2WB50 Stochastic Simulation
Assignment 2: Reinforcement Learning in a Queueing System

Version 1.0

Jaron Sanders

February 26, 2024

1 Objective

You will implement an intelligent dispatcher in a queueing system based on reinforcement learning.

If you're curious: this assignment is based on recent research of ours [1].

2 Preliminaries

Suppose that customers arrive at a queueing system according to a homogeneous Poisson process with rate $\lambda \in (0, \infty)$. If T_n denotes the arrival time of the nth customer, then this implies that for $n \in \mathbb{N}_+ := \{1, 2, 3, \ldots\}$, the customers' interarrival times $T_{n+1} - T_n$ are independent and identically distributed $\text{Exp}(\lambda)$ random variables.

Suppose furthermore that each new customer arrives at an *intelligent dispatcher*. This dispatcher can choose to either dispatch an arriving customer to one of $m \in \mathbb{N}_+$ queues, or to deny the arriving customer access into the queueing system. Each queue has a single server; and rejected customers leave the system forever. [1]

Suppose that a customer is routed to queue $j \in [m] := \{1, \ldots, m\}$. This customer then either (i) enters service immediately (if the server is available), or (ii) joins the waiting line there (if the server is not available). Let us also assume that it takes a server at a queue j an exponentially distributed random amount of time to complete a service, with mean $1/\mu_j$. Here, $\mu_j \in (0, \infty)$.

Whenever a server completes their service for a customer, (i) the serviced customer firstly departs, and (ii) the server secondly inspects their queue to see if there are any customers still waiting for service. If there are customers waiting, then the server will start servicing the next customer that is first in line. Otherwise, the server will go idle until a new customer is dispatched to them by the intelligent dispatcher.

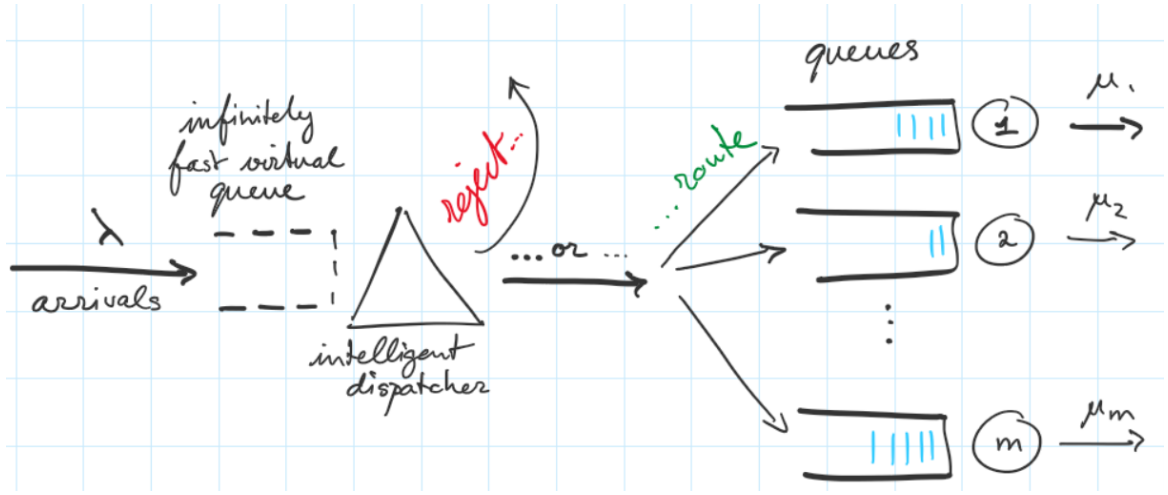

Figure 1: Here, at some time t, the number of customers waiting are $N_1(t) = 4, N_2(t) = 2, \ldots, N_m(t) = 5$. We may capture this in a vector $N(t) = (4, 2, \ldots, 5)$. Note that we do not count customers that are in service. These are, after all, not waiting.

[1] **Hint.** You could program the dispatcher as a "virtual queue" that takes zero processing time. This is depicted in Figure 1.

Tasks

1. [Easy] First, program the environment. To do so, consider at this stage a *random dispatcher*: one that sends an arriving customer to one of the queues uniformly at random with probability $\theta \in [0, 1]$, or rejects an arriving customer with probability $1 - \theta$.

(15pts) (a) Program a discrete-event simulation for the queueing system described above. Discuss your implementation/its design in your report. You are free to use any programming language of your choice.

Note, however, that you are not allowed to use a ready-made software library to simulate a queue such as SimPy. You should actually program the environment by yourselves, without outside help. The same holds true for the Reinforcement Learning (RL)agent that you will program later.

(5pts) (b) Test your simulation for $m = 2$ with parameters $\theta = 0.40$, $\lambda = 2.0$, $\mu_1 = 3.0$, $\mu_2 = 4.0$.

The long-term average number of customers waiting in queue 1 should then approximately equal 0.02051, and that of queue 2 should approximately equal 0.01111. Discuss the outcome of your test in your report.

(10pts) (c) Run your simulation for $m = 5$ with parameters $\lambda = 3.0$, $\mu_j = j + 3.0$.

For each of $\theta \in \{0.60, 0.85\}$, generate the following table, include it in your report, and discuss your findings:

Queue	Sample mean of the long-term average number of customers	Confidence interval
1	...	...
2	...	...
...	...	...
m	...	...

3 Reinforcement learning (RL)

Artificial intelligence techniques are actively being researched, and for instance, the field of *machine learning* is concerned with the development of algorithms that can learn (optimal decisions) from data. Within the field of machine learning, "RL" refers to certain type of algorithm—a certain approach to decision taking/statistical learning, if you will.

There are consequently many different RL algorithms. If you are broadly interested in the topic, then I recommend for you to read [2]. Related courses, e.g. on decision theory, are also available later in your study program. But for now and for simplicity, this assignment focuses on a specific version of the so-called "State–Action–Reward–State–Action (SARSA) algorithm".

SARSA is a RL algorithm for learning a good policy for a *Markov decision process*. It works by estimating so-called *qualities* of different actions taken in different states, and simultaneously using its belief of the qualities of different actions to choose next actions.

The qualities are typically tracked in a map $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Here, $\mathcal{S}$ represents the "state space" of a system, and $\mathcal{A}$ the "action space" of a system. SARSA's name reflects that the function for updating the Q-map depends on:

- The current state of the system $s_n \in \mathcal{S}$ at iteration $n \in \mathbb{N}_+$ of the algorithm.
- The action $a_n \in \mathcal{A}$ the agent[2] chooses in the state.
- The reward $r_n \in \mathbb{R}$ the agent gets for choosing the action.
- The state s_{n+1} the agent is in at iteration $n + 1$ of the algorithm, after having taken the action.
- And finally the next action a_{n+1} the agent chooses at the algorithm's $(n + 1)$st iteration.

The acronym for the quintuple $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$ is SARSA. [3]

[2] I.e., the intelligent dispatcher in this assignment.

[3] Alternatively, $(s_n, a_n, r_{n+1}, s_{n+1}, a_{n+1})$, depending to which time step the reward is formally assigned.

3.1 Implementing SARSA

Here is pseudocode that describes one implementation of SARSA:

Input: An initial state $s_1 \in \mathcal{S}$; a sequence of exploration probabilities $\varepsilon_1, \varepsilon_2, \ldots \in [0,1]$; a sequence of discount factors $\alpha_1, \alpha_2, \ldots \in (0,1]$; a sequence of learning rates $\gamma_1, \gamma_2, \ldots \in (0,\infty)$; and an iteration horizon $N \in \mathbb{N}_+$.
Output: A sequence of actions taken $a_1, a_2, \ldots, a_N$.

begin
  For as long as some state–action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ hasn't been seen, treat $Q(s,a) = -\infty$.
  Take the initial action a_1 uniformly at random: $a_1 \sim \text{Unif}(\mathcal{A})$.
  Set $Q(s_1, a_1) = 0$.
  for $n \leftarrow 1$ **to** $N-1$ **do**
    Observe reward r_n.
    Observe state s_{n+1}.
    Flip a biased coin that shows "Heads" with probability ε_n.
    if *the coin shows heads* **then**
      "Explore" by choosing the next action uniformly at random: $a_{n+1} \sim \text{Unif}(\mathcal{A})$.
    else
      "Exploit" by taking an action you currently think has the best quality: $a_{n+1} \in \arg\max_{a \in \mathcal{A}} Q(s_{n+1}, a)$. Break ties uniformly at random.
    end
    if $Q(s_{n+1}, a_{n+1}) = -\infty$ **then**
      Set $Q(s_{n+1}, a_{n+1}) = 0$.
    end
    Update $Q(s_n, a_n) \leftarrow (1-\gamma_n) Q(s_n, a_n) + \gamma_n \big(r_n + \alpha_n Q(s_{n+1}, a_{n+1}) \big)$;
  end
end

3.2 Configuring SARSA

In order to use the pseudocode above and to actually be able to implement the SARSA algorithm, one needs to specify (i) the exact moments at which actions are taken and (ii) one's goals. Together, this implies (iii) a certain reward structure. [4]

3.2.1 The exact moments that actions are taken

Let $N(t) \in \mathbb{N}_0^m$ denote the vector describing the number of customers waiting at time $t \in [0,\infty)$. Here, $\mathbb{N}_0 := \{0,1,2,\ldots\}$. Have a look at the caption of Figure 1 for an illustration of $N(t)$. Also, let $I(t) \in \{0,1\}^m$ denote the vector describing which servers are idle (available). That is, if $I_j(t) = 0$, then server j is idle; or if $I_j(t) = 1$, then server j is busy.

Recall next that arrivals occur at times $T_1, T_2, \ldots$, *et cetera*. We will assume that the dispatcher's nth routing decision occurs at time T_n, and specifically that at iteration $n \in \mathbb{N}_+$, the system state

$$s_n = \big(N(T_n^-); I(T_n^-) \big) \in \mathbb{N}_0^m \times \{0,1\}^m,$$

i.e., the queue lengths and server statuses within the system right before the arrival.

Suppose for example that $m = 2$ and a new customer arrives at time T_n that encounters system state $s_n = (3,0;1,1)$. This means that currently, there are three customers waiting in front of queue 1; there are zero customers waiting in front of queue 2; server 1 is busy with a customer; and server 2 is also busy with a customer. Note in particular that there are five customers in the system of which three are actually waiting.

3.2.2 The actions that the dispatcher can take

The SARSA dispatcher can either *reject* a customer (say $a = -1$), or *route* a customer to a queue $j \in [m]$ (say $a = j$). In other words,

$$\mathcal{A} = \{-1\} \cup \{1, 2, \ldots, m\}.$$

Note that with this action space, SARSA agent will <u>not</u> be searching for "optimal state-based rejection probabilities". Instead, SARSA will search for "optimal state-based deterministic routing decisions".

[4] Alternatively, (i) and (iii) are sometimes given in a system, and (ii) is then implied.

3.2.3 Our goal and the associated rewards

Now let $\xi \in \mathbb{N}_0^m$ and suppose that our goal is to maximize the long-term probability of observing ξ customers waiting in the system, i.e., to

$$\max_{\text{all possible policies}} \lim_{t\to\infty} \mathbb{P}[N(t) = \xi].$$

The aforementioned SARSA algorithm tries to achieve exactly this when the dispatcher receives reward

$$r_{n+1} = \int_{T_n}^{T_{n+1}} \mathbb{1}[N(t) = \xi]\,\mathrm{d}t$$

right before the dispatcher makes its routing decision (so again, at time T_{n+1}^-). You can interpret R_{n+1} as being the "total amount of time that the event $\{N_1(t) = \xi_1 \wedge N_2(t) = \xi_2 \wedge \cdots \wedge N_m(t) = \xi_m\}$ was true in the interval $[T_n, T_{n+1})$".

Tasks

(20pts) 2. [Medium] Having programmed your environment in the previous task, now program your agent: a *reinforcement learning dispatcher* that uses SARSA. Discuss your implementation in your report.

3. [Hard] Suppose that $m = 1$, $\lambda_1 = 7/10$, and $\mu_1 = 1$. For $n \in \mathbb{N}_+$, let the discount factor be $\alpha_n = 0.9$, the learning rate $\gamma_n = 0.2$, and the exploration probability

$$\varepsilon_n = \begin{cases} 1 & \text{if } n < 1000, \\ \max\{0.99(1 - \frac{n-1000}{10000}), 0.01\} & \text{otherwise.} \end{cases}$$

Conduct and report on the following experiments:

(10pts) (a) For $\xi_1 = 2$, measure the performance over time of the SARSA dispatcher that you have implemented. Do enough independent replications to get reliable conclusions.

(5pts) (b) Play around a bit by varying $\xi_1 = 2, 3, 4, \ldots$, and inspect the quality map Q that SARSA finds after many iterations each time. What deterministic policy appears to be optimal, if any?

(15pts) (c) Design and conduct an experiment with SARSA for a system with at least $m \geq 3$ queues and (on average) $\sum_j \mu_j \leq 1$. You may do any experiment that seems scientifically interesting to you, as long as the two previous criteria are met *and* you get the RL algorithm to work. You are most certainly allowed to change the rewards, and indeed, even SARSA itself.

Yes: you have considerable freedom in this question, so you can go wild! The sky is the limit ☺.

Be warned that you may suffer from the curse of dimensionality at larger system sizes, such as $m \gtrsim 5$. The state–action space can quickly become quite large. If you come up with a clever solution and are able to do RL in a relatively large system, then you will impress us. (Yes, such solutions exist, and are actively being researched.)

4 Deliverable(s)

Tasks

(20pts) 4. Write a report in LaTeX that describes your results and hand it in as a compiled PDF.

These points are awarded to reports (i) that have a well-designed, canonical structure such as "abstract, introduction, methodology, discussion, conclusion", and (ii) in which have all explanations/observations/discussions/conclusions flow naturally from one to the other, have depth, and are well-presented.

Merely handing in a collection of results, e.g. with tasks numbers as section titles, is insufficient.

Knockout criteria

You must do the following in order to be marked:

(i) Meet the deadline, by:
Submitting a PDF file containing an electronic copy of your report to Canvas, on time.
Submitting a ZIP file containing all of your code to Canvas, on time.

Also, your report must meet the following criteria in order to be marked:

(i) It includes a title, author names, student numbers, and a bibliography when citing.
(ii) It includes an acknowledgments section, in which any entity that contributed in some manner is mentioned (any entity other than the authors). Examples include other programmers and/or students, but also generative artificial intelligence and public discussion forums from which you may have sourced material. *Remember that as a team you must try to do the entire assignment by yourselves, i.e., without outside help. Still, if you do decide to partake in some discussion or if you opt to use some tool, then you must acknowledge the help that you received.*
(iii) It includes a paragraph in the Appendix, in which every student briefly explains their specific contributions. In particular, tell us which percentage of the (a) analysis, (b) report writing, and (c) programming you did. *Every student is expected to contribute significantly to every component.*
(iv) It finally includes all of your code, verbatim, at the end of the Appendix.
(v) There is a hard page limit of **eight A4 papers**, single column. This excludes the bibliography, assignment contribution statement, acknowledgments section, and code in the Appendix.
Focus on what is important: it is about quality of content, not quantity.
(vi) Margins should be at least 2 cm, and font size at least 10 pt.
(vii) The overall presentation is clean / neat / orderly.
(viii) Your texts are legible, in English, and contain few spelling mistakes.

Grading

Try your best and don't give up if you find this assignment challenging! I will judge you for effort, so understand that you may report difficulties or even failure.

Still, your aim should most certainly be a perfect implementation for that perfect mark.

References

[1] Céline Comte, Matthieu Jonckheere, Jaron Sanders, and Albert Senen–Cerda. "Score-Aware Policy-Gradient Methods and Performance Guarantees using Local Lyapunov Conditions". In: (2023).

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.