Internship report

Vo Van Nghia

Date

29 Sep, 2022

Table of contents

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. 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.

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

This is the internship description: *"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 model free approach."

2 System settings

In this internship, we are interested in learning an efficient policy for some dynamic systems where the internal settings (transitions rate) of the system depend on some environments (or which environment the system is in). We could take one decision at a specific time (doing some things or even doing nothing) to make the system run more efficiently (faster or cheaper depends on each type of problem). However, we can only observe the state of the system (for example: the number of remaining tasks) and not the environments when taking decisions. And therefore, we will explorer Q-learning and derived techniques since they does not need to know the internal settings of the systems. In this report, we focus ourselves on two following system models.

2.1 Queuing system

2.1.1 Parameters

We have n classes of queue $Q_1, ..., Q_n$, and $L_1, ..., L_n$ the maximum number of jobs (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 $m_1, ..., m_m$. And the environment of class Q_i is a random variable, denoted by M_i, which is in one of the states $m_1, ..., m_m$. For each environment m_j, Q_i has their own holding cost $c_{i,j}$ (the cost of one unfinished job on the queue), arrival rate $\lambda_{i,j}$ (the rate of one more job arriving to the queue) and departure rate $\mu_{i,j}$ (the rate that a job departs the system (if served)). Furthermore, the environment M_i can change from m_j and m_k with the rate of $\xi_{i,j,k}$. 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	$\mu_{1,1}$	...	$\mu_{1,m}$
⋮				
Q_n	Holding cost	$c_{n,1}$	...	$c_{n,m}$
	Arrival rate	$\lambda_{n,1}$	...	$\lambda_{n,m}$
	Departure rate	$\mu_{n,1}$	...	$\mu_{n,m}$

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:

- The state $S = (X_1, ..., X_n)$ where X_i is a random variable represents the current number of class Q_i jobs and is observable.
- The environment vector $E = (M_1, ..., M_n)$ 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 and the only information provided to him is the observable state S. Only jobs 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.

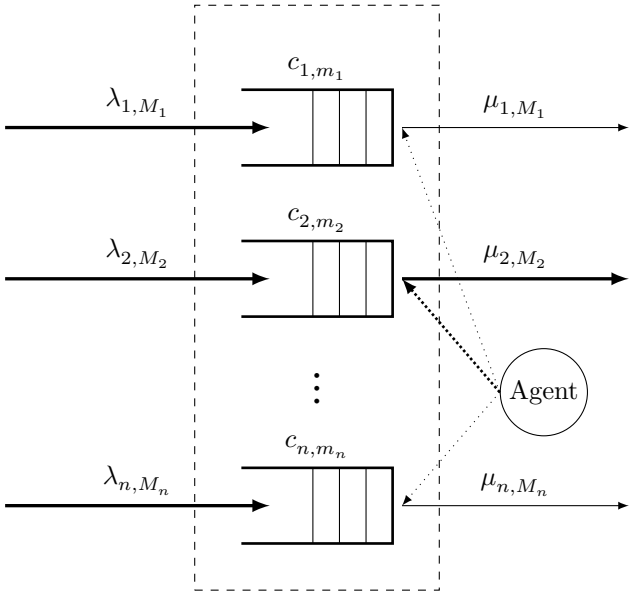


Figure 1: Visualization of the queueing system when the agent activates queue 2, the bold lines represent the flow of jobs inside the system.

2.1.4 Evolution

In the continuos-time scale, each type of event (job arrival, job 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 job 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 job of class Q_i reaches an upper limit, we consider $\lambda_{i,M_i} = 0$).
- The job 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 to evolve 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 the load-balancing system, we only have one stream of job arrivals with rate λ and n queues. Same as above, The queues capacities (holding cost and departure rate) also depend on which environment they are in. Therefore, we have similar parameters to 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. He also receive the same information: the observable state S. 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 job to. Only the chosen queue will have new job arriving, the other queues only have to process the remaining jobs of them.

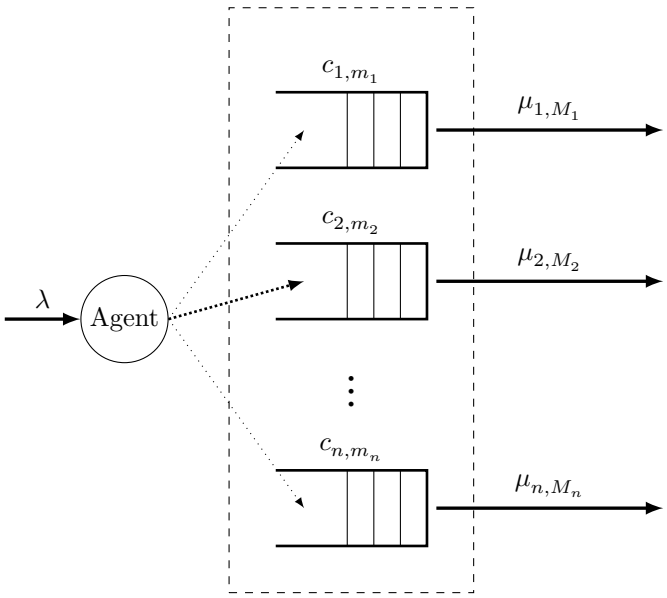


Figure 2: Visualization of the load-balancing system when the agent sends job to queue 2, the bold lines represent the flow of jobs inside the system.

2.2.4 Evolution

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

- The job of class Q_a increases by 1. Because we can only send job to one queue, there is only one event of this type and its rate is λ.
- The job 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 job 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 where 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.

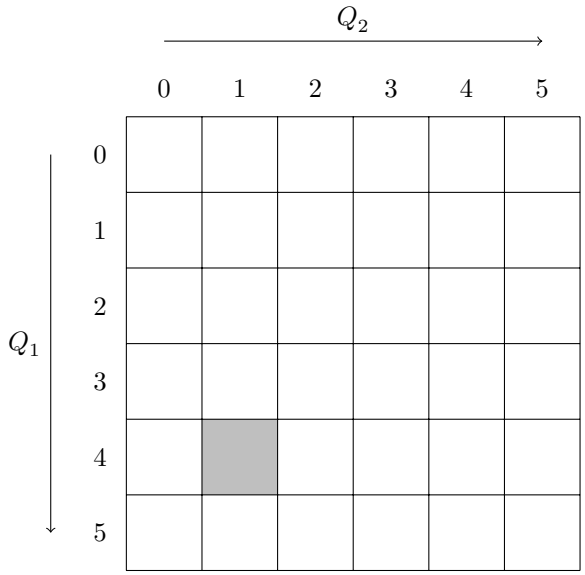


Figure 3: Representation of a system with 2 queues and the length limit of both queues are 5, the gray cell represents an observable state of the system $S = (4, 1)$.

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 no longer an MDP, but instead a partial observable markov decision process (POMDP). We will now continue discussing about the MDP and return to the POMDP in later section.

3.2 Bellman equation

First, we want to try the solving method of a MDP.

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 for MDP

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:

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

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

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

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

3.4 Efficient policy when having unobservable environment

Our problem is a 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.

Since the goal of the internship is not about solving the POMDP, in the later sections, we will explore how learning algorithms of MDP work in the context of unobservable environment.

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 for a MDP, 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 visits 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.

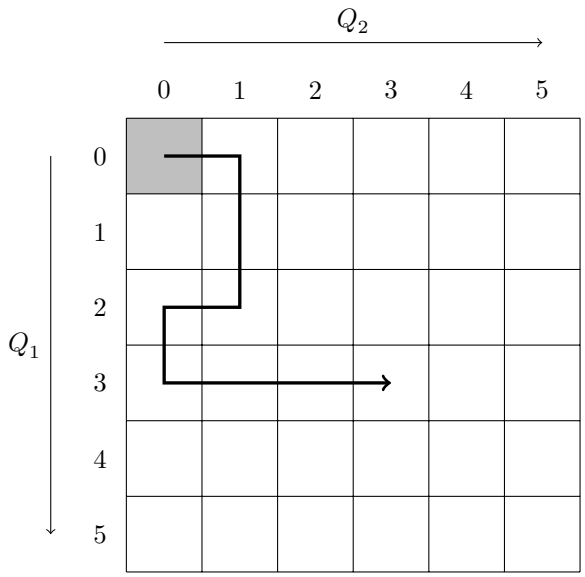


Figure 4: An example of the path which the agent might take. It starts from $(0, 0)$ and moves to $(3, 3)$. The next state could be either $(3, 4), (4, 3), (3, 2), (2, 3)$ depends on the action it chooses and the transition probabilities.

In this report, we want to investigate the Q-learning algorithm when applied to our POMDP. We used the following algorithm:

Algorithm 1: Online Q-Learning

Input: $T > 0$ number iterations, γ discount factor, ϵ exploration factor, η learning rate
Output: Q_*, π_*
for $t \leftarrow 0$ **to** T **do**
  $S, E \leftarrow \mathfrak{S}$ current state of the system;
  $a \leftarrow \begin{cases} \text{one random possible action,} & \text{with the probability of } \epsilon \text{ (exploration)} \\ \arg\max_{a \in \mathcal{A}} Q_t(S, a), & \text{with the probability of } 1 - \epsilon \text{ (learning)} \end{cases}$
  `/*` $S' = S$ `if` $E' \neq E$ `and vice versa` `*/`
  $R, S', E' \leftarrow \mathfrak{S}(a, S, E)$;
  $Q_{t+1}(S, a) \leftarrow Q_t(S, a) + \eta(R + \gamma \max_{a' \in \mathcal{A}} Q_t(S', a') - Q_t(S, a))$;
end

4.2 Implementation

4.2.1 From Python to C++

In this section, we document 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.

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

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

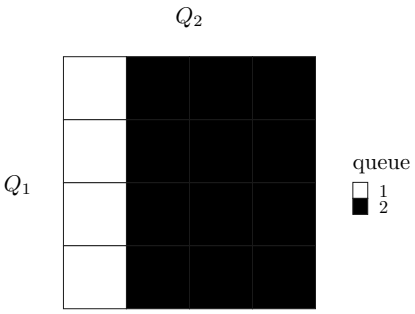
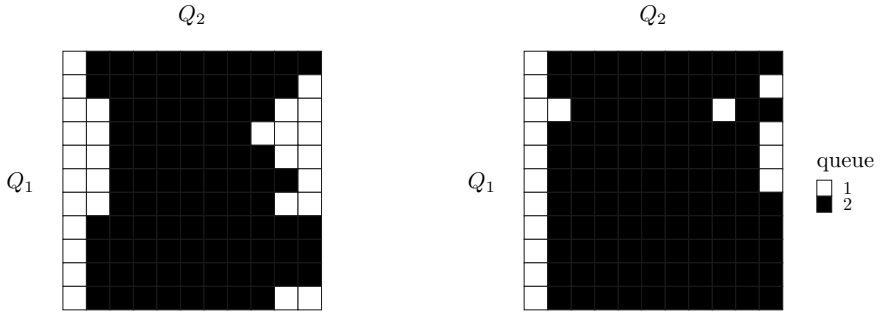


Figure 5: Result of online Q-learning using Python for 3x3 case. $\gamma = 0.999$, $\epsilon = 0.99$

First, we see that on the first column where $Q_2 = 0$, our agent always chooses to activate the first queue, because this is the only option possible, because there is no job 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.

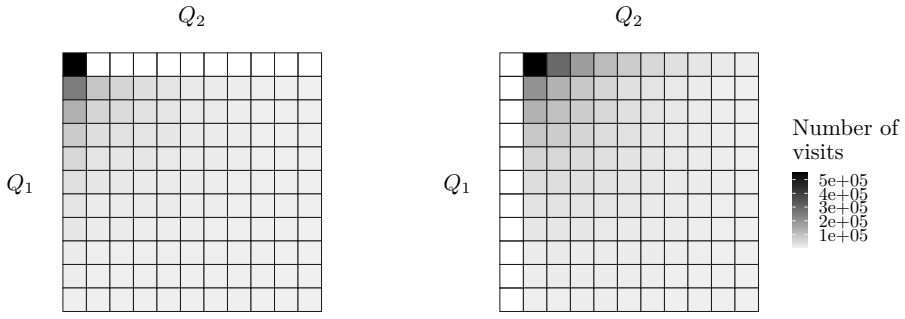


(a) For 10^6 iterations

(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 visits. The rest have relatively low number of visits 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 minute, 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 memory 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 do 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.

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 the calculation of the Q-value for all pairs (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 is 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') − Q_t(S, a));
        end
    end
end
```

6 Value iteration

6.1 About the algorithm

This algorithm is directly based on equation (7) and used to estimate the value of $V_*(S)$. After obtaining an estimate of $V_*(S)$, the policy is deduced as $\pi_*(S) = \arg\max_a R(S) + \gamma \sum_{S' \in \mathcal{S}} P(S'|S,a)V_*(S')$. The algorithm is written as follow:

Algorithm 4: Value iteration

Input: γ discount factor
Output: V_*, π_*
$\Delta \leftarrow \infty$;
while $>$ **do**
  for $S \in \mathcal{S}$ **do**
    $V_{t+1}(S) \leftarrow \max_{a \in \mathcal{A}} R(S) + \gamma \sum_{S' \in \mathcal{S}} P(S'|S,a)V_*(S')$;
  end
  $\Delta \leftarrow \|V_{t+1} - V_t\|\_2$;
end

We can see above that the algorithm makes use of $P(S'|S,a)$, because of this, this algorithm converge a lot faster than both Q-learning methods but we have to know the transition probability, so this method is use only as a baseline to compare with the result of Q-learning.

If we have only one constant environment E_c (or no environment changing), we could rewrite $P(S', E_c|S, E_c, a) = P(S'|S,a)$ and value iterations work correctly. However, as we have unobservable environment, algorithm 4 can not work without modification because we have $P(S', E'|S, E, a)$ but we want to estimate $V_*(S)$ and not $V_*(S, E)$. In the next section, we discuss about how to get an empirical estimate of $P(S'|S,a)$ based on $P(S', E'|S, E, a)$.

6.2 Empirical transition probability

Our idea is to calculate the probability of the system stay in a particular state $E \in \mathcal{E}$ and then multiply it with the transition probability to get the average probability. First, we have some equations:

$$P(S'|S,E,a) = \begin{cases} P(S', E|S, E, a) & \text{if } S' \neq S \\ 1 - \sum_{S' \neq S} P(S', E|S, E, a) = \sum_{E'} P(S, E'|S, E, a) & \text{otherwise} \end{cases} \tag{9}$$

$$P(E'|S,E,a) = \begin{cases} P(S, E'|S, E, a) & \text{if } E' \neq E \\ 1 - \sum_{E' \neq E} P(S, E'|S, E, a) & \text{otherwise} \end{cases} \tag{10}$$

The logic behind equation (9) and (10) is the fact that only one event can happen at a time.

$$P(S, E'|S, E, a) = \begin{cases} \xi_{i, M_i, M'_i} & \text{if } E' \text{ and } E \text{ differ by only one entry at index } i \\ 0 & \text{else} \end{cases} \tag{11}$$

By rewriting $P(S, E'|S, E, a)$ as in equation (11), we see that $P(E'|S, E, a)$ does not depend on S, a and therefore $P(E'|S, E, a) = P(E'|E)$. From that, we could calculate the steady state by an iterative method using the equation below:

$$\pi_{t+1}(E') = \sum_{E \in \mathcal{E}} P(E'|E)\pi_t(E) \tag{12}$$

After $\pi(E)$ converges to $\tilde{\pi}(E)$ and using (9), we have the final empirical value we want:

$$\tilde{P}(S'|S,a) = \sum_{E \in \mathcal{E}} \tilde{\pi}(E)P(S'|S,E,a) \tag{13}$$

7 Result

7.1 No environment changing (MDP)

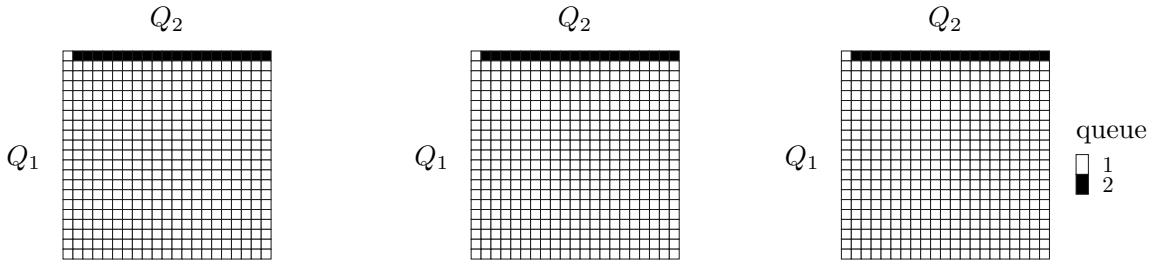
7.1.1 Queueing

First we will try with the linear cost. This is the simplest case and since it is proven mathematically (?), it is only a test to show that all algorithms works correctly. We will try two tuples of parameters, the first one will be:

Table 5: Simple queuing 1

Q_1	Holding cost c_1	3
	Arrival rate λ_1	0.13
	Departure rate μ_1	0.3
Q_2	Holding cost c_2	1
	Arrival rate λ_2	0.13
	Departure rate μ_2	0.3

The theoretical optimal action here is 1.



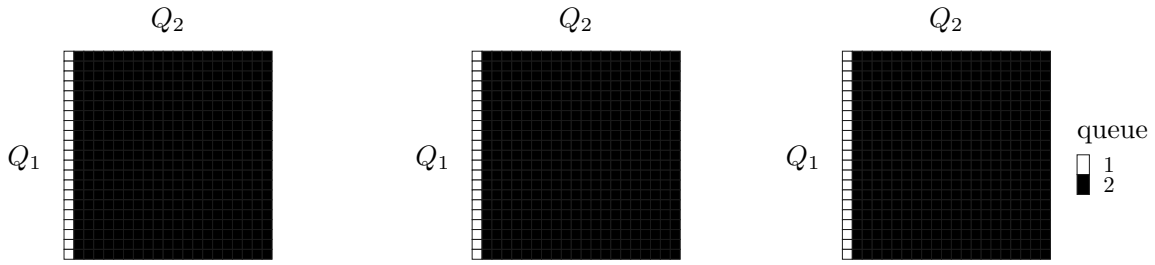
(a) Online Q-learning (b) Offline Q-learning (c) Value iterations

Figure 8: Result of table 5 for 20x20 case. $\gamma = 0.999$, $\epsilon = 0.999$.

And the second one is nearly the same, except the holding cost is reverse, and therefore the optimal action here is 2:

Table 6: Simple queuing 2

Q_1	Holding cost c_1	1
	Arrival rate λ_1	0.13
	Departure rate μ_1	0.3
Q_2	Holding cost c_2	3
	Arrival rate λ_2	0.13
	Departure rate μ_2	0.3



(a) Online Q-learning (b) Offline Q-learning (c) Value iterations

Figure 9: Result of table 6 for 20x20 case. $\gamma = 0.999$, $\epsilon = 0.999$.

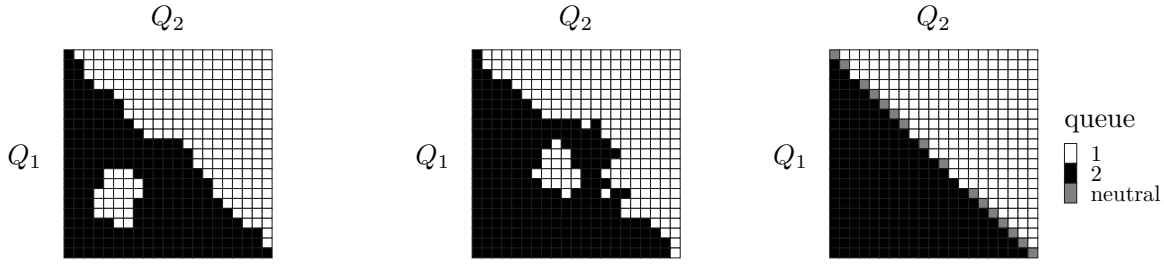
As we can see, all three algorithms work well for this simple case. We will move to more complex case.

7.1.2 Load-balancing

For this system, We first try a system where two queues have the same parameters:

Table 7: Simple load-balancing 1

Q_1	Holding cost c_1	3
	Departure rate μ_1	0.3
Q_2	Holding cost c_2	3
	Departure rate μ_2	0.3
	Global arrival rate	0.5



(a) Online Q-learning (b) Offline Q-learning (c) Value iterations

Figure 10: Result of table 7 for 20x20 case. $\gamma = 0.999$, $\epsilon = 0.999$.

Intuitively, if two queues has the same parameters, the optimal action will be sending job to the queue that has less jobs, which is the result of value iterations method (the right grid) of figure 10. There is also a gray color indicates that at those states, either action 1 (sending new job to Q_1) or 2 (sending to new job to Q_2) leads to the same result, because they have the same number of jobs.

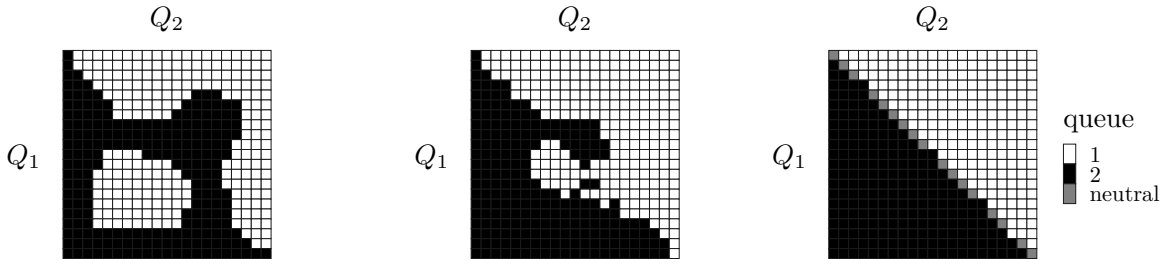
For two Q-learning methods, the online and offline version has been trained for 3×10^{10} and 5×10^7 iterations respectively and takes 30 minutes each. They give us a similar result and both are close to the optimal policy of value iterations.

However, since both methods depends on the evolution of the system, they are very sensitive to the parameters of systems. For example, if we increase the arrival rate to 0.55:

Although the optimal policy should be the same as above, the same number of iterations give us a different result.

Table 8: Simple load-balancing 2

Q_1	Holding cost c_1	3
	Departure rate μ_1	0.3
Q_2	Holding cost c_2	3
	Departure rate μ_2	0.3
	Global arrival rate	0.55

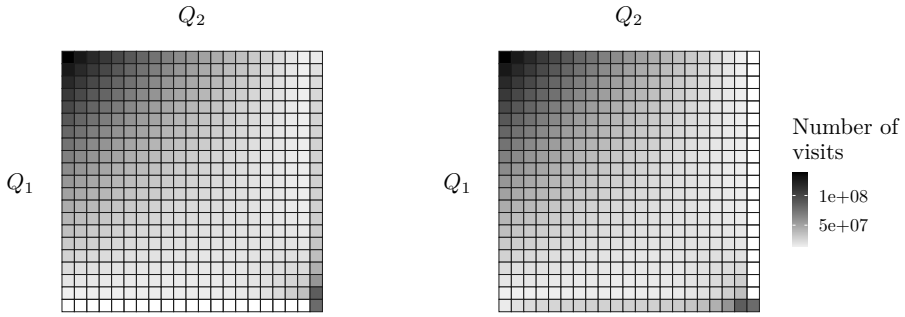


(a) Online Q-learning (b) Offline Q-learning (c) Value iterations

Figure 11: Result of table 8 for 20x20 case. $\gamma = 0.999$, $\epsilon = 0.999$.

The offline version is similar to the policy of value iterations method and has a smaller hole compared to the online version. The difference can be explained by the fact that in the offline version, each pair state-action is visited 5×10^7 times, so the total number of iterations will be $2 \times 41^2 \times 5 \times 10^7 \approx 1.6 \times 10^{11}$, which is $\gg 3 \times 10^{10}$, the total number of iterations of the online version.

Furthermore, even if we run the online version with that much iterations, it will still not guarantee that we will have a good approximation of the offline result. This is because we are following the trajectory (or the evolution of the system) and therefore it is impossible to visit all states equally. We can see the number of visits of each pair state-action below.



(a) Sending to Q_1 (b) Sending to Q_2

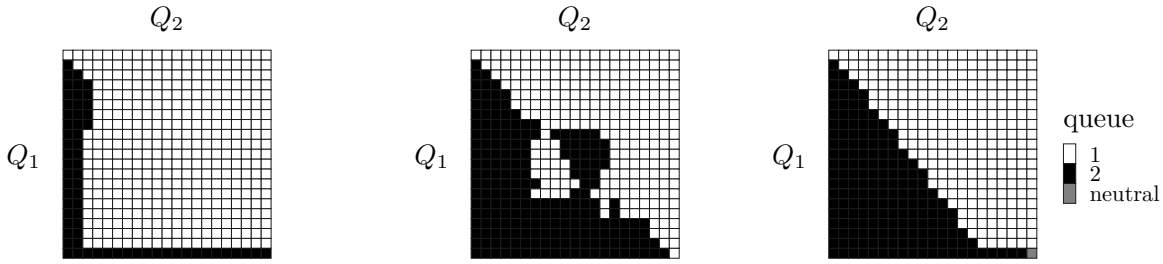
Figure 12: Number of visits of online Q-learning on table 8 for 20x20 case.

The number of visits is high around $(0, 0)$ and $(40, 40)$. Central region of the grid receives much less visits. Therefore, we see in figure 10 that the diagonal is beginning to form from these two points but not the region near the grid center.

We will the finish this section with an unbalance example. We fix the cost and make the departure rate of the first queue higher than the second one as follow:

Table 9: Simple load-balancing 3

Q_1	Holding cost c_1	3
	Departure rate μ_1	0.4
Q_2	Holding cost c_2	3
	Departure rate μ_2	0.3
	Global arrival rate	0.6



(a) Online Q-learning (b) Offline Q-learning (c) Value iterations

Figure 13: Result of table 9 for 20x20 case. $\gamma = 0.999$, $\epsilon = 0.999$.

Once again, we see the sensitivity of Q-learning with the system settings. Intuitively, since class 1 works faster than class 2 with the same cost, we will send more jobs to the first one as in the value iterations method. However, the learning capacity of both Q-learning methods is not sufficient to capture all the information of the system.

We then now move to the cases of multiple environments.

7.2 Multiple environments (POMDP)

We will introduce some new notations of environment transitions. We note that after uniformization, the probability of transitioning from one environment to itself (the diagonal of table 2) does not matter any more since it will be taken account into the dummy event. Futhermore, the sum of the other probabilities does not need to be less than 1 as we will divide them with the normalization constant.

The more important thing now is the ratio between each probabilities pair. We can rewrite table 2 as follow:

Table 10: Uniformized matrix transition of the environment of Q_i

	m_1	m_2	...	m_m
m_1	$*$	$\xi_{i,1,2}$	...	$\xi_{i,1,m}$
m_2	$\xi_{i,2,1}$	$*$	...	$\xi_{i,2,m}$
$\vdots$				
m_m	$\xi_{i,m,1}$	$\xi_{i,m,2}$	...	$*$

Furthermore, because we want to see if the speed of environment changing affects the optimal policy of each method, we multiply all transition matrices of each Q_i of the system with α. This does not change the ratio mentioned above and only takes effect when uniformization. In this process, it acts as a ratio between the change in the observable state and non observable (environment) state. If α is small, the rate of environment changing will only take a small part in the uniformized probability and vice versa.

From all the small transition matrices of each Q_i, we can get a big environment transition matrix $\mathcal{M}_{\mathcal{E}}$ where each row and column represents an environment state encoded using the method present in From

n-dimensions to 2-dimensions section. Since all the entries except the ones on the diagonal are multiplied by α, we can express $\mathcal{M}_{\mathcal{E}}$ as follow:

$$\mathcal{M}_{\mathcal{E}} = \begin{pmatrix} \rho_{1,1} & \alpha\rho_{1,2} & \dots & \alpha\rho_{1,|\mathcal{E}|} \\ \alpha\rho_{2,1} & \rho_{2,2} & \dots & \alpha\rho_{2,|\mathcal{E}|} \\ \vdots & & & \vdots \\ \alpha\rho_{|\mathcal{E}|,1} & \alpha\rho_{2,2} & \dots & \rho_{|\mathcal{E}|,|\mathcal{E}|} \end{pmatrix}$$

Where for example if $(1, 1, \dots, 1, 1) \mapsto i$ and $(1, 1, \dots, 1, 2) \mapsto j$, then $\rho_{i,j}$ will be equal to $\xi_{n,1,2}$ (the probability that the environment of queue n moves from m_1 to m_2).

If $\Pi_{\mathcal{E}}$ is the solution of equation (12) then $\mathcal{M}_{\mathcal{E}}\Pi_{\mathcal{E}} = \Pi_{\mathcal{E}}$ (because $P(E'|E)$ is actually ρ).

Lastly, with all the notion above, we can prove that the solution of equation (12) does not depend on α.

Lemma. *Given a set of all $\rho_{i,j}\,(i \neq j)$ defined as above.*

We call a pair of $(\alpha, (\rho_{1,1}, \dots, \rho_{|\mathcal{E}|,|\mathcal{E}|}))$ satisfied if $\mathcal{M}_\alpha$ is a right stochastic matrix where:

$$\mathcal{M}_\alpha = \begin{pmatrix} \rho_{1,1} & \alpha\rho_{1,2} & \dots & \alpha\rho_{1,|\mathcal{E}|} \\ \alpha\rho_{2,1} & \rho_{2,2} & \dots & \alpha\rho_{2,|\mathcal{E}|} \\ \vdots & & & \vdots \\ \alpha\rho_{|\mathcal{E}|,1} & \alpha\rho_{2,2} & \dots & \rho_{|\mathcal{E}|,|\mathcal{E}|} \end{pmatrix}$$

If $(\alpha, (\rho_{1,1}, \dots, \rho_{|\mathcal{E}|,|\mathcal{E}|}))$ and $(\alpha', (\rho'_{1,1}, \dots, \rho'_{|\mathcal{E}|,|\mathcal{E}|}))$ are satisfied, and there is T such that $\mathcal{M}_\alpha \mathrm{T} = \mathrm{T}$, then $\mathcal{M}_{\alpha'} \mathrm{T} = \mathrm{T}$

Proof. From $\mathcal{M}_\alpha \mathrm{T} = \mathrm{T}$, we have:

$$\begin{aligned}
&\iff \alpha \sum_{j,j\neq i} \rho_{i,j}\tau_j + \rho_{i,i}\tau_i = \tau_i \\
&\iff \alpha \sum_{j,j\neq i} \rho_{i,j}\tau_j + (1 - \alpha \sum_{j,j\neq i} \rho_{i,j})\tau_i = \tau_i \\
&\iff \alpha \sum_{j,j\neq i} \rho_{i,j}\tau_j - \alpha\tau_i \sum_{j,j\neq i} \rho_{i,j} = 0 \\
&\iff \sum_{j,j\neq i} \rho_{i,j}\tau_j = \tau_i \sum_{j,j\neq i} \rho_{i,j} \\
&\iff \alpha' \sum_{j,j\neq i} \rho_{i,j}\tau_j = \alpha'\tau_i \sum_{j,j\neq i} \rho_{i,j} \\
&\iff \alpha' \sum_{j,j\neq i} \rho_{i,j}\tau_j + (1 - \alpha' \sum_{j,j\neq i} \rho_{i,j})\tau_i = \tau_i \\
&\iff \alpha' \sum_{j,j\neq i} \rho_{i,j}\tau_j + \rho'_{i,i}\tau_i = \tau_i
\end{aligned}$$

Therefore, $\mathcal{M}_{\alpha'} \mathrm{T} = \mathrm{T}$. □

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.