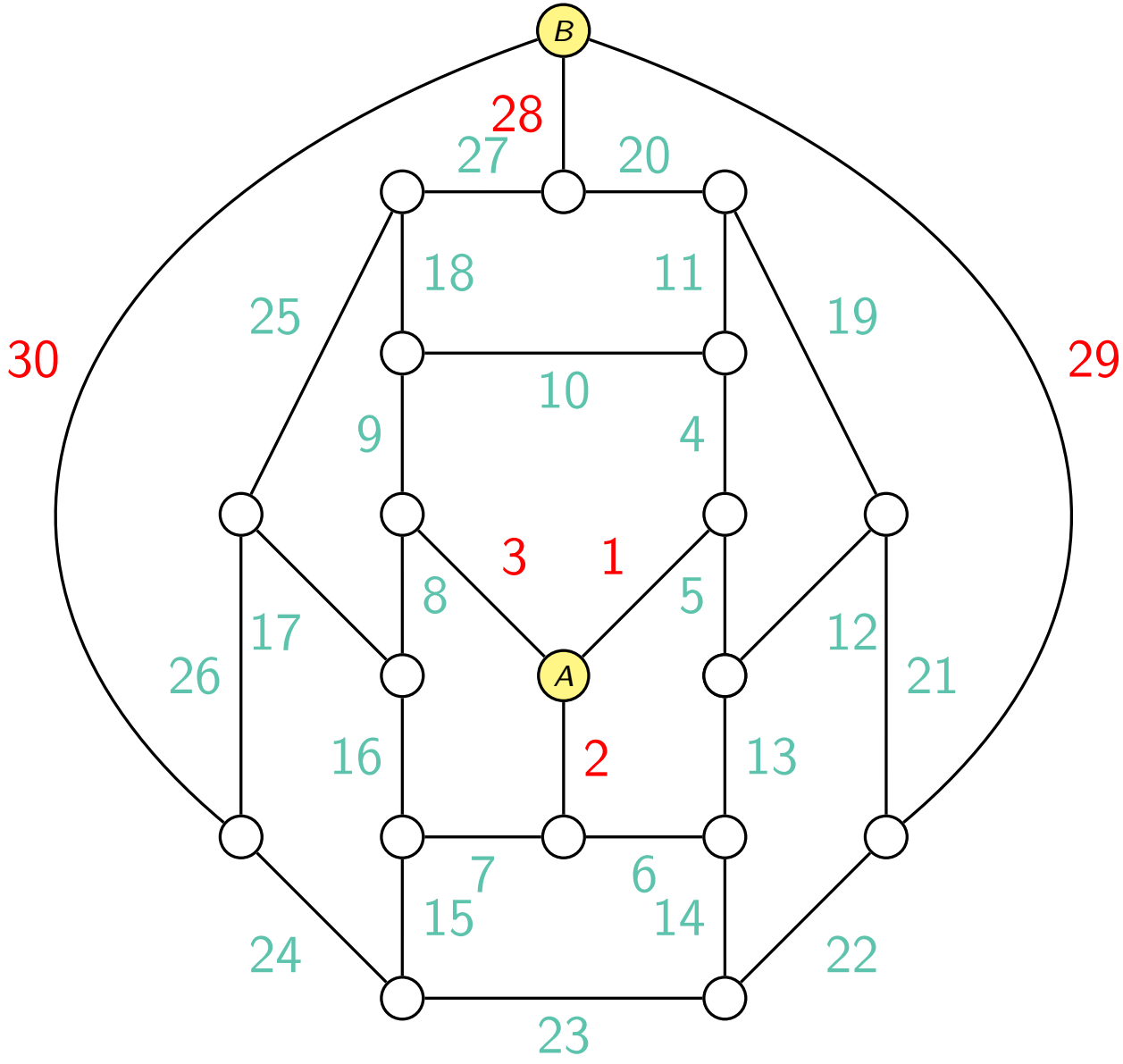
This formula becomes unstable when c is large and/or the Λ_j are small. The product terms are very large and have alternate signs $(-1)^{j-1}$.

Higham (2009) propose a stable method for matrix exponential. More reliable, but significantly slower.

For the case where the above prob is close to 1, we also have

$$\mathbb{P}\left[A_1 + \cdots + A_c \leq 1 \mid \pi,\ C = c\right] = \sum_{j=1}^{c} (1 - e^{-\Lambda_j}) \prod_{k=1,\, k\neq j}^{c} \frac{\Lambda_k}{\Lambda_k - \Lambda_j}.$$

A dodecahedron network



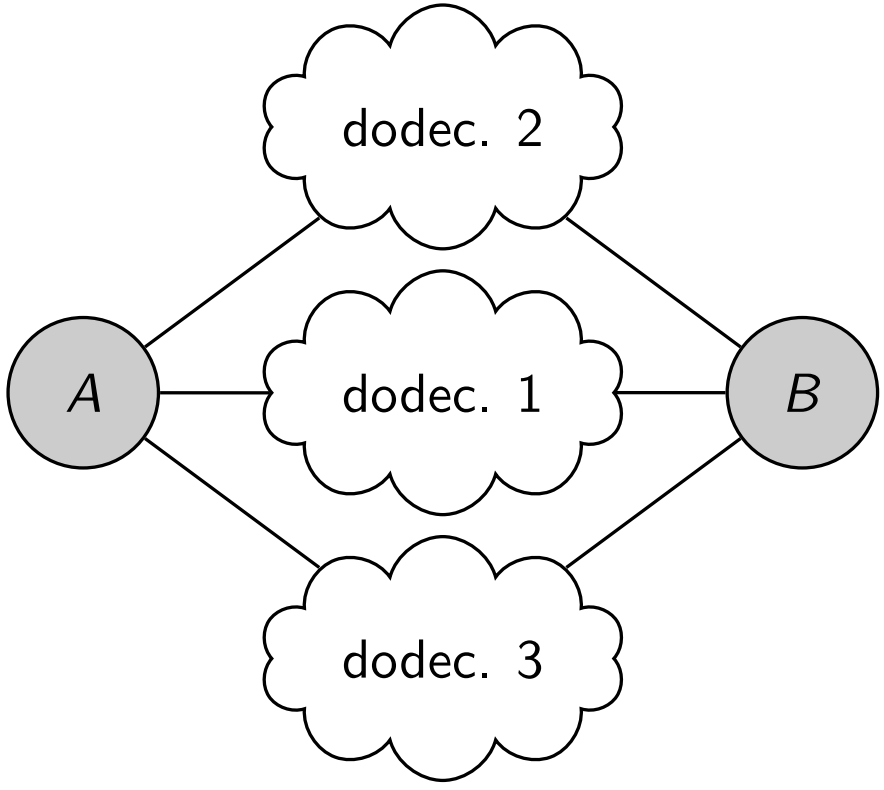
Turnip method for dodecahedron graph: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

$u_i = \epsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
$\bar{W}_n$	2.881e-3	2.065e-6	2.006e-9	1.992e-12	1.999e-15	2.005e-18
RE$[\bar{W}_n]$	0.00302	0.00421	0.00433	0.00436	0.00435	0.00434
T (sec)	15.6	15.5	15.5	15.5	15.5	15.5

We see that $u \approx 2 \times 10^{-3\epsilon}$ and RE is bounded (proved).

Three dodecahedron graphs in parallel.

60 nodes and 90 links.



Turnip for three dodecahedrons in parallel: $n = 10^8$, $\mathcal{V}_0 = \{1, 20\}$

$u_i = \epsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
$\bar{W}_n$	2.39e-8	8.80e-18	8.20e-27	8.34e-36	8.07e-45	7.92e-54
RE$[\bar{W}_n]$	0.0074	0.0194	0.0211	0.0210	0.0212	0.0215
T (sec)	6236	6227	6229	6546	6408	6289

We have $u \approx 2 \times 10^{-9\epsilon}$ and RE is bounded (proved).
Total CPU time is about 2 hours, regardless of ϵ.

However, for very large graphs (thousands of links), the turnip method fails, because the important permutations π, for which the conditional probability contributes significantly, are rare, and hitting them becomes a rare event.

BRE does not hold for an asymptotic regime where the size of the graph increases. Splitting will come to the rescue (later on).

Dependent Links: A Marshall-Olkin Copula Model

Goal: Define a model where the X_i's may have positive dependence.

Dependent Links: A Marshall-Olkin Copula Model

Goal: Define a model where the X_i's may have positive dependence.

We use an auxiliary dynamic model to specify the dependence. Suppose all links are initially operational. For each $\mathbf{s} \subseteq \{1, \ldots, m\}$, a shock that takes down all links in $\mathbf{s}$ occurs at an exponential time with rate $\lambda_{\mathbf{s}}$. Let $\mathcal{L} = \{\mathbf{s} : \lambda_{\mathbf{s}} > 0\} = \{\mathbf{s}(1), \ldots, \mathbf{s}(\kappa)\}$.

Denote $\lambda_j = \lambda_{\mathbf{s}(j)}$, let Y_j be the shock time for subset $\mathbf{s}(j)$, and $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$ (the latent state of the system).

X_i is the the indicator that component i is operational at time 1:

$$X_i = \mathbb{I}[Y_j > 1 \text{ for all shocks } j \text{ such that } i \in \mathbf{s}(j)\}.$$

Dependent Links: A Marshall-Olkin Copula Model

Goal: Define a model where the X_i's may have positive dependence.

We use an auxiliary dynamic model to specify the dependence. Suppose all links are initially operational. For each $\mathbf{s} \subseteq \{1, \ldots, m\}$, a shock that takes down all links in $\mathbf{s}$ occurs at an exponential time with rate $\lambda_{\mathbf{s}}$. Let $\mathcal{L} = \{\mathbf{s} : \lambda_{\mathbf{s}} > 0\} = \{\mathbf{s}(1), \ldots, \mathbf{s}(\kappa)\}$.

Denote $\lambda_j = \lambda_{\mathbf{s}(j)}$, let Y_j be the shock time for subset $\mathbf{s}(j)$, and $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$ (the latent state of the system).

X_i is the the indicator that component i is operational at time 1:

$$X_i = \mathbb{I}[Y_j > 1 \text{ for all shocks } j \text{ such that } i \in \mathbf{s}(j)\}.$$

This can represent group failures and cascading failures (quite natural).

Dependent Links: A Marshall-Olkin Copula Model

Goal: Define a model where the X_i's may have positive dependence.

We use an auxiliary dynamic model to specify the dependence. Suppose all links are initially operational. For each $\mathbf{s} \subseteq \{1, \ldots, m\}$, a shock that takes down all links in $\mathbf{s}$ occurs at an exponential time with rate $\lambda_{\mathbf{s}}$. Let $\mathcal{L} = \{\mathbf{s} : \lambda_{\mathbf{s}} > 0\} = \{\mathbf{s}(1), \ldots, \mathbf{s}(\kappa)\}$.

Denote $\lambda_j = \lambda_{\mathbf{s}(j)}$, let Y_j be the shock time for subset $\mathbf{s}(j)$, and $\mathbf{Y} = (Y_1, \ldots, Y_\kappa)$ (the latent state of the system).

X_i is the the indicator that component i is operational at time 1:

$$X_i = \mathbb{I}[Y_j > 1 \text{ for all shocks } j \text{ such that } i \in \mathbf{s}(j)\}.$$

This can represent group failures and cascading failures (quite natural).

However, the previous PMC and turnip methods do not apply here, because the "repairs" or failures of links are not independent!

PMC method, now a destruction process

Generate the shock times Y_j (instead of link failure or repair times), sort them to get $Y_{\pi(1)} < \cdots < Y_{\pi(\kappa)}$, and retain only the permutation π.

PMC estimator: $\mathbb{P}[\text{graph is failed at time 1} \mid \pi]$.

PMC method, now a destruction process

Generate the shock times Y_j (instead of link failure or repair times), sort them to get $Y_{\pi(1)} < \cdots < Y_{\pi(\kappa)}$, and retain only the permutation π.

PMC estimator: $\mathbb{P}[\text{graph is failed at time 1} \mid \pi]$.

To compute it, add the shocks $\pi(1)$, $\pi(2)$, ..., and remove corresponding links $i \in \mathbf{s}(j)$, until the system fails, at critical shock number C_{s}.

Data structure: forest of spanning trees.
When removing a link: breath-first search for alternative path.

The time $A_j = Y_{\pi(j)} - Y_{\pi(j-1)}$ between two successive shocks is exponential with rate Λ_j equal to the sum of rates of all forthcoming shocks. That is, $\Lambda_1 = \lambda_1 + \cdots + \lambda_\kappa$ and $\Lambda_{j+1} = \Lambda_j - \lambda_{\pi(j)}$ for $j \geq 1$.
PMC estimator of u:

$$U = \mathbb{P}\left[A_1 + \cdots + A_c \leq 1 \mid \pi, C_{\mathrm{s}} = c\right] = \sum_{j=1}^{c} (1 - e^{-\Lambda_j}) \prod_{k=1,\, k\neq j}^{c} \frac{\Lambda_k}{\Lambda_k - \Lambda_j}.$$

Generating the permutation π directly

At step k, the kth shock is selected with probability λ_j/Λ_k for shock j, where Λ_k is the sum of rates for the shocks that remain. This avoids the sort, and we stop when we reach C_{s}.

However, the probabilities λ_j/Λ_k change at each step, so they must be updated to generate the next shock. Could bring significant overhead: $\mathcal{O}(\kappa)$ time at each step; $\mathcal{O}(C_{\mathrm{s}}\kappa)$ time overall. So it is slower in some situations.

A special case: If the λ_j are all equal, the next shock is always selected uniformly. This amounts to generating a random permutation, which is easy to do efficiently.
We also have a formula to compute the hypoexponential cdf must faster in this case.

Scanning the shocks in reverse order

Instead of adding shocks until the system fails, we can generate all the shocks to know π, then assume that all shocks have already occurred, and remove them one by one until $\mathcal{V}_0$ is connected. Reconstructing the network like this is sometimes much faster.

But for a link to be repaired, we must remove all the shocks that affect it! How do we know when the link is repaired?

Scanning the shocks in reverse order

Instead of adding shocks until the system fails, we can generate all the shocks to know π, then assume that all shocks have already occurred, and remove them one by one until $\mathcal{V}_0$ is connected. Reconstructing the network like this is sometimes much faster.

But for a link to be repaired, we must remove all the shocks that affect it! How do we know when the link is repaired?

If c_i shocks can affect link i, start a counter f_i at c_i, and decrease it each time a shock that affects i is removed. Link i is repaired when $f_i = 0$.

C_{S} is the number of shocks that remain when the system becomes operational, plus 1.

This gives a faster way to compute C_{S} when it is large (close to κ). The estimator U remains the same.

PMC with anti-shocks

Here we change the estimator. Assume all the shocks have occurred and generate independent anti-shocks that remove the shocks, one by one. Idea: repair the shocks rather than the links.

Anti-shock j occurs at exponential time R_j, with rate $\mu_j = -\ln(1 - e^{-\lambda_j})$. This gives $\mathbb{P}[R_j \le 1] = \mathbb{P}[Y_j > 1] = \mathbb{P}[\text{shock } j \text{ has occurred}]$.

Sorting the times R_j gives a permutation π' ($\equiv$ reverse of π).
$C_{\mathrm{a}} = \kappa + 1 - C_{\mathrm{s}} =$ anti-shock number when system becomes operational.
Times between successive anti-shocks: $A'_k = R_{\pi'(k)} - R_{\pi'(k-1)}$, exponential with rate $\Lambda_k = \mu_{\pi(k)} + \cdots + \mu_{\pi(\kappa)}$. Estimator of u:

$$U' = \mathbb{P}[A'_1 + \cdots + A'_{C_{\mathrm{a}}} > 1 \mid \pi'].$$

When u is very small, we can often compute U' accurately and not U.

Adapting the turnip method

When generating the shocks [or anti-shocks] in increasing order of occurrence, at each step j, discard the future shocks [or anti-shocks] that can no longer contribute to system failure [or repair].

For instance, when removing a link, if there are nodes that become disconnected from $\mathcal{V}_0$, those nodes can be removed for further consideration. And future shocks k that only affect removed links can be discarded, and their rate λ_k subtracted from Λ_j.

Adapting the turnip method

When generating the shocks [or anti-shocks] in increasing order of occurrence, at each step j, discard the future shocks [or anti-shocks] that can no longer contribute to system failure [or repair].

For instance, when removing a link, if there are nodes that become disconnected from $\mathcal{V}_0$, those nodes can be removed for further consideration. And future shocks k that only affect removed links can be discarded, and their rate λ_k subtracted from Λ_j.

When an anti-shock occurs, if it repairs a link that connects two groups of nodes, all links that connect the same groups can be discarded, and anti-shocks that only affect discarded links can be discarded.

Overhead: Must maintain data structures to identify shocks [or anti-shocks] that can be discarded.

Removing links from the graph is more time consuming than adding links.

A generalized splitting (GS) algorithm

Uses latent variables $\mathbf{Y}$. Let

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

the time at which the network fails, and $S(\mathbf{Y}) = 1/\tilde{S}(\mathbf{Y})$.

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

$$\rho_t \stackrel{\text{def}}{=} \mathbb{P}[S(\mathbf{Y}) > \gamma_t \mid S(\mathbf{Y}) > \gamma_{t-1}] \approx 1/2$$

for $t = 1, \ldots, \tau$. The γ_t's are estimated by pilot runs.

For each level γ_t, construct (via MCMC) a Markov chain $\{\mathbf{Y}_{t,j}, j \geq 0\}$ with transition density κ_t and whose stationary density is the density of $\mathbf{Y}$ conditional on $S(\mathbf{Y}) > \gamma_t$:

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

GS algorithm with shocks

Generate $\mathbf{Y}$ from density f
if $S(\mathbf{Y}) > \gamma_1$ **then** $\mathcal{X}_1 \leftarrow \{\mathbf{Y}\}$ **else return** $U \leftarrow 0$
for $t = 2$ **to** τ **do**
    $\mathcal{X}_t \leftarrow \emptyset$ // set of states that have reached level γ_t
    for all $\mathbf{Y}_0 \in \mathcal{X}_{t-1}$ **do**
        for $\ell = 1$ **to** 2 **do**
            sample $\mathbf{Y}_\ell$ from density $\kappa_{t-1}(\cdot \mid \mathbf{Y}_{\ell-1})$
            if $S(\mathbf{Y}_\ell) > \gamma_t$ **then** add $\mathbf{Y}_\ell$ to $\mathcal{X}_t$
return $U \leftarrow |\mathcal{X}_\tau|/2^{\tau-1}$ as an unbiased estimator of u.

Repeat this n times, independently, and take the average.
Can compute a confidence interval, etc.

Defining κ_{t-1} via Gibbs sampling:

Require: $\mathbf{Y}$ for which $S(\mathbf{Y}) > \gamma_{t-1}$
  for $j = 1$ **to** κ **do**
    if $S(Y_1, \ldots, Y_{j-1}, \infty, Y_{j+1}, \ldots, Y_\kappa) < \gamma_{t-1}$ **then**
      // removing shock j would connect $\mathcal{V}_0$
      resample Y_j from its density truncated to $(0, 1/\gamma_{t-1})$
    else
      resample Y_j from its original density
  return $\mathbf{Y}$ as the resampled vector.

Data structure: forest of spanning trees.

GS algorithm with anti-shocks

Same idea, but evolution and resampling is based on $\mathbf{R}$ instead of $\mathbf{Y}$.

$$S(\mathbf{R}) = \inf\{\gamma \geq 0 : \Psi(\mathbf{X}(\gamma)) = 1\}.$$

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

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

Require: $\mathbf{R} = (R_1, \ldots, R_\kappa)$ for which $S(\mathbf{R}) > \gamma_{t-1}$.
  for $j = 1$ **to** κ **do**
    if $S(R_1, \ldots, R_{j-1}, 0, R_{j+1}, \ldots, R_\kappa) \leq \gamma_{t-1}$ **then**
      resample R_j from its density truncated to (γ_{t-1}, ∞)
    else
      resample R_j from its original density
  return $\mathbf{R}$ as the resampled vector.

Example: dodecahedron graph

GS for the dodecahedron, shocks on links only: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

$u_j = \epsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
τ	9	19	29	39	49	59
$\bar{W}_n$	2.877e-3	2.054e-6	2.022e-9	2.01e-12	1.987e-15	1.969e-18
RE$[\bar{W}_n]$	0.0040	0.0062	0.0077	0.0089	0.0099	0.0112
T (sec)	93	167	224	278	334	376

GS, three dodeca. in parallel, shocks on links: $n = 10^6$, $\mathcal{V}_0 = \{1, 20\}$

$u_j = \epsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
τ	26	57	87	117	147	176
$\bar{W}_n$	2.38e-8	8.87e-18	8.18e-27	8.09e-36	8.24e-45	7.93e-54
RE$[\bar{W}_n]$	0.0071	0.0109	0.0137	0.0158	0.0185	0.0208
T (sec)	1202	2015	2362	2820	3041	3287

Turnip for three dodecahedrons in parallel: $n = 10^8$, $\mathcal{V}_0 = \{1, 20\}$

$u_i = \epsilon$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
$\bar{W}_n$	2.39e-8	8.80e-18	8.20e-27	8.34e-36	8.07e-45	7.92e-54
RE$[\bar{W}_n]$	0.0074	0.0194	0.0211	0.0210	0.0212	0.0215
T (sec)	6236	6227	6229	6546	6408	6289

Example: dodecahedron graph

Shocks on nodes and on links, all at rate λ. $\mathcal{V}_0 = \{1, 20\}$, $n = 10^6$.

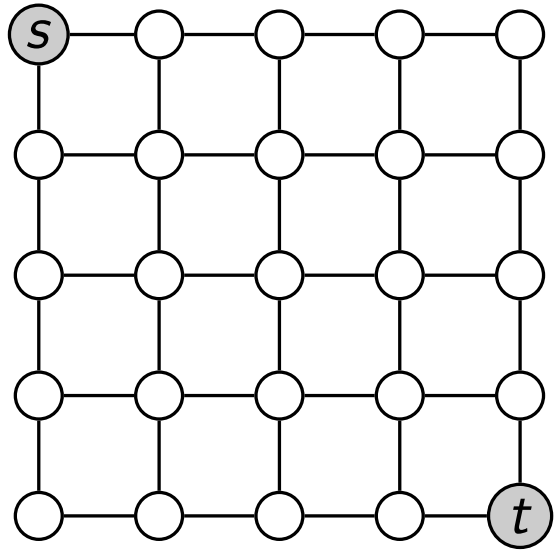
$\lambda = 10^{-5}$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
PMC	2.014e-5	23.8	0.0049	9.75	* 49	0.00117
PMC-π	1.996e-5	24.0	0.0049	9.75	* 35	0.00084
PMC-rev	2.014e-5	23.8	0.0049	9.75	* 50	0.00119
PMC-anti	2.012e-5	23.8	0.0049	41.25	33	0.00079
turnip	1.993e-5	24.1	0.0049	8.62	* 58	0.00140
turnip-π	1.998e-5	24.0	0.0049	8.51	* 52	0.00125
turnip-anti	2.000e-5	12.6	0.0035	40.18	53	0.00066
GS	2.002e-5	31.2	0.0056		230	0.00719
GS-anti	2.022e-5	30.7	0.0055		239	0.00732

Dodecahedron, shocks on links and nodes

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-15}$						
PMC	1.988e-15	24.2	0.0049	9.75	* 46	0.00112
PMC-π	1.998e-15	24.1	0.0049	9.75	* 35	0.00086
PMC-rev	2.063e-15	22.4	0.0047	9.75	* 52	0.00116
PMC-anti	1.988e-15	24.1	0.0049	41.2	33	0.00079
turnip	2.080e-15	22.2	0.0047	8.62	* 57	0.00127
turnip-π	2.079e-15	22.2	0.0047	8.51	* 51	0.00113
turnip-anti	1.984e-15	12.6	0.0036	40.18	49	0.00062
GS	2.014e-15	90.9	0.0095		688	0.0625
GS-anti	1.990e-15	102.3	0.0101		614	0.0629
$\lambda = 10^{-20}$						
PMC-anti	2.008e-20	23.9	0.0049	41.3	32	0.00077
turnip-anti	2.003e-20	12.6	0.0035	40.2	49	0.00062
GS	2.034e-20	136.2	0.012		849	0.116
GS-anti	1.962e-20	125.5	0.011		892	0.112

Square lattice graphs



20×20 lattice: 400 nodes, 760 links, and 1160 different shocks.
40×40 lattice: 1600 nodes, 3120 links, and 4720 different shocks.

For $\lambda_j = 10^{-10}$ and 10^{-20}, we have $\mu_j = 23.0259$ and 46.0517.
Computing U is much faster for these μ's than for the corresponding λ's.

20×20 lattice graph, $n = 10^5$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-10}$						
PMC	1.995e-10	580	0.076	166	* 2668	15.5
PMC-π	2.018e-10	574	0.076	166	* 2252	12.9
PMC-rev	1.995e-10	580	0.076	166	* 2030	11.8
PMC-anti	2.076e-10	558	0.075	995	898	5.0
turnip	1.972e-10	587	0.077	148	* 3237	19.0
turnip-π	2.123e-10	545	0.074	147	* 2844	15.5
GS	2.021e-10	63	0.025		3033	1.9
GS-anti	2.006e-10	65	0.025		2919	1.9

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-20}$						
PMC-anti	1.984e-20	584	0.0764	995	900	5.3
GS	2.14e-20	134	0.0366		3504	4.7
GS-anti	1.992e-20	116	0.0341		3562	4.1

40×40 **lattice graph,** $n = 10^4$

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-10}$						
PMC-anti	1.888e-10	2499	0.5	4044	1437	359
GS	2.151e-10	62	0.079		4473	28
GS-anti	2.085e-10	65	0.081		4402	29
algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-20}$						
PMC-anti	1.416e-20	3333	0.577	4053	1431	477
GS	1.748e-20	163	0.128		4785	78
GS-anti	1.935e-20	121	0.110		4869	59

20×20 lattice graph, 400 nodes and 760 links.
One shock per node at rate λ and one shock per link at rate 10λ.
$\mathcal{V}_0 = \{1, 400\}$, GS with shocks, $n = 10^4$.

λ	$\bar{W}_n$	RE$[\bar{W}_n]$	T (sec)
10^{-2}	4.66e-2	0.0283	102
10^{-3}	2.16e-3	0.0480	133
10^{-4}	2.00e-4	0.0624	122
10^{-5}	1.95e-5	0.0629	153
10^{-6}	2.17e-6	0.0653	168
10^{-7}	2.14e-7	0.0634	184
10^{-8}	2.05e-8	0.1203	105
10^{-9}	1.97e-9	0.1093	150
10^{-10}	1.94e-10	0.0696	266
10^{-11}	1.97e-11	0.0819	187
10^{-12}	2.16e-12	0.0629	359
10^{-18}	1.93e-18	0.0712	811

PMC and turnip do not work here when λ is too small.

Complete graphs

Complete graph with n_0 nodes, one link for each pair of nodes.

$n_0 = 30$ gives 435 links and 465 shocks.

$n_0 = 100$ gives 4950 links and 5050 shocks.

$n = 10^5$.

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-10}$						
PMC	1.916e-10	242	0.0492	154	2246	5.43
PMC-π	1.893e-10	245	0.0495	153	1890	4.63
PMC-rev	1.916e-10	242	0.0492	154	2026	4.90
PMC-anti	1.934e-10	239	0.0489	313	110	0.26
turnip	2.065e-10	224	0.0474	100	790	1.77
turnip-π	1.911e-10	242	0.0492	100	761	1.84
turnip-anti	1.994e-10	36	0.0190	259	220	0.08
GS	1.962e-10	60	0.0244		689	0.41
GS-anti	2.061e-10	66	0.0257		580	0.38
$\lambda = 10^{-20}$						
PMC-anti	2.041e-20	227	0.0476	312	110	0.25
turnip-anti	1.961e-20	37	0.0191	260	209	0.08

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

algorithm	$\bar{W}_n$	$S_n^2/\bar{W}_n^2$	RE$[\bar{W}_n]$	C	T(sec)	WNRV
$\lambda = 10^{-10}$						
PMC-anti	2.02e-10	2499	0.5	3361	1088	272
GS	1.943e-10	67	0.082		1116	7.5
GS-anti	1.935e-10	65	0.081		1107	7.2
$\lambda = 10^{-20}$						
PMC-anti	2.02e-20	2499	0.5	3379	1099	275
GS	2.13e-20	158	0.13		1475	23
GS-anti	2.15e-20	144	0.12		1385	20

Extensions

PMC, turnip, and GS could be adapted to rare-event simulation in more general shock-based reliability models, e.g., where shocks only alter the state of the system, may change the future shock rates, etc. Several applications in sight.

Example: Probability that max flow is under a given threshold in a network where links have random capacities.

Example: Probability of overflow in a communication network where links have capacities and demand is random.