Meta Strategy Guided Deep Reinforcement Learning in Green Security Games

Tianyu Gu

CMU-CS-20-114

May 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Fei Fang (Chair)
Zico Kolter

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Keywords: Security Game, Reinforcement Learning

Abstract

While multi-agent reinforcement learning algorithms have attracted many research interests, very few algorithms in the field were deployed in real-world scenarios due to their uninterpretablilty and sample inefficiency in the training process. In this work, we propose an algorithm to use meta-strategy as regulators to train multi-agent deep reinforcement learning agents to account for these challenges. We also propose several approaches to solve for meta-strategies, including linear program based approaches and shortest cycle based approaches. Through experiments, we discuss the effectiveness of incorporating meta-strategy in reinforcement learning.

Acknowledgments

I would like to thank Professor Fei Fang for her great support on this project. Professor Fang is a great advisor and mentor. She has been super supportive during many hard times and has provided many valuable advice to guide my learning. Without her support, this project would not been able to complete

I would like to thank Professor Zico Kolter for being my committee member. Professor Kolter provided many guidance on this project and I learned many techniques from his Graduate Artificial Intelligence class, which greatly helped this project.

Contents

1 Introduction **1**

2 Preliminaries and Related Work **3**
2.1 Stackelberg Security Games 3
2.2 Patrolling Security Games 3
2.3 Deep Reinforcement Learning and Multi-Agent Reinforcement Learning 3
2.4 KL-Divergence 4

3 Green Security Game with Continuous Space **7**

4 Solving Meta-Strategy in Discrete Space **9**
4.1 Solving for Markovian Patrolling Strategies 9
4.1.1 Abstracted Game Setting 9
4.1.2 Linear Program Formulation with Implicit Stationary Equations 9
4.1.3 Linear Program Formulation with Explicit Stationary Equations 10
4.1.4 Markovian Strategy Non Linear Program Formulation 10
4.1.5 Comparison of Implicit Stationary Equation and Explicit Stationary Formulations 12
4.2 Solving for Shortest Cycle Based Strategies 12

5 Meta Strategy Guided Reinforcement Learning **15**
5.1 KL-Divergence regularized reinforcement learning 15
5.1.1 Q-function update for defender 15
5.1.2 Policy update for defender 16
5.1.3 Complete Algorithm 16

6 Experiments **19**

7 Disscussions and Future Work **23**

Bibliography **25**

List of Figures

6.1 Learning curve 20
6.2 Solutions of meta-strategies 21
6.3 Example runs of trained defenders 22

List of Tables

6.1 Environment setting 20
6.2 Mean and standard deviation of trained defender's utility against each attacker . . 21

Chapter 1

Introduction

Stackelberg security game is a game model in which the leader first commits to a strategy and the follower then observes the leader's policy before committing to his own strategy. Stackelberg security game models have been successfully deployed in many real-world settings to protect important infrastructure. [12] helps schedule patrolling and monitoring for the Los Angeles International Airport (LAX). [1] is used by the United States Coast Guard (USCG) to combat terrorism. [6] is deployed to protect wildlife in conservation sites. Outside of guarding physical infrastructure, game theoretic models have also been used to protect virtual targets. [13] developed a browser extension to combat social engineering attacks.

Despite Stackelberg security game models' success , deep reinforcement learning algorithms have attracted an increasing amount of interests because these algorithms do not require domain knowledge and are applicable in more than one scenarios. [8] developed an actor-critic, model-free algorithm based on the deterministic policy gradient. [7] builds upon [8] to maximize the agent's policy's entropy. [9] extends the line of work to incorporate coordination and competition of multiple agents.

While deep reinforcement learning algorithms have attracted many research interests, very few real-world applications have been built based upon these algorithms because of their uninterpretablility and sample inefficiency during the training process. In this work, we propose an algorithm to use meta-strategy in training deep reinforcement learning agents to increase interpretability and sample efficiency. We also propose several different approaches to solve for meta-strategies, including linear program based approaches and shortest cycel based approaches.

The algorithm follows two steps. In the first step, we solve for a meta-strategy. We then use the meta-strategy as a regulator to train the reinforcement learning algorithm based on [7, 9] in the second step.

Chapter 2

Preliminaries and Related Work

2.1 Stackelberg Security Games

Security games are characterized by two players, a defender and an attacker, and a set of targets that the defender tries to protect from the attacker [5, 15]. Solving security games relies on the concept of Stackelberg equilibrium in which the defender first commits to a strategy that the attacker can observe. A pair of strategies of the defender and the attacker is in Stackelberg equilibrium if the attacker's strategy is a best response to the defender's strategy and the defender's strategy maximizes his utility when best responded by the attacker. When the game is zero-sum, the solution concept Stackelberg equilibrium coincide with Nash equilibrium.

2.2 Patrolling Security Games

Patrolling security games are proposed as extensions to the normal form security games. A patrolling security game is a two-player multi-stage game with infinite horizon, such that the defender moves a single resource along the edges of an arbitrary graph to protect certain targets and the attacker intrudes the environment by placing a resource on a selected target node [2]. Alarms which inform the defender the presence of the attacker at some location are then incorporated in the patrolling security game model [11]. [3] extends the alarm patrolling security games by considering spatially uncertain alarm signals, which is able to detect an attack but is uncertain on the exact location of the attacker. Recent progress in alarm patrolling security games also considers false negatives of alarm detection. Other than alarms, latest progress in patrolling security games also incorporates deep reinforcement learning to account for real-time information such as footprints [17].

2.3 Deep Reinforcement Learning and Multi-Agent Reinforcement Learning

A reinforcement learning problem is usually formulated as a Markov Decision Process, with the transition probability P, the state space S, the action space A, the discount factor γ, and the

reward function r. The goal of reinforcement learning algorithm is to generate a policy π from state s to action a for the player to take to maximize the expected total reward $\mathbb{E}_{\pi,P}[\sum_{t=0}^{\infty}\gamma^t R_t]$. Q-learning is one of the early breakthroughs in reinforcement learning [14]. The Q term refers to the value of a state-action pair (s,a) under policy π and is defined to be

$$Q^{\pi}(s_t,a_t)=\mathbb{E}_{\pi,P}[\sum_{n=0}^{\infty}\gamma^n r(s_{t+n},a_{t+n})]$$

Another breakthrough in reinforcement learning is the introduction of using deep neural network as approximation to the Q function [10]. Deep Q-Network learns the optimal policy by minimizing the loss.

$$\mathcal{L}(\theta)=\mathbb{E}_{s,a,s'}[(Q^{\theta}(s,a)-(r+\gamma\max_{a'}Q^{\tilde{\theta}}(s',a')))^2]$$

such that the Q is the target network used for training stabilization. The line of work of Q-learning aims to generate an accurate Q-function, and take the action with the highest state-action pair.

Another line of work called policy gradient methods are able to learn parameterized policy directly without consulting approximate value functions. Recent breakthroughs in reinforcement learning often involve actor-critic method which learns the policy and the value functions at the same time. Actor refers to the learned policy and critic refers to the learned value function [14]. The algorithm Deep Deterministic Policy Gradient (DDPG) follows the actor-critic method and learns the best policy by solving

$$\max_{\theta}\mathbb{E}_s[Q_{\theta}(s,\mu_{\phi}(s))]$$

such that Q_θ is the value function that is learned in a similar manner as Q-learning and μ_ϕ is the policy function. An extension to the DDPG algorithm is called the Soft Actor Critic (SAC) algorithm [7]. SAC generate policies that maximize the expected future reward and the entropy of the learned policy by solving

$$\max_{\theta}\mathbb{E}_s[Q_{\theta}(s,\pi_{\phi}(s))+\alpha H(\cdot|\pi_{\theta})]$$

$H(\cdot|\pi_\theta)$ represents the policy's entropy.

In the multi-agent reinforcement learning domain, one of the biggest breakthrough is the introduction of Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm. MADDPG considers action policies of other agents and is able to successfully learn policies that require complex multi-agent interaction [9]. MADDPG learns policy μ_i for agent i by updating the gradient in which J is the expected utility and o_i is agent i's observation.

$$\nabla_{\theta_i}J(\mu_i)=\mathbb{E}_{s,a}[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i}Q_i^{\mu}(s,a_1,\cdots,a_N)|_{a_i}=\mu_i(o_i)]$$

2.4 KL-Divergence

The Kullback-Leibler divergence (KL-Divergence) is a metrics used to measure similarity of two probability distributions. For distributions P and Q of a continuous random variable, the

KL-Divergence is defined to be

$$\begin{aligned} D_{KL}(P\|Q) &= \mathbb{E}_P\left[\frac{P}{Q}\right] \\ &= \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right)dx \end{aligned}$$

Chapter 3

Green Security Game with Continuous Space

The game takes place in a continuous space with discrete time steps. There are two players in the game, the defender and the attacker. The defender controls one resource, the patroller and the attacker controls one resource, the poacher. All agents move simultaneously with continuous displacement. There are N targets in the game that are susceptible to be attacked by the attacker. The attack is completed when the attacker moves close enough to one of the targets for penetration time k time steps. The attacker is stealthy when not penetrating a target such that even if the defender and the attacker meet in the continuous space, the defender cannot catch the attacker. The attacker can only be caught at the target that she is attacking. The game ends when the attacker is caught or an attack is completed. When the game starts, both the defender and the attacker start from random initial locations. If the attacker attacks target n without being caught, the attacker gains utility $U_a^u(n) > 0$, the defender gains utility $U_d^u(n) < 0$. If the attacker is caught by the patroller, the attacker gains utility $U_a^c < 0$, and the defender gains utility $U_d^c > 0$.

Chapter 4

Solving Meta-Strategy in Discrete Space

In this chapter, we present several different approaches to solve for meta-strategies including Markovian patrolling strategies and shortest cycle based strategies.

4.1 Solving for Markovian Patrolling Strategies

In this section, we abstract the continuous space game into a discrete space game. We then use various linear program based approaches to solve the abstracted game. The solved meta-strategies are Markovian patrolling strategies.

4.1.1 Abstracted Game Setting

In the abstract game, the continuous space is abstracted as an undirected graph $G = (E, V)$. The defender moves discretely along the edges and the attacker becomes a one-shot attacker, which only chooses a node to land and attack. The game becomes zero-sum. The Attacker is able to observe the defender's strategy, but not able to observe the defender's exact location on the graph. Assume when the attacker is attacking, the defender has already patrolled long enough so that the probability that the defender is in each node follows the stationary distribution.

4.1.2 Linear Program Formulation with Implicit Stationary Equations

In this section, we present the linear program formulation with implicit stationary equations. Variable v represents the maximum expected payoff for the attacker. x_{ij} represent the flow of the defender moving to node j given he is currently in node i. A_{ij} is a constant which equals to 1 if there exists an edge between node i and node j and 0 otherwise. μ_i is the probability that defender is in node i.

$$\min_{x,v} v \tag{4.1}$$

$$\mu_i = \sum_j x_{ji} \qquad \forall i \tag{4.2}$$

$$\sum_j x_{ij} = \sum_k x_{ki} \qquad \forall i,j,k \tag{4.3}$$

$$x_{ij} \leq A_{ij} \qquad \forall i,j \tag{4.4}$$

$$\sum_i \sum_j x_{ij} = 1 \qquad \forall i,j \tag{4.5}$$

$$v \geq (1-\mu_j)U_a^u(j) + \mu_j U_a^c(j) \qquad \forall j \tag{4.6}$$

Constraint (4.2) states that the probability of defender in node i is equal to the sum of incoming flow into node i. Constrain (4.3) ensures that the defender is only moving along the edges. Constraint (4.4) ensures that μ is a valid probability distribution. Constraint (4.5) ensures that the attacker is best responding to the defender's patrolling strategy.

4.1.3 Linear Program Formulation with Explicit Stationary Equations

In this section, we present the linear program formulation with explicit stationary equations.

4.1.4 Markovian Strategy Non Linear Program Formulation

The non-linear program formulation to solve for the optimal Markovian defender strategy is listed below. v represents the maximum expected payoff for the attacker. π_{ij} represent the probability of the defender moving to node j given he is currently in node i. μ_i is the stationary probability that defender is in node i.

$$\min_{\mu,\pi,v} v \tag{4.7}$$

$$\text{s.t. } \pi_{ij} \geq 0 \qquad \forall i,j \tag{4.8}$$

$$\sum_j \pi_{ij} = 1 \qquad \forall i \tag{4.9}$$

$$\pi_{ij} \leq A_{ij} \qquad \forall i,j \tag{4.10}$$

$$\mu_i = \sum_j \pi_{ji}\mu_j \qquad \forall i \tag{4.11}$$

$$\sum_i \mu_i = 1 \qquad \forall i \tag{4.12}$$

$$\mu_i \in [0,1] \qquad \forall i \tag{4.13}$$

$$v \geq (1-\mu_j)U_a^u(j) + \mu_j U_a^c(j) \qquad \forall j \tag{4.14}$$

Since the abstracted game is zero sum, minimizing the expected attacker payoff in equation (1) is the same as minimizing defender payoff. equation (4.7) and (4.8) ensures that the transition probability π_{ij} is a valid probability distribution. Equation (4.9) ensures that the defender can only travel along the edges. Equation (4.10), (4.11) and (4.12) are the stationary equations to compute the defender's stationary probability μ_i. Equation (4.13) ensures that the attacker is maximizing her expected utility.

Converting to Mixed Integer Linear Programming

To remove the non-convex constraints in the above formulations, we apply a technique used in [16] to discretize variables and converting the program to a Mixed Integer Linear Program. The solution will be approximate solutions to the original formulation. Let variables $p_l \in [0, 1]$ be the discrete levels with $p_0 = 0$ and $p_L = 1$, define $d_{ijl} \in \{0, 1\}$ to be binary variables such that d_{ijl} indicates a particular discrete probability choice p_l Let $\pi_{ij} = \sum_l p_l d_{ijl}$ Replace π_{ij} in the original formulation and define $w_{ijl} = d_{ijl}\mu_i$. Then adding the following constraints completes the MILP.

$$\mu_i - Z(1 - d_{ijl}) \leq w_{ijl} \leq \mu_i + z(1 - d_{ijl}) \qquad \forall i, j \qquad (4.15)$$

$$-Z d_{ijl} \leq w_{ijl} \leq Z d_{ijl} \qquad \forall i, j \qquad (4.16)$$

Complete MILP formulation

We are now ready to present the complete mixed integer linear program for the abstract game for Markovian patrolling strategy.

Below is the complete formulation for the single type attacker case.

$$\min_{\mu,\pi,v} v \tag{4.17}$$

$$\text{s.t. } d_{ijl} \in \{0,1\} \qquad \forall i,j,l \tag{4.18}$$

$$\sum_j \sum_l p_l d_{ijl} = 1 \qquad \forall i \tag{4.19}$$

$$\sum_l d_{ijl} = 1 \qquad \forall i,j \tag{4.20}$$

$$\sum_l p_l d_{ijl} \leq A_{ij} \qquad \forall i,j \tag{4.21}$$

$$\mu_j = \sum_i \sum_l p_l w_{ijl} \qquad \forall j \tag{4.22}$$

$$\sum_i \mu_i = 1 \qquad \forall i \tag{4.23}$$

$$\mu_i \in [0,1] \qquad \forall i \tag{4.24}$$

$$\mu_i - Z(1-d_{ijl}) \leq w_{ijl} \leq \mu_i + z(1-d_{ijl}) \qquad \forall i,j \tag{4.25}$$

$$-Zd_{ijl} \leq w_{ijl} \leq Zd_{ijl} \qquad \forall i,j \tag{4.26}$$

$$v \geq (1-\mu_j)U_a^u(j) + \mu_j U_a^c(j) \qquad \forall j \tag{4.27}$$

4.1.5 Comparison of Implicit Stationary Equation and Explicit Stationary Formulations

The size of the linear program of the implicit stationary formulation is much smaller than that of the explicit stationary formulation. Therefore, the implicit stationary equation linear program formulation runs faster. However, the abstract policy solved from implicit stationary equation Formulation could result to multiple edges with flow 0 in the case of sparse targets, leading to a weak guide for the reinforcement learning training. To provide stronger guides, for each node with 0 outgoing flow, we pick the outgoing edge that leads the defender to the closest node with non-zero outgoing flow. The explicit stationary equation formulation generates a valid action probability distribution for each node on the graph, but the generated policy is an approximation to the optimal Markovian patrolling strategy. Generating a better approximation would result in larger program and cost more computation time. Since the generated defender's stationary distribution is similar in both settings, we use the implicit stationary equation linear program formulation in later experiments because of its low running time.

4.2 Solving for Shortest Cycle Based Strategies

In this section, we do not consider a game setting. Instead, we use search algorithms to directly solve for defender's patrolling strategies. The continuous space is discretized into a graph $G = (E, V)$. A subset V_r of the nodes represent the set of target nodes. We use Dijkstra's algorithm

[4] to find the shortest paths from one node $v_i \in V_r$ to another node $v_j \in V_r$ such that covers all the nodes in V_r. We can then compute the shortest cycle c^* covering the set V_r. The cycle c^* becomes part of the defender's patrolling strategy. For node $v_n \notin V_r$, we compute the shortest path from v_n to c^*, the defender would then would follow the cycle to patrol. Note when using this patrolling strategy, the defender considers not just his current location. Therefore, this is a non-Markovian patrolling strategy.

Chapter 5

Meta Strategy Guided Reinforcement Learning

5.1 KL-Divergence regularized reinforcement learning

Let ψ denote the abstract policy solved using MILP and α denote the regularization coefficient. Then the RL problem we are trying to solve becomes

$$\pi^* = \arg\max_{\pi} E_{\pi}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha D_{KL}(\pi(\cdot|s_t) \| \psi(\cdot|s_t)))]$$

The Bellman equation for Q^π becomes

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_{s'}\left[\mathbb{E}_{a'}\left[R(s,a,s') + \gamma(Q^\pi(s',a') + \alpha D_{KL}(\pi(\cdot|s') \| \psi(\cdot|s')))\right]\right] \\ &= \mathbb{E}_{s'}\left[\mathbb{E}_{a'}\left[R(s,a,s') + \gamma(Q^\pi(s',a') + \alpha\, \mathbb{E}_{a'}\left[\log \frac{\pi(a'|s')}{\psi(a'|s')}\right])\right]\right] \\ &= \mathbb{E}_{s'}\left[\mathbb{E}_{a'}\left[R(s,a,s') + \gamma(Q^\pi(s',a') + \alpha\left(\log \frac{\pi(a'|s')}{\psi(a'|s')}\right))\right]\right] \end{aligned}$$

5.1.1 Q-function update for defender

Since the above Q function is an expectation from the environment dynamics and the policy, it can be approximated with samples.

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_{s'}\left[\mathbb{E}_{a'}\left[R(s,a,s') + \gamma(Q^\pi(s',a') + \alpha\left(\log \frac{\pi(a'|s')}{\psi(a'|s')}\right))\right]\right] \\ &\approx r + \gamma\left(Q_\phi^{\text{targ}}(s',\tilde{a}') + \alpha\left(\log \frac{\pi(\tilde{a}'|s')}{\psi(\tilde{a}'|s')}\right)\right) \text{ s.t. } \tilde{a}' \sim \pi(\cdot|s') \end{aligned}$$

Suppose we have a random batch of transitions $B = (s, a, r, s', d)$ from the replay buffer. We first compute targets for the Q function

$$y = r + \gamma(Q_\phi^{targ}(s', \tilde{a}') + \alpha \log \frac{\pi_\theta(\tilde{a}'|s')}{\psi(\tilde{a}'|s')})$$

note $\tilde{a}' \sim \pi_\theta(\cdot|s')$, is sampled from the current policy. we then update Q-function by one step of gradient descent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (Q_\theta(s', \tilde{a}') - y)$$

5.1.2 Policy update for defender

We update the policy to maximize the expected future return plus the expected future KL-Divergence.

$$E_{a \sim \pi_\theta}[Q(s, a) + \alpha \log \frac{\pi_\theta(\tilde{a}|s)}{\psi(\tilde{a}|s)}]$$

Using the reparameterization trick, this term is also equal to

$$E_{\xi \sim N}[Q(s, \tilde{a}_\theta(s, \xi)) + \alpha \log \frac{\pi_\theta(\tilde{a}_\theta(s, \xi)|s)}{\psi(\tilde{a}_\theta(s, \xi)|s)}]$$

Thus, to maximize this term, we apply gradient ascent using the gradient

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} [Q(s, \tilde{a}_\theta(s, \xi)) + \alpha \log \frac{\pi_\theta(\tilde{a}_\theta(s, \xi)|s)}{\psi(\tilde{a}_\theta(s, \xi)|s)}]$$

5.1.3 Complete Algorithm

In this section we present the complete algorithm. Algorithm 1 presents the whole training procedure. Algorithm 1 is a standard training procedure for multi-agent reinforcement learning settings. We loop until convergence and in each loop, each agent observes the current state, executes action and gets reward and next state from the environment. Each agent will then train parameters accordingly.

Algorithm 1 Training procedure

Initialize policy parameters θ^i, Q-function parameters ϕ_1^i, ϕ_2^i, replay buffer D_i for each agent i
Set target parameters equal to main parameters $\phi^i_{\text{targ},1} \leftarrow \phi^i_1$, $\phi^i_{\text{targ},2} \leftarrow \phi^i_2$
repeat
  for each agent i **do**
    Obtains observation s_i and select action $a_i \sim \pi_\theta(\cdot|s_i)$
  end for
  Execute actions $\{a_1, \cdots, a_n\}$ in the environment
  for each agent i **do**
    Obtains next observation s'_i, reward r_i, and done signal d
    Execute TRAINAGENT_i$(s_i, a_1, \cdots, a_n, r, s'_i, d)$
  end for
  if done signal d is terminal **then**
    Reset environment
  end if
until convergence

Algorithm 2 presents training of an agent with meta strategy.

Algorithm 2 TRAINAGENT_i with meta strategy

1: Input: observation s_i, actions of all agents $a_1, \cdots, a_n$, reward r, next observation s_i', done signal d

2: Store $(s_1, \cdots, s_n, a_1, \cdots, a_n, r, s_i', d)$ in replay buffer D_i

3: **if** it's time to update **then**

4: **for** number of updates **do**

5: Randomly sample $B = (s_1, \cdots, s_n, a_1, \cdots, a_n, r, s_i', d)$ from D

6: Compute targets for the Q-functions:

$$y = r + \lambda(1-d)\left(\min_{k=1,2} Q^i_{\phi^i_{\text{targ},k}}(s_i', \tilde{a_1}', \cdots, \tilde{a_i}', \cdots, \tilde{a_n}') + \alpha \log \frac{\pi_{\theta^i}(\tilde{a_i}'|s_i')}{\psi^i(\tilde{a_i}'|s_i')}\right)$$

$$\text{s.t. } \tilde{a_i}' \sim \pi_{\theta^i}(\cdot|s_i')$$

7: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi^i_k} \frac{1}{|B|} \sum_{(s_i, a_1, \cdots, a_n, r, s_i', d) \in B} \left(Q_{\phi^i_k}(s_i, a_1, \cdots, a_n) - y\right)^2 \text{ for } k = 1, 2$$

8: Update policy by one step of gradient ascent using

$$\nabla_{\theta^i} \frac{1}{|B|} \sum_{(s_i, a_1, \cdots, a_n, r, s_i', d) \in B} \left(\min_{k=1,2} Q_{\phi^i_k}(s, a_1, \cdots, \tilde{a}_{i\theta^i}(s_i), \cdots, a_n) + \alpha \log \frac{\pi_{\theta^i}(\tilde{a}_{i\theta^i}(s_i)|s_i)}{\psi^i(\tilde{a}_{i\theta^i}(s_i)|s_i)}\right)$$

$$\text{s.t. } \tilde{a}_{i\theta^i}(s_i) \sim \pi_{\theta^i}(\cdot|s_i)$$

9: Update target network with

$$\phi_{\text{targ},k} \leftarrow \rho \phi^i_{\text{targ},k} + (1-\rho)\phi^i_k$$

10: **end for**

11: **end if**

In line 5, unlike usual replay buffers, we store the observations and actions of all agents along with reward of agent i from current step, agent i's next state and whether the current episode has reached an end. In line 6 and line 8, we make use of a centralized critic function similar to the one used in [9]. We use double-Q trick to take the minimum of two Q-functions to stabilize training. In line 9, the parameters of the target network is applied a soft update as in [8] to stabilize training.

Chapter 6

Experiments

In this section, we evaluate the effectiveness of the proposed algorithms through experiments. Each defender is trained together with an attacker using [7, 9] in the competitive setting using Algorithm 1. The list of defenders we evaluate is presented below:

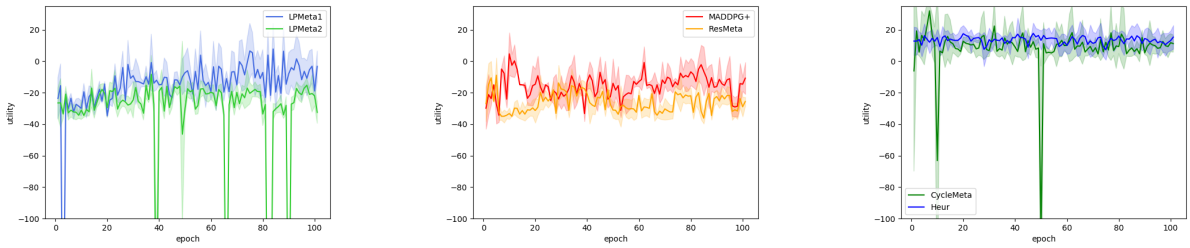
- **LPMeta1**: Defender trained using Algorithm 2 with implicit stationary equation based LP meta strategy (section 4.1.2). The number of nodes in the abstract graph is set to 100.
- **LPMeta2**: Defender trained using Algorithm 2 with implicit stationary equation based LP meta strategy (section 4.1.2) but without flow conservation constraints. In this LP formulation, we allow flow to stay at one node. The number of nodes in the abstract graph is set to 100.
- **CycleMeta**: Defender trained using Algorithm 2 with shortest cycle based meta strategy (section 4.2). The number of nodes in the abstract graph is set to 100. Since the shortest cycle based meta strategy is a deterministic strategy, we fit a multivariate Gaussian distribution in each node with mean being the action of the shortest cycle meta strategy in that node and variance being the same as the variance of the trained policy. Distance based reward shaping is also applied to help train the defender.
- **MADDPG+**: Defender trained using [7, 9]
- **ResMeta**: Defender trained using [9] but with solution of implicit stationary equation based LP as residual.
- **Heur**: Heuristic defender that patrols along the shortest cycle between targets.

The setting of the environment is summarized in the table below.

Table 6.1: Environment setting

Environment setting	Details
Size	500 x 500
Maximum time steps	70
Number of targets	5
Target 0 location and reward	$(400, 400)$, 20
Target 1 location and reward	$(200, 100)$, 10
Target 2 location and reward	$(100, 200)$, 10
Target 3 location and reward	$(260, 50)$, 10
Target 4 location and reward	$(200, 200)$, 15
Penetration time	20
Defender speed	20
Attacker speed	20
Attacker appearance time range	$[0, 10]$

The learning curves for different defenders are presented below. The y-axis is the mean and standard deviation of defender's utility of 100 test episodes after each training epoch. One epoch of training consists of 4000 game steps.



(a) (b) (c)

Figure 6.1: Learning curve

We test each defender against the attackers listed below:

- **MADDPG+**: Attacker trained using [7, 9].
- **Greedy**: Heuristic attacker that attacks the target with the highest reward.
- **Rand**: Heuristic attacker that attacks a random target.

After training, each pair of defender and attacker plays 100 rounds of games against each other. The mean and standard deviation of the defender's utilities are listed in Table 6.2. We observe that using meta-strategies as regulators could improve defender's utility while it could also hurt defender's utility. The choice of meta-strategy heavily influences defender's performance. We also test a heuristic defender that patrols along the shortest cycle among targets. We found that the performance of the heuristic defender is comparable to that of the CycleMeta defender.

Table 6.2: Mean and standard deviation of trained defender's utility against each attacker

Defender \ Attacker	MADDPG+	Greedy	Rand
LPMeta1	3.27, 14.18	-14.28, 5.19	-4.64, 10.79
LPMeta2	-22.16, 9.25	-23.61, 6.65	-2.75, 18.86
CycleMeta	11.26, 8.67	-1.36, 5.35	8.79, 15.99
MADDPG+	-10.73, 9.71	-17.43, 4.18	-4.47, 13.41
ResMeta	-23.21, 5.04	-23.99, 6.58	-4.81, 16.47

We now present the solutions of the meta-strategies and example runs of the defenders. Here, we fix the defender's starting location to be the center of the environment. The solutions of the implicit stationary equations based LP and shortest cycle meta-strategy is shown in Figure 6.2

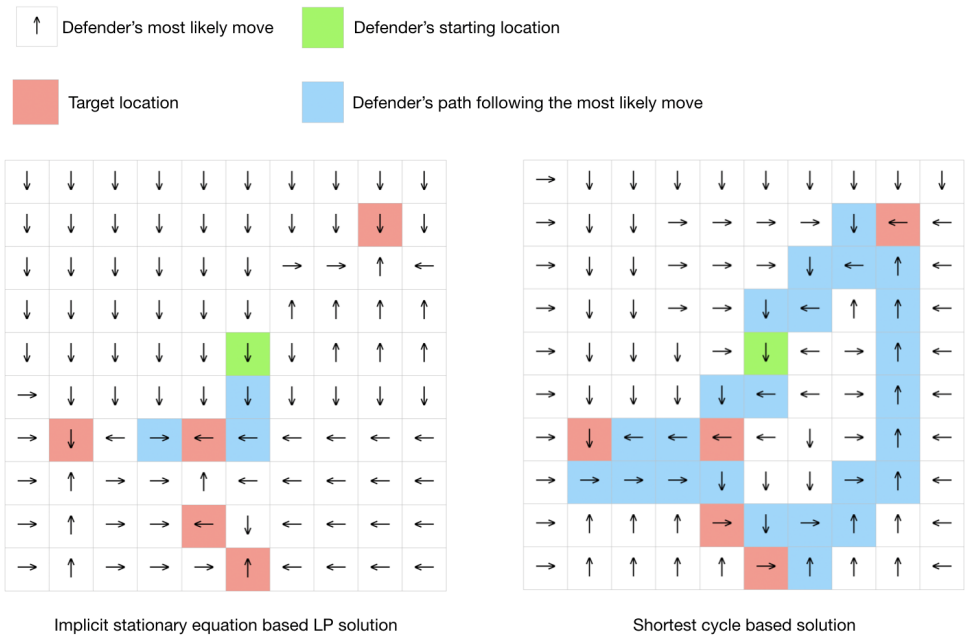


Figure 6.2: Solutions of meta-strategies

Example runs of each trained defender is shown in Figure 6.3. The solutions of the implicit stationary equation LP formulation encourage self-loops after the defender reaching the target. The same self-looping behavior could be observed in LPMeta2 defender and ResMeta defender, which uses the solutions of the implicit stationary equation LP formulation as meta-strategies. The CycleMeta defender uses the shortest cycle based meta-strategy in training and exhibits similar behaviors as the shortest cycle based meta-strategy, which is to patrol the targets along the shorter paths. However, one improvement that is found by CycleMeta during training is revisiting a nearby visited target before going for the next one if the next target is far away, as we observed in Figure 6.3. The MADDPG+ attacker is showing similar behaviors as moving to the closest target from its initial location.

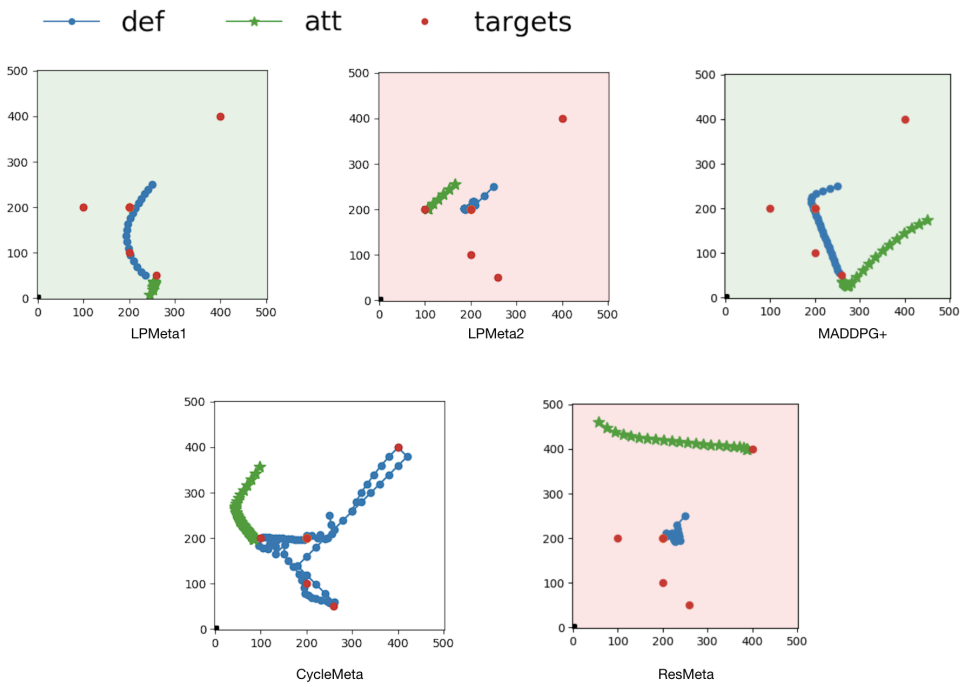


Figure 6.3: Example runs of trained defenders

Through experiments, we have shown that meta-strategy guided reinforcement learning agents are able to gain higher utilities than traditional reinforcement learning agents. We also note that the choice of meta-strategy has a significant effect on the outcome of the training. Not all meta-strategies could have a positive effect in terms of defender's utility. We have also found that while trained meta-strategy guided reinforcement learning agents show certain similarities in patrol paths to its meta-strategy, these agents are able to improve based on the input strategy by patrolling scholastically and taking more efficient routes to patrol targets. LPMeta1 defender and CycelMeta defender are able to perform better than MADDPG+ defender since during training, meta-strategy guided agents are able to locate and patrol the targets more efficiently through the help of meta-strategy regulator while the MADDPG+ defender could only establish a patrolling strategy by exploration.

We have shown that meta-strategy guided reinforcement learning performs better than traditional deep reinforcement learning approaches in our patrolling environment. Meta-strategy guided reinforcement learning could also help outside our proposed environment since it allows us to provide an initial strategy to the agents and allow the agents to improve based on the input meta-strategy.

Chapter 7

Disscussions and Future Work

In this work, we presented some challenges of multi-agent reinforcement learning algorithms including uninterpretability and sparse rewards. We presented a continuous space green security game with two players and several approaches to solve for meta-strategies in the simplified game including linear program based strategies and shortest cycle based strategies. We then use the results from the simpler setting to guide defender training in a multi-agent reinforcement learning setting. Through experiments, we show that using meta-strategies as regulators could speed up training for multi-agent reinforcement learning agents and lead to higher expected utility. While using meta-strategies could lead to improvements in defender's utility, it could also result in decrease of defender's utility. The choice of meta-strategy heavily affects defender's performance. Through experiments, we have shown that meta-strategy guided reinforcement learning agents have the ability to take in an initial meta-strategy and make improvements based upon the input strategy.

For future work, human-UAV coordination is gaining significance in the green security field. Having a UAV could help find locations of attacker and could send out alarming signals to deter attackers. Incorporating recent work on human-UAV coordination would help further improve defender's patrolling strategy.

Bibliography

[1] Bo An, Eric Shieh, Milind Tambe, Rong Yang, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. Protect – a deployed game theoretic system for strategic security allocation for the united states coast guard. *AI Magazine*, 33(4):96, Dec. 2012. doi: 10.1609/aimag.v33i4.2401. URL https://www.aaai.org/ojs/index.php/aimagazine/article/view/2401. 1

[2] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artif. Intell.*, 184–185:78–123, June 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.03.003. URL https://doi.org/10.1016/j.artint.2012.03.003. 2.2

[3] Nicola Basilico, Giuseppe De Nittis, and Nicola Gatti. Adversarial patrolling with spatially uncertain alarm signals. *CoRR*, abs/1506.02850, 2015. URL http://arxiv.org/abs/1506.02850. 2.2

[4] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. 4.2

[5] Fei Fang, Albert Xin Jiang, and Milind Tambe. Optimal patrol strategy for protecting moving targets with multiple mobile resources. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 957–964, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450319935. 2.1

[6] Fei Fang, Thanh H. Nguyen, Rob Pickles, Wai Y. Lam, Gopalasamy R. Clements, Bo An, Amandeep Singh, and Milind Tambe. Deploying paws to combat poaching: Game-theoretic patrolling in areas with complex terrain (demonstration). In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 4355–4356. AAAI Press, 2016. 1

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290. 1, 2.3, 6, 6

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. 1, 5.1.3

[9] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

URL http://arxiv.org/abs/1706.02275. 1, 2.3, 5.1.3, 6, 6

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602. 2.3

[11] Enrique Munoz de Cote, Ruben Stranders, Nicola Basilico, Nicola Gatti, and Nick Jennings. Introducing alarms in adversarial patrolling games: Extended abstract. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1275–1276, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450319935. 2.2

[12] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, AAMAS '08, page 125–132, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. 1

[13] Zheyuan Ryan Shi, Aaron Schlenker, Brian Hay, and Fei Fang. Towards thwarting social engineering attacks. *CoRR*, abs/1901.00586, 2019. URL http://arxiv.org/abs/1901.00586. 1

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249. 2.3

[15] Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, USA, 1st edition, 2011. ISBN 1107096421. 2.1

[16] Yevgeniy Vorobeychik, Bo An, and Milind Tambe. Adversarial patrolling games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, page 1307–1308, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0981738133. 4.1.4

[17] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. *CoRR*, abs/1811.02483, 2018. URL http://arxiv.org/abs/1811.02483. 2.2