INST0060 Investigation report: Hospital Queuing problem

Group T1*

University College London, WC1E 6BT

April 29, 2020

Abstract

Queuing systems are a common subject of study in reinforcement learning. Here, we simulate a hospital with doctors and patients, each with their own characteristics, and use reinforcement learning to optimize the allocation of arriving patients to a queue.

1 Introduction

1.1 Background

In recent times, hospital queuing times have become a concern. Because hospital represent clusters of vulnerable individuals, it can be of vital interest to limit the number of people standing in line or in a waiting room.

According to a 2015 study, on average Americans spent 84 minutes per visit in the hospital during the 2003-2010 period. [3]

A long-term goal to accommodate the increasingly large flow of patients would be to improve healthcare infrastructures. Yet, because staffing and finance issues are so context-sensitive in many countries, it is not possible to directly ascertain which strategy to follow. However, a more efficient allocation of pre-existing resources can be beneficial in many cases. Furthermore, some decisions in such complex systems could potentially be automated for higher reactivity and better performance.

Reinforcement learning is a form of goal-directed learning with interaction; an agent learns how to respond to situations with certain actions so as to maximize a reward. [5] In a sense, a reinforcement learning agent explores and exploits past experience.

Queuing problems have been optimised using reinforcement learning in the past (such as traffic intersections [4]), and hospitals have been modelled statistically as well. [1, 2]

In this work, we use reinforcement learning to allocate patients to queues as efficiently as possible. First, we will describe the system as modelled (with simplifying assumptions) and the methods used to learn policies. Then, we will analyse the resulting policies in simple cases, to gauge the ability of learners compared to systematic or random policies and to human intuition. We will also reflect on our experience of the project, discussing the various difficulties encountered and what steps were taken in response.

2 Methods

In the first instance, we tried modelling the hospital system as a Markov decision process (or MDP) in continuous time, in order to maintain proximity with both a real hospital system and the material we took away from the INST0060 lectures. However, we quickly realised that we were not equipped to simulate a system in continuous time, and that the complexity of the system (even with our simplifying assumptions) prevented us from using an MDP or table-lookup approach. Hence, we decided to approximate a hospital system by a discrete-time process with a patient arrival every timestep and developed featurisations that allowed for the use of function approximation.

2.1 Model characteristics

The hospital contains servers (doctors) of different types, which correspond to their abilities. Similarly, jobs (patients) have different priorities that warrant different services. For example, a doctor of type 1 can treat a patient with priority 1 or lower, but a doctor of type 0 *cannot* treat a patient with priority 1 or higher.

In general, if the highest doctor type is N at initialisation, then it is expected that there be at least one doctor of each type $n \in \{0, 1, \ldots, N\}$. Each *type of* doctor has an associated queue, hence thereafter we call "queue n" the unique queue which doctors of type n draw from. Newly arrived patients have needs sampled from $\{0, 1, \ldots, N\}$ according to a

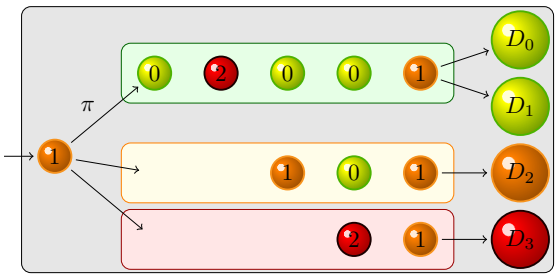

Figure 1: A hospital with 4 doctors: two of type 0 (D_0 and D_1, green), one of type 1 (D_2, orange) and one of type 2 (D_3, red). A newly arrived patient with priority 1 is being allocated to a queue according to policy π. Note the priority 2 patient mistakenly sent to queue 0; they will not be sent away until they have reached the end of the queue.

known distribution and are allocated to a given queue in $\mathcal{A} = \{0, 1, \ldots, N\}$ by an agent following a policy π. Note that, once allocated, a patient may not change queues, though they *may be allocated to a doctor who cannot treat them* (see Fig. 1). Once they are received by a doctor, if there is a discrepancy, the patient is sent away.

The state space is considered discrete, for simplicity. At each time step:

- a new patient arrives, with priority level sampled at random;
- one or more doctors might be finished treating a patient; when that is the case, the "cured" patient leaves and is replaced by the next-in-line.

Each individual doctor has their own efficiency, which is modelled by the probability of being done with a patient at each timestep.

Hence, if a skilled doctor happens to also be very fast, it might be beneficial to allocate low priority patients to their queue, to deplete the size of queues more quickly; however, this runs the risk of making patients with high priority wait longer.

In this study, we restrict ourselves one doctor per type, as this still offers ample flexibility. The general case could be the subject of future work. Similarly, though this study started with one queue *per doctor* rather than per *type* of doctor, we simplified the system after discussions with the module administrators and TAs.

One of the challenges associated with learning a policy to set balanced priorities between curing patients and keeping the hospital from getting too full. Because of the behaviour of misallocated patients (see Fig. 1), it can be more efficient for the agent to just misallocate patients from the start so as to keep the hospital empty.

The agent's actions tend to have very delayed consequences, which makes it difficult for it to link rewards and policy. Further, originally the penalty for misallocation was set to trigger only when patients reached the doctor. This mechanic was modified midway through the project to investigate its effect, and this will be the subject of Experiment 3.

2.2 Learning

Two well-known algorithms of RL are the Q-learning (QL) and the SARSA algorithms. [5] SARSA is an on-policy algorithm, meaning the agent learns the value of the state-action pair on the basis of the performed action. By contrast, QL is an off-policy algorithm, where the state-action-value function (or q-function) is estimated based on a greedy policy, which is not necessarily the currently implemented policy. Generally, SARSA tends to exhibit faster convergence, while the performance of Q-learned policies tend to be higher. However, in SARSA the learner can easily get stuck in local minima, and QL tends to take longer to converge.

2.3 Featurisation

Even with the limited complexity of its modelling, the number of individual states in the hospital is extremely large, since each state includes the number of people in each queue, but also the types of patients and their waiting times. In fact, it is countably infinite if the hospital occupancy in not bounded (which it is during training), so that a simpler representation (a simple vector) must be used for the agent to learn efficiently.

While at the beginning a few real numbers were used (such as the average number of patients per queue), later on an effort was made to include more information (real numbers, but converted to discrete values) and finally, one-hot vectors. This last attempt was the most successful and will be applied throughout this work; the impact of featurisations will be the subject of experiment 1.

In this work, the systems of interest will be:

Experiment 1: Two doctors (one of type 0 and one of type 1) with one featurisation and one learning algorithm. The

Experiment 2: We analyse the behaviour of the agent as the frequency of urgent patients increases.

3 Results

3.1 Experiment 1: Featurisations

Here we demonstrate how the importance of the choice of feature vector. The feature vectors below gener-

ally fall into two categories: one-hot vectors (featurisations 7-12) and non-one-hot vectors (featurisations 1-6). These can be found in the `learning.py` file. EXPLAIN EXPERIMENTAL CONDITIONS: which reward system, how many tests for each featurisation, which learning algo... try to justify choices

The data shown in Table 1 implies that one-hot vectors are clearly the better choice since most of them achieve close to maximum reward. The upper bound of the reward in these simulations is 10 000. However, the difference between individual one-hot featurisations can also be significant.

Feature 12 appears to be the clear winner.It does not only achieve a higher reward on average but is also more consistent (given the lower deviation).The information that it takes into account is the level of sickness of the next patient to be allocated, the wait time for the first patient on each queue and the number of patients by types in each queue.

Mb should add a picture of the histogram of the allocated patients? reward evolution ? Adding/removing some parameters in the table ?

The table below presents useful statistics about some of the featurisations.Each experiment is conducted 100 times in order to obtain this data.

Rewards	Mean	Median	Std.dev
feature 12	9260	9478	647
feature 7	9041	9357	1247
feature 8	8698	9141	1362
feature 9	9067	9485	1072
feature 10	941	1636	2540
feature 11	8876	9523	1532
feature 1	6953	7182	1271

Table 1: Rewards depending on the featurisation

3.2 Experiment 2: Model behaviour (code in `fastdoc_exp.py`)

In this experiment, we try to characterise the behaviour of the agent in ambiguous situations. The hospital has four doctors, of types 0 through 3. Doctor 3 (the most highly skilled) is also very efficient, while lower skilled doctors are equally slow.

TBD We set the featurisation to `feature_12`, the learning algorithm to SARSA and a reward system to penalise misallocations early and not penalise when occupancy is reached, as these seem to yield better performance overall.

Then, we gradually increase the frequency of type 3 patients. When there are few, the agent should allocate many low priority patients to doctor 3 because it treats more quickly. However, when there are many, allocating low-priority patients to doctor 3 causes type 3 patients to wait longer, so we expect a decrease in total allocations to queue 3.

Figure 2: Behaviour of agent as priorities change. For each probability, the agent was trained for ??? episodes with ??? steps each.

The results are shown in Figure 2

4 Discussion

5 Conclusion

Declaration

This document represents the group report submission from the named authors for the project assignment of module: Foundations of Machine Learning and Data Science (INST0060), 2019-20. In submitting this document, the authors certify that all submissions for this project are a fair representation of their own work and satisfy the UCL regulations on plagiarism.

References

[1] Jiekun Feng and Pengyi Shi. Steady-state diffusion approximations for discrete-time queue in hospital inpatient flow management. *Naval Research Logistics (NRL)*, 65(1):26–65, 2018.

[2] Matthew S. Hagen, Jeffrey K. Jopling, Timothy G. Buchman, and Eva K. Lee. Priority queuing models for hospital intensive care units and impacts to severe case patients. *AMIA Annual Symposium proceedings Archive*, 2013:841–850, 2013.

[3] Kristin N. Ray, Amalavoyal V. Chari, John Engberg, Marnie Bertolet, and Ateev Mehrotra. Opportunity costs of ambulatory medical care in the United States. *The American Journal of Managed Care*, 21(8):567–574, 2015.

[4] Azura Che Soh, Mohammad Hamiruce Marhaban, Marzuki Khalid, and Rubiyah Yusof. Modelling and optimisation of a traffic intersection based on queue theory and Markov decision control methods. In *First Asia International Conference on Modelling & Simulation (AMS'07)*, pages 478–483. IEEE, 2007.

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