Internship report

Vo Van Nghia

Date

29 Sep, 2022

Table of contents

1 Introduction **1**

1.1 Institut de Recherche en Informatique de Toulouse (IRIT) . . . 1

1.1.1 About the institut . . . 1

1.1.2 Organization . . . 1

1.2 The internship . . . 1

2 System settings **2**

2.1 Queuing system . . . 2

2.1.1 Parameters . . . 2

2.1.2 Cost . . . 3

2.1.3 Agent . . . 3

2.1.4 Evolution . . . 3

2.2 Load-balancing system . . . 4

2.2.1 Parameters . . . 4

2.2.2 Cost . . . 4

2.2.3 Agent . . . 5

2.2.4 Evolution . . . 5

2.3 System representation . . . 5

3 Reinforcement learning **6**

3.1 Parameters . . . 6

3.2 Bellman equation . . . 7

3.3 Optimal policy . . . 8

4 Online Q-learning **8**

4.1 About the algorithm . . . 8

4.2 Implementation . . . 9

4.2.1 From Python to C++ . . . 9

4.2.2 From n-dimensions to 2-dimensions . . . 11

4.3 Result . . . 12

5 Offline Q-learning **12**

5.1 About the algorithm . . . 12

5.2 Result . . . 12

1 Introduction

1.1 Institut de Recherche en Informatique de Toulouse (IRIT)

1.1.1 About the institut

The Institut de Recherche en Informatique de Toulouse (IRIT), created in 1990, is a Joint Research Unit (UMR 5505) of the Centre National de la Recherche Scientifique (CNRS), the Institut National Polytechnique de Toulouse (INP), the Université Paul Sabatier Toulouse3 (UT3), the Université Toulouse1 Capitole (UT1) and the Université de Toulouse Jean Jaurès (UT2J).

IRIT is one of the largest UMR at the national level, is one of the pillars of research in Occitanie with its 600 members, permanent and non-permanent, and about 100 external collaborators. Due to its multi-tutorial nature (CNRS, Toulouse Universities), its scientific impact and its interactions with other fields, the laboratory constitutes one of the structuring forces of the IT landscape and its applications in the digital world, both at regional and national level.

Through its cutting-edge work and dynamics, our unit has been able to define its identity and acquire undeniable visibility, while positioning itself at the heart of changes in local structures: University of Toulouse, as well as the various mechanisms resulting from future investments (LabEx CIMI, IRT Saint-Exupéry, SAT TTT, 3IA ANITI).

IRIT has focused its research on five major scientific issues and six strategic application areas.

- Health, Autonomy, Living, Well-being
- Smart City
- Aerospace and Transportation
- Social Media, Digital Social Ecosystems
- e-Education for learning and teaching
- Heritage and People Safety

As well as strategic action:

- Scientific Computing, Big Data and AI

1.1.2 Organization

The 24 research groups of the laboratory are dispatched in seven scientific departments:

- Dpt ASR : Architecture, Systems, Networks
- Dpt CISO : HPC, Simulation, Optimization
- Dpt FSL : Reliability of Systems and Software
- Dpt GD : Data Management
- Dpt ICI : Interaction, Collective Intelligence
- Dpt IA : Artificial Intelligence
- Dpt SI : Signals, Images

1.2 The internship

Markov decisions processes (MDPs) and their model free counterpart in reinforcement learning (RL) have known a large success in the last two decades. Although research in these two areas has been taking place for more than fifty years, the field gained momentum only recently following the advent of powerful hardware and algorithms with which suprahuman performance were obtained in games like Chess or Go.

However, these impressive successes often rely on quite exceptional hardware possibilities and cannot be applied in many "usual" contexts, where, for instance, the volume of data available or the amount of computing power is more restricted. To define the next generation of more "democratic" and widely applicable algorithms, such methods still need to deal with very demanding exploration issues as soon as the state/action spaces are not small. One way around this is to use underlying knowledge and structure present in many MDPs. This is especially true for problems related to scheduling and resources sharing in among others server farms, clouds, and cellular wireless networks. The internships will revolve around this theme of improving the efficiency of learning algorithms by leveraging the structure of the underlying problem and focus mainly on modelfree approach.

2 System settings

Two type of systems are studied in this internship: queuing system and load-balancing system.

2.1 Queuing system

2.1.1 Parameters

We have n classes of queue $Q_1, \dots, Q_n$, and $L_1, \dots, L_n$ the maximum number of work (limit) on each class of queue. For each Q_i, its behavior is fully controlled by which environment it is in. An environment can be in one of the states $m_1, \dots, m_m$. And the environment of class Q_i is a random variable, denoted by M_i, which is in one of the states $m_1, \dots, m_m$. For each environment, Q_i has their own holding cost $c_{i,j}$ (the cost of one unfinished unit of work on the queue), arrival rate $\lambda_{i,j}$ (the rate of one more unit of work arriving to the queue) and departure rate $\mu_{i,j}$ (the rate of one unit of work on the queue has finished). Furthermore, the environment M_i can change from m_j and m_k with the rate of $\xi_{i,j,k}$. In this internship, we focus on the case where $n := 2$. A table summarizing all parameters is shown below.

Table 1: Queuing system parameters

		m_1	...	m_m
Q_1	Holding cost	$c_{1,1}$	...	$c_{1,m}$
	Arrival rate	$\lambda_{1,1}$	...	$\lambda_{1,m}$
	Departure rate	$\mu_{1,1}$	...	$\mu_{1,m}$
$\vdots$				
Q_n	Holding cost	$c_{n,1}$	...	$c_{n,m}$
	Arrival rate	$\lambda_{n,1}$	...	$\lambda_{n,m}$
	Departure rate	$\mu_{n,1}$	...	$\mu_{n,m}$

For each Q_i, we have a matrix transition as below.

Table 2: Matrix transition of the environment of Q_i

	m_1	...	m_m
m_1	$\xi_{i,1,1}$	...	$\xi_{i,1,m}$
m_m	$\xi_{i,m,1}$	...	$\xi_{i,m,m}$

The state of the system is represented by two vectors:

- $S = (X_1, \dots, X_n)$ where X_i is a random variable represents the current number of works of class Q_i and is observable.
- $E = (M_1, \dots, M_n)$ and this vector is not observable.

2.1.2 Cost

The cost of the system is a function of S and E. We propose two functions of cost. The first one is a simple linear function.

$$f_1(S, E) = \sum_{1}^{n} c_{i,M_i} X_i$$

And the second one is a convex function which is specialized for the case $n = 2$, where ϵ is a fixed positive constant.

$$f_2(S, E) = c_{1,M_1} X_1 + c_{2,M_2}(\epsilon X_2^2 + X_2)$$

2.1.3 Agent

The agent will decide which queues should be activated. His goal is to minimize the cost of the whole system. Only works on activating queues can be processed and finished. In more generic problems, the agent is allowed to activate / deactivate multiple queues at the same time based on some conditions. However, in this internship, the agent can only activate one queue at a time.

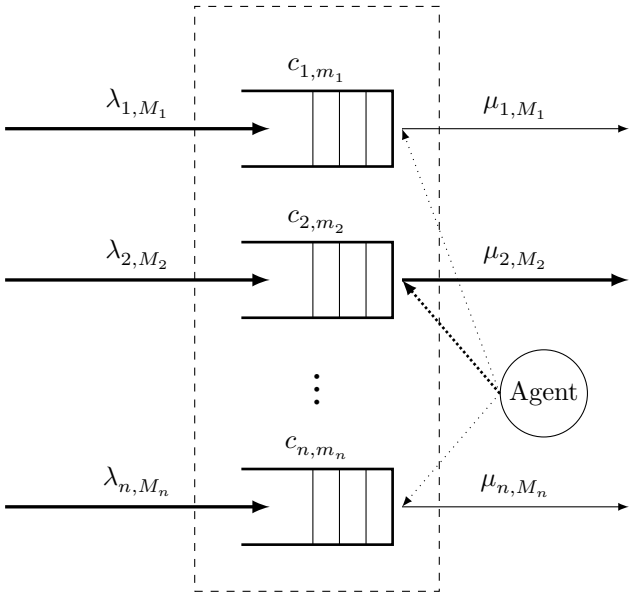


Figure 1: Visualization of the queueing system when the agent activates queue 2, the bold lines represent the flow of works inside the system.

2.1.4 Evolution

In the continuos-time scale, each type of event (work arrival, work departure and environment changing) happens independently. That system is quite hard to program, therefore, we used a technique called "uniformization" to move from the continuos-time scale to discrete-time scale. In this time scale, at a given time, only one event can take place, regardless of their type.

In particular, given the system state is (S, E), the agent decides to activate queue a. One of these $n + 1 + n(m - 1) + 1$ events can happen.

- The work of class Q_i increases by 1. Because there are n classes, we have n events of this kind with the rate $\lambda_{1,M_1}, ..., \lambda_{n,M_n}$ respectively (if the number of work of class Q_i reaches an upper limit, we

consider $\lambda_{i,M_i} = 0$).

- The work of class Q_a decreases by 1. Because we can only activate only one class, there is only one event of this type and its rate is μ_{a,M_a}.
- The environment of class Q_i changes to a different environment other than m_i and the rate changing to environment j is $\xi_{i,M_i,j}$. Because there are $m-1$ possible changes for each class and there are n classes, the number of this kind of event is $n(m-1)$.
- And a special dummy event where nothing changes.

A discrete probability distribution is used to express that. In order to satisfy the condition of a probability distribution, all the rates above are divided by a normalization constant C to make sure that their sum are not greater than 1. If that sum is smaller than 1, the special dummy event is used to fill the gap so that the final sum will be equal to 1.

The normalization constant has the form as follow, which is deduced from the above evolution of the system.

$$C = \sum_{i=1}^{n} \max_{j} \lambda_{i,j} + \max_{i,j} \mu_{i,j} + \sum_{i=1}^{n} \max_{j} \sum_{k=1,k\neq j}^{m} \xi_{i,j,k} + \epsilon$$

If $\epsilon > 0$, the probability of the dummy event will always be greater than 0. In this internship, as we do not want the system evolves too slowly, we choose $\epsilon := 0$.

After obtaining all the information above, we use that discrete probability distribution to obtain the next transition T of the system and denote S' and E' the next state of the system.

2.2 Load-balancing system

2.2.1 Parameters

For this system, we have a similar parameters with the one of the queuing system. The only difference in this system is that instead of having multiple arrival rate that depends on the class of queue as well as the environment, we have only one global arrival rate λ. A table summarizing the parameters of this system is shown below.

Table 3: Load-balancing system parameters

		m_1	...	m_m
Q_1	Holding cost	$c_{1,1}$	...	$c_{1,m}$
	Departure rate	$\mu_{1,1}$	...	$\mu_{1,m}$
⋮				
Q_n	Holding cost	$c_{n,1}$	...	$c_{n,m}$
	Departure rate	$\mu_{n,1}$	...	$\mu_{n,m}$
	Global arrival rate		λ	

2.2.2 Cost

Same as above, the cost of the system is a function of S and E. However, in this system, only the simple linear version is used.

$$f_1(S, E) = \sum_{1}^{n} c_{i,M_i} X_i$$

2.2.3 Agent

The goal of the agent is the same: minimizing the cost of the whole system. The difference here is instead of having to deactivating some queues, all queues now run continuously. His mission is to choose which queue to send the new arriving work to. Only the chosen queue will have new work arriving, the other queues only have to process the remaining works of them.

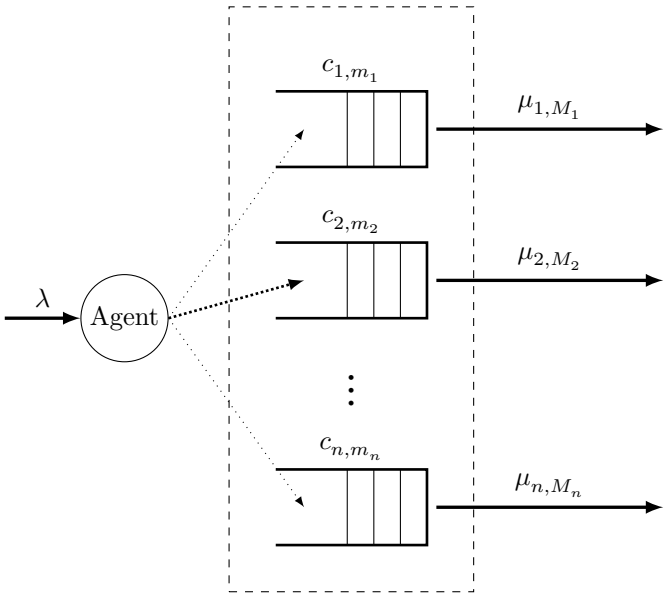


Figure 2: Visualization of the load-balancing system when the agent sends work to queue 2, the bold lines represent the flow of works inside the system.

2.2.4 Evolution

In this system, given its state is (S, E), the agent decides to send work to queue a. One of these $1 + n + n(m-1) + 1$ events can happen.

- The work of class Q_a increases by 1. Because we can only send work to one queue, there is only one event of this type and its rate is λ.
- The work of class Q_i decreases by 1. Because there are n classes, we have n events of this kind with the rate $\mu_{1,M_1}, \dots, \mu_{n,M_n}$ respectively (if the number of work of class Q_i reaches 0, we consider $\mu_{i,M_i} = 0$).
- The environment of class Q_i changes to a different environment other than m_i and the rate changing to environment j is $\xi_{i,M_i,j}$. Because there are $m-1$ possible changes for each class and there are n classes, the number of this kind of event is $n(m-1)$.
- And a special dummy event where nothing changes.

The form of the normalization constant also changes according to the evolution of the system.

$$C = \lambda + \sum_{i=1}^{n} \max_j \mu_{i,j} + \sum_{i=1}^{n} \max_j \sum_{k=1, k\neq j}^{m} \xi_{i,j,k} + \epsilon$$

2.3 System representation

In this internship, we limited ourselves on the cases where $n = 2$ for both systems. A system of that kind can be represented as a grid whose each side represents the evolution of each queue. In this report, we use

the vertical side for the first queue, and the horizontal one for the second. Each cell of that grid represent a specific observable state of the system.

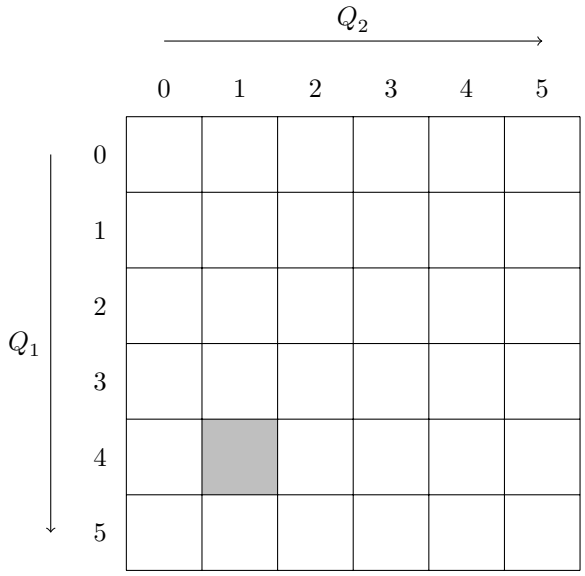


Figure 3: Representation of a system with 2 queues and the length limit of both queues are 5, the gray cell represents an observable state of the system $S = (4, 1)$

3 Reinforcement learning

Reinforcement learning is used for training the agent to attain his goal. In this session, we present a short summary and introduce some related notions that could be useful for later.

3.1 Parameters

A simple reinforcement learning is modeled by a markov decision process (MDP) whose parameters include as follow:

- $\mathcal{S}$: the set of all states of the system.
- $\mathcal{A}$: the set of all actions, and A_S, the set of all actions available from state S.
- $P(S_{t+1} = S'|S_t = S, a_t = a)$: the probability that action a in state S at time t will lead to state S' at time $t + 1$.
- $R(S'|S, a)$: the immediate reward received after transitioning from state S to state S', due to action a.

In addition, we have policy $\pi(a|S) = \mathbb{P}[A_t = a|S_t = S]$, a distribution over actions given states. In our problem, there is also an unobservable state inside our system, therefore, our problem is a generalization of the MDP, called partial observable markov decision process (POMDP). The parameters could be seen as follow:

- $\mathcal{S}$: the set of observable states is represented as the grid.
- $\mathcal{E}$: the set of unobservable environment states.
- $\mathcal{A}$: the action of activating / sending work to a queue.
- $P(S', E'|S, E, a)$: the transition probability that is clarified in the evolution section and depends on the both visible and hidden state (S, E).

- $R(S'|S, E, a)$: the immediate reward is $-C$ where C is the total holding cost of the system. Since this function depends only on the the state (S, E) of system, we could write $R(S, E)$ instead.

3.2 Bellman equation

First, we want to try the solving method of a MDP to our problem, so in this section, we assume that E is a constant state and will not consider E.

The accumulate discounted future reward (return) from time t with the discount factor $0 \leq \gamma \leq 1$ is:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots = r_{t+1} + \gamma G_{t+1}$$

From that, we have state-value function $V_\pi(S)$ is the expected return starting from state S, and then following policy π:

$$V_\pi(S) = \mathbb{E}_\pi[G_t|S_t = S] \tag{1}$$

And action-value function $Q_\pi(S, a)$ is the expected return starting from state S, taking action a, and then following policy π:

$$Q_\pi(S, a) = \mathbb{E}_\pi[G_t|S_t = S, a_t = a] \tag{2}$$

From the fact that $\pi(a|S)$ is a distribution over $\mathcal{A}$ given S, we have:

$$\begin{aligned} V_\pi(S) &= \mathbb{E}_\pi[G_t|S_t = S] \\ &= \sum_{a \in \mathcal{A}} \pi(a|S) \mathbb{E}_\pi[G_t|S_t = S, a_t = a] \\ &= \sum_{a \in \mathcal{A}} \pi(a|S) Q_\pi(S, a) \end{aligned} \tag{3}$$

Furthermore, we see that the expected future return of $Q_\pi(S, a)$ is the sum of the current reward $R(S)$ as well as the expected return of the next state regardless action multiply by the probability of moving to that state. Therefore, we have the equation below:

$$Q_\pi(S, a) = R(S) + \gamma \sum_{S' \in \mathcal{S}} P(S'|S, a) V_\pi(S') \tag{4}$$

Finally, the bellman equations are obtained from (3) and (4):

$$
\begin{aligned}
V_\pi(S) &= \sum_{a\in\mathcal{A}} \pi(a|S) Q_\pi(S,a) \\
&= \sum_{a\in\mathcal{A}} \pi(a|S)(R(S) + \gamma \sum_{S'\in\mathcal{S}} P(S'|S,a) V_\pi(S')) \\
&= R(S) + \gamma \sum_{a\in\mathcal{A}} \pi(a|S) \sum_{S'\in\mathcal{S}} P(S'|S,a) V_\pi(S')
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
Q_\pi(S,a) &= R(S) + \gamma \sum_{S'\in\mathcal{S}} P(S'|S,a) V_\pi(S') \\
&= R(S) + \gamma \sum_{S'\in\mathcal{S}} P(S'|S,a) \sum_{a'\in\mathcal{A}} \pi(a'|S') Q_\pi(S',a')
\end{aligned} \tag{6}
$$

3.3 Optimal policy

The core idea of this problem is to find an optimal policy π_* that maximizes the expected reward or $V(S)$. Mathematically, we have

$$\pi > \pi' \text{ if } V_\pi(S) > V_{\pi'}(S) \ \forall S \in \mathcal{S}$$

And we want to find

$$\pi_* \text{ such that } \pi_* \geq \pi \ \forall \pi$$

We can do that by finding the optimal state-value function $V_*(S) = \max_\pi V_\pi(S)$ or action-value function $Q_*(S,a) = \max_\pi Q_\pi(S,a)$ and define the optimal policy as follow:

$$\pi_*(S) = \begin{cases} 1 & \text{if } a = \arg\max_{a\in\mathcal{A}} Q_*(S,a) \\ 0 & otherwise \end{cases}$$

Plug all together into equation (5) and (6), we have:

$$V_*(S) = R(S) + \max_{a\in\mathcal{A}} \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V_*(s') \tag{7}$$

$$Q_*(s,a) = R(S) + \gamma \sum_{S'\in\mathcal{S}} P(S'|S,a) \max_{a'\in\mathcal{A}} Q_*(S',a') \tag{8}$$

We then move to next sections where we discuss about several training methods based on these equations.

4 Online Q-learning

4.1 About the algorithm

Online Q-learning is probably one of the most simple and popular algorithm for reinforcement learning problem. It is used to estimate $Q_*(S,a)\ \forall S \in \mathcal{S}$ and $a \in \mathcal{A}$ while interacting with the system. This algorithm only requires the real system or a simulator to interact with, and not the internal settings of that system, which makes it fit the context of this internship and more broadly real-life, where these settings can change at any time. Beside its simplicity and close to real life context, the algorithm is also proven to converge almost surely when the number of visit to each state goes to ∞ in [WD92]. Visually, we can imagine that our agent will follow a grid trajectory of states and try to discover what is the action-value at that state with a specific action.

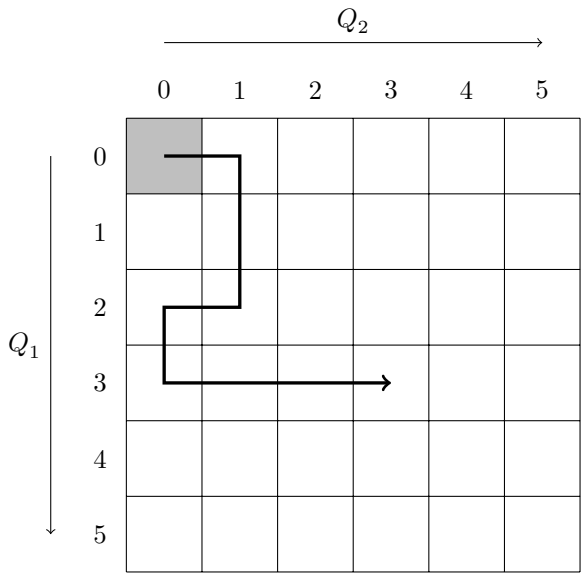


Figure 4: An example of the path which the agent might take. It starts from $(0, 0)$ and moves to $(3, 3)$. The next state could be either $(3, 4), (4, 3), (3, 2), (2, 3)$ depends on the action it chooses and the transition probabilities.

Formally, the algorithm can be written as follow:

Algorithm 1: Online Q-Learning

Input: $T > 0$ number iterations, γ discount factor, ϵ exploration factor, η learning rate
Output: Q_*, π_*

```
/* Internally, 𝔖 keeps track of the unobservable state E as well but the algorithm
   can not extract it here.                                                    */
```

$\mathfrak{S}$ simulator of the system;
for $t \leftarrow 0$ **to** T **do**
  $S \leftarrow \mathfrak{S}$ current observable state of the system;
  $a \leftarrow \begin{cases} \text{one random possible action}, & \text{with the probability of } \epsilon \text{ (exploration)} \\ \arg\max_{a \in \mathcal{A}} Q_t(S, a), & \text{with the probability of } 1 - \epsilon \text{ (learning)} \end{cases}$

```
  /* The state E might change in this step but from the algorithm's perspective,
     nothing has changed.                                                      */
```

  $R, S' \leftarrow \mathfrak{S}(a)$;
  $Q_{t+1}(S, a) \leftarrow Q_t(S, a) + \eta(R + \gamma \max_{a' \in \mathcal{A}} Q_t(S', a'))$;
end

4.2 Implementation

4.2.1 From Python to C++

In this section, we document the some difficulties while implementing the framework and how we overcome it. For illustration purpose, we will test the algorithm with a very simple problem of the queuing system and linear cost function with no environment (or constant environment). The parameter of this system is show in table 4.

It has been proven that if we have $c_1\mu_1 \geq c_2\mu_2$, the optimal policy for all cases is activating the first queue if possible, and vice versa in the case where $c_1\mu_1 \leq c_2\mu_2$. From table 4, we know that our optimal action here is activating the second queue.

Table 4: Simple queuing problem for testing

Q_1	Holding cost c_1	2
	Arrival rate λ_1	0.135
	Departure rate μ_1	0.3
Q_2	Holding cost c_2	3
	Arrival rate λ_2	0.135
	Departure rate μ_2	0.3

With a naive Python implementation, the algorithm works correctly when the limit of both queue are small.

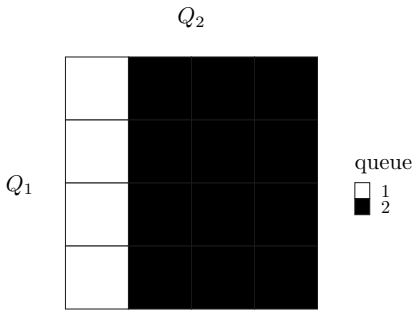
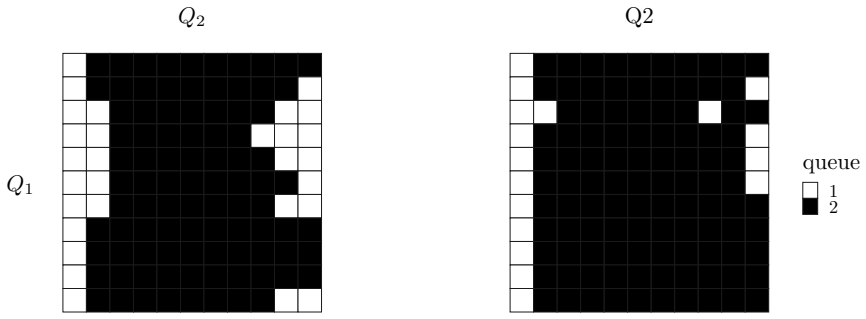


Figure 5: Result of online Q-learning using Python for 3x3 case. $\gamma = 0.999$, $\epsilon = 0.99$

First, we see that on the first column where $Q_2 = 0$, our agent always chooses to activate the first queue, not because this is the optimal action but this is the only option possible, because there is no work on the second queue. Same thing holds for the first row where $Q_1 = 0$. For the other cells, we have the black color which means the agent chooses the second queue which is aligned with the analytical solution as above.

This implementation runs 10^6 iterations for approximately 2 minutes. This is quite good for number with small case. However, if we increase the limit to 10, with the same number of iterations gives a imperfect result.

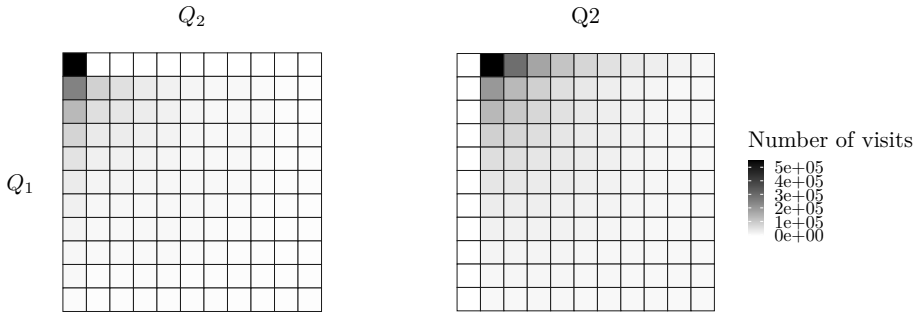


(a) For 10^6 iterations

(b) For 10^7 iterations

Figure 6: Result of online Q-learning using Python for 10x10 case. $\gamma = 0.99$, $\epsilon = 0.999$

In the figure above, we can see that even after 20 minutes with 10^7 iterations, the algorithm still does not converge completely yet. To have a clear view about the reason why, we have figure 7.



(a) Activating Q_1 (b) Activating Q_2

Figure 7: Number of visit for the system above. $\gamma = 0.99$, $\epsilon = 0.999$

We can see that, our agent is stuck at $(0, 0)$, only the area around the origin has a high number of visit. The rest have relatively low visit and therefore, does not converge because of the nonfulfillment of the condition in [WD92].

To speed up the algorithm, we decided to switch to C++, which is famous for its speed and its mature support for high-speed computation with library like Eigen and then expose a Python interface for more ease of use. This is also the same approach for many scientific libraries in Python such as Numpy or Tensorflow. After switching to C++ and some improvements later, the algorithm now can run 10^9 iterations in only 1 minutes, a 2×10^3 speed up.

4.2.2 From n-dimensions to 2-dimensions

With C++, the simulator is already running faster. However, there are still room for improvements. Now, the probability $P(S', E'|S, E, a)$ is calculated on-the-fly when we meet that state (S, E) while running the algorithm, we could make it faster by calculate the matrix transition for all the states beforehand. However, the state S and E of the system are each represented by a vector of n entries, if we keep using vector like that, we will need a tensor of $n + n + 1 = 2N + 1$ dimensions (first n dimensions for saving the observable state S, the next n dimensions for saving the unobservable state S and the last one for action). In addition, since there is no sparse version for high-dimensional tensor, it will take a lot of memories to save that tensor, more precisely:

$$(L_1 + 1) \times \cdots \times (L_n + 1) \times \underbrace{n^m}_{n \text{ queues and } m \text{ environments}} \times \underbrace{n}_{n \text{ actions}}$$

If we try to allocate a tensor of that size, we will get an "out of memory" error.

Furthermore, in other algorithms, we are required to iterate all over the set of states. If n, the number of queues is static, we could something similar to algorithm 2. However our goal is to let n as a input for the program. So we can not use that algorithm there.

In the end, we decide to encode the whole state into an integer by extending the notion of base of number. Given state (S, E), we introduce 3 functions:

$$\begin{aligned} f_{\mathcal{S}}(S) &= f(X_1, \dots, X_n) = X_1 L_2 \dots L_n + X_2 L_3 \dots L_n + \cdots + X_{n-1} L_n + X_n \\ f_{\mathcal{E}}(E) &= f(M_1, \dots, M_n) = M_1 m^{n-1} + M_2 m^{n-2} + \cdots + M_{n-1} m + M_n \\ f_{\mathcal{S},\mathcal{E}(S,E)} &= f(X_1, \dots, X_n, M_1, \dots, M_n) = m^n f_{\mathcal{S}}(S) + f_{\mathcal{E}}(E) \end{aligned}$$

Algorithm 2: Iteration over the set of visible states $\mathcal{S}$ when $n = 3$

```
for i ← 0 to L_1 do
    for j ← 0 to L_2 do
        for k ← 0 to L_3 do
            S ← (i, j, k);
            ...;
        end
    end
end
```

All three functions are one-to-one mapping, so we can replace the state with the result of these function without losing anything. Because of doing so, we have achieved several benefits:

- We can calculate the transition matrix right now and use a sparse matrix to store it for efficiency in both storage and performance (since the transition matrix will be full of 0).
- Turn a problem with unknown dimension to a 2-dimensional one which makes accessing and looping through the states easier.
- Overall speed improvement since we only need to deal with integer instead of a vector.

4.3 Result

5 Offline Q-learning

5.1 About the algorithm

In this version of Q-learning, we do not explore or follow the trajectory as in the online version, but we will force calculate the q value for all pair (S, a). This method is faster than online Q-learning, however, it requires a reliable simulator of the system to be able to work. The algorithm could be described below:

Algorithm 3: Offline Q-Learning

Input: $T > 0$ number iterations, γ discount factor, η learning rate
Output: Q_*, π_*
$\mathfrak{S}$ simulator of the system;
E initial environment state of the system;

```
for t ← 0 to T do
    −, −, E' ← 𝔖(−, −, E);
    if E ≠ E' then
        E ← E' ;
    end
    for s ∈ 𝒮 do
        for a ∈ 𝒜 do
            S', R, − ← 𝔖(a, S, E);
            Q_{t+1}(S, a) ← Q_t(S, a) + η(R + γ max_{a'} Q_t(S', a'));
        end
    end
end
```

5.2 Result

References

[WD92] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.