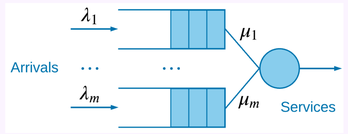
Finding Optimal Policy for Queueing Models: New Parameterization

Trang H. Tran[1] · Lam M. Nguyen[2] · Katya Scheinberg [1]

[1] Cornell University, School of Operations Research and Information Engineering [2] IBM Research, Thomas J. Watson Research Center

Problem Statement

The optimal control problem of queueing networks has many important applications in real life, including communications networks, transportation, and manufacturing systems [1]. These problems can be modeled effectively as Reinforcement Learning (RL) environments. We are interested in the optimization aspects of this approach: how the choices of reward function and policy parameters influence the efficiency of the optimization process.



We consider a simple parallel queueing system with one server, exponential inter-arrival and service times: the jobs of class i arrive to the system following Poisson processes with respective rates λ_i, $i = 1, \dots, m$. The processing times for class i jobs are i.i.d., having exponential distribution with the respective service rates $\mu_1, \mu_2, \dots, \mu_m$. The corresponding load condition for this system is:

$$\rho = \frac{\lambda_1}{\mu_1} + \frac{\lambda_2}{\mu_2} + \cdots + \frac{\lambda_m}{\mu_m} < 1 \tag{1}$$

We assume that a decision time occurs when a new job arrives to the system or when the server completes a service. We let $c \in \mathbb{R}^m$ be the holding cost vector, and let the holding cost be $c^\top s$ where $s \in \mathbb{R}^m$ is the observable state. The following Theorem characterizes the optimal policy for this system.

Theorem 1 - Optimal Policy

For a parallel single server queueing system with infinite buffer, the optimal policy is the priority policy based on the c-μ rule: The server selects the job in queue $i^* = \arg\max\{c_i\mu_i |$ for all i such that $s_i > 0\}$ (i.e. choose the job in the class with the largest cost - expected processing time ratio).

We denote the priority policy by π_P.

Linear Policy Representation

First we consider a simple class of linear parameterized policies: given the state vector $s \in \mathbb{N}^m$, we define $\pi(A) = \text{softmax}(As)$, where A is a matrix and the parameter. This is a special case of the parameterization by neural networks with softmax activation which is widely used in RL.

Under such a policy, the probability of any action for any state vector is a number between 0 and 1. In contrast, the priority policy π_P is a threshold policy which chooses one action with probability one. Next, we show that π_P can be approximated arbitrarily closely with our class of parameterized policies.

Theorem 2 - Policy Representation

Consider the call of linear policies described in Section 3.1 and assume that $s_i \le Q$ for every $i = 1, \dots, m$ for a given state vector. Let $\{A_k, k \ge 0\}$ be the sequence

$$A_k = \begin{bmatrix} (Q+1)^m \cdot k + 1 & \dots & 1 \\ \dots & (Q+1)^i \cdot k + 1 & \dots \\ 1 & \dots & (Q+1) \cdot k + 1 \\ 1 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m},$$

and $\pi_k = \pi(A_k) \in \mathbb{R}^{m+1}$ be the policy corresponding to A_k. We have

- The starting point of the sequence π_0 is a random policy that chooses every action with equal probability.
- The sequence π_k converges to the priority policy: $\pi_k \to \pi_P$ when $k \to \infty$.
- There does not exist a bounded sequence of matrices A'_k such that sequence $\pi(A'_k)$ converges to the priority policy: $\pi_k \to \pi_P$ when $k \to \infty$.

Alternative Parameterization

Theorem 2 shows that the in order to converge to the priority sequence, the policy parameters have to grow arbitrarily large, moreover, their growth may be as fast as exponential in the number of queues. Thus we consider an alternative parameterization for π. Let $B = \ln(A)$ where ln is an element-wise operator and A is a matrix with positive elements. Let B be the parameters of policy $\bar{\pi}$, and the parameterization be

$$\bar{\pi}(B) = \text{softmax}(e^B s), \tag{2}$$

We denote this scheme as the logarithm-scale parameterization of policy $\bar{\pi}$. The sequence of matrices B_k associated with A_k is

$$B_k = \begin{bmatrix} \ln[(Q+1)^m \cdot k + 1] & \dots & 0 \\ \dots & \ln[(Q+1)^i \cdot k + 1] & \dots \\ 0 & \dots & \ln[(Q+1) \cdot k + 1] \\ 0 & \dots & 0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}.$$

The diagonal elements B_k also grow infinitely large but much slower than those of A_k, which may result in better behavior of the optimization problem (e.g. smoother objective) when using this parameterization.

Gradient Estimators for Policy Optimization

The function value estimator, inexact zeroth-order oracle is the average cost function induced by the parameterized policy $\pi(A)$. Let n be the number of sample paths and $T-1$ be the time horizon, the zeroth-order oracle $\tilde{J}(A)$ is

$$\tilde{J}(A) = \frac{1}{n}\sum_{i=1}^{n} C(\tau_i) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{T}\sum_{t=0}^{T-1} c(s_{i,t}, a_{i,t})\right], \tag{3}$$

where τ_i is the i-th sample path following policy $\pi(A)$. We now describe two popular ways of estimating gradients, thus providing inexact first-order oracles.

Finite Difference Estimator

The Finite Difference (FD) estimator $\widetilde{\nabla} J_1(A)$ has the form

$$\widetilde{\nabla} J_1(A) = \frac{1}{M}\sum_{i=1}^{M} \frac{\tilde{J}(A + u\cdot\vec{v_i}) - \tilde{J}(A)}{u}\vec{v_i}, \tag{4}$$

where $\vec{v_i}, i = 1, \dots, M$ is a set of vectors and u is the finite difference step parameter. We choose the canonical setting where $\vec{v_i}, i = 1, \dots, M$ are the unit vectors of the parameter space.

Policy Gradient Estimator

For the time horizon T and number of sample paths N, our Policy Gradient (PG) estimator $\widetilde{\nabla} J_2(A)$ is computed as

$$\widetilde{\nabla} J_2(A) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]\left[\frac{1}{T}\sum_{t=0}^{T-1} r(s_{i,t}, a_{i,t}) - b\right],$$

$$\text{with } b = \frac{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]^2\left[\frac{1}{T}\sum_{t=0}^{T-1} r(s_{i,t}, a_{i,t})\right]}{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]^2}.$$

Key References

[1] J. G. Dai and Mark Gluzman. Queueing network controls via deep reinforcement learning, *Stochastic Systems 12(1):30-67, 2022.*

[2] Billy Jin, Katya Scheinberg, and Miaolan Xie. High probability complexity bounds for line search based on stochastic oracles, *Advances in Neural Information Processing Systems, 2021.*

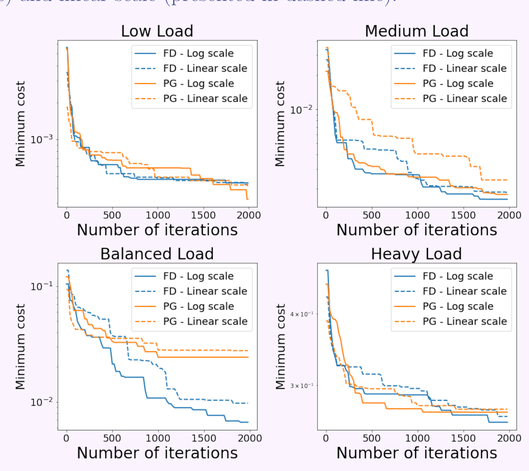
Experiments

We experiment with two parameterizations: the standard linear parameterization and the proposed logarithm-scale parameterization. We implement two gradient estimators and employ an adaptive line-search algorithm (ALOE) [2] to optimize the cost function.

Below we summarize the load conditions in our experiment setting, and the best performance of each parameterization (Linear scale and Logarithm scale). The cost function is normalized to $[0, 1]$. We report the best cost function achieved in each setting and note the first order oracle that yields the best result. The last row reports the confidence intervals of the cost function at the optimal (priority) policy in each setting.

Settings	Low Load	Medium Load	Balanced Load	Heavy Load
Service rates	$(18, 9, 6)$	$(9, 4.5, 3)$	$(6, 3, 2)$	$(3, 2, 1)$
Load param.ρ	$1/3$	$2/3$	1	$11/6$
Linear scale	0.0003 (PG)	0.0018 (FD)	0.0097 (FD)	0.2647 (FD)
Log. scale	0.0004 (PG)	0.0016 (FD)	0.0066 (FD)	0.2586 (FD)
Optimal CI	$[0.0002, 0.0015]$	$[0.0012, 0.0032]$	$[0.0079, 0.0093]$	$[0.2611, 0.4037]$

The figure below compares the minimum cost function achieved by iteration, for two parameterization scales: logarithm (proposed scale, presented in continuous solid lines) and linear scale (presented in dashed line).



The figure below compares the correct rate (in percent) achieved by iteration for two parameterization scales. In most cases, the logarithm-scales perform better than the standard linear-scales.