INST0060 Investigation report: Hospital Queuing problem

Auguste Baum, Yongxin Chen, Isabel Jiang, and Todor Todorov

University College London, WC1E 6BT

May 21, 2020

Abstract

The queuing problem is a common subject in reinforcement learning study. Here, we simulate a hospital with doctors and patients that each have their own characteristics, and use reinforcement learning to optimize the allocation of arriving patients to a queue. The behaviour of the system is confronted to intuition, and various featurisations and learning algorithms are compared. Provided it has had enough training, an agent trained with our best featurisation and with the right reward system can achieve high efficiency.

1 Introduction

1.1 Motivations

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. We then study the impact of

1. featurisation,
2. model parameters,
3. learning algorithm and
4. reward system

on the resulting policy.

We will also reflect on our experience of the project, discussing the various difficulties encountered and what steps were taken in response.

1.2 Background

In recent times, hospital queuing time 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. [1]

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. [3] 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 [2]), and hospitals have been modelled statistically as well.

2 Methods

In the first instance, we tried modelling the hospital system as a Markov Decision Process (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 a 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 develop featurisations that allow 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 $n \in \{0, 1, \dots, N\}$. 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 $\{0, 1, \dots, N\}$ according to a known distribution and are allocated to a given queue in $\mathcal{A} = \{0, 1, \dots, N\}$ by an agent following a policy π. Note that, once allocated, a patient may not change queues, even though they *may be allocated to a doctor who cannot treat them* (see Figure 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 finish 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 time step.

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 priorities wait longer. Though our implementation does allow for more, we restrict the study to one doctor per type, as this still offers ample flexibility. 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.

In order to guarantee termination, we set the episode length to a given number of steps, but the episode also terminates if the number of people in the hospital becomes equal to the occupancy. Whether this should be penalised or not is the subject of Experiment 3 (among other things).

One of the challenges associated with learning an efficient policy is to balance the priorities between curing patients (i.e. not misallocating

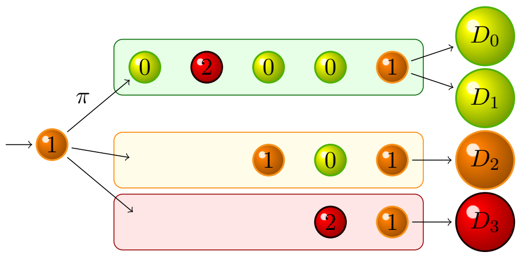

Figure 1: A hospital with 4 doctors: two of type 0 (D_0 and D_1, green), one of type 1 (D_2, orange) and one of type 2 (D_3, red). A newly arrived patient with priority 1 is being allocated to a queue according to policy π. Note the priority 2 patient mistakenly sent to queue 0; they will not be sent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

them) and keeping the hospital from getting too full. Indeed, because of the behaviour of misallocated patients (see Figure 1), it can be more efficient for the agent to just misallocate patients from the start so as to keep the hospital empty. The consequences of the agent's actions tend to be very delayed, 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. In order to investigate effect of this mechanic, we tried to modify it; this will be also be part of Experiment 3.

2.2 Learning

Two well-known algorithms of RL are the Q-learning and the SARSA algorithms. [3] SARSA is an on-policy algorithm, meaning the agent learns the value of the state-action pair on the basis of the performed action. Q-learning is an off-policy algorithm, with the state-action-value function (or q-function) estimated based on a greedy policy, which is not necessarily the same as the implemented policy. The performance of each will be evaluated in experiment 3, along with the reward system.

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 time. In fact, it is countably infinite if the number of people in the hospital is 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 were implemented. 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.

2.4 Performance criteria

Throughout our experiments, policies are evaluated by simulating a hospital identical to that used for training (except the simulation is not ended if the capacity is reached), and by recording a number of variables. These include the "misallocation rate" (the proportion of patients that were allocated to an unqualified doctor), the waiting time of patients, the evolution of rewards and the number of patients that received treatment, among others. However, these measurements should be interpreted with caution as their values are often dependent on multiple factors. For example, the waiting time of patients could be high because the policy is inefficient, but also just because the doctors are intrinsically slow. The following experiments are performed:

Experiment 1: We compare different featurisations for the hospital.

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

Experiment 3: We study the effect of both the reward system and learning algorithm on policy performance.

3 Evaluation

3.1 Experiment 1: Featurisations (code in `feat_exp.py`)

In this experiment, we compare the performance of different featurisations. The hospital has six doctors, of types 0 through 5. Here, doctors 0, 1 and 4 are inefficient (probability of being finished treating is 0.1 at each timestep), doctors 3 and 5 are efficient (0.4 and 0.5, respectively) and doctor 2 is highly efficient (0.9). An arriving patient has equal chances of having any need (0 through 5). The learning process consists of 100 episodes with 100 steps each, using SARSA. The resulting policy is then used in a 100-step-long simulation, and the average reward is recorded. For each feature, the learning and evaluation process is repeated 300 times. This is essential since our model incorporates a certain level of randomness.

The most important characteristic of the current model, apart from the definition of the used featurisation, is the reward system. In its current state, the system gives a penalty as soon as a misallocation happens, another based on the waiting time of patients and it gives a constant positive reward for each treated patient.

The variety of featurisations that were developed can be found in `learning.py`. They generally fall into two categories: one-hot vectors (featurisations 7 through 13) and non-one-hot vectors (1 through 6). We will now describe further the construction of some of them and their efficiency.

`feature_1`, fittingly, constitutes our first attempt at a featurisation. The information it holds is, essentially, the type of the newly arrived patient to be allocated (as an integer) and integers representing how many people of each type are in any given queue. Using this feature the algorithm does seem to exhibit some learning behaviour but it is far from being satisfactory or consistent. Namely, the agent seems to recognize which doctors are more efficient but fails to properly allocate the patients. This is not surprising considering how limited the representation is:

- The new patient's type is only represented by a single integer;
- the feature vector conveys proper information about the distribution of patients in different queues only when they are approximately equally distributed and in small numbers (up to 2 persons of a particular type in a queue);
- there is no information about the patient waiting times.

By contrast, `feature_7` is our first attempt at a one-hot vector featurisation, which is a concatenation of all the information from `feature_1` along with the mean waiting time of the waiting patient, as one-hot vectors. For example, in our experiment we have 6 distinct doctor types, so if in a given state the new patient is of type 1 then the first 6 elements of the featurisation would be `[0,1,0,0,0,0]`. As seen in Figure 2, the resulting policy seems to produce much better results than the naïve policy (which allocates all patients of type n to queue n) or `feature_1`.

`feature_13` holds the same data as `feature_7` but encoded using slightly different conditions. However, the empirical data does not suggest a significant performance increase compared to `feature_7`.

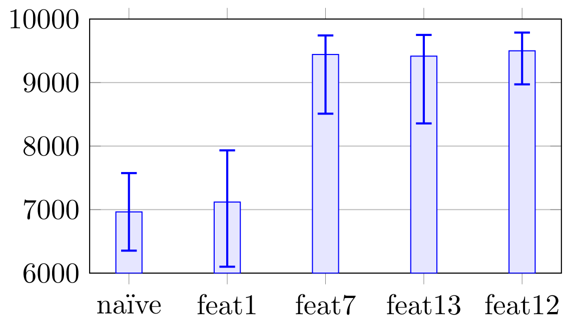

Figure 2: Median simulation reward using different features. Error bars show 20th and 80th percentile (to represent the high skew). The maximum possible reward in this case is 10000.

`feature_12` is essentially a slightly upgraded version of the previous two. Instead of the mean waiting time, this featurisation includes the waiting time of the next-in-line in each queue. This allows for a better distinction between the queues. As can be seen in Figure 2, this only increases the median reward marginally, but it does reduce the deviation from the median. We also note that policies learned using this featurisation consistently allocate patients to doctors who can actually treat them: the misallocation rate is 0% consistently, whereas for `feature_7` and `feature_13` it is usually about 1–2%.

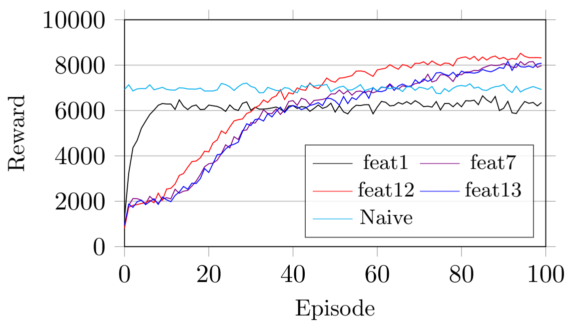

Figure 3: Episodic rewards during the learning process (data averaged from 50 simulations, each with 100 episodes).

The expressiveness of one-hot vectors allows the agent to form near-optimal policies, provided that the featurisation includes all the relevant characteristics of the system. As seen in Figure 2 and Figure 3, `feature_12` performs best out of all trialled featurisations, which is expected considering it contains the most useful information in the most expressive way possible.

In Figure 4 we show an optimal policy for this experiment, with regards to the present reward system. Though it might seem odd for the agent to leave some doctors without any work, this is expected behaviour in this instance: the agent considers it too expensive to keep people waiting for the slow doctors rather than assigning them to the faster ones directly, even though they might be overqualified (note that there is no penalty for allocating to an overqualified doctor).

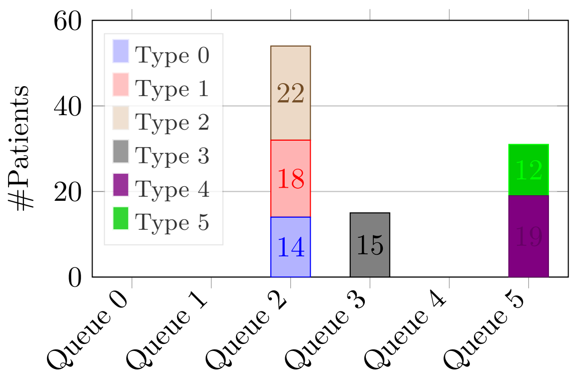

Figure 4: Example distribution (policy) of patients' types for the current experiment using `feature_12`. There are 100 patients in total.

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. Here, the hospital has four doctors, of types 0 through 3. Doctor 3 (the most highly skilled) is very efficient (80% chance of treating a patient at each time step), while all lower skilled doctors are equally slow (40%).

We set the featurisation to `feature_12`, the learning algorithm to SARSA and the reward system to penalise misallocations early and not penalise when occupancy is reached, as these seem to yield better performance overall. These assumptions are the object of experiments 1 and 3.

New patients are equally likely to be of type 0, 1 and 2, and we vary the probability of *type 3 patient arrivals* during training, from 10% to 90%. When there are few urgent patients, we would expect the agent to 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, which is detrimental to the reward. Hence, the dependent variable of interest is the proportion of patients *allocated to queue 3*.

For each data point, we train an agent on a hospital with a given arrival probability and then apply the learned policy to a hospital with *equiprobable* arrivals, so that policies can be compared on equal footing.

We start by running a "short-term" trial, in which for each probability, the simulation (training and testing) was carried out 300 times; for each simulation the agent was trained for 50 episodes with 500 steps each. The results are shown in Figure 5 (blue bars).

For low probability, the behaviour seems to agree with our predictions: as urgent patients become more frequent, fewer low-priority patients are given to the high-skill doctor.

However, contrary to what was announced, comparing policies the way we do in Figure 5 is not necessarily viable, considering *the agent might not have had equal access to each state* in each circumstance: for instance, when type 3 patients arrive 90% of the time (see far right), the agent gets little experience with treating lower priority patients. In this case, since patients of type 3 can only be put in queue 3 (any other decision would result in a misallocation penalty), the agent learns to put any arriving patient in queue 3, irrespective of their type, which leads to what is observed and goes against our intuition.

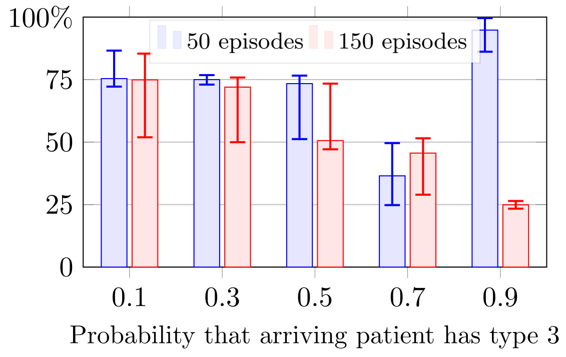

Figure 5: Median proportion of patients allocated to queue 3 as priorities change. Error bars show 20th and 80th percentile (to represent the high skew).

In response to this unexpected, though easily rationalised behaviour, we also ran a "long-term" trial, with the same parameters except with training lasting 150 episodes instead of 50. This way, the model can gather more meaningful experience with each type of patient during training, even when arrivals of some patient type are sparse. For each point, the simulation was carried out 100 times (which took around 4 processor hours on a modern laptop). The result is in Figure 5 (red bars).

It would seem that in general the agent is more cautious about allocating patients to queue 3, particularly when urgent patients are frequent: in that case the proportion is about 25%, which corresponds to only the type 3 patients being put in queue 3 (since during testing all patient types are equally probable).

Accordingly, the waiting time of type 3 patients is drastically decreased when the type 3 queue is reserved for them: in this case it is 0.1 timesteps whereas it sits between 30 and 50 timesteps when all patients are allocated to queue 3.

3.3 Experiment 3: Penalty check (code in `rewards_exp.py`)

Experiment 3 compares learned policies with different combinations of two penalty mechanics, the penalty for misallocation (governed by the `earlyRewards` parameter) and the penalty for reaching the occupancy of the hospital (by the `capacity_penalty` parameter).

`earlyRewards=True` corresponds to penalising any misallocation as soon as it is performed (the patient is still put in the queue); the alternative is to issue the penalty only once the patient has reached the doctor.

`capacity_penalty=True` causes the agent to be harshly penalised when there are as many patients in the hospital as the maximum occupancy. The alternative is no penalty. In all cases, this also causes the premature end of the current episode.

We need to study a system that can saturate when treated naively (otherwise the size penalty is never applied) but where the policy can be optimised to avoid saturation: if it is not possible, then the agent is never penalised, and so the penalty loses its meaning. Hence, here we use three doctors from type 0 to type 2, with the type 1 doctor having an efficiency of 0.4 (40% chance of treating a patient at each time step) and the other two equally efficient with 10% chance of treating a patient at each time step. An arriving patient has equal chances to be of any type (0 through 2). The capacity of hospital (`capacity_hospital`) is set to be 40.

The featurisation is `feature_7` and the two implemented algorithms, SARSA and Q-learning, are contrasted in the experiment. Four combinations are possible for the reward system: misallocations can be penalised early or late, and the capacity being reached can be penalised or not.

In this experiment, we examine the influence of the reward system and learning algorithm on the misallocation rate and the waiting time of patients.

For each combination, we train and test an agent 40 times and compute the both quantities. For the waiting time specifically, the median is computed for each queue and the estimate is the median of those.

We expect that in general, when the penalty for misallocation is issued as soon as the patient is assigned to the queue, the misallocation rate should be low. The rationale here is that when the misallocation penalty is issued early, the agent gets clearer feedback (otherwise, the other penalties make it difficult for the agent to link the penalty to its actions and update its policy accordingly).

Similarly, the size penalty is meant to be large enough to clearly indicate to the agent not to let the occupancy be reached, so that when it is active, the average waiting time should be low.

In fact, we expect there to be some degrees of trade-off between misallocation rate and waiting time, as our hospital model can be abused somewhat: although a misallocated patient must go through the whole queue regardless of the reward system, when it reaches a doctor it is sent away *immediately*. Hence, misallocating a patient to an empty queue *immediately* removes that patient from the hospital (albeit resulting in a misallocation penalty).

In Figure 6, both measurements taken using

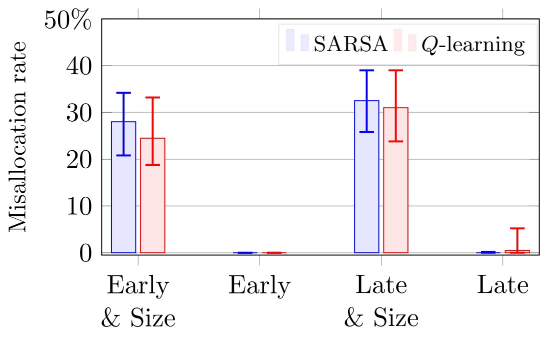

*Figure 6: Median proportion of misallocated patients, depending on learning algorithm and reward system.**

the `capacity_penalty` yield similarly high misallocation, while those without it show almost none at all. It would seem that, as predicted, the size penalty drives the agent to empty the hospital by any means necessary.

In this particular case, it would seem that using Q-learning with early misallocation penalties and no capacity yields the best performance, as the policies tend not to misallocate and to yield comparatively lower waiting times.

By contrast, all other combinations yield similar waiting times (see Figure 7).

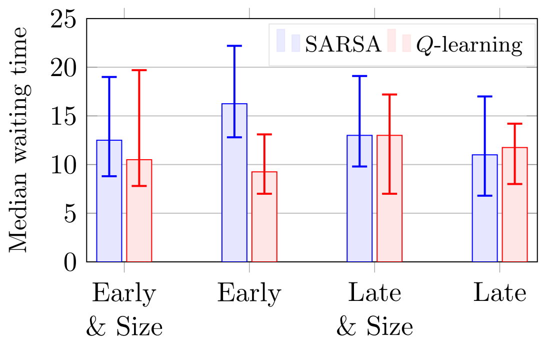

*Figure 7: Median waiting time of patients depending on learning algorithm and reward system.**

4 Discussion

In this work, we modelled a complex queuing system and applied reinforcement learning techniques to find policies for efficiently allocating jobs to servers. The focus was on the result of the learning, how varying learning methods could influence the performance, and whether or not the agent's behaviour was consistent with our intuition.

Our experiments shed some light on various aspects of our approach to modelling and controlling the hospital queuing system. First, using the internal state directly to learn a policy, even for our simplified model, is intractable. This lead us to experiment with different representations for the hospital.

It would seem that featurisations in the form of a one-hot vector are superior in terms of the performance of the resulting policy. A reason for this is their greater number of degrees of freedom, which makes feature space complex enough. This was the subject of experiment 1,

* *"Early" and "Late" refer to when the penalty for misallocation is issued. "Size" corresponds to the capacity penalty mechanic being active. Error bars show 20th and 80th percentile.*

in which we could also see that policies learned with the right featurisation can outperform the naive policy.

On the other hand, Figure 4 shows that designing a system that can perfectly discriminate between all available policies is rather difficult: a real hospital would obviously not want some of its doctors working constantly and others doing nothing. One might choose to implement a simple heuristic that keeps all the doctors busy or alternatively devise a more elaborate reward system including other parameters, like fatigue or salary.

Furthermore, we note that increasing the model complexity (by taking into account more parameters, as suggested earlier) could render the featurisations we developed inefficient. The scalability of our project could be enhanced by applying a secondary mapping to the currently implemented state representation (like radial basis functions or monomials).

The single most powerful trait of the current reward system is that it assigns a constant positive reward for each treated patient. This both mitigates misallocation of patients and reduces waiting time as the algorithm strives to maximise the total reward at the end. The size of this constant, though, needs to be chosen relative to the size of the potential penalties.

In experiment 2, we analysed the behaviour of the learned agent in an intuitive yet nontrivial example, where one doctor is both very experienced and very efficient. Though the measurements are affected by the training processes itself, it seems that with enough training the agent adopts coherent behaviour with respect to the reward system (which is itself based on our subjective intuition, admittedly).

Experiment 3 tested the influence of different penalty systems and reward systems on the misallocation rate and waiting time. Interestingly, in most cases, no evident discrepancies between the two algorithms was found, the exception being with early rewards and no size penalty, where Q-learning was superior. Most notably, a large penalty for letting the hospital get full leads to a large misallocation rate; we suspect that this undesirable behaviour is heavily dependent on the size of this penalty.

Initially, the misallocation penalty was always issued late—the investigation would have consisted of consistently finding a good policy even with delayed feedback. We then decided to try issuing rewards early, which seemed to yield much more satisfying performance (especially with respect to misallocation).

The fact that the final reward is a sum of individual contributions (i.e. from misallocations or full hospital) probably makes it more difficult for the agent to link its actions to its consequences. An interesting avenue would be to partition the rewards, so as for the learner to have access to the “marking scheme”, so to speak.

Additionally, though we decided to keep it fixed for the whole duration of the project, the influence of the discount factor γ on the learning with late misallocation penalty could have been investigated. Indeed, γ governs the importance given to events far in the past.

In conclusion, despite the simplicity of its implementation, in its current state, we believe our project to be a success, with a learning procedure able to train agents to behave nearly optimally in a large array of situations, if given enough time.

Declaration

This document represents the group report submission from Group T1 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] 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.

[2] 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.

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