Loss Networks

Jeffrey Minten and Jeroen van Riel

March 2023

(a) loss network model

First of all, the problem statement is a little bit unusual, because most often maintenance requests are controlled, i.e. they are planned according to some policy that tries to minimize the risk of failure. In the current setting it seems there are two types of failures. Let us refer to the maintenance requests as type-A failures. It is stated that ignoring these requests (due to lack of capacity) will eventually lead to a more high-cost type-B failure.

Let the specialized maintenance groups be indexed by $\mathcal{L} = \{1, \dots, L\}$ and let C_l denote the number of workers in this group $l \in \mathcal{L}$. The groups correspond to the *links* in the loss network model. The *customer classes* may be used to model K different types of maintenance requests. Each request type $1 \leq k \leq K$ is characterized by the level of *importance* w_k and the required number of workers b_{kl} from each maintenance group l. Note that the routes $R_k = \{l \in \mathcal{L} : b_{kl} > 0\}$ have no *network* interpretation, but are merely the set of maintenance groups required by type-k requests. In order to use the provided theory, we must assume that requests of type k arrive as a Poisson process with rate λ_k.

(b) optimization

Using the loss network model, we cannot directly model the influence of a lost service request on the failure rate of the hardware. However, we can use the importance w_k as a proxy by penalizing important lost requests more. Let B_k denote the stationary probability of blocking type-k requests, then we aim to minimize the objective

$$\sum_{k=1}^{K} w_k B_k.$$

For instances with small C and K, we can sum over all states to obtain G^k and G as in part (e), to calculate B_k directly. However, this method quickly becomes computationally infeasible, so it is no good candidate for optimization. When we want to use the alternative Erlang fixed-point method for approximating the stationary blocking probabilities, we need to assume $b_{kl} \in \{0, 1\}$, which may be prohibitive in light of the application. However, the relative speed of this method allows us to use some heuristics-based local optimization method to minimize the objective.

(c) loss network specification

We have $L = 4$ links and $K = 4$ user classes. Let the link capacities be given by

$$C = \begin{pmatrix} 3 & 1 & 1 & 1 \end{pmatrix}$$

Note that we need to use $b_{kl} \in \{0, 1\}$ in order to be able to apply the Erlang fixed-point method later on in the assignment.

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix},$$

from which we deduce the routes to be

$$\begin{aligned} R_1 =& \{1\}, & R_3 =& \{1, 2, 3\}, \\ R_2 =& \{1, 2\}, & R_4 =& \{1, 3, 4\}. \end{aligned}$$

(d) stationary joint distribution

In general, Proposition 4 of the lecture notes provides us with the stationary distribution

$$\pi(n) = G^{-1} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!},$$

with

$$G = \sum_{n \in S_L} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}. \tag{1}$$

Note that the state space S_L depends on B and C and is defined as the lattice points of the 4-dimensional polyhedron

$$S_L = \{n \in \mathbb{N}^4 : nB \leq C\},$$

where n and C are row vectors.

In the current specific loss network, this means that we could enumerate all states (n_1, n_2, n_3, n_4) and sum over the corresponding terms

$$\frac{\rho_1^{n_1}}{n_1!} \cdot \frac{\rho_2^{n_2}}{n_2!} \cdot \frac{\rho_3^{n_3}}{n_3!} \cdot \frac{\rho_4^{n_4}}{n_4!}.$$

(e) blocking probabilities

We first show how this holds in general before giving the specific formula for the current loss network. The argument is very similar to the one given in the lecture notes for multi-class Erlang-B models. Substituting the formula for $\pi(n)$ from equation (3.11), we get

$$B_r = \sum_{\substack{n \in S_L \\ n+e_r \notin S_L}} \pi(n) = 1 - \sum_{n+e_r \in S_L} \pi(n)$$

with

$$\sum_{n+e_r \in S_L} \pi(n) = G^{-1} \underbrace{\sum_{n+e_r \in S_L} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}}_{(*)}$$

We now show that $(*)$ is actually $G^r = G(C_1^r, \dots, C_L^r)$, which is the normalization constant of an amended loss network with link capacities $C_l^r = C_l - b_{rl}$ for all $1 \leq l \leq L$. Recall that the state space is given by

$$S_L = \{n \in \mathbb{N}^K : \sum_{k=1}^{K} b_{kl} n_k \leq C_l \text{ for all } 1 \leq l \leq L\}.$$

Let S_L^r denote the state space of the amended loss network. Now observe that

$$n + e_r \in S_L$$

is equivalent to

$$\begin{aligned} C_l &\geq \sum_{k=1}^{K} b_{kl}(n_k + \mathbf{1}\{k = r\}) \\ &= b_{rl} + \sum_{k=1}^{K} b_{kl} n_k \quad \text{for all } 1 \leq l \leq L \end{aligned}$$

which is in turn equivalent to

$$n \in S_L^r.$$

(f) computational complexity estimate

Note that the computation is linear $O(n)$ in the number of states n. There are efficient procedures for computing the number of lattice points in a polyhedron [3]. For example, we are able to explicitly count and enumerate the states S_L using the `polymake` [1] software package. This allows us to simply compute the

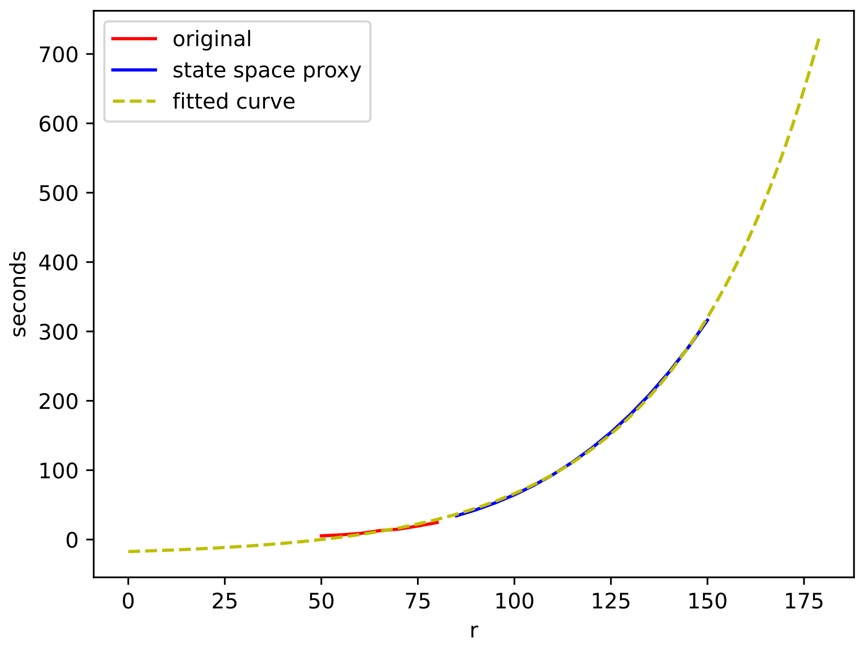

Figure 1: Estimation of runtime as function the capacity $C = (r, r, r, r)$.

sum (1) by going over these states. We measured the running run of this procedure with capacity settings $C = (r, r, r, r)$ for increasing $r \in \{50, 55, \ldots, 80\}$, see the red curve in Figure 1. The average number of points processed per second turned out to be roughly $P = 71274$.

We now estimate the running time complexity for large values of r by computing the size $|S_L(r)|$ of the corresponding state space using `lattE` [2] (through `polymake`). Next, we simply divide $|S_L(r)|$ by P to obtain the blue curve in Figure 1. Because computing the number of lattice points is costly itself, we fit a exponential curve (dashed).

Note that our main assumption here is that the complexity of counting and enumerating lattice points is comparable. Because we are not familiar with the exact methods used by `lattE`, we are not sure about the validity of this assumption. The code used for this part can be found at `https://github.com/jeroenvanriel/stochastic-networks`.

(g) Erlang fixed-point

In the loss network that is described in exercise (c) there are four classes and four links. Furthermore, we have $b_{kl} \in \{0, 1\}$, so that we can use the Erlang fixed-point approximation method, which is based on a recursive expression for all blocking probabilities B_k, in general given by

$$1 - B_k = \prod_{l:b_{kl}=1} (1 - B_l'),$$

$$B_l' = \frac{\frac{\sigma_l^{C_l}}{C_l!}}{\sum_{i=0}^{C_l} \frac{\sigma_l^i}{i!}},$$

$$\sigma_l = \sum_{k=1}^{K} b_{kl} \sigma_{kl}',$$

$$\sigma_{kl}' = \rho_k \frac{(1 - B_k)}{(1 - B_l')},$$

for all $k = 1, \ldots K$. To be able to give a clear view of the approximations for all k, we will work from bottom to top. We start by constructing σ_l for all l, which gives

$$\begin{aligned}
\sigma_1 &= \sum_{k=1}^{K} b_{k1}\sigma_{k1}' = \sigma_{11}' + \sigma_{21}' + \sigma_{31}' + \sigma_{41} \\
&= \frac{\rho_1(1 - B_1) + \rho_2(1 - B_2) + \rho_3(1 - B_3) + \rho_4(1 - B_4)}{(1 - B_1')}, \\
\sigma_2 &= \sum_{k=1}^{K} b_{k2}\sigma_{k2}' = \sigma_{22}' + \sigma_{32}' = \frac{\rho_2(1 - B_2) + \rho_3(1 - B_3)}{(1 - B_2')}, \\
\sigma_3 &= \sum_{k=1}^{K} b_{k3}\sigma_{k3}' = \sigma_{33}' + \sigma_{43}' = \frac{\rho_3(1 - B_3) + \rho_4(1 - B_4)}{(1 - B_3')}, \\
\sigma_4 &= \sum_{k=1}^{K} b_{k4}\sigma_{k4}' = \sigma_{44}' = \frac{\rho_4(1 - B_4)}{(1 - B_4')}.
\end{aligned}$$

These σ_l can now be substituted in each formula for B_l', so we only need the main expression for this. At last, we need to construct all expressions for the different values of k, giving

$$\begin{aligned}
1 - B_1 &= \prod_{l:b_{1l=1}} (1 - B_l') &&= (1 - B_1'), \\
1 - B_2 &= \prod_{l:b_{2l=1}} (1 - B_l') &&= (1 - B_1')(1 - B_2'), \\
1 - B_3 &= \prod_{l:b_{3l=1}} (1 - B_l') &&= (1 - B_1')(1 - B_2')(1 - B_3'), \\
1 - B_4 &= \prod_{l:b_{4l=1}} (1 - B_l') &&= (1 - B_1')(1 - B_3')(1 - B_4').
\end{aligned}$$

Combining all above formulae gives the Erlang fixed-point approximation specifically tailored for the loss network considered in exercise (c).

(h) convergence of fixed-point

Note that we are dealing with two types of convergence in this case, namely (i) the convergence of the fixed-point computation itself and (ii) convergence of these fixed-point approximations for the *growing loss networks* when N tends to infinity.

First of all, observe that our matrix b_{kl} has full rank, so the theory says we should obtain converging blocking probabilities. Setting $\rho = \begin{pmatrix} 2.0 & 0.8 & 0.8 & 0.8 \end{pmatrix}$, such that $\rho \leq C$, but somewhat close to obtain non-negligible blocking probabilities. We then simply consider linear scaling of the loss systems by setting

$$\rho(N) = N\rho,$$
$$C(N) = NC.$$

We compute the blocking probabilities using Matlab (actually, GNU Octave version 5.2.0). As a stopping criterion for the inner iteration, we used a threshold on the maximum difference, i.e. continue while $\max |B^{(i)} - B^{(i-1)}| > \epsilon$, where $B^{(i)}$ denotes the vector of blocking probability approximations after the ith iteration We start the iteration with $B^{(0)} = B'^{(0)} = 0$ and use $\epsilon = 0.001$. The code used for this part can be found at `https://github.com/jeroenvanriel/stochastic-networks`.

As you can see in Figure 2, the approximations seem to converge. However, we observed that our computations break down after around 44 iterations, when we receive non-representable numbers (NaN).

References

[1] Tutorial for lattice polytopes. `https://polymake.org/doku.php/user_guide/tutorials/latest/lattice_polytopes_tutorial#lattice_points_in_rational_polytopes`.

[2] V. Baldoni, N. Berline, J. De Loera, B. Dutra, M. Koppe, S. Moreinis, G. Pinto, M. Vergne, and J. Wu. *A User's Guide for LattE integrale v1.7.2*, 2013.

[3] M. Köppe. A primal barvinok algorithm based on irrational decompositions. *SIAM Journal on Discrete Mathematics*, 21(1):220–236, jan 2007.

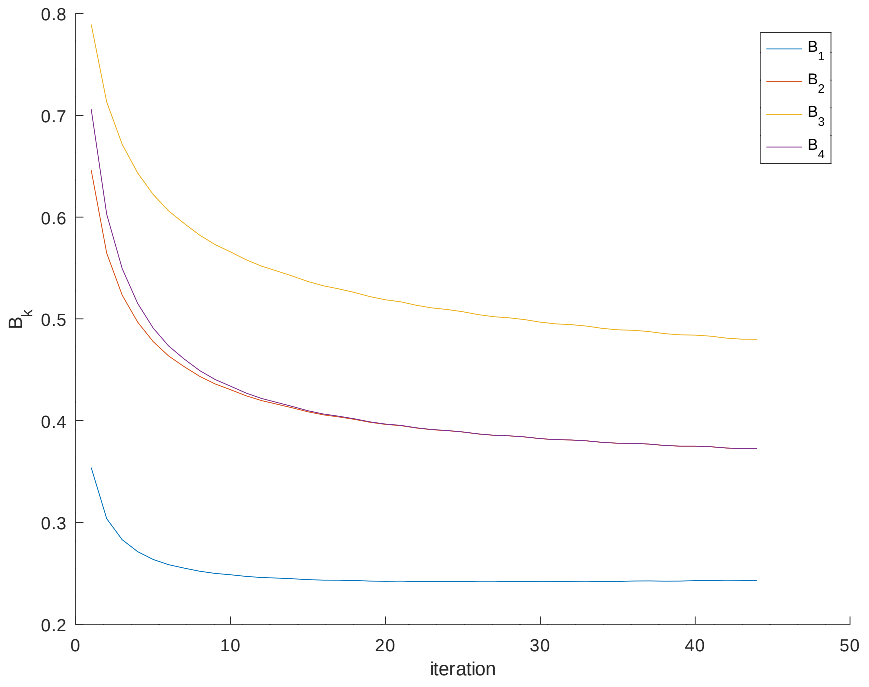

Figure 2: Convergence of blocking probability approximations for linearly scaling loss networks.