Optimal Solution to the Dec-POMDP Approached with Reinforcement Learning

Dylan Kim

December 20th, 2024

1. Introduction

The Decentralized Partially-Observable Markov Decision Process(Dec-POMDP) framework is applied in several real-world tasks, especially where multiple decision makers collaborate to manage the total system event based on partial observability. However, since exhaustively searching for optimal strategy is an exponential time complexity, several scholars researched deriving optimal solutions through Reinforcement Learning(RL). Japanese researchers Toshimitsu Ushio and Tatsushi Yamasaki were among them. They claimed to find an optimal solution to the Cat and Mouse problem using Q Learning, one of the RL algorithms.

However, the proposed algorithm is not aligned with Dec-POMDP and control theory principles and they have not provided any implementation, which questions the validity of their work. Therefore, the motivation behind this research is to reproduce their algorithm and demonstrate it. This paper provides the basic background behind this idea, and then introduces the algorithm and mathematical formula from their paper. Several experiments and demonstrations follows and conclude with an analysis of the result.

2. Background

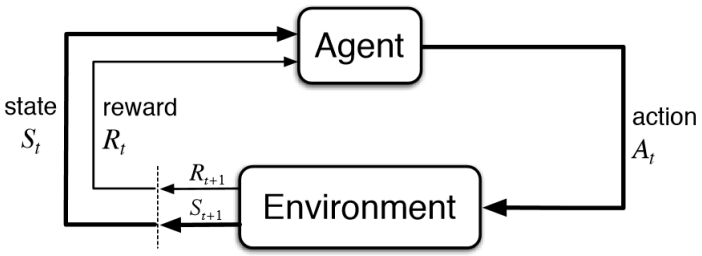
This section explains the concepts required to understand Ushio and Yamasaki's work as well as the argument of this paper.

2-1. Reinforcement Learning(RL)

Reinforcement Learning(RL) is one of the artificial intelligence algorithms that learns what to do and how to map from situations to actions, to maximize the numerical reward signal. An agent, the learner and decision maker, is not told which actions to take but must

discover which actions yield the most reward by trying them[1]. As the agent takes some **action**, the environment –the world in which the agent can operate– responds to the action and returns the **numerical reward** and the current condition of the environment(**state**), back to the agent. The positive reward encourages the agent to choose the same action and the negative reward encourages the agent to avoid that action. Then, the agent records how "good" the action was in that state based on the total reward it got. Repetition of this process allows the agent to construct how to behave in a certain state, called **policy**. The agent repeats this trial-and-error process, aiming to maximize the long-run cumulative rewards.

2-2. Markov Decision Process(MDP)



[Figure 1. Key Concept of MDP]

Markov Decision Process(MDP) is a mathematical model that formalizes sequential decision-making; thus, it can mathematically idealize RL problems. It proposes that any problem with learning goal-directed behaviours, regardless of all redundant details, can be abstracted to three signals passing back and forth between an agent and environment: environment-generated pairs of states, and reward, alternating with agent-controllable action.

At each discrete time step t, the agent receives some state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$ based on that state, then sends it to the environment. In the next step, as a

consequence, the agent receives some numerical reward $R_{t+1} \in \mathcal{R}$ with a new state S_{t+1}. As a result, the signal interaction in MDP can be expressed as a sequence below:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \cdots$$

Recall that the agent's goal is to maximize the long-run cumulative rewards. In MDP, the agent's goal is more formally defined as maximizing the expected Return G_t, a function that represents some particular aspect of the sequence of future rewards. This paper uses the simplest Return, the sum of all future rewards.

$$G_t := R_{t+1} + R_{t+2} + R_{t+3} \cdots R_T \tag{1}$$

where T is a terminal time step. Based on the existence of the terminal time step, the entire agent-environment interactions break into finite repetitive cycles, from the initial to the terminal time step. Here, each cycle is called an Episode. Once the agent reaches the state of the terminal time step, called a terminal state, a new Episode begins independently of the previous one. Tasks of identifiable episodes like this are called Episodic Tasks.

Unfortunately, many real-world tasks are not Episodic, but Continuing, meaning that it does not break into identifiable episodes but goes continuously without limit. For Continuing tasks, maximizing Eq.1 can lead the Return to become infinity since $T = \infty$. To avoid this, the agent aims to maximize *discounted return*:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

$$
\begin{aligned}
G_t :=& R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots \gamma^{T-1} R_T \\
=& \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}
\end{aligned}
\tag{3}
$$

where $\gamma \in [0, 1]$ refers to the discount factor, which determines whether short-term reward is valued more, or long-term reward.

Using those Returns, the agent estimates how "good" it is to be in a given state and/or to perform a given action. This is called a **Value** and the function that estimates the value is called the **Value function**. There are two types of value functions; whether its input is a state or a (state, action) pair, this paper utilizes the latter one called **(State) Action Value Function**. Since the agent will eventually choose its action that leads to the highest value provided by the Value function, the Value function is typically defined with respect to **Policy** π, the way the agent would behave in each state. Formally, a policy is a mapping from states to the probability of selecting each possible action; $\pi(a|s)$ means the probability of taking action a if the current state is s. Then, the value of taking action a at a state s under policy π, denoted as $q_\pi(s, a)$, is defined as:

$$
q_\pi(s, a) := \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = a, A_t = a \Big] \quad \text{(by Eq (2))} \tag{4}
$$

$$
\begin{aligned}
q_\pi(s, a) :=& \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
=& \mathbb{E}_\pi\Big[\sum_{k=0}^{T-1} \gamma^k R_{t+k+1} \Big| S_t = a, A_t = a \Big] \quad \text{(by Eq (2))}
\end{aligned}
\tag{5}
$$

where $\mathbb{E}$ is denoted as *expected* value. Since there is more than one way to estimate the value, one may be better than the others ("better" means if one gives a greater expected return than the other for every state). Then, there is always at least one value function that is better than or equal to all other policies, this is called **Optimal Value Function**,

denoted as $q_*(s, a)$. The policy defined with respect to the optimal value function is called **Optimal Policy**, denoted as π_*.

$$q_*(s, a) := \max_{\pi} q_\pi(s, a) \tag{6}$$

Applying Eq.3 for every single state to find the optimal value function & optimal policy is, in fact, problematic. This is because if the number of states gets larger, the number of value calculations for all possible states and all possible value functions, then finding the optimal one increases exponentially[2]. American applied mathematician Richard Bellman tackled this and found a way to optimize the number of calculations based on the following principle:

$$\begin{aligned} G_t &:= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{7}$$

This equation can replace the exhaustive calculations into a single calculation between the current reward and the expected return of the next state, meaning that calculating the return of the current state becomes one calculation given the return of the next state. Formally, this type of problem-solving is called Dynamic Programming, which divides the current problem into smaller subproblems and reduces the amount of calculation to solve the current problem by utilizing the solution of the subproblem. Based on Eq.7, Eq.3 can be expressed as

$$\begin{aligned} q_\pi(s, a) &:= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, a) | S_t = s, A_t = a] \end{aligned} \tag{8}$$

Since this allows recursively simplify the value function for the current state based on the

value function for the next state, S_{t+1}, this can simplify the optimal value function as follows:

$$q_*(s,a) := \max_{\pi} q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} q_*(S_{t+1}, a') | S_t = s, A_t = a] \tag{9}$$

[Note that $\max_{a' \in \mathcal{A}(S_{t+1})} q_*(S_{t+1}, a')$ means the optimal value function of the next state, meaning that the value of taking action a', action that gives the highest value, at state S_{t+1}]

This equation is called the Bellman Optimality Equation. Observe that the optimal value function for the current state is now dependent on the one for the next state[3]. Therefore, once the agent reaches the terminal state that it was looking for, it redefines the value function of the smallest subproblem, which is the immediately preceding state from the terminal state. Then, it redefines the value function for the next smallest subproblem, the state two steps before the terminal state, using the solution of the smallest subproblem it just calculated. It recursively redefines all subproblems, in increasing order of its scale, until it redefines the value function for the current state.

Suppose that the agent committed n steps to reach the terminal state and start calculating the return value for the initial state. Calculation with Eq.3 would require linear time $\mathcal{O}(n)$, while Eq.7 costs constant time $\mathcal{O}(1)$ given that all the solutions for the required subproblems are stored as it backtracks.

2-3. Q-Learning(QL)

Q-learning(QL) is one of the RL algorithms that can be used when there is a finite number of possible states and actions. In QL, the agent constructs the policy directly by filling in a Q table based on the interaction with the environment. The Q table is a table in which rows and columns represent the states and actions, respectively, and Q values stored in each slot. A Q value, $Q(s,a)$, represents the state-action value, the expected amount of

future reward as a particular action a is taken in a given state s[4]. The higher the Q value, the higher future rewards can be expected, thus making the agent more likely to choose that action. Updating the system of Q value respects the Dynamic Programming paradigm, once the agent reaches the terminal state, it backtracks its steps to its initial state. During this, it updates the Q values of all (state, action) combinations that it visited until the terminal state. Not surprisingly, updating the function for Q values satisfies the Bellman Optimality Equation as follows:

$$Q^{new}(s,a) = Q^{old}(s,a) + \alpha\Big[R_{t+1} + \gamma \cdot \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a) - Q^{old}(s,a)\Big] \tag{10}$$

where α is learning rate and γ is discount factor ($\alpha, \gamma \in [0,1]$). A learning rate is a hyper-parameter that determines how fast the algorithm learns, in other words, how much the old value affects updating. If the learning rate is 1, then the old Q value does not take action to update but is simply replaced by a new Q value. As the learning rate value decreases, the portion for the old value would increase, and updating would gradually reflect a new Q value. The pseudocode of the Q-Learning algorithm follows:

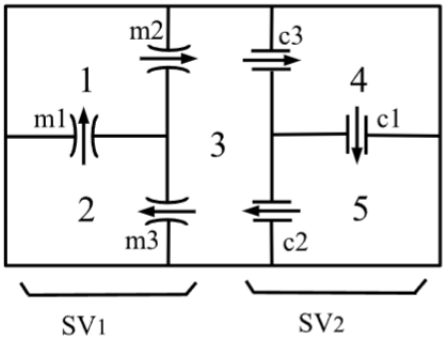
QL Algorithm

1. Initialize Q-table (Initialize all Q-values to 0)
2. For each episode do
   a. Set state to the initial state
   b. Choose an action
   c. Convey the action to the environment
   d. Receive a new state and reward
   e. Update Q-value using Eq.(8)
   f. Check whether the current episode is terminated

[Figure 2. Q-Learning Algorithm]

2-4. Cat and Mouse Problem

This section introduces the proposed problem that Ushio and Yamasaki used to demonstrate their algorithm, called the Cat and Mouse problem.



[Figure 3. Maze for cat and mouse with 2 supervisors]

There are 5 rooms partitioned by doors as in Figure 3. Each door is one-way, used by a cat or a mouse exclusively. Mouse and cat locates 2 and 4, denoted as $(2, 4)$ or 24, at the initial state. This system terminates if both cat and mouse are located in room 3, i.e. when the current state becomes $(3, 3)$. The goal of this problem is to control doors to avoid this. Defining what "controlling" the door refers to is essential, however, Ushio and Yamasaki were not consistent in their definition. When they introduced this problem, they defined that the control policy in this problem means which door should be closed[5] while the figure that represents their result implies that each event refers to which door should be opened.[6]. This paper follows later, i.e. enabling the event represents opening the corresponding door.

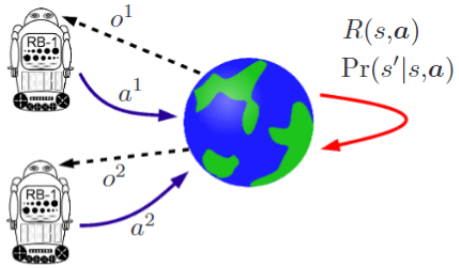
To reach the solution, two decision-makers, called supervisors, are allowed to control each kind of event, c_i's and m_i's (in this paper, the term "supervisor" is used interchangeably with the "agent" from RL). SV_1, mouse's supervisor, can observe $\{m_1, m_2, m_3, c_2, c_3\}$ are happening and control $\{m_1, m_2, m_3\}$. On the other hand, SV_2, the one for cat, can observe

$\{m_2, m_3, , c_1, c_2, c_3\}$ and control $\{c_1, c_2, c_3\}$. Since c_2 is uncontrollable for SV_1, it cannot present a cat entering room 3 from room 2. Cat and Mouse move if supervisors enable the event, and open the door, which allows them to move to the next room. For example, the cat moves from room 4 to room 5 only when c_1 is just enabled.

Since each supervisor can observe only part of the system, they would receive its local perspective of the state from the environment. For example, suppose the current state of the is (2,4) and event m_1 is selected. Since SV_1 does observe m_1, it would receive its local state of (1,4). However, since SV_2 cannot observe m_1 is happening, its local state would maintain (2,4).

2-6. Decentralized Partially Observable Markov Decision Process

This section highlights the difference between the structures of the usual MDP and the Cat and cat-and-mouse problem. As introduced in Sections 2-2, the MDP system is controlled by a single, centralized agent. However, the system of the cat-and-mouse problem is controlled by multiple agents where each has partial control of the system, thus it is called a Decentralized system. Furthermore, each controller not only partially controls but partially observes the system. This is called Decentralized Partially-Observable Markov Decision Process(Dec-POMDP).



[Figure 4. Execution of a Dec POMDP][7]

At each step, the agents *independently* take an action, The environment undergoes a state transition and generates a reward depending on the state and the actions of both agents. One of the key differences between MDP and Dec-POMDP is that the **Value** is defined as the expected return of the **Joint Policy**, a tuple of all local policies. Hence, multiple agents collaborate to maximize the value corresponding to the global policy.

A common assumption in Dec-POMDP is a separation of *off-line* phase and *on-line* phase. First, agents choose local policies and send those to the environment. Then, during the *off-line* phase, which is centralized (the Red line in Figure 4), another single computer computes the joint policy and subsequently distributes it to the agent. Then the decentralized *on-line* phase follows; each agent receives its part of the joint policy and its history of actions and observations[8]. A key point of Dec POMDP is that even though each agent receives its own observation of the state, they share one **reward function** so that they collaborate for the same goal.

Based on that, each agent aims to maximize the value of the joint policy while keeping the policies of the other agents fixed. Then, the next agent is chosen to maximize the value by finding its best policy. This is called alternating maximization; this process is repeated until the joint policy converges to Nash equilibrium[9]. Nash equilibrium is a concept in game theory in which the game reaches the optimal outcome. This is a state where all players(agents) keep optimal strategies(policies) even after consideration of other's strategies[10]. Therefore, the joint policy reaches the **Optimal Joint Policy** when all agents choose their local policy that is optimal for the value(of joint policy) while it does not affect each other's policy.

3. Method

This section introduces the proposed algorithm and equations in Yamasaki and Ushio's paper. Since the ultimate purpose of this paper is to analyze the difference between their equation and typical Dec-POMDP and how that difference affects the result, an in-depth comparison between the two is introduced in Section 5 while this section simply introduces and summarizes their work.

3-1. Notation Summary

The following table shows what each symbol and notation stands for in their paper.

Notation	Meaning
SV_i	i^{th} Supervisor
Σ / Σ_i^o	Set of events / Set of observable events of SV_i
$\sigma \in \Sigma$ / $\sigma_i^o \in \Sigma_i^o$	An event / An event that can be observed by SV_i
S_i	Set of states of SV_i
g_i	$S_i \times \sum_i^{o*} \rightarrow S_i$; State Transition function
$Q_i(s_i, \pi_i)$	Discounted expected total reward in the case that SV_i selects π_i at s_i
$R_i^1(s_i, \pi_i)$	Expected reward of π_i at s_i
$T_i(s_i, \sigma_i^o)$	Discounted expected total reward when SV_i observes σ_i^o at state s_i and selects the optimal control policy
$\eta_i(s_i, \sigma_i^{o'})$	Probability of SV_i observes event σ_i^0 at s_i
$\mathcal{P}_i^1(s_i, \pi_i, \sigma_i^o)$	probability that SV_i observes the occurrence of an event $\sigma_i^o \in \{\pi_i \cap \sum_i^o\}$, when SV_i selects π_i at s_i
$\Pi_i(s_i)$	A set of control policy at state $s_i \in S_i$
$\pi_i \in \Pi_i(s_i)$	A control policy based on current state s_i
π	Net control policy; intersection of each supervisor's control policy
$F_i(s_i)$	Active Event set over $\sum$ at state s_i ; set of feasible events that an make a transition from s_i to some legal state
r_i^1	Rewards that SV_i get from evaluation for the control policy π_i
r_i^2	Rewards that SV_i get as it observes σ_i^o

[Figure 3. Table of Notations from the Paper]

3-2. Proposed Reward System in the paper

This section introduces how Ushio and Yamasaki construct the reward system for their RL learning algorithm. For closing each controllable door, it takes a cost(reward) of -2. They also mentioned that r_i^1 is defined as the sum of the cost to close doors. If the cat and mouse move to the next room, the corresponding supervisor will receive a reward $r_i^2 = 1$, encouraging them to open the door. However, if the cat and mouse are encountered in room 3, both supervisors will obtain a negative reward(punishment) of $r_i^2 = -10$. By assigning punishment to the state that they should avoid, each supervisor will try to disable events that lead to the encounter.

3-3. Algorithm & Equation from Paper

At state $s_i \in S_i$, each supervisor selects its local control policy, π_i, representing which doors should be opened or closed. It utilizes ϵ-greedy selection, letting the supervisor choose the policy that shows the highest Q value with a probability of $1 - \epsilon$. This is called Exploitation. Otherwise, it selects the policy randomly, which is called Exploration. Exploration & Exploitation trade-off is one of the fundamental concepts in decision-making about balancing between greedily selecting the option that is assumed to be optimal at the current state and exploring undiscovered choices for a better option with the risk of taking a worse one. Ushio and Yamasaki set $\epsilon = 0.1$, which means that their algorithm will exploit with 90% or explore for a new option with 10%.

Once n supervisors select local policies, the algorithm then calculates the net control policy, i.e global control policy π, by taking the intersection of all local policies as follows:

$$\pi = \bigcap_{i=1}^{n} \pi_i \tag{11}$$

If there are more than events in the net control policy, then it is further narrowed down into a single event based on a new parameter η. Unfortunately, they have not defined η value in their paper, but provided the following equations which explain the properties of η:

$$\mathcal{P}_i^1(s_i, \pi_i, \sigma_i^o) = \frac{\eta_i^*(s_i, \sigma_i^o)}{\sum_{\sigma_i^{o'} \in \pi_i \cap \sum_i^o} \eta_i^*(s_i, \sigma_i^{o'})} \qquad \text{(Eq.16 in the paper)} \tag{12}$$

where $\mathcal{P}_i^1$ is a probability that SV_i observes the occurrence of an event $\sigma_i^o \in \{\pi_i \cap \sum_i^o\}$, when SV_i selects π_i at s_i, and

$$\eta_i^*(s_i, \sigma_i^o) > 0, \quad \sum_{\sigma_i^{o'} \in F_i(s_i) \cap \sum_i^o} \eta_i^*(s_i, \sigma_i^{o'}) = 1 \tag{13}$$

Given that $F_i(s_i) \cap \sum_i^o$ represents the set of all possible legal(feasible) and observable events that can happen at s_i, this paper assumed η refers to the probability of SV_i observes event σ_i^0 at s_i since the sum of η for all observable & feasible events at s_i is 1. Note that each supervisor defines local η value for all (state, feasible event) pairs. Their algorithm also lacks specification about how those η inform the algorithm to narrow the net control policy to a single event. They wrote that "An event $\sigma \in \pi$ occurs in the DES. This occurrence is not affected by supervisors, but restricted by Eq.16."[11] Since they have not explained how P_i^1 values are utilized for restriction, this paper assumes that the algorithm selects an event whose local η_i is the highest, to be aligned with the greedy selection for local control policy.

As the environment receives the selected event, it makes a transition to the next state. If SV_i can observe the event, then its local perspective(local state) will also be transitioned, otherwise maintain the current local state. This is followed by returning that state with the numerical rewards to each SV_i. Specifically, Ushio and Yamasaki proposed two types of rewards: one is evaluation for control policy π_i, denoted as r_i^1, and the other is for observation

of σ_i^o, denoted by r_i^2. As each supervisor receives states and rewards, it updates the following local parameters:

- $T_i(s_i, \sigma_i^o)$: Discounted expected total reward when SV_i observes σ_i^o at state s_i and selects the optimal control policy
- $R_i^1(s_i, \pi_i)$: Expected reward of π_i at s_i
- $\eta_i(s_i, \sigma_i^{o'})$: Probability of SV_i observes event σ_i^0 at s_i

Since these are local parameters, each supervisor stores its values corresponding to all possible input pairs. Updating equations for each parameter follows:

$$T_i(s_i, \sigma_i^o) \leftarrow T_i(s_i, \sigma_i^o) + \alpha\big[r_i^2 + \gamma \max_{\pi' \in \Pi_i(s_i')} Q_i(s_i', \pi_i') - T_i(s_i, \sigma_i^o)\big] \tag{14}$$

$$R_i^1(s_i, \pi_i) \leftarrow R_i^1(s_i, \pi_i) + \beta\big[r_i^1 - R_i^1(s_i, \pi_i)\big] \tag{15}$$

for all $\sigma_i^{o'} \in \pi_i \cap \Sigma_i^o$

$$\eta_i(s_i, \sigma_i^{o'}) \leftarrow \begin{cases} (1-\delta)\eta_i(s_i, \sigma_i^{o'}) & (\text{if } \sigma_i^{o'} \neq \sigma_i^o) \\ \eta_i(s_i, \sigma_i^{o'}) + \delta \left[\sum_{\sigma_i^{o''} \in \pi_i \cap \Sigma_i^o} \eta_i(s_i, \sigma_i^{o''}) - \eta_i(s_i, \sigma_i^{o'}) \right] & (\text{if } \sigma_i^{o'} = \sigma_i^o)) \end{cases} \tag{16}$$

where α, β, δ are learning rate of 0.1. Lastly, with updated local parameters, supervisors update local Q-values if they can observe the event based on the following equation:

$$\begin{aligned} &\forall \pi_i' \in \Pi_i(s_i) \quad \text{s.t} \quad \pi_i' \cap \pi_i \neq \emptyset, \\ &Q_i(s_i, \pi_i') \longleftarrow R_i^1(s_i, \pi_i') + \sum_{\sigma_i^{o''} \in \pi_i' \cap \Sigma_i^o} \frac{\eta_i(s_i, \sigma_i^{o''})}{\sum_{\sigma_i^{o'''} \in \pi_i \cap \Sigma_i^o} \eta_i(s_i, \sigma_i^{o'''})} T_i(s_i, \sigma_i^{o''}) \end{aligned} \tag{17}$$

This process is repeated until the environment reaches the terminal state, and one episode ends. The pseudocode of their algorithm follows:

Algorithm in Ushio's Paper

1. Initialize T_i, R_i^1, and η_i for all SV_i
2. Initialize Q values of all SV_i by Eq.17
3. Repeat until any s_i is a terminal state

   (for each episode):

   a. Initialize a state $s_i \leftarrow s_0$(Initial State) for all SV_i

   b. Repeat for each SV_i (for each step of an episode):

      i. Select a control policy $\pi_i \in \Pi_i(s_i)$ based on Q_i values by SV_i

      (As a result, a net control policy π is assigned to the DES G)

      ii. Observe the occurence of event $\sigma_i^o \in \sum_i^o$

      iii. Acquire reward rewards r_i^1 and r_i^2

      iv. Make a transition $s_i \xrightarrow{\sigma_i^o} s_i'(= g_i(s, \sigma_i^o))$ in SV_i

      v. Update $T_i(s_i, \sigma_i^o)$, $R_i^1(s_i, \pi_i)$, and $\eta_i(s_i, \sigma_i^{o'})$ by Eq.14, Eq.15, and Eq.16 respectively

      vi. Update the Q_i values by Eq.17

      vii. $s_i \leftarrow s_i'$

[Figure 4. Learning Algorithm created by Ushio and Yamasaki][12]

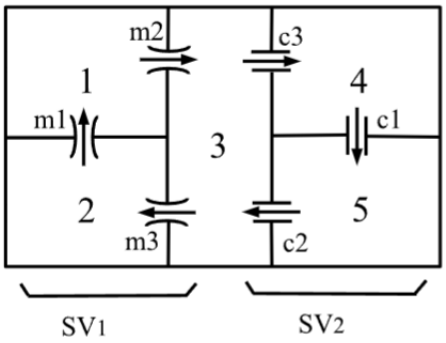
4. Results

This section introduces the reproduction of their algorithm. Based on the result, the code has been updated if the result was considered an error in implementation. Therefore, 3 different experiments were conducted. After each model had been trained, it was demon-

strated in the main file, 20 steps were simulated and its performance was observed. All experiments have the same value of hyper-parameters as follows:

- Epoch(# Episode): 1000
- $\alpha = 0.1$
- $\beta = 0.1$
- $\gamma = 0.9$
- $\delta = 0.1$
- $\epsilon = 0.1$

4-1. Experiment 1



[From Figure 3]

The Q table is implemented as 6×8 metric, 6 refers to the number of local states and 8 refers to the number of possible control policies for each supervisor. Since each mouse and cat can be in 3 different rooms, the total number of states is $3 \times 3 = 9$. However, in the perspective of SV_1, since it cannot observe c_1, it cannot distinguish the cat being in room 4 and room 5. Thus, it cannot distinguish (1,4) & (1,5), (2,4) & (2,5), and (3,4) & (3,5). Similarly, since SV_2 cannot observe m_1, it cannot distinguish (1,3) & (2,3), (1,4) & (2,4), and (1,5) & (2,5). Therefore, each supervisor has 6 different local states. Meanwhile, since each supervisor has 3 controllable doors and each door can be either opened or closed, there

are $2^3 = 8$ different control policies, provided that uncontrollable doors are assumed to be opened.

Similar to Q-table, since $T_i(s_i, \sigma_i^o)$, $R_i^1(s_i, \pi_i)$, and $\eta_i(s_i, \sigma_i^{o'})$ values are dependent on either (state, policy) pair or (state, event), each supervisor constructs 3 more tables, one for each parameter. Hence, the R^1 table is constructed as 6×8 metric, T table and η table are constructed as 6×5 metrics since each supervisor has 5 observable events.

Result

The following sequence is a history of the global states and selected events.

$$\{(2,4), m_1, \quad (1,4), c_1, \quad (1,5), c_1, \quad (1,5), c_1, \quad (1,5), c_1 \quad \cdots (\text{Repeated until 20 steps})\}$$

At the initial state, m_1 was selected, then after that, c_1 was continuously selected, even at the state of (1,5), in which feasible events are m_2 and c_2. Since a mouse or cat would move only when those two events happen, they stopped moving as c_1 was repetitively selected. This can conclude that an event is not feasible in the state can still be selected because the agent can consider a non-feasible policy(a policy that enables non-feasible events) without recognizing that it does not lead cat and mouse to move. This result motivated the next experiment to restrict the agent to select the policy that only controls feasible events of the current state.

4-2. Experiment 2

In this experiment, each agent can only control the feasible events. For example, at (2,4), the feasible event for SV_1 is m_1; thus, it can choose either $[c_1, m_1, m_2, m_3]$ or $[m_1, m_2, m_3]$. Since the supervisor has only two possible local policies at each state, the

Q table and R^1 table are reduced into 6×2 metric. T table and η table are maintained as 6×5 metric.

Result

The following sequence is a history of the global states and selected events.

$$\{(2,4), m_1, \quad (1,4), m_2, \quad (3,4), m_3, \quad (2,4), m_1, \quad (1,4), m_2, \quad (3,4), m_3, \quad \cdots (\text{Repeated until 20 steps})\}$$

The problem of selecting the non-feasible policy is resolved, but the result above is not sufficient to be called the optimal solution because only the mouse moves while the cat stays. The in-depth analysis detected an implementation issue regarding η tables below:

States	m_1	m_2	m_3	c_2	c_3
(1,3)	23.19	5.95	1.99	0	0.5
(1,4) /(1,5)	5.25	0.81	0	0.5	0
(2,3)	10.93	1.53	0	0	0.5
(2,4) /(2,5)	6.68	4.42e-201	0	0.5	0
(3,3)	0	0	0.5	0	0.5
(3,4) /(3,5)	8.34	0	6.66e-154	0.5	0

[Table 1. η table for SV_1]

States	m_2	m_3	c_1	c_2	c_3
(1,3) /(2,3)	2.44	0	0.85	0	0.5
(1,4) /(2,4)	2.64e-124	0	3.72e-125	0	0
(1,5) /(2,5)	0.29	0	0.4	0.5	0
(3,3)	0	0.5	0	0	0.5
(3,4)	0	1.009	0.16	0	0
(3,5)	0	50.03	3.1	0.5	0

[Table 2. η table for SV_2]

Since the η value is defined to be the probability, the result above is a clear error. The following code is the implementation of updating the η algorithm:

```
1  def update_eta(eta_table, state, local_policy, event, delta, is_mouse):
2
3      observable_policy = local_policy.copy()
4      if is_mouse:
5          observable_policy.remove("c1")
6      else:
7          observable_policy.remove("m1")
8
9      for event_prime in observable_policy:
10         event_prime_index = observable_policy.index(event_prime)
11         if event_prime == event:
12             eta_table[state, event_prime_index] = eta_table[state,
       event_prime_index]+delta*(np.sum(eta_table[state])-eta_table[state,
       event_prime_index]) # Lower branch of Eq.16
13         else:
14             eta_table[state, event_prime_index] = (1-delta)*eta_table[
       state, event_prime_index] # Upper branch of Eq.16
```

[Code 1. Updating η value from Experiment 2]

Suppose SV_1's local policy is $[m_2, c_1, c_2, c_3]$ at some state. Since c_1 is removed by line 5, *obervable_policy* would be $[m_2, c_2, c_3]$. The main issue of the result above was indexing. Here, *event_prime_index* of 0 would represent m_2 based on line 10, while the actual index of m_2 in the η table is not 0, but 1 from Table 1. Thus, the algorithm was supposed to update the η value for m_2, but actually it updated m_1. Because the algorithm only updates up to the third column of the table, given that the length of the *observable_policy* would be 3 ($[m_i, c_2, c_3]$) or 2 ($[c_2, c_3]$) after c_1 being removed, this explains the reason why η value for c_2 and c_3 maintained the initial value of 0.5. Therefore, this paper conducted another experiment after fixing this issue.

4-3. Experiment 3

In this experiment, to be consistent on the indexing, new constant lists *MOUSE_OBSERVABLE_EVENTS* and *CAT_OBSERVABLE_EVENTS* are defined as

follows:

- MOUSE_OBSERVABLE_EVENTS = $[m_1, m_2, m_3, c_2, c_3]$
- CAT_OBSERVABLE_EVENTS = $[m_2, m_3, c_1, c_2, c_3]$

and the algorithm will derive the index of events only based on two lists. Furthermore, to simplify the code, each supervisor has a function for updating η. Updated *update_eta* function follows:

```
def update_eta_mouse(eta_mouse, old_state, event, delta):
    # Skip if the supervisor cannot observe the selected event
    if event not in MOUSE_OBSERVABLE_EVENTS:
        return

    # Utilize the constant list to be consistent on the indexing
    event_index = MOUSE_OBSERVABLE_EVENTS.index(event)

    # Calculate new eta value based on the Lower branch of Eq.16
    new_event_eta = eta_mouse[old_state, event_index]+delta*(np.sum(
   eta_mouse[old_state])-eta_mouse[old_state, event_index])

    #Upper branch of Eq.16
    eta_mouse[old_state] = eta_mouse[old_state]*(1-delta)

    # Updating eta value for the selected event
    eta_mouse[old_state, event_index] = new_event_eta
```

[Code 2. Updating η value for the mouse from Experiment 3]

Result

The following are a history sequence of the (global states, selected events) and updated η tables:

$\{(2,4), m_1, \quad (1,4), c_1, \quad (1,5), c_2, \quad (1,3), c_3, \quad (1,4), c_1 \quad (1,5), c_2, \quad \cdots (\text{Repeated until 20 steps})\}$

States	m_1	m_2	m_3	c_2	c_3
(1,3)	0	0.02	0	0	0.98
(1,4) /(1,5)	0	0.11	0	0.89	0
(2,3)	0.1	0	0	0	0.9
(2,4) /(2,5)	0.3	0	0	0.7	0
(3,3)	0	0	0.5	0	0.5
(3,4) /(3,5)	0	0	0.94	0.06	0

[Table 3. η table for SV_1]

States	m_2	m_3	c_1	c_2	c_3
(1,3) /(2,3)	0.003	0	0	0	0.997
(1,4) /(2,4)	0.02	0	0	0.98	0
(1,5) /(2,5)	0.03	0	0	0.97	0
(3,3)	0	0.5	0	0	0.5
(3,4)	0	0.96	0.04	0	0
(3,5)	0	0.88	0	0.12	0

[Table 4. η table for SV_2]

Unfortunately, fixing η function did not change the result. Several more training was conducted, but the results were either the cat is cycling and the mouse stays, or vice versa. However, additional training detected an interesting tendency about training count and its correlation with the state-event history.

Training 1

Training Counts

$m_1 : 5949 \quad m_2 : 5863 \quad m_3 : 5223 \quad c_1 : 101 \quad c_2 : 705 \quad c_3 : 373$

State-Event History

$\{(2,4), m_1, \quad (1,4), m_2, \quad (3,4), m_3, \quad (2,4), m_1, \quad (1,4), m_2, \quad \cdots (\text{Mouse cycles})\}$

Training 2

Training Counts

$m_1 : 1078 \quad m_2 : 721 \quad m_3 : 394 \quad c_1 : 6174 \quad c_2 : 6055 \quad c_3 : 5430$

State-Event History

$\{(2,4), m_1, \quad (1,4), c_1, \quad (1,5), c_2, \quad (1,3), c_3, \quad (1,4), c_1 \quad \cdots (\text{Cat cycles})\}$

Not only there were a clear unbalance of training counts between c_i's and m_i's, but cat or mouse whose events trained more cycles while the other stayed.

Since no other implementation issue has been detected, this experiment concluded that Ushio and Yamasaki's algorithm did not work.

5. Discussion

This section will propose possible errors that cause the result from Section 4-3. According to Section 2-2, the supervisor(or the agent in RL) aims to maximize the value for every state. Therefore, comparing which values are maximized in Dec-POMDP and Ushio and Yamasaki's paper may explain the result.

As Section 2-6 introduced, in typical Dec-POMDP, each agent aims to find a local policy that is optimal for the value for the joint policy and eventually reaches Nash Equilibrium. The fundamental concept behind this idea is a shared reward function through all agents.

On the other hand, according to Eq.17, supervisors in their algorithm aim to maximize their own local Q values based on independent reward function. This implies that they are maximizing two different things. Hence, two supervisors actually compete with each other instead of collaborating, appealing its local policy over others. This possibly leads to a conflict between two policies and confuses the system about which policy it should choose.

As introduced in Section 3-3, an event is selected from multiple local policies by its η value. According to their algorithm, the η value for a selected event is incremented while the others are decremented by Eq.16. In other words, η increases as the event gets selected and the event with higher η is more likely to be chosen; this implies that the event is selected more frequently, it is more likely to be selected while others are ignored. The

result from Section 4-3 reflects this logic, as either c_i's or m_i's selected more in the early training, they obtained higher η. Then, they start dominating the opponent, making the system selects them. That is why the training counts were unbalanced.

As a result, Ushio and Yamasaki not only did not follow Dec-POMDP principles, but also could not find a proper way to derive global optimal policy based on local optimal policies.

6. Reflection

The biggest difficulty throughout this project was the paper's readability. Not only were they not consistent with the concept they were proposing, but unnecessarily complicated notation made the overall equation hard to understand, even the simple one. Overall, poor readability increased the time spent to analyze their algorithm. However, the implementation proposed in this paper would not be 100% equivalent to their work due to the inconsistency and ambiguity of their paper.

Nonetheless, the reproduced implementation demonstrated that their learning algorithm did not derive the optimal solution to the Cat and Mouse Problem since the trained model was neither consistent nor resolved the problem. Furthermore, the previous discussion section argued the reason for the result based on the possible bias from their equations.

Unfortunately, the only possible option to improve this research is conducting another experiment with different assumptions made regarding η. Therefore, instead, this paper wishes to motivate the question, "So, how can we derive the actual optimal policy?" This paper has provided enough knowledge and tools about what to do and not to do.

References

[1] Sutton, Richard and Barto, Andrew. Reinforcement Learning An Introduction. 1. The MIT Press, 2018.

[2] Oliehoek, Frans A. "Decentralized POMDPs." Reinforcement Learning State of Art, edited by Marco. Wiering and Martin v. Otterlo, 11 Section 4.1. Springer Berlin, Heidelberg, 2012.

[3] Brunton, Steve. "Model Based Reinforcement Learning: Policy Iteration, Value Iteration, and Dynamic Programming." YouTube, 2022. https://www.youtube.com/watch?v=sJIFUTITfBc%5C&t=1365s.

[4] Daniel Soper, "Foundations of Q-Learning," April 22, 2020. educational video, https://www.youtube.com/watch?v=__t2XRxXGxI

[5] Ushio, Toshimitsu and Yamasaki, Tatsushi. "Decentralized Supervisory Control in Discrete Event Systems Based on Reinforcement Learning." 3049. In Transactions of the Institute of Systems Control and Information Engineers, ISCIE, November 2005.

[6] Ushio, Toshimitsu and Yamasaki, Tatsushi. "Decentralized Supervisory Control in Discrete Event Systems Based on Reinforcement Learning." 3049 Figure 4. In Transactions of the Institute of Systems Control and Information Engineers, ISCIE, November 2005.

[7] Oliehoek, Frans A. "Decentralized POMDPs." Reinforcement Learning State of Art, edited by Marco. Wiering and Martin v. Otterlo, 2 Figure 1. Springer Berlin, Heidelberg, 2012.

[8] Oliehoek, Frans A. "Decentralized POMDPs." Reinforcement Learning State of Art,

edited by Marco. Wiering and Martin v. Otterlo, 2. Springer Berlin, Heidelberg, 2012.

[9] Oliehoek, Frans A. "Decentralized POMDPs." Reinforcement Learning State of Art, edited by Marco. Wiering and Martin v. Otterlo, 11 Section 4.2. Springer Berlin, Heidelberg, 2012.

[10] Chen, James. "Nash Equilibrium: How It Works in Game Theory, Examples, plus Prisoner's Dilemma." Investopedia, January 5, 2024. https://www.investopedia.com/terms/n/nash-equilibrium.asp.

[11] Ushio, Toshimitsu and Yamasaki, Tatsushi. "Decentralized Supervisory Control in Discrete Event Systems Based on Reinforcement Learning." 3048. In Transactions of the Institute of Systems Control and Information Engineers, ISCIE, November 2005.

[12] Ushio, Toshimitsu and Yamasaki, Tatsushi. "Decentralized Supervisory Control in Discrete Event Systems Based on Reinforcement Learning." 3049 Figure 2. In Transactions of the Institute of Systems Control and Information Engineers, ISCIE, November 2005.