Robust multi-agent temporal difference learning in cooperative systems with adversaries

Eleni Nisioti, Daan Bloembergen, Michael Kaisers

July 8, 2020

1 Introduction

Most real-world problems involve multi-agent systems (MAS), where uncertainty is an emerging property arising due to interactions among agents, rather than an external model misspecification. In such systems, agents use reinforcement learning to co-evolve optimal policies aligned with their expectations of rewards given by the environment. Interactions during training affect the learning process and often lead to brittle policies, which perform well during training but fail remarkably in novel evaluation environments. Cooperation in multi-agent systems is a design principle in many real-world problems, such as power grids, communication networks and traffic networks, that simplifies the training process, but can lead to optimal policies vulnerable to adversaries aiming at harming the operation of the MAS.

Robustness is a long-standing pursuit in the control and reinforcement learning theory Zhou et al. (1996); Morimoto and Doya (2005). While single-agent approaches pursue robustness during learning or planning by considering stochastic perturbations in transition probabilities and rewards (Abbasi Yadkori et al., 2013; Abdullah et al., 2019) or time-variant Markov dynamics Lecarpentier and Rachelson (2019), multi-agent reinforcement learning (MARL) studies robustness primarily in terms of performance in the presence of different types of agents. All robust learning approaches share, however, a common ground: the environment is governed by some sort of uncertainty. In MARL, a policy is considered robust when agents perform well in various multi-agent environments, not necessarily encountered during the training process.

Minimax decisdion rules are a common approach to designing robust policies in MARL (Li et al., 2019). Devised in game theory to compute best-response policies in zero-sum games (von Neumann and Morgenstern, 1947), minimax strategies can be straightforwardly adopted to design agents acting in their best interests and best-responding to other agents that behave as zero-sum opponents (Littman, 1994). In this way, the system can learn conservative policies, where each agent acts safely in an egocentric way. As is customary, we refer to malicious agents behaving as zero-sum opponents as adversaries.

Many real-world cooperative multi-agent systems are critical: the operation of communication networks, power grids, traffic and transport networks is characterized by unsafe regions, which are usually associated with loss of information due to over-flows or physical damage of components due to over-loads. Recent innovations in traditional infrastructures, such as the smart grid and the Internet of Things, have improved the efficiency and usability of these systems, but also inadvertently increased their vulnerabilities. Due to practical issues such as safety and cost, policies are traditionally learned offline, and then deployed in the real-world. Thus, intelligent policies deployed on these systems need to be apriori robust to potential attacks. Communication during deployment is limited by the hardware and bandwidth characteristics of the MAS, so execution of these policies needs to bet decentralized. For a comprehensive review of safe reinforcement learning, we refer readers to (García et al., 2015).

Cooperative systems are trained with the assumption that all agents aim at maximizing a common reward signal, which leads to policies that are not robust to a misbehavior of nodes and, therefore, vulnerable to attacks. In this work, we focus on misbehavior due to a fixed number of adversaries that arrive at some random time step in the system, perform an adversarial selection of agents and, then, directly manipulate their actions. We refer to this type of attack as a *multi-agent*

adversarial attack. This is a novel type of attack that extends the classical notion of worst-case selection of actions in an adversarial attack to an adversarial choice among multiple agents. It can be of particular interest in MAS where different assets may be more vulnerable or vital to the operation of the networks, such as financial markets, computer cluster and communication networks where the attack of central nodes can bring cascading failures.

To learn robust multi-agent policies we design a temporal difference learning operator, where the value of the target policy is computed assuming that a multi-agent adversarial attack is taking place. We refer to this operator as robust multi-agent Q-learning (Rom-Q). Training is performed in a centralized manner, where an agent observing the system state and actions of all nodes learns a Q-value function in the joint state-action space. To evaluate the target policy, the agent considers all possible selections of a given number of adversaries and performs an update that augments the selection of actions in minimax-Q with an adversarial selection of nodes based on the system value function. Thus, a learning update requires the solution of a number of linear programs, identical to the ones solved by minimax-Q, whose number depends on the number of present agents and assumed number of adversaries. A different policy is learned for the actions of each agent, execution is therefore decentralized.

In the remainder of this report, we focus on the analysis of Rom-Q. In Section 2, we compare our operator to other operators pursuing robustness or considering adversarial attacks in MARL. In Section 3, we provide the theoretical background required to understand Rom-Q. In Section 3, we analyze our robust multi-agent operator, Rom-Q, and provide its algorithmic implementation. Finally, Section 6 contains our simulations aiming at evaluating the ability of Rom-Q in finding optimal and robust policies and comparing it to the.

2 Related work

When studying robustness in MARL, there's two essentially distinct ways to view adversaries. The Bayesian approach views adversaries as players in a game played between themselves and the cooperative agents comprising the system (Johanson et al., 2008). Under this direction, agents need to follow some form of opponent-aware reinforcement learning and attempt to learn policies that converge to the Nash equilibria of a general-sum or zero-sum game. Naturally, the model of the adversaries needs to be known and correct. This approach has the advantage of finding policies that are less brittle and easier to compute than best-responses. These advantages are vital when policies are adapted in an online manner, but lose their importance when multi-agent policies are learned off-line, in safe simulation environments with sufficient resources.

In this latter scenario, best-response policies to attacks based on the agents' current estimation of the values of the optimal policy can be found apriori. A robust operator "imagines" attacks during training, in order to come up with a policy robust to this type of attack (Klíma et al., 2019) and learns conservative policies without requiring the simulation of attacks during training. As the effect of an attack is calculated based on an agent's own value function, a model of the adversary is not required. There is of course a trade-off between the attack profiles a policy is robust to and the optimality of the solution.

A variety of RL algorithms has already been combined with minimax updates. The stage was set by the seminal work of (Littman, 1994) that introduced minimax-Q by blending the framework of MDPs with Markov games. Minimax policy gradients were introduced in (Li et al., 2019) to ensures robustness to various types of opponents in a multi-agent setting. This work focuses on continuous action spaces, as it employs deep learning for function approximation. Robust temporal difference learning (Klíma et al., 2019) considered security games where agents are attacked with a certain probability and modified classical temporal difference learning with minimax updates. Our work resembles this approach, as we also define a temporal difference algorithm for being robust to a certain profile of attack. However, there are some notable differences. First, in (Klíma et al., 2019), all agents are attacked with an equal probability, which decreases in the number of agents. In contrast, adversaries in our formulation find the most vulnerable agents to attack.

When studying safety in RL, in addition to the policies followed by adversaries, we also need to define the *type of attack*. In our definition, we consider that policies define the actions that an adversary performs when controlling an agent, while the type of attack includes all other

parameters required to fully describe the attack, which, among others, may include the number of attackers, selection of nodes and probability of occurrence. We can argue that safety is in the eye of the designer, with different approaches to robustness anticipating different types of attacks. In the spirit of the recent deep reinforcement learning bloom, adversaries often manipulate the observations of agents with the aim of fooling the function approximators used for decision making. This type of attack can be at most as effective as the direct manipulation of actions considered in our multi-agent adversarial attack. The work in (Lin et al., 2020) introduces a novel attack in cooperative systems, where an attacker first uses reinforcement learning to find the actions that will have the worst long-term effect on the system reward and, then, manipulates the observations of an agent to lead the victim to taking the wrong action. In contrast, our attack is short-sighted, choosing the actions that will bring the lowest reward in the current state. This is more appropriate in critical systems, where adversarial attacks attempt to bring the largest damage in a short time span, and do not usually have the luxury of interacting with the system.

3 Background

For our mathematical notation, we use $\mathcal{S}$ to represent a set, $\vec{s}$ for a vector, lower-case letters for random variables and upper-case letters for functions. We only use vector notation when referring to variables of a MAS, ignoring the fact that a single agent can have a multi-dimensional state or action space while the state of a MAS can be one-dimensional. We denote that a variable refers to agent i with a subscript, s_i, and use superscripts to denote other indexes, such as the time step.

Temporal difference learning Reinforcement learning agents interact with an environment that can be modeled as a Markov Decision Process (MDP). At a given point in time, an agent is in a state s, performs an action a and observes a reward r and next state s'. The environment is characterized by the reward function $R(s, a)$ and the transition probability function $T(s, a, s')$. Agents need to solve two problems. The prediction problem consists in evaluating the expected reward when the agent is in state s and follows policy π thereafter. This information is stored in the value function $V^\pi(s)$. In many learning variants, the agent also computes the action value function $Q^\pi(s, a)$, which contains the expected reward when the agent is in state s, performs action a and follows policy π thereafter. The control problem aims at computing the optimal π^* among all possible policies. Value functions satisfy the following recursive relationship:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s')V(s') \tag{1}$$

$$V(s') = \max_{a' \in \mathcal{A}} Q(s, a') \tag{2}$$

In temporal difference learning, an agent computes the optimal policy in the absence of knowledge of the reward function $R(s, a)$ and transition probability function $T(s, a, s')$. To achieve this, the agent continuously refines its estimation of the Q-function by interacting with the environment. The update equation for Q-values is:

$$Q^\pi(s, a) = Q^\pi(s, a) + \alpha\big(r + \gamma V^T(s') - Q^\pi(s, a)\big) \tag{3}$$

where α is the learning rate, quantifying how quickly the agent forgets information about the past, and γ is the discount factor, quantifying how much the agent discounts future information. The quantity in the parenthesis is the temporal difference error and denotes the difference between our previous estimate of the policy and our improved estimate after the current time step. $V^T(s')$ is the value of the target policy, which differs among learning algorithms. In Q-learning the target policy is equal to $\max_a Q(s, a)$.

Stochastic games Stochastic games are a generalization over Markov Decision Processes (MDPs) and repeated (matrix) games, useful as a learning framework in MAS. At a given point in time, the MAS is in a state $\vec{s}$ that gives rise to a game played between N agents. In our zero-sum setting, we can assume that agents are divided into D defenders and K adversaries. Then, $\pi_D(s)$

is the joint policy followed by all defenders, which is a mapping from the joint state space $\mathcal{S}_D$ to the joint action space $\mathcal{A}_D$.

The value of a state $\vec{s}$ from the perspective of a defender can be computed as:

$$V(\vec{s}) = \max_{\pi_D(s)} \min_{\vec{a}_K} \sum_{\vec{a}_D} Q(\vec{s}, \vec{a}_K, \vec{a}_D)\pi_D(\vec{s}) \tag{4}$$

Similar to single agents in an MDP, agents in a stochastic game can learn the quality of an action $\vec{a}_D$ against an adversarial action $\vec{a}_K$ as:

$$Q(\vec{s}, \vec{a}_K, \vec{a}_D) = R(\vec{s}, \vec{a}_K, \vec{a}_D) + \gamma \sum_{\vec{s'} \in \mathcal{S}} T(\vec{s}, \vec{a}_K, \vec{a}_D, , \vec{s'})V(\vec{s'}) \tag{5}$$

Minimax-Q Minimax-Q is an extension of Q-learning to zero-sum stochastic games, where the policies of the two players are proven to converge to the Nash Equilibrium of the game (Littman, 1994). The target value in this case is as defined in Eq. (4). At each learning iteration, the player chooses an action to execute based on its policy $\pi(s)$ and the exploration scheme, observes the state reward, the transition of the game state and the action of the adversaries a_K, and updates its Q-value function based on its current estimation of the value function. It then uses linear programming to update the policy and value function as follows:

$$\pi(s) = \arg\max_{\pi'_D}(\vec{s}) \min_{\vec{a}'_K} \sum_{\vec{a}'_D} Q(\vec{s}, \vec{a}'_K, \vec{a}'_D)\pi_s(\vec{a}'_D) \tag{6}$$

$$V^{\pi}(\vec{s}) = \min_{\vec{a}'_K} \sum_{\vec{a}'_D} Q(\vec{s}, \vec{a}'_K, \vec{a}'_D)\pi_{\vec{a}'_D}(\vec{s}) \tag{7}$$

4 Robust multi-agent Q-learning

In our game formulation, N agents are defenders, with each agent i performing actions based on the policy defined over its own action space and the system state, i.e. $\pi_i = \pi(a_i|\vec{s})$. We denote the set of possible partitions of the of agents into K adversaries and D defenders as $\mathcal{C}$ and c_K denotes such a partition.

We employ temporal difference learning, as presented in (3), and define the value of the target functions as:

$$V^T(\vec{s}) = \min_{\vec{a}_j} \min_{j \in c_K} \max_{\pi_{-j}} \sum_{\vec{a}_{-j}} Q(\vec{s}, \vec{a}_{-j}, \vec{a}_j)\pi_{-j}(\vec{a}_{-j}|\vec{s}) \tag{8}$$

Our formulation is therefore very similar to minimax-Q, with one notable difference: the type of an agent is not known prior to updating the Q-value function. Instead, agents are attacked adversarially, by picking the partition c_K that will give the minimum Q-value. While minimax-Q is a linear program, our addition of minimizing over all possible partitions renders the calculation of $V^T(s)$, a mixed integer linear program. Solving Eq. (8) exactly can be done by solving a linear program for each possible partition c_K, the number of possible partitions being $(N!)/(K!N!)$. This exhaustive enumeration entails high complexity. Algorithm 1 contains the pseudocode for Rom-Q, which requires as input the set of nodes $\mathcal{N}$, the state and action space $(\mathcal{S}, \mathcal{A})$, the learning hyper-parameters $(\alpha, \gamma, \epsilon)$ and the size of the adversarial attack K.

Algorithm 1: Rom-Q

Data: $\mathcal{N}, \mathcal{S}, \mathcal{A}, \alpha, \gamma, \epsilon, K$
Result: N policies $\pi_i(s)$
for $\vec{s} \in \mathcal{S}, \vec{a} \in \mathcal{A}, i \in \mathcal{N}$ **do**
  Initialize $Q(\vec{s}, \vec{a}), V(\vec{s}), \pi_i(\vec{s})$
end
while *learning has not converged* **do**
  with probability ϵ select random action $\vec{a}$
  otherwise, **for** *node* $i \in \mathcal{N}$ **do**
    select action $\vec{a}_i$ with probability $\pi_i(\vec{s})$
  end
  Perform action $\vec{a}$ and observe reward r, next state $\vec{s}'$
  if s' *is terminal* **then**
    $target = r$
    Sample new initial state s'
  else
    $target = r + \gamma V(\vec{s}')$
  end
  $Q(\vec{s}, \vec{a}) = (1 - \alpha)Q(\vec{s}, \vec{a}) + \alpha(target)$
  Find all possible subsets c_k of K adversaries out of N nodes
  for $c_k \in \mathcal{C}$ **do**
    $\pi_{-c_k}(\vec{s}) = \arg\max_{\pi_{-c_k}} \min_{\vec{a}_{c_k}} \sum_{\vec{a}_{-c_k}} \pi_{-c_k}(\vec{s}) Q(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k})$
    $V_{-c_k}(\vec{s}) = \min_{\vec{a}_{c_k}} \sum_{\vec{a}_{-c_k}} \pi_{-c_k}[s] Q(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k})$
  end
  Choose adversarial subset $\bar{c_k} = \arg\min_{c_k}(V_{-c_k}(\vec{s}))$
  for *node* $i \in \{\mathcal{N} - c_k\}$ **do**
    $\pi_i(\vec{s}) = \pi_{-\bar{c}_k[i]}(\vec{s})$
  end
  $V(\vec{s}) = \min_{\vec{a}_{\bar{c}_k}} \sum_{\vec{a}_{-\bar{c}_k}} \pi_{-\bar{c}_k}(\vec{s}) Q(\vec{s}, \vec{a}_{\bar{c}_k}, \vec{a}_{-\bar{c}_k})$
end

5 Computing multi-agent adversarial attacks

In this section, we describe the process of computing a multi-agent adversarial attack during the evaluation of a learned policy. Adversaries arrive randomly at a time step t with a probability δ and: (i) pick the nodes that they will attack (ii) manipulate the chosen nodes In our definition of this attack, we consider that advsersaries select both nodes and actions adversarially. These two selection procedures steps are not independent: the adversarial selection of victims takes into account the effect of the performed actions, so that the attack brings about the largest decrease in the immediate reward in hindsight, i.e. after the remaining defenders perform actions according to the learned policy.

We consider a cooperative team of N agents, where each agent i has a_i available actions, observes the system state $\vec{s}$ and receives the system reward $r = \sum_i r$. After training, the system has learned the optimal Q-value function $Q^*(\vec{s}, \vec{a})$, where $\vec{a}$ is the joint action space, and individual policies $\pi_i^*(s)$. Adversaries arrive at a random state $\vec{s}$ and consider all possible partitions $\mathcal{C}$ of the N agents into K adversaries and $N - K$ defenders and, for each partition $c_K \in \mathcal{C}$ solve the following problem:

$$\min_{c_K} V_{-c_K}(\vec{s}) = \min_{\vec{a}_{c_K}} \sum_{-c_K} Q^*(\vec{s}, \vec{a}_{-c_k}, \vec{a}_{c_K}) \pi^*_{-c_k}(\vec{s}) \tag{9}$$

This means that adversaries marginalize over the policies of defenders and, then, perform the actions that will incur the minimum expected reward, based on the optimal joint policy. We

denote this quantity as $V_{c_k}(s)$, as it is a type of value function, indicating the expected reward when adversarial agents and defenders perform their actions and the optimal policy is followed thereafter. Finally, adversaries choose the partition $\arg\min_{c_k} u_{c_k}(s)$ and perform their chosen action a_{c_k}. Thus, adversaries learn deterministic attack policies $\sigma(s) = P(c_k, a_{c_k}|s)$, which consist in a mapping from states to node and action selections. We present the pseudocode for finding the optimal adversarial policy $\sigma^*(\vec{s})$ for a given learned policy $\pi^*(\vec{s})$ and Q-value function $Q^*(\vec{s}, \vec{a})$ in Algorithm 2.

Algorithm 2: Computing the optimal policy of a multi-agent adversarial attack.

Data: $\pi^*(\vec{s}), Q^*(\vec{s}, \vec{a}), K$
Result: $\sigma(\vec{s}), \forall \vec{s} \in \mathcal{S}$
Find all possible subsets c_K of K adversaries out of N nodes
for $\vec{s} \in \mathcal{S}$ **do**
  for $c_k \in \mathcal{C}$ **do**
    $V_{c_k}(\vec{s}) = \min_{a_{c_k}} \sum_{-c_k} Q^*(\vec{s}, \vec{a}_{c_k}, \vec{a}_{-c_k}) \pi^*_{-c_k}(\vec{s})$
  end
  Choose adversarial set $\bar{c}_k = \arg\min_{c_k}(V_{c_k}(\vec{s}))$
  $\vec{a}_{\bar{c}_k} = \arg\min_{\vec{a}_{\bar{c}_k}} \sum_{\vec{a}_{-\bar{c}_k}} Q^*(\vec{s}, \vec{a}_{\bar{c}_k}, \vec{a}_{-\bar{c}_k}) \pi^*_{-c_k}(\vec{s})$
  $\sigma(\vec{s}) = [\bar{c}_k, \vec{a}_{\bar{c}_k}]$
end

6 Experiments and results

In this section we evaluate the ability of RoM-Q to find multi-agent policies that are robust to worst-case adversarial attacks of a fixed size. We experiment with a load balancing problem for a toy network characterized by a critical overflow area. We compare the performance of policies learned by a multi-agent system using RoM-Q to the performance of policies learned by Q-learning and minimax-Q, where evaluation takes into account the ability of these methods to get optimal rewards in the absence of attacks, as well as their ability to perform well when attacks take place during evaluation.

Load balancing Our toy network consists of two inter-connected nodes, as presented in Fig. 1. Each node i is modeled by its capacity c_i, i.e. the maximum number of tasks it can hold, and the probability of arrival of a new task in a given time step, p_i. The number of tasks currently stored in a node define its state and the system state consists of the states of all nodes. Nodes are capable of executing a task using the action a_i^{e} and off-loading a task using the action a_i^{o}. Executing a task incurs cost c_i^{e}, while off-loading a task from node i to node j incurs a cost $e_{i \to j}$, equal to the weight of the edge connecting the two nodes. Finally, over-flows incur a very large cost c_i^{h}. In addition to these costs received per time step, nodes also receive a reward u_i if the node has not over-flown in the current time step. This toy problem can be viewed as a multi-agent variant of the classical cliff walking problem, where the cliff region is determined by the capacities of the nodes.

Training consists of S_{train} learning samples. If one of the nodes over-flows, the system is reset to a zero state for all nodes. At the end of the training process, we find an optimal policy $\pi_i^*(\vec{s})$ for each node i. Policies are defined over the joint state space and the action space of that node, which is the cartesian product of its two actions, i.e. $\pi_i(s) : \mathcal{S} \to \mathcal{A}_i^e \times \mathcal{A}_i^o$. No attacks take place during training.

Evaluation requires the definition of the adversarial policy, which is used to perform attacks during S_{eval} evaluation steps. At each step, a greedy multi-agent policy $\pi^* \vec{s}$ competes against an adversarial policy $\sigma^*(\vec{s})$. Adversaries arrive randomly with a probability δ in a given evaluation

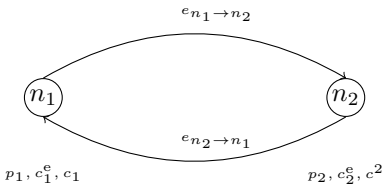

Figure 1: Schematic of a toy network consisting of two nodes.

Parameter	Value
training samples S_{train}	500000
evaluation samples S_{eval}	50000
trials I	10
learning rate α	0.01
exploration rate ϵ	0.1
discount factor γ	0.9

Table 1: Learning hyper-parameters

Parameter	Value
utility u	[8, 8]
overflow cost c_h	[100,100]
capacity c	[3,3]
arrival rate p	[0.9, 0.7]
execution cost c^e	[5,1]
off-loading cost $e_{i \to j}$	[3,3]

Table 2: Node modeling parameters

step. We generate a pool of adversarial policies for evaluation: $\{\sigma_{\text{Q}}^*(s), \sigma_{\text{M}}^*(s), \sigma_{\text{R}}^*(s)\}$. These are the adversarial policies found based on the optimal policies learned using Q-learning, minimax-Q and Rom-Q, respectively. In addition to evaluating the optimal policy learned by each method, we also save intermediate policies during training to examine how training converges to robust solutions. Finally, we compute confidence intervals at a 90% confidence level by performing I independent training and evaluation trials, where each trial has a different seed, fixed at the beginning of the experiment. We present all hyper-parameters associated with learning in Table 1 and all parameters related to the modeling of the nodes and system in Table 2.

6.1 Optimality analysis

We now analyze the different learning methods in terms of their ability to find policies with optimal rewards in the absence of attacks. We also visualize these optimal policies, the state visits when following them during evaluation, and their optimal adversarial policy, computed as was previously described in Section 5.

The heatmaps in the first row of Fig. 2 illustrate the number of visits per state using color, as well as the optimal policies of the two nodes, using arrows. There are two arrows starting from each square of the state space: the green arrow visualizes the actions of n_1 and the blue arrow the actions of n_2. When node i executes a task, i.e. $a_i^e = 1$, the arrow points to a decrease in the value of its state (leftwards for n_1 and downwards for n_2). Similarly, when node i off-loads a task, i.e. $a_i^o = 1$, the arrow points to an increase in the value of the other node's state (upwards for n_1 and rightwards for n_2). For states that have been visited by virtually contain no actions, the optimal action is $\vec{a} = [a_1^e = 0, a_1^o = 0, a_2^e = 0, a_2^o = 0]$. The lower row presents the respective optimal adversarial policies σ_{Q}^*, σ_{M}^* and σ_{R}^*, where, in each state, the actions of the node controlled by the adversary are shown.

We observe that none of the three methods over-flows in the absence of attacks. Q-learning learns policies that keep the two nodes close to capacity, as nodes prefer to remain inactive when they have a small number of tasks, in order to avoid costs associated with the off-loading or execution of tasks. The adversarial policy chooses to attack node 1 when its load is low, while node 2 is attacked when node 1 has a high load. Thus, adversaries following σ_{Q}^* attack nodes in order to manipulate them into over-flowing the other node. We can also observe that adversaries sometimes choose to execute tasks: although this appears to come in contrast to the adversary's objective of over-flowing the system, it incurs an additional cost, and is therefore worst-case for an one-step attacker. Minimax-Q, on the other hand, learns more conservative policies, restricting nodes to visiting states with low and intermediate node values. The optimal adversary for minimax-Q, σ_{M}^* again appears to have the objective of over-flowing a node by attacking the other node

and transmitting packets, but differs from σ^*_{Q} in its adversarial choice of nodes. Finally, RoM-Q learns a policy where most state visits have high loads for node 2 and low load for node 1. The optimal adversary in this case, σ^*_{Q}, always chooses to attack node 2 in order to over-flow node 1. By inspecting Table 2, we can see that n_1 has a high arrival rate of tasks and a high cost for executing tasks, which explains why it opts for always off-loading tasks to n_2.

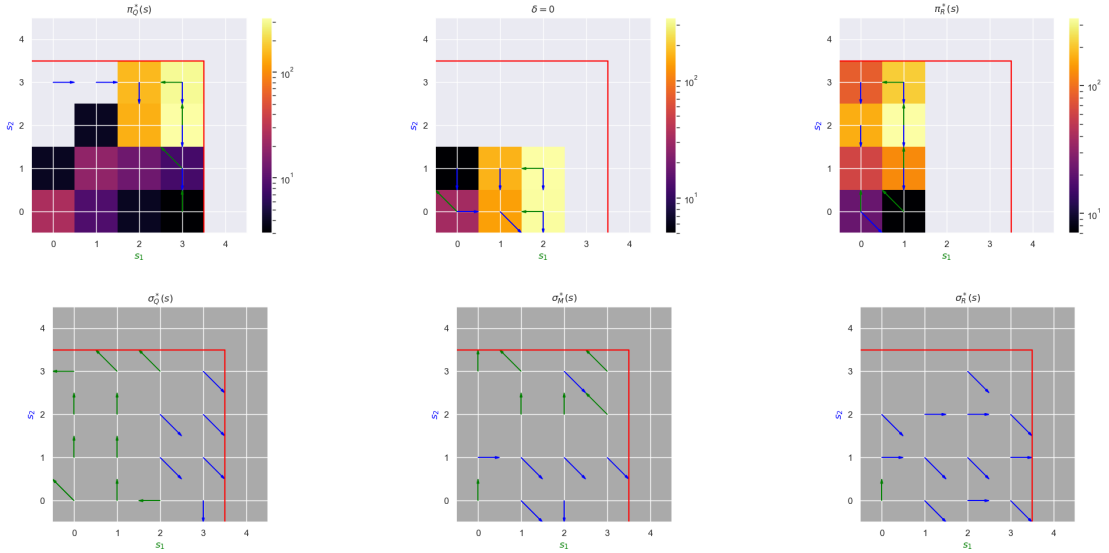

Figure 2: Heatmap of state visits and actions of the optimal policy (upper row) and the adversarial policies (lower row) for Q-learning, minimax-Q and RoM-Q (from left to right).

To get a better understanding of the learning process, we visualize the improvement in the performance of policies during training time. Figure 3 presents the evolution of rewards, averaged over I trials, with training time measured in samples of experience. Performance is measured as the average system reward per evaluation sample. In addition, Figure 4 presents the evolution of the duration of episodes, in order to visualize the ability of the policies to avoid over-flow during the course of an evaluation episode. Note that by episode, we mean the period between two system restarts, which occur either when a node over-flows or when 50 consecutive samples have been observed without over-flow. By inspecting these figures we observe that all three methods converge to similar values and at a similar speed.

6.2 Behavior under attacks

In this section we visualize the behavior of the learned policies under different probabilities of attack taking place during evaluation. We restrict ourselves to the case where adversaries follow the policies that are optimal to the method being evaluated. In other words, Q-learning competes against $\sigma^*_Q(s)$, minimax-Q competes against $\sigma^*_M(s)$ and Rom-Q competes against $\sigma^*_R(s)$. Note that the actions presented in this sections are the ones under the optimal policy, and differ from the actions executed in reality when an adversary is controlling a node.

In Fig. 5, we observe that Q-learning over-flows often when being under attack. This is because its optimal policy keeps both nodes close to capacity. Similarly, state visits under minimax-Q visit the over-flow area, but with smaller frequency. In contrast, Rom-Q remains in the safe region regardless of the frequency of attacks. This is intuitive: the operator used during training is considering attacks of the same type taking place during evaluation. Thus, evaluation with $\delta = 1$ is identical to the environment "imagined" by the agent during training. What is more interesting however, is that Rom-Q remains robust under any probability of attack δ, as we can see in the lower part of 5, which suggests that it is robust to uncertainty in the probability of attack.

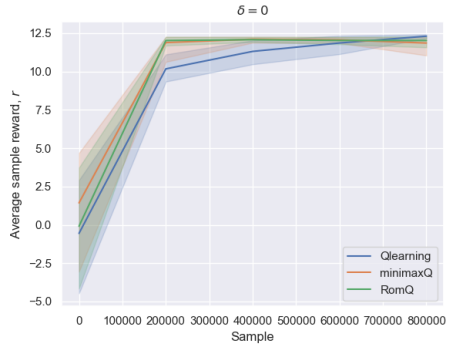

Figure 3: Evolution of rewards for Q-learning, minimax-Q and RoM-Q. Rewards are calculated during evaluation episodes. The optimal reward for the toy network is 655.

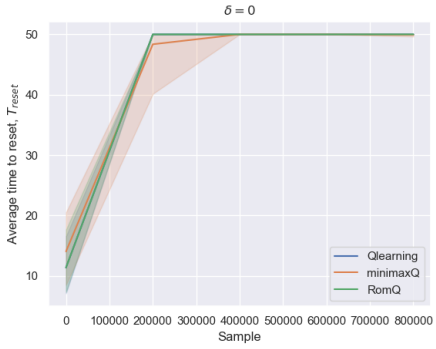

Figure 4: Evolution of the duration of evaluation episodes for Q-learning, minimax-Q and RoM-Q. The optimal episode duration for the toy network is 50.

6.3 Robustness analysis

We now evaluate the performance of policies learned by Rom-Q when attacks based on different adversarial policies take place during evaluation. Adversarial policies for evaluation are sampled from the following pool: $\{\sigma^*_{\mathrm{Q}}(s), \sigma^*_{\mathrm{M}}(s), \sigma^*_{\mathrm{R}}(s)$. To put robustness of policies learned using Rom-Q into perspective, we contrast their performance to the one achieved by policies using Q-learning and minimax-Q. Note that, to produce Fig. 6, both defenders and adversarial actions are chosen deterministically, by choosing the greedy action based on the agents' Q-tables. In simulations that will follow up, minimax-Q and Rom-Q will also be evaluated when optimal and adversarial policies are probabilistic.

We can derive various interesting observations by closely inspecting Fig. 6. First, the performance of Q-learning drops drastically for $\delta > 0.1$ for any type of attack. Second, minimax-Q and Rom-Q exhibit better robustness to attacks, with Rom-Q having the best performance among all, for all types of attacks considered. Third, in the absence of attacks during evaluation, minimax-Q and Rom-Q achieve slightly lower rewards than Q-learning, due to the fact that their optimal policies are more conservative and nodes execute tasks often in order to stay far from the over-flow area.

7 Discussion and conclusions

We presented a robust operator for multi-agent reinforcement learning that leads to policies safe in the presence of adversarial attacks taking place during deployments. Attacks in our framework occur with arbitrary probability and consist of a pre-determined number of adversaries that choose both nodes and their actions adversarially. We demonstrated through simulations on a toy network that taking into account the different vulnerabilities of nodes comprising the MAS is important when adversaries choose their victims adversarially. Our simulations also clearly illustrate some limitations of classical learning algorithm when it comes to robustness: Q-learning, being oblivious to potential attacks during deployment, converges to policies that are optimal and very close to the over-flow area, even though policies with similar expected rewards exist. On the other hand, minimax-Q finds policies that are overly conservative, assuming that 2 adversaries are present in the system and therefore robust to various attacks. In addition to settling for policies with lower rewards, minimax-Q does not lead to robustness to various probabilities of attacks and number of attackers, especially when attacker choose nodes adversarially.

A limitation of Rom-Q is the complexity of solving at each learning iteration a number of linear programs that scales with the number of possible combinations of K adversaries out of N agents. Although policies are learned off-line, and can thus profit from simulation environments rich in resources, it would be useful to find a variant of Rom-Q with reduced complexity. If one considers

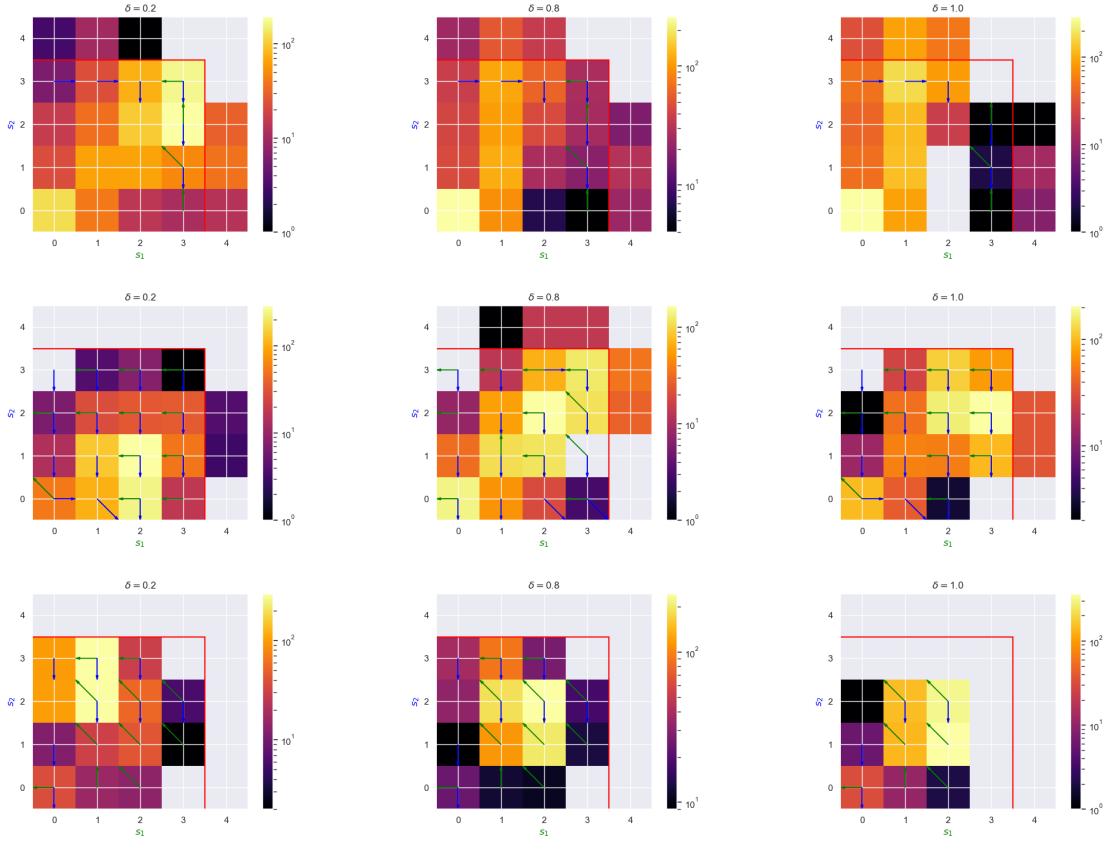

Figure 5: State visits and performed actions for different probabilities of attack for Q-learning (first row), minimax-Q (second row), Rom-Q (third row).

that the learning update in Rom-Q involves a mixed integer linear program, a promising direction would be to express it in a standard form that allows for approximate solutions and comes with optimality guarantees. We believe that knapsack problems are an appropriate optimization for our formulation, where we can map the decision of viewing a node as an adversary or defender as an insertion to one of two knapsacks, the value of which is associated with the Q-value function for a considered adversarial partition.

Finally, examining the behavior of Rom-Q policies in more complex networks, where nodes are organized into clusters and bottlenecks in the network can bring cascading failures, can make its contribution to robustness more prominent.

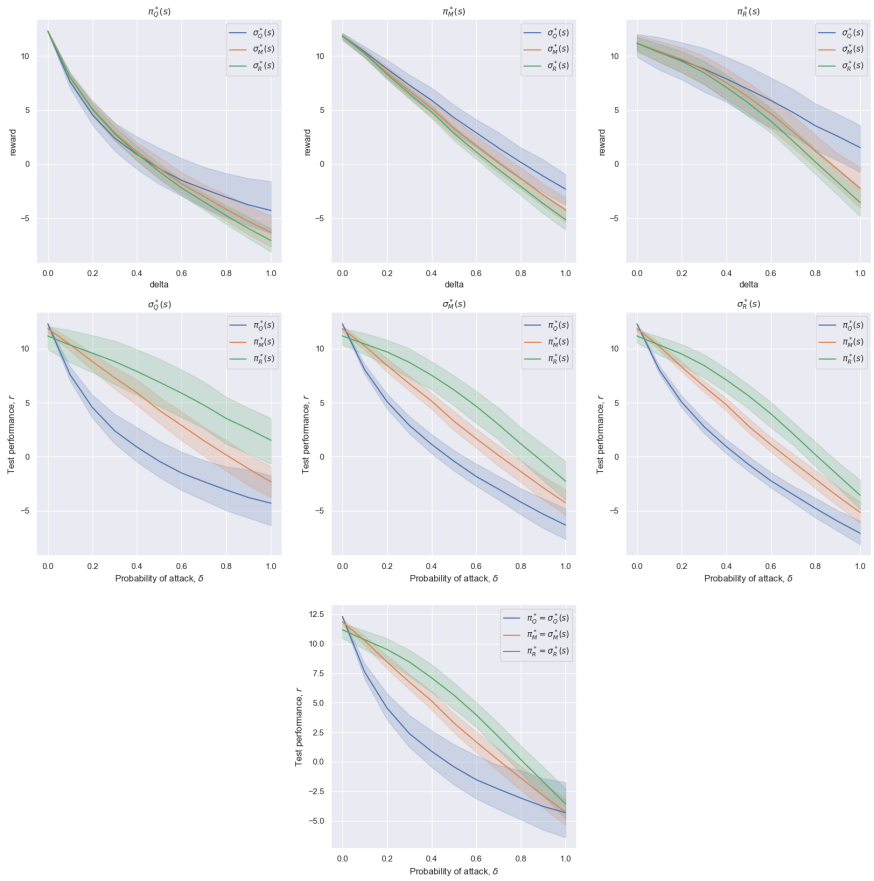

Figure 6: Performance of Q-learning, minimax-Q and Rom-Q evaluated against $\sigma^*_{\mathrm{Q}}(s), \sigma^*_{\mathrm{M}}(s), \sigma^*_{\mathrm{R}}(s)$. In the first row, one method competes against all three possible adversarial policies in each sub-plot. In the second row, all three methods compete against a different adversarial policy in each sub-plot. In the last row, each methods competes against the adversarial policy computed by attacking its own optimal Q-value function.

References

Abbasi Yadkori, Y., Bartlett, P. L., Kanade, V., Seldin, Y., and Szepesvari, C. (2013). Online learning in markov decision processes with adversarially chosen transition probability distributions. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2508–2516. Curran Associates, Inc.

Abdullah, M., Ren, H., Bou-Ammar, H., Milenkovic, V., Luo, R., Zhang, M., and Wang, J. (2019). Wasserstein robust reinforcement learning. *ArXiv*, abs/1907.13196.

García, J., Fern, and o Fernández (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480.

Johanson, M., Zinkevich, M., and Bowling, M. (2008). Computing robust counter-strategies. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 721–728. Curran Associates, Inc.

Klíma, R., Bloembergen, D., Kaisers, M., and Tuyls, K. (2019). Robust temporal difference learning for critical domains. *CoRR*, abs/1901.08021.

Lecarpentier, E. and Rachelson, E. (2019). Non-stationary markov decision processes a worst-case approach using model-based reinforcement learning. In *NeurIPS*.

Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., and Russell, S. (2019). Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4213–4220.

Lin, J., Dzeparoska, K., Zhang, S. Q., Leon-Garcia, A., and Papernot, N. (2020). On the robustness of cooperative multi-agent reinforcement learning. *ArXiv*, abs/2003.03722.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann.

Morimoto, J. and Doya, K. (2005). Robust reinforcement learning. *Neural Comput.*, 17(2):335–359.

von Neumann, J. and Morgenstern, O. (1947). *Theory of games and economic behavior*. Princeton University Press.

Zhou, K., Doyle, J. C., and Glover, K. (1996). *Robust and Optimal Control*. Prentice-Hall, Inc., USA.