DOUBLE-ORACLE DEEP REINFORCEMENT LEARNING FOR HANDLING EXPONENTIAL ACTION SPACE IN SEQUENTIAL STACKELBERG SECURITY GAMES

by

CZANDER TAN

A THESIS

Presented to the Department of Computer and Information Science
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2022

THESIS APPROVAL PAGE

Student: Czander Tan

Title: Double-Oracle Deep Reinforcement Learning for Handling Exponential Action Space in Sequential Stackelberg Security Games

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

Thanh H. Nguyen	Chair
Daniel Lowd	Core Member

and

Krista M. Chronister	Vice Provost for Graduate Studies

Original approval signatures are on file with the Division of Graduate Studies of the University of Oregon.

Degree awarded June 2022

THESIS ABSTRACT

Czander Tan

Master of Science

Department of Computer and Information Science

June 2022

Title: Double-Oracle Deep Reinforcement Learning for Handling Exponential Action Space in Sequential Stackelberg Security Games

Standard Stackelberg Security Games (SSGs) assume attackers to be *myopic* players that select only a single target based on the defender's strategy. In this paper, we consider sequential SSGs, in which attackers launch multiple attacks sequentially. With a sequence of events, however, the defender's action space grows exponentially with the number of time steps, making the problem computationally intractable.

To handle this issue, this paper presents the following contributions. First, we use the Double Oracle algorithm to iteratively derive player strategies. Second, we use Advantage Actor-Critic models to approximate best response policies for both players. Lastly, we represent the defender action space compactly with *marginal probabilities* instead of enumerating all possible actions.

Overall, our experiments show that the Double Oracle algorithm not only allows us to search through defender strategies effectively and efficiently, but also provides optimal solutions that outperform other models at scalable settings.

ACKNOWLEDGMENTS

I would like to thank my research advisor Thanh H. Nguyen for her assistance in my research and the writing of this work, as well as her understanding during a difficult part of my life. I would also like to thank Heidi Kaufman, who helped me with my first publication and always encouraged my interdisciplinary explorations, and Stephen Fickas, who saw my potential and encouraged me to pursue a graduate degree in Computer Science. Finally, I would like to thank my dear family and cherished friends, without whose love and support I would not be here.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

1 INTRODUCTION

In real-world security domains, law enforcement organizations work to protect critical infrastructure from human attackers. They do this by monitoring and allocating resources to specific targets not only to engage with security threats, but also to strategically deter and control attacks. An airport, for example, includes multiple terminals, control towers, hangars, lobbies, and runways. In order to establish security at an airport, human and material resources are distributed to checkpoints, patrols, and various forms of security systems. While the number of potential targets at an airport is substantial, security resources are often extremely limited, so organizations have to figure out which resources to apportion to patrols, how much for checkpoints, where to place those checkpoints, etc., to achieve optimal coverage of the entire territory.

Since defensive agencies have to establish security preemptive to any attack, this kind of leader-follower security game, known as Stackelberg Security Games (SSG), puts the defenders at a disadvantage. Malicious actors, or attackers, often spend time observing general patterns of security resources specific to their targets; therefore, in SSGs, attackers decide on which target to attack with knowledge of the general defender strategy. On the other hand, defenders are modeled to commit to their resource allocation strategy without knowledge of the attacker's behavior. As such, game theory has been utilized as a formal method for developing strategies to handle such disadvantages. Much research has been done with developing game theoretical models for security scenarios to optimize defender strategies based on the values of their targets, randomization, and other methods (Fang et al., 2017; Shieh et al., 2012; Pita et al., 2009; Tsai et al., 2009; Sinha et al., 2018).

Increased attention to defensive strategies, however, also pressures malicious actors to develop more sophisticated strategies of their own. One of these strategies involves *sequential attacks*, where a malicious actor attacks multiple targets in a specific order (Nguyen et al., 2019). For example, a malicious actor could launch an attack on a less valuable target to attract the attention of security resources, after which they would launch an attack on a more valuable target. Even if the value of targets were more or less uniform, these sophisticated attack strategies could be formed and executed to manipulate security resources.

Standard SSGs assume attackers to be *myopic* players that select only a single target based on the defender's strategy. That is, attackers in standard SSGs only consider one action. Similarly, defenders in standard SSGs only consider the present moment, hence the possibility of allocating resources to a select few targets. In sequential SSGs, however, both

attackers and defenders consider future possibilities and make decisions based on the sequence of events that have happened and might happen. As a result, both sides can adapt their strategies according to real-time observations. For an attacker, partial observation of whether or not a defender resource is at an attacked location could change their next attack. Similarly, a sequence of attacks reveals information to the defender over time, allowing the defender to update its resource allocations accordingly.

Sequential SSGs not only provide models for more realistic security scenarios, but they also level the playing field between attacker and defender. While the attacker has preemptive knowledge of general defender behaviors, their belief of whether a resource is allocated to a specific target is only confirmed when they make an attack. On the other hand, the defender knows where all of its resources are, as well as where an attacker launches their attack. Therefore, in the sequence of events, the defender has full observation of the game while the attacker only has a partial observation.

Being able to model more realistic scenarios, however, also introduces exponential complexity to the problem of deriving optimal defender strategies. The action space for defenders in standard SSGs is already large, being a function of the number of targets and resources. By adding a sequence of events, the action space grows exponentially according to the number of time steps. For example, let a defender have 5 resources, with the goal of protecting 10 targets. In a standard SSG, the action space of the defender strategy is 252. In a sequential SSG, with 3 attacks, the action space of the defender strategy would be $(252)^3 \approx 10^7$, over a ten million combinations.

In order to handle the exponential action space of the defender in sequential SSGs, our work focuses on the following main contributions. First, we use the Double Oracle algorithm (Jain et al., 2011) to derive the most likely subsets of attacker and defender strategies. Using this algorithm, we bypass the need to enumerate over all strategies and take advantage of the Monte Carlo sampling methods that base the Double Oracle algorithm.

Second, we use a deep reinforcement learning model, the Advantage Actor-Critic (A2C) (Mnih et al., 2016), to approximate best response policies for the defender and the attacker. This model merges the benefits of value-based learning with that of policy gradient methods, presenting a robust solution for each oracle in the Double Oracle algorithm.

Lastly, we transform the representation of the defender action space into *marginal probabilities* instead of probabilities across all possible actions. While this approach aids the intractability of enumerating over all defender actions, it is specific to the SSG setting, where defender marginal probabilities represent their coverage over the targets.

CHAPTER 2

2 BACKGROUND

Sinha et al. (2018) in (Sinha et al., 2018) provide a comprehensive overview of SSGs and technical advances in SSG-based applications over the years. In this section, I briefly describe these applications to give the reader a sense of where SSG research has focused. Then I discuss a work that has applied the Double Oracle algorithm to standard SSG games. Lastly, I discuss recent work in sequential SSGs.

2.1 Stackelberg Security Game Applications

The use of SSG-based applications for real-world domains began primarily with the deployment of ARMOR at the Los Angeles International Airport (LAX) in 2007, with the deployment of IRIS for scheduling the U.S. Federal Air Marshal Service in 2009, and with the deployment of PROTECT for scheduling the U.S. Coast guard in 2012.

ARMOR – Pita et al. (2008) in (Pita et al., 2009) present ARMOR (Assistant for Randomized Monitoring Over Routes), a scheduling system to randomize checkpoints and canine patrols at LAX. A key contribution of Pita et al., in relation to this paper, is the consideration of multiple adversary types in SSGs (e.g. smugglers, criminals, terrorists). While Pita et al. only consider one attack, considering multiple adversary types means that the defender strategy provided by ARMOR must be adaptive to various attacker priorities, similar to how the shifting environment in sequential SSGs affects defender strategies.

IRIS – An extension of the considerations of ARMOR, Tsai et al. (2009) apply a SSG model to transportation networks with IRIS (Intelligent Randomizing In Scheduling) in (Tsai et al., 2009). They face the issue of exponential state space since transportation networks, particularly tens of thousands of commercial flights, contain numerous routes. One way IRIS solves this issue is to represent the strategy space of the leader (i.e. the defender) compactly, based on scheduling constraints, instead of enumerating over all possible schedules. This solution is similar to how this paper compactly represents defender actions using marginal probabilities due to exponential state and action spaces.

PROTECT – To address the bounded rationality of attackers, rather than assuming perfect rationality as the previous applications have done, Shieh et al. (2012) in (Shieh et al., 2012) use quantal response (QR) to model attacker behavior. With this new assumption, Shieh et al. present PROTECT (Port Resilience Operational/Tactical Enforcement to Combat Terrorism) to aid the U.S. Coast Guard in protecting the ports around Boston and New York. In addition to modeling QR attackers, PROTECT also uses

compact representation for defender actions. In the experiments of this paper, we use QR as a baseline against which we compare our defender strategy computed by the Double Oracle algorithm.

2.2 Double Oracle for Stackelberg Games

In their paper, Karwowski and Mandziuk apply the Double Oracle algorithm to approximate equilibria for general-sum extensive-form games in the Stackelberg setting (Karwowski and Mandziuk, 2020). They move beyond conventional methods for solving Stackelberg equilibria that use mixed integer linear programs (MILP), instead leveraging Monte Carlo tree search to sample player strategies. While their methods are able to generate high-quality solutions that scale better than MILP-based methods, their setting is only for one-shot Stackelberg games. Thus, in this paper, we seek to apply the Double Oracle algorithm to the sequential setting.

2.3 Sequential Stackelberg Security Games

The first foray into sequential SSGs is the paper by Nguyen et al. (Nguyen et al., 2019), where they explore two-step sequential SSGs in two settings: one where the defender can move resources between time steps, and one where the defender cannot move resources. For both settings, they find that considering attacks sequentially derives a strategy no worse for the defender than considering the attacks as a series of one-shot games. In this paper, we consider the setting where the defender can move resources between time steps. This setting not only allows for more complex and realistic scenarios through non-zero-sum games, but also results in higher quality defender strategies than computing from a series of one-shot SSGs.

To go beyond the two-step sequential SSG setting, Tomasek et al. consider sequential SSGs as one-sided partially observable stochastic games (Tomasek et al., 2020). They base their method on Heuristic Search Value Iteration and adapt it to the finite-horizon sequential SSG setting to design a heuristic for scaling towards larger games. However, they acknowledge that their methods do not return the optimal defender strategies for larger games. Therefore, in this paper, we seek to develop a method that generates an optimal solution by implementing reinforcement learning models in a Double Oracle algorithm.

CHAPTER 3

3 PROBLEM DESCRIPTION

A sequential Stackelberg Security Game (seSSG) involves a fixed number of targets and two players, an attacker and a defender, where the defender has a limited number of resources to defend the targets. At each time step, the defender first commits to a strategy, and then the attacker selects a target to attack. If there is a defender resource at that target, the attack fails; otherwise, the attack is successful. Then the attacker selects another target to attack, and the process repeats until all time steps are finished.

Formally speaking, a seSSG is defined by a set of targets T, a fixed number of time steps I, a set of defender resources K, a set of rewards R over the targets, and a set of penalties P over the targets. In terms of game theory, players engage the game by selecting actions against each other, and each executed action is known as a *pure strategy* (e.g. a resource allocation). Let G and H be the set of all pure strategies for the defender and the attacker respectively. To introduce uncertainty into the game setting, players employ a *mixed strategy*, which is a randomization over all pure strategies. In our case, a mixed strategy for the defender is denoted as $x = \{Pr(g)\}$, where g is a pure strategy in G and $Pr(g)$ is the probability that the defender plays pure strategy g. Similarly, let a mixed strategy for the attacker be denoted as $y = \{Pr(h)\}$, where h is a pure strategy in H and $Pr(h)$ is the probability an attacker plays pure strategy h.

At each turn i of a game, each player simultaneously takes an action based on the state s_i of the game. The defender's action $d_i \sim x_i$ allocates k defense resources to T targets, where $k < T$ and d_i refers to the set of targets protected by the defender at i. At the same time, the attacker's action $a_i \sim y_i$ chooses a target to attack. If the attack meets a defender resource at a_i, i.e. $\exists\ l \in d_i$ where $a_i = l$, the attack fails. Additionally, the resource at l is removed from play. The defender is rewarded according to the utility for that target $R^D(a_i)$, and the attacker is penalized for $P^A(a_i)$. However, if the attacker does not meet a defender resource at a_i, i.e. $a_i \neq l, \forall l \in d_i$, the attack is successful and the target is destroyed. In that case, the defender is penalized according to the defender penalty for that target $P^D(a_i)$, and the attacker is rewarded with $R^A(a_i)$. Within one time step, the process described is that of a standard non-sequential SSG.

For a seSSG, however, each player takes an action again, repeating the process until the last time-step is finished. Therefore, the trajectory of the game must be taken into account, as it reflects how defender and attacker mixed strategies, x_i and y_i, would be updated. Furthermore, to simulate real-world scenarios, we also introduce a resource movement cost for the defender. That is, if the defender's sampled action moves a resource

from one target to another, a cost is deducted from the immediate reward or penalty.

After players take their actions, then, each of them sees an independent observation of the results. Since the defender knows where all the resources were allocated during the attack, the defender has a complete observation of the game turn, knowing where the attack happened in relation to the resource allocation. Let $o_i^d = \{(d_i, a_i)\}$ denote the history of the game so far at time step i, representing the defender's observation. At the next time step, the defender's mixed strategy would be $x_{i+1} = \{Pr(g|o_i^d)_{i+1}\}$, which shows how the probabilities of playing a pure strategy g are updated given the history of the game.

The attacker, on the other hand, sees only the outcome of the attack, gaining knowledge of the defender's resource allocation only at the target attacked. That is, if the attack does not meet any defender resource at target a_i, the only information gained about the defender's strategy is that a resource was not allocated at a_i. Since the attacker does not have information about the entire resource allocation, they have incomplete information regarding the game turn. Let $o_i^a = \{[a_i, b(a_i)]\}$ denote the attacker observation at time step i, where $b(a_i)$ represents whether or not there was a defender resource at a_i. The attacker's mixed strategy at the next time step, then, would be $y_{i+1} = \{Pr(h|o_i^a)_{i+1}\}$, which shows how the probabilities of playing a pure strategy h are updated given partial observations at each time step.

As such, each player takes an action and updates their strategies each turn until the last time step is reached. Within each game, then, the objective is to compute a mixed defender strategy x^* to determine the defender's actions each time step such that the defender's utility is maximized, given a rational minimizing attacker in the game setting specified. With U^D to denote the defender utility after all time steps, and U^A as the same for the attacker, a Strong Stackelberg Equilibrium (SSE) for a seSSG can be formulated as:

$$\begin{aligned} x^* &= argmax_x U^D(x, a^*(x)) \\ a^*(x) &= argmax_a U^A(x, a) \end{aligned} \tag{1}$$

A SSE denotes a point where it would not be advantageous for the defender to deviate from mixed strategy x^*, given the same for the attacker playing a^*. However, as mentioned previously, to enumerate over all pure strategies for the defender in G is infeasible for an exponentially large action space, making the computations for x intractable. Therefore, in Chapter 4, we introduce the Double Oracle approach, which iteratively enumerates through best-response strategies to derive an equilibrium. Before introducing the Double Oracle approach, however, we describe a simple example comparing the seSSG setting to the standard SSG, incorporating resource movement costs for the defender.

3.1 Example with Resource Movement Cost

Consider a non-zero-sum toy example game setting with 3 targets, 2 time steps, and 2 defender resources, where the value of each of the targets is 1, 2, and 3 respectively. In this example, if the attacker attacks a target and there is not a defender resource allocated to it, they receive a reward equal to the value of the target; otherwise, they get a utility of zero. Likewise, if the defender is able to block an attack, the defender gets a reward equal to the value of the defended target; otherwise, the defender gets a utility zero. Real-world values will have penalties less than zero, but we use zero here for convenience and to make it a non-zero-sum game.

In a standard SSG setting, the defender's mixed strategy in a SSE would make it such that the expected utility of the attacker is equal across the targets. With 2 resources and 3 targets, the defender has 3 possible actions to allocate resources accordingly: $\{(t^1, t^2), (t^1, t^3), (t^2, t^3)\}$. This game can be represented in the following matrix, with the row player as the attacker and the column player as the defender:

	(t^1, t^2)	(t^1, t^3)	(t^2, t^3)
t^1	$0, 1$	$0, 1$	$1, 0$
t^2	$0, 2$	$2, 0$	$0, 2$
t^3	$3, 0$	$0, 3$	$0, 3$

With this setting, the defender's mixed strategy across their 3 possible actions would calculate to be $(\frac{2}{11}, \frac{3}{11}, \frac{6}{11})$ respectively.

Let's say the attacker chooses to attack target t^1 at the first time step, and a defender resource was present. In the second time step, the defender would only have one resource remaining and two targets left to defend, each with the values 2 and 3 respectively. In a standard SSG setting, the defender would have to recalibrate their mixed strategy for the next attack, for defending either t^2 or t^3 respectively, to $(\frac{2}{5}, \frac{3}{5})$.

Let us now introduce the resource movement cost of -0.5 to determine the expected utility of the defender for the second attack. Consider the case where the defender's move in the first round was (t^1, t^2). The utility matrix of the second round, including resource movement cost for the defender, would then be:

	$t^2 \to t^2$	$t^2 \to t^3$
t^2	$0, 2$	$2, -0.5$
t^3	$3, 0$	$0, 2.5$

Even if we fixed the attacker's first attack to target t^1 such that the defender always covers t^1 in the first round and gets a reward of 1, the defender's total expected utility for the two-step game would be 1.4. This value results from the defender's expected utility of 0.4 in the second round for playing $(\frac{2}{5}, \frac{3}{5})$. If the defender played (t^1, t^3) for the first round, however, the resulting utility matrix for the second round would be:

	$t^3 \rightarrow t^2$	$t^3 \rightarrow t^3$
t^2	$0, 1.5$	$2, 0$
t^3	$3, -0.5$	$0, 3$

In this case, the defender's expected utility for the second round would be 1.1. Given the case that the attacker's first target is fixed to t^1 such that the defender always gets a reward of 1 in the first round, the total expected utility of the defender for the two-step game if they played (t^1, t^3) the first round would be 2.1. Using the initially calculated mixed strategy of the defender to select only between (t^1, t^2) and (t^1, t^3), i.e. $(\frac{2}{11}, \frac{3}{11}) \rightarrow (\frac{2}{5}, \frac{3}{5})$ respectively, the defender's expected utility considering both scenarios is 1.82.

Let's say we keep the attacker's first attack fixed to target t^1, and we considered a sequential SSG setting. In terms of a real-world scenario, t^1 might be a checkpoint that the attacker must pass through in order to get to either t^2 or t^3. In that case, we can represent the simplified sequential game through the following matrix, with the row player as the attacker and the column player as the defender:

	$(t^1, t^2), t^2$	$(t^1, t^2), t^3$	$(t^1, t^3), t^2$	$(t^1, t^3), t^3$
t^1, t^2	$0, 3$	$2, 0.5$	$0, 2.5$	$2, 1$
t^1, t^3	$3, 1$	$0, 3.5$	$3, 0.5$	$0, 4$

Knowing that the attacker will first attack target t^1 before either t^2 or t^3, the defender has 4 possible sequential actions shown in the matrix above. To render the attacker's choice between t^2 and t^3 indifferent, the defender can play the mixed strategy $(\frac{1}{5}, \frac{3}{10}, \frac{1}{5}, \frac{3}{10})$. In this case, the defender's total expected utility for the two-step game would be 2.125, which is greater than the 1.82 derived from playing two one-shot games.

Furthermore, the sequential SSG setting introduces an additional advantage in that the mixed strategy equilibrium computation need only be computed once, instead of being recalibrated every time step according to the attack's outcomes. The latter option is not feasible for real-world scenarios, where response to sequential attacks must often be

immediate. Since the defender's mixed strategy for a sequential SSG setting must include resource allocations for every time step, and take into account every possible attack sequence, the space of the matrix grows exponentially. If we did not fix the first attack to t^1 in our toy game setting, for example, the attacker would have 6 possible sequences of attacks and the defender would have 18 possible sequences of resource allocations, making the size of the matrix 108. From these calculations, we can see that beyond the toy game setting, the computation for the SSE would be intractable. Therefore, the next section introduces the Double Oracle algorithm to solve this issue.

CHAPTER 4

4 DOUBLE ORACLE

The Double Oracle algorithm was first developed to solve simultaneous two-player adversarial games (McMahan et al., 2003). In our case, an *oracle* O is essentially a function that returns a policy π given a set of inputs S, that is, $O : S \rightarrow \pi$. And a policy maps a given state $s \in S$ to a mixed strategy x, from which an action a is sampled, giving $\pi : s \rightarrow (a \sim x)$. Basically, the Double Oracle algorithm employs an oracle for each player that returns a best response policy given the strategies of their opponent. Then, with the best response policies being added to a set of policies of the respective players at each iteration, the algorithm keeps generating best response policies for each player until no better can be found. As such, the iterative process of the Double Oracle algorithm aids with the issue of exploring an exponentially large action space by simulating only best response policies against the respective opponent's set of policies.

To explain the Double Oracle algorithm in more detail, let X_j and A_j be the set of defender and attacker policies, respectively, generated so far at iteration j. First, a utility matrix, M_j, between the defender and attacker is calculated by simulating each defender's policy $\pi^D \in X_j$ against each attacker's policy $\pi^A \in A_j$. Then, we use a linear program solver, $CoreLP(X_j, A_j, M_j)$, which calculates the Nash equilibrium of the two-player game. The Nash equilibrium returned by the $CoreLP$ gives p_j^* and q_j^*, which are probability distributions for the defender and attacker over their respective sets of policies X_j and A_j.

After the $CoreLP$ step, the main part of the Double Oracle algorithm is run: we compute a defender oracle that generates a best response policy π^{D*} against q_j^*; at the same time, we compute an attacker oracle that generates a best response policy π^{A*} against p_i^*. Both new policies are subsequently added to X_{j+1} and A_{j+1} respectively, and the utility matrix is updated to M_{j+1}. Subsequently, the Nash equilibrium probability distribution for each player is updated with $CoreLP(X_{j+1}, A_{j+1}, M_{j+1})$, which provides p_{j+1}^* and q_{j+1}^*. As such, X and A increase in size each iteration of the Double Oracle algorithm, and new best response policies are generated for each player each time until convergence is reached.

Convergence results when the policies generated by the oracles stop improving over existing policies in the set, or when the probability distributions given by the $CoreLP$ stop changing. At that point, neither the defender oracle nor the attacker oracle can produce a policy better than the current solution provided by the $CoreLP$. This way, the optimal probability distribution over best response policies for either player can be derived without having to enumerate over all possible policies or compute all the possible probability distributions over those policies.

Computing the oracles, however, can be quite expensive. In order to make it tractable, we use deep neural networks to approximate the oracles of each player. This means that the modified algorithm looks as follows:

1. Generate initial policies for defender and attacker sets.
2. Compute respective Nash equilibrium probability distributions using the $CoreLP$.
3. Train a neural network against the attacker probability distribution to approximate the defender oracle.
4. Train a neural network against the defender probability distribution to approximate the attacker oracle.
5. Add generated policies (i.e. the weights and biases of the networks) to the sets of policies of each respective player.
6. Repeat steps 2-5 until convergence, where each oracle's policy returns a utility no better than that from the $CoreLP$ for each player.
7. Return the optimal defender and attacker mixed strategies (i.e. probability distributions over their set of respective policies represented by neural networks).

The correctness of the Double Oracle algorithm for two-player zero-sum games has been presented in (Jain et al., 2011) and (McMahan et al., 2003). Basically, when the algorithm converges, the defender and attacker mixed strategies at that iteration form an equilibrium of the game. And since each oracle cannot find a better policy after searching over all possible policies against the opponent's mixed strategy, those defender and attacker mixed strategies are optimal. Additionally, the algorithm must converge, because at worst, it searches over all policies in the entire exponentially large state and action space.

4.1 CoreLP

To compute Nash equilibria for our experiments, we use the *gambit* software, specifically their linear complementarity program (LCP) as our $CoreLP$. Basically, the program generates a probability distribution for each player by optimizing the player's expected utility against the set of policies of the opponent. Finding these probability distributions can be formulated as a maximin problem between defender and attacker utilities as follows:

$$\begin{gathered} \max_{p,\ v} v^* \\ \text{such that } v \leq \sum_k p_k \cdot U(d_k,\ a_l), \quad \forall a_l \in A \\ \text{where } \sum_k p_k = 1 \text{ and } p_k \in [0,\ 1] \end{gathered} \tag{2}$$

The variables here represent the following:

- v: The player's expected utility
- p: The player's probability distribution, formed from a random distribution over the set of policies D

- k: The index of a policy for the player
- l: The index of a policy for the opponent
- p_k: The probability to play a given policy d_k
- d_k: A single policy for the player, indexed by k from the set of player policies in the restricted problem.
- a_l: A single policy for the opponent, indexed by l from the set of opponent policies in the restricted problem.
- $U(d_k, a_l)$: The total utility of the player when playing policy d_k against opponent policy a_l.

As the first line signifies, our main objective is to maximize the player expected utility v that corresponds to some probability distribution, or randomization, p. Our primary variables, then, are v and p.

While the second and third constraints ensure that p adds up to one and each $p_k,\ \forall k \in |D|$, is a valid probability, the first constraint ensures the opponent minimizes the player's expected utility for each opponent policy, across all opponent policies. That is, for every policy d played according to some overarching randomization p, we assume the worst case scenario where the opponent always plays their best policies. If we derive the optimal player strategy according to the worst case scenario, any other action the opponent takes will result in a better outcome for the player. Thus, for every randomization p, we derive the minimum expected utility for the player. Then, out of the set of derived minimized player expected utilities, the objective selects the randomization p that corresponds to the maximum expected utility v^*.

As such, at each step j of the Double Oracle algorithm, the $CoreLP$ returns probability distributions p_j^* and q_j^* (along with corresponding maximin expected utilities v_j^{D*} and v_j^{A*}) for the defender and attacker, each according to their respective set of policies X_j and A_j. This process is compactly represented in Algorithm 1.

Algorithm 1: Double Oracle Algorithm with Neural Networks

Initialize defender strategies X
Initialize attacker strategies A
for *each iteration* **do**
    $M_j \leftarrow Game(x, a), \forall x \in X, \forall a \in A$
    $(p_j^*, v_j^{D*}), (q_j^*, v_j^{A*}) \leftarrow CoreLP(X, A, M_j)$
    Initialize and train defender oracle neural network D_j:
     $\pi_j^{D*}, v_j^d \leftarrow D_j(q_j^*)$
    Initialize and train attacker oracle neural network E_j:
     $\pi_j^{A*}, v_j^a \leftarrow E_j(p_j^*)$
    if $v_j^d \leq v_j^{D*} AND\ v_j^a \leq v_j^{A*}$ **then**
        return p_j^*, q_j^*
    $X \leftarrow X \cup \{\pi_j^{D*}\}$
    $A \leftarrow A \cup \{\pi_j^{A*}\}$

CHAPTER 5

5 ORACLE DESIGN

With the attacker's equilibrium distribution q_j^* over their set of policies A_j, the defender oracle must generate a best response policy d^*. More specifically, this best response policy d^* must maximize the defender expected utility $U(d^*, q_j^*)$ such that the expected utility is better than that from the equilibrium, i.e. $U(d^*, q_j^*) > v_j^{D*}$.

As shown in (Jain et al., 2011), however, this problem is NP-hard as it reduces to the Set-Cover problem. While Jain et al. derive a Mixed Integer Linear Program to solve the defender oracle, their problem is zero-sum and not sequential. Furthermore, as previously mentioned, the state and action space for seSSGs are much larger than standard SSGs. Therefore, we use reinforcement learning, specifically Policy Gradient methods, to approximate the best response policy for the defender oracle.

5.1 Reinforcement Learning - Policy Gradient Methods

Due to the exponentiality of the state and action space for seSSGs, we cannot implement a standard reinforcement learning approach, which derives a policy based on approximating the state (Value Function Approximation) or state-action values (Q-learning) of the entire game. Since computing all state-action values for large seSSGs is intractable, we turn to Policy Gradient methods, which directly approximate the policy itself. In Policy Gradient methods, the oracle learns *parameters* θ for policy π, and the goal is to find the θ for the optimal policy π^*, which provides the maximum value function. As our oracles are implemented using neural networks, θ represents the weights and biases of the network.

With a game state s_i as input to the network, then, the defender gets a mixed strategy x_i from its parameterized policy, with the probability of the mixed strategy being represented by $\pi(x_i|s_i, \theta_i)$; then the defender samples a discrete resource allocation $d_i \sim x_i$. The outcome of playing d_i, which includes defender utility r_i, is then used to update the policy parameters based on a learning rate α and the gradient of some function $J(*)$ that seeks to maximize the defender's utility, giving $\theta_{i+1} = \theta_i + \alpha \nabla J(\theta_i)$.

The disadvantage of using Policy Gradient methods over value-based methods, however, is that they are more likely to get stuck in local optima. Without direct optimization of the values of states or state-action pairs, it is difficult to determine if a current policy is the globally optimal one. In our case, while $J(*)$ aims to maximize the defender's utility, there is no absolute target against which to compare the state-action

values of a given policy.

One solution is to leverage the benefits of both Policy Gradient methods and value-based methods by approximating both a value function and a parameterized policy. In this way, the oracle has a target to aid it in getting out of local optima, and it does not have to compute the state-action values of the entire game. This combined approach has led to Actor-Critic methods, which is explained further in the section below.

5.1.1 Advantage Actor-Critic (A2C)

In Actor-Critic methods, we have two primary components: the parameterized policy (actor), and the value function (critic). While the policy and value function each could have their own parameters, for the sake of simplicity, we use θ to represent parameters for both of them. Practically speaking, the oracle can be thought of as a single model that generates two outputs; in terms of neural networks, therefore, the two outputs share most of the network layers' weights and biases.

With the value function denoted as $V(s|\theta)$ and the policy as $\pi(x|s,\theta)$, an implementation of $J(\theta)$ that corresponds to actor-critic reinforcement learning as shown in (Sutton and Barto, 2018) is:

$$\nabla J(\theta_i) = [r_i + \gamma V(s_{i+1}|\hat{\theta}) - V(s_i|\theta)] \ \nabla \ log \ \pi(x_i|s_i,\theta_i) \tag{3}$$

where r_i is the reward for employing mixed strategy x_i to transition from state s_i to s_{i+1}, and γ is a discount factor. The term $\nabla \ log \ \pi(x_i|s_i,\theta_i)$ incorporates the *actor* part of the approach, where the probability of the state-action pair is factored into the gradient.

The segment of the equation $r_i + \gamma V(s_{i+1}|\hat{\theta}) - V(s_i|\theta)$ accounts for the *critic* part of the approach, where the value function is trained to encapsulate the value of the immediate reward at s_i and the discounted future reward. Note that the parameters in $\gamma V(s_{i+1}|\hat{\theta})$ are labeled with a hat symbol to distinguish that they are fixed for that value, so that the gradient step for the value function only ascends once for $V(s_i|\theta)$. The parameters in $\gamma V(s_{i+1}|\hat{\theta})$ are fixed also so that the target for $V(s_i|\theta)$ does not vary substantially. Furthermore, this segment of the equation corresponds to the Q-value of the state-action pair, where, generally speaking in reinforcement learning terms,
$Q(s,a) = R(s,a,s') + \gamma V(s') - V(s)$. Therefore, we can rewrite the above equation as:

$$\nabla J(\theta_i) = Q(s_i,x_i|\theta) \ \nabla \ log \ \pi(x_i|s_i,\theta_i) \tag{4}$$

As $Q(s_i,x_i|\theta)$ has a high variance from update to update, learning can be stabilized by introducing a baseline. Mnih et al. show in (Mnih et al., 2016) that subtracting $V(s_i)$ from

$Q(s_i, x_i)$ decreases overall variance, deriving the *advantage* $A(s_i, x_i) = Q(s_i, x_i) - V(s_i)$. The parameter update can thus be written as:

$$\theta_{i+1} = \theta_i + \alpha[A(s_i, x_i|\theta) \nabla log\ \pi(x_i|s_i, \theta_i)] \tag{5}$$

As such, we implement the Advantage Actor-Critic (A2C) method for the defender oracle's neural network by updating the network's parameters θ_i from its outputs $V(s_i|\theta_i)$ and $\pi(x_i|s_i, \theta_i)$, as well as the immediate utility of the time step r_i.

5.2 Compact Representation for Defender Actions

In regular reinforcement learning settings, the mixed strategy generated by Actor-Critic oracles is a probability distribution over all possible actions. Since enumerating all actions is intractable for larger instances of this problem, we require a compact representation for defender actions.

In this paper, we follow Nguyen et al. in (Nguyen et al., 2019), which implements compact representations for defender actions in resource movement settings of seSSGs. Instead of denoting defender mixed strategy x as a probability distribution over all possible actions, x can represent the probability distribution of moving a resource from one target to another over all targets. The justification for this approach, as proposed in (Nguyen et al., 2019), is that the probability a defender resource will be at a given target represents the marginal probability that that target will be protected by the defender before an attack at that time step.

Representing defender actions compactly with marginal probabilities also allows us to derive $\pi(x_i|s_i, \theta_i)$ easily. As such, we would not need to enumerate over all possible resource allocations for the defender.

5.3 Attacker Oracle

Similar to the defender oracle, the attacker oracle also uses an Actor-Critic method to train its neural network by updating parameters λ for its respective value function and policy.

With a game state s_i as input to the network, then, the attacker samples an action a_i from the mixed strategy the parameterized policy outputs, with the probability of the mixed strategy being represented by $\pi(a_i|s_i, \lambda_i)$. The outcome of playing a_i, which includes attacker utility u_i, is then used to update the policy parameters based on a learning rate α and the gradient of some function $J(*)$ that seeks to maximize the attacker's utility, giving $\lambda_{i+1} = \lambda_i + \alpha\nabla J(\lambda_i)$. The network update for the attacker oracle follows equation (5) for

its respective parameters.

One thing to note is that we do not need a compact representation for the attacker's action at each time step, since the attacker only attacks one target per time step. As such, the attacker's action space is defined by the number of targets in the game, and is not intractable. Therefore, the method for training the attacker oracle is much simpler than that for the defender oracle.

CHAPTER 6

6 EXPERIMENTAL OVERVIEW

In this section, we first outline the technical specifications of the game. Then, we discuss the states or observations that are input to the defender and attacker oracles respectively, as well as the details of the neural network architecture and training program for each of the oracles.

6.1 Game Representation

The following game setting features encode necessary information about the seSSG, which is naturally representable as a scale-free network graph. Specifically for our experiment, we have an *adjacency matrix*, which encodes the connectivity of the graph, as well as a *payoff matrix*, which encodes the utilities each player receives at each node of the graph depending on the outcome of the players' actions at a given time step.

6.1.1 Adjacency Matrix

The connectivity of the seSSG game setting is represented by a $T \times T$ matrix, which shows the ability of a player to move from one target to another. A fixed positive value in position ij of the matrix shows that the player cannot move from target t_i to target t_j. A negative value at position ij of the matrix represents a resource movement cost, showing the penalty for the defender to reallocate a resource from target t_i to target t_j. The cost to keep a resource at a target, i.e. moving resource from target t_i to target t_i at a given time step, is zero. This resource movement cost is meant to represent real-life security scenarios, where it takes time and energy for a security resource to be redistributed to another location apart from some set schedule. On the other hand, since the attacker's sequential attacks are represented as preemptively coordinated, they do not suffer any movement costs.

6.1.2 Payoff Matrix

The payoffs for each player in the seSSG game setting is represented by a $T \times 4$ matrix, which shows the rewards and penalties for the defender and attacker for each target location. The first column shows the rewards for the defender for defending each attacked target, while the second column shows the penalties for the defender if each attacked target is not defended. The third column shows the rewards for the attacker for attacking an undefended target, and the fourth column shows the penalties for the attacker for attacking

a defended target. The varying payoffs for each target is meant to represent real-life security scenarios, where various target locations have different priorities or values. An example would be the baggage claim at an airport, where high pedestrian traffic is located regularly and the location is widely accessible to the public, as opposed to an airline lounge.

6.2 Defender Oracle

The following subsections describe the representation of the input features of the defender oracle (DO).

6.2.1 Input Representation - Defender Observation

Recall that the defender has full observation of the game state. That is, at each time step, the defender knows the locations of all defender resources, as well as whether or not each target has been attacked (i.e. the history of sequential attacks). As such, we represent the defender observation as the game state:

- Game State: A $T \times 2$ matrix that shows the position of defender resources before making the next action. The first column indicates the number of defender resources at each target. For the second column, the defender observes the targets that have been attacked; a value of 0 shows that the target has not been attacked, while a value of 1 shows that a target has been attacked.

6.2.2 Output Representation

Using the A2C framework, the output of the DO is an *actor*, which represents the policy by which the defender samples an action, and a *critic*, which represents the value of the input state. Since we are not enumerating over all possible defender actions, the compact representation schema described in section 5.2 is reflected in the *actor* component of the output, as described below:

- Actor: A $T \times T$ matrix where each entry in position ij of the matrix represents the probability the defender moves a resource from target t_i to target t_j. Each row i of the matrix signifies the potential coverage of the defender for that target t_i, and therefore, each row i of the matrix must sum up to 1.

- Critic: A continuous value that represents the network's prediction of the value of the current game state.

6.3 Attacker Oracle

The following subsections describe the representation of the input features of the Attacker Oracle (AO).

6.3.1 Input Representation - Attacker Observation

Recall that the attacker does not have full knowledge of the game state; they know the defender mixed strategy but they do not know the exact resource allocation of the defender at each time step (i.e. the action that is sampled from the defender's mixed strategy). As such, the observation of the attacker updates only at the target they choose to attack at each time step, as described below:

- Partial Observation: A $T \times 2$ vector that shows whether or not targets have been defended. In the first column, a value of -1 shows that the target has not yet been observed, a value of 0 shows that the target has been observed during a time step to *not* have been defended, and a value of 1 shows that the target has been observed during a time step to have been defended. The second column is similar to the representation of the defender's observation, showing the targets that have been attacked. A value of 0 shows that the target has not been attacked, while a value of 1 shows that a target has been attacked.

6.3.2 Output Representation

Using the A2C framework, the output of the AO is an *actor*, which represents the policy by which the attacker samples an action, and a *critic*, which represents the value of the input state. Since the action space of the attacker corresponds to the number of targets, that is, it is not exponential, we do not require compact representation of the attacker's policy. Therefore, the output of the AO follows a standard A2C schema in the reinforcement learning context, as described below:

- Actor: A $1 \times T$ vector where each entry at position i represents the probability that the attacker will attack target t_i at a given time step. Therefore, this component represents the full policy for the attacker across all possible actions at a given time step.

- Critic: A continuous value that represents the network's prediction of the value of the current game state.

6.4 A2C Architecture for Oracles

The network architectures for the AO and the DO are mostly the same, excepting a couple of differences regarding the *actor* component of the output. Otherwise, each network architectures contains three sections.

The first section is a graph convolutional network (GCN), which incorporates connectivity information from the adjacency matrix. Since the seSSG setting is represented as a scale-free network graph, we encode the information of the state using graph convolutional techniques, which are generalized CNNs (Li et al., 2018). Our GCN contains two graph convolutional layers, each batch-normalized, as well as a feed-forward linear layer, all activated by Rectified Linear Units (ReLU).

The output of the GCN is then sent to two separate sections: the actor network and the critic network. The critic network contains two feed-forward linear layers, and is the same for the AO and the DO. The first of these layers is ReLU activated, and the second layer compresses the vector input to generate the state value.

The actor network for the both AO and the DO also contains two linear layers. The first layer is ReLU activated, but the process of the second layer is where the AO and the DO diverge. Since the output of the actor network of the DO is a $T \times T$ matrix, it is activated by a Softmax function over each row of the input vector. For the DO, then, the last layer generates a probability distribution across each target for all targets in order to produce a compact representation of the defender's policy. On the other hand, since the output of the actor network of the AO is a $1 \times T$ matrix, it is activated by a Softmax function over the entire vector space, thus generating a probability distribution over all targets.

For this experiment, we build the network architectures for the AO and the DO to be similar in order to test the efficacy of the Double Oracle algorithm for handling the exponential strategy space of the defender. Building the networks for the AO and DO differently could introduce noise into the experiment, since there would be more hyperparameters that affect the results of the Double Oracle algorithm. As such, we also train each algorithm by the same procedure, as discussed in the following section.

6.5 Oracle Training Algorithm

As explained in section 5.1.1, each oracle network is trained using the A2C framework. At each time step of each training episode, the *actor* and *critic* is generated by the oracle network. The player action is sampled from the *actor* policy, an opponent action is sampled from their set of best response policies, and those actions are executed to obtain the reward and the next state. Using the outcome, the loss is calculated, which involves the

log of the probability the player plays the sampled action as well as the advantage value. Then, the parameters of the oracle network are updated with the loss. This training procedure is represented compactly in the figure Algorithm 2.

6.6 Baselines

To compare the optimal solution returned by the Double Oracle algorithm, specifically the resulting mixed strategy of the defender, we also train one A2C network against a uniformly randomizing attacker and one A2C network against an attacker that plays according to subjective utility quantal response (SUQR).

The baseline A2C network trained against a uniformly randomizing attacker generates a defense strategy that is not based on any information from the attacker or the environment. In other words, this baseline network simulates a reactive defender that performs actions and updates strategies only according to the immediate situation. Similarly, the uniformly randomizing attacker does not perform actions based on information from the environment, providing adequate circumstances to generate a defender trained on negligible information. By comparing the mixed strategy from the Double Oracle algorithm against the uniformly randomized baseline, we contrast a reactive defender against one that is trained against many optimal attacker strategies.

For the second baseline, we generate a defense strategy trained against an attacker that plays according to SUQR (Nguyen et al., 2013). Basically, the attacker first takes into account the mixed strategy of the defender at a given time step, deriving the marginal probabilities that each target is covered by the defender. Then the attacker calculates their expected utility for each target according to their coverage probabilities, and determines their own mixed strategy based on those expected utilities. Simply put, the attacker calculates its subjective utility to simulate a boundedly rational player based on human behavior (i.e. quantal response). While the uniformly randomizing attacker performs actions despite any environmental information, the SUQR attacker simulates a malicious actor that has full knowledge of the defender strategies, allowing a defender to be trained against an attacker that responds to environmental changes. However, the SUQR attacker is still a reactive player, since it updates its strategy according to the current defender strategy. Comparing this baseline against the mixed strategy returned by the Double Oracle algorithm, then, contrasts a defender that plays one optimized policy with a defender that samples across many policies.

In order to compare the three resulting models, we have them play games against an optimal attacker, that is, the one returned by the Double Oracle algorithm. We then compare each of the models' resulting utilities across multiple simulated games.

Algorithm 2: A2C Reinforcement Learning for Each Player Oracle Given Opponent Policy Set OP

Initialize player oracle network W_θ
for *each episode* k **do**
    Initialize game state s
    Sample opponent policy $Z_k \sim OP$
    for *each time-step* i **do**
        Updated round information s.t. $s \rightarrow s_i$
        Get actor policy and critic value $x_i, v_i = W_\theta(s_i)$
        Sample player action $a_i \sim x_i$
        Generate opponent action z_i from $Z_k(s_i)$
        Execute actions a_i and z_i for s_i,
         observe player reward r_i and next state s_{i+1}
        Calculate actor loss:
         $\nabla\ log\ \pi_\theta(a_i|s_x)$
        Calculate critic loss:
         Get $v_{i+1} = W_\theta(s_{i+1})$
         $A_\theta(s_i, x_i) = (r_i + \gamma v_{i+1} - v_i) - v_i$
        Update oracle network parameters:
         $\theta \leftarrow \theta + \alpha[A_\theta(s_i, x_i)\ \nabla\ log\ \pi_\theta(x_i|s_i)]$

CHAPTER 7

7 RESULTS

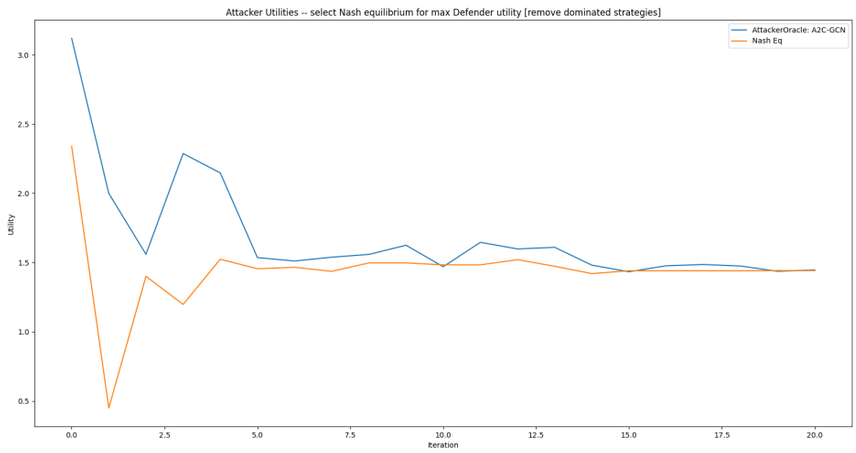
For this problem, we run the experiment with a small game, a mid-sized game, and a larger game. Experimenting with progressively larger games shows that the Double Oracle algorithm provides a scalable solution that is able to handle exponential action spaces. The following pages show our results for each of the game size settings.

7.1 Small Game

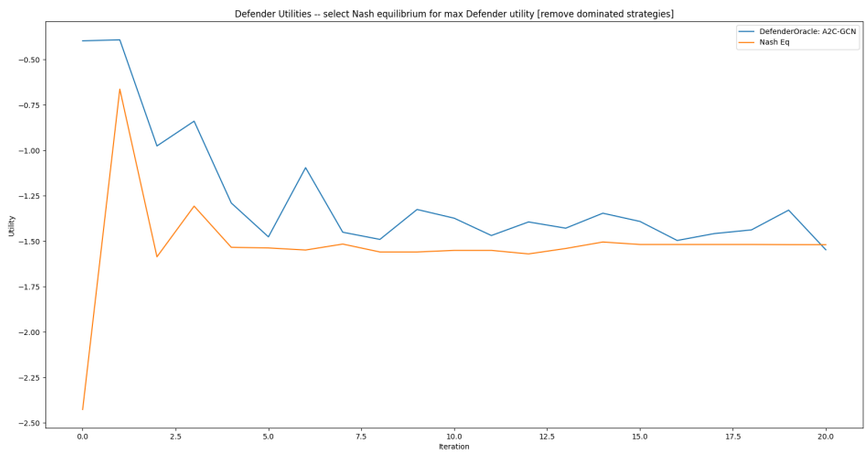
For the small game setting, there are 10 targets and 5 time steps, and the defender has 5 resources. While the small game is not trivial, it is often not applicable to real-world scenarios, in which there are many more targets and resources available.

In this section, we first show the plots of how the attacker oracle and the defender oracle converge their training utilities towards the Nash Equilibrium utilities to derive an optimal solution for the Double Oracle algorithm. Then, we compare the optimal defender solution with the network trained against a uniformly randomizing attacker and the network trained against a SUQR attacker by showing their utilities playing against an optimal attacker.

7.1.1 Double Oracle



(a) Attacker Oracle



(b) Defender Oracle

Figure 1: Oracle Convergence for 10-target Games

For the small game setting, the Double Oracle algorithm took 21 iterations to converge, with a running time of approximately 35 minutes.

7.1.2 Model Comparison

In the following plot, **MIX** labels the defender strategy returned by the Double Oracle algorithm, **RAND** labels the defender strategy trained against a uniformly randomizing attacker, and **SUQR** labels the defender strategy trained against a SUQR attacker.

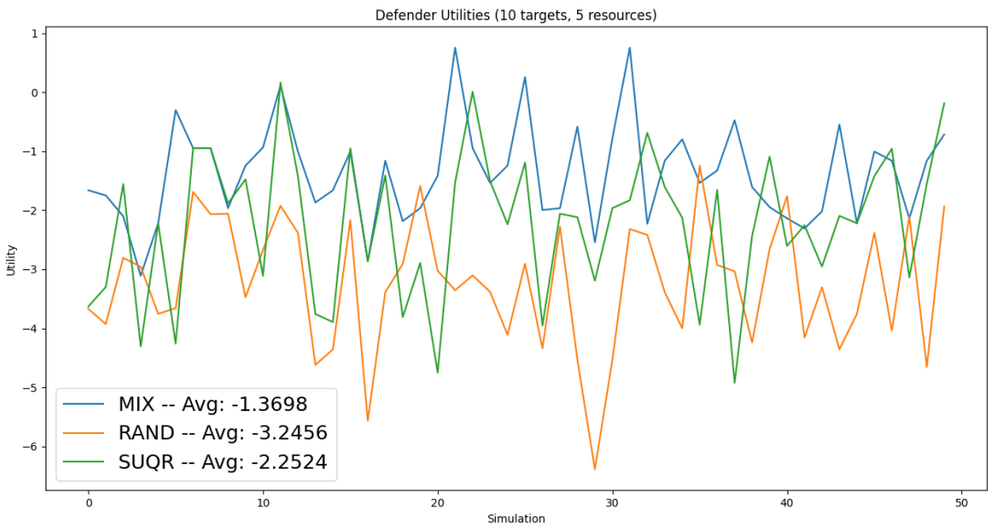


Figure 2: Comparison of Models against an Optimal Attacker for 10-target Games

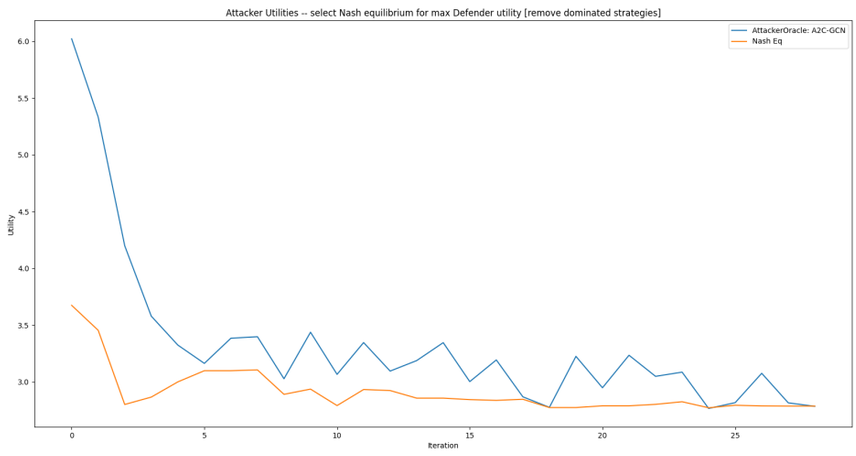
As shown in the average utilities in Figure 2, the defender strategy returned by the Double Oracle algorithm outperforms the baselines. Across 50 simulations, the average utility of the Double Oracle defender was -1.3698, while that for the randomly-trained defender and the SUQR-trained defender were -3.2456 and -2.2524 respectively. Therefore, the Double Oracle defender performed approximately twice as well as a SUQR-trained defender, and almost thrice as well as the randomly-trained defender.

7.2 Mid-sized Game

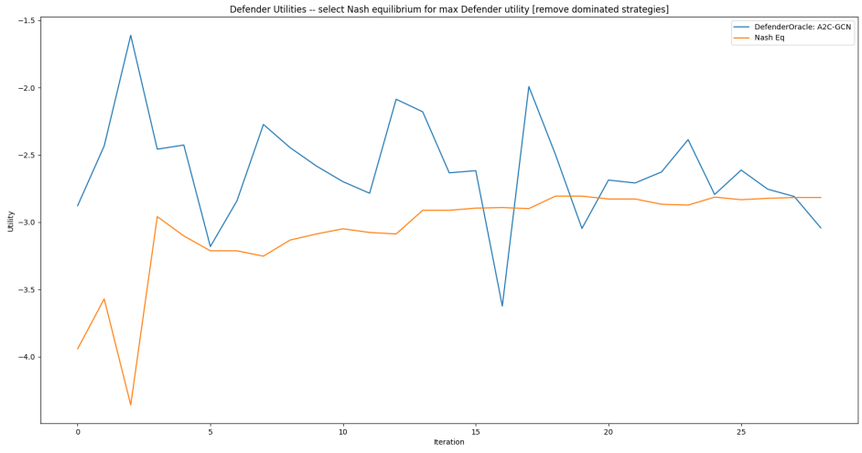
For the mid-sized game setting, there are 20 targets with 10 time steps, and 10 defender resources.

In this section, we first show the plots of how the attacker oracle and the defender oracle converge their training utilities towards the Nash Equilibrium utilities to derive an optimal solution for the Double Oracle algorithm. Then, we compare the optimal defender solution with the network trained against a uniformly randomizing attacker and the network trained against a SUQR attacker by showing their utilities playing against an optimal attacker.

7.2.1 Double Oracle



(a) Attacker Oracle



(b) Defender Oracle

Figure 3: Oracle Convergence for 20-target Games

For the mid-sized game setting, the Double Oracle algorithm took 29 iterations to converge, with a running time of approximately 3.7 hours.

7.2.2 Model Comparison

In the following plot, **MIX** labels the defender strategy returned by the Double Oracle algorithm, **RAND** labels the defender strategy trained against a uniformly randomizing attacker, and **SUQR** labels the defender strategy trained against a SUQR attacker.

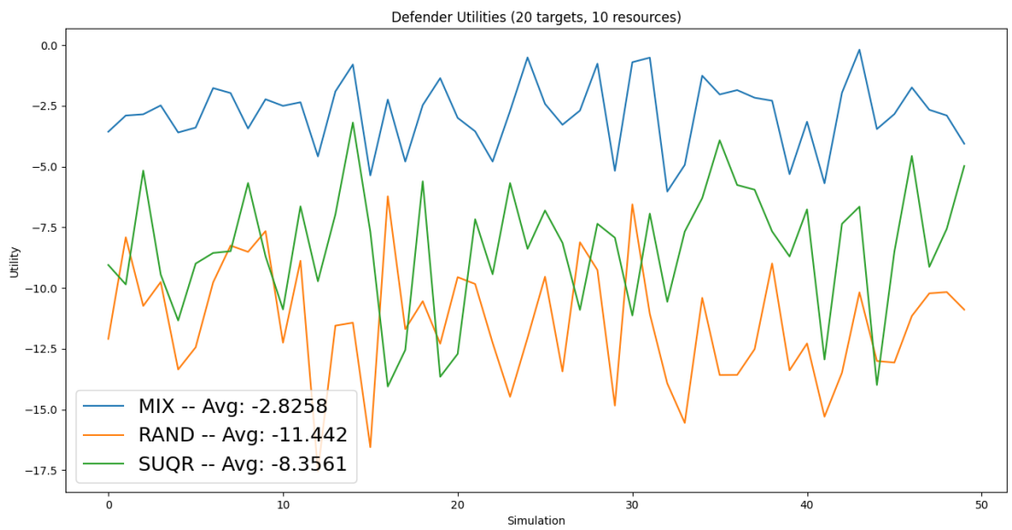


Figure 4: Comparison of Models against an Optimal Attacker for 20-target Games

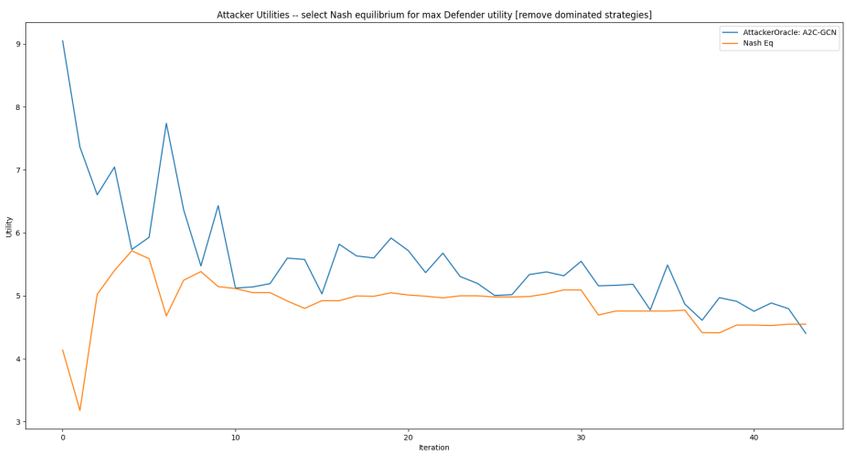
As shown in the average utilities in Figure 4, the defender strategy returned by the Double Oracle algorithm outperforms the baselines. Across 50 simulations, the average utility of the Double Oracle defender was -2.8258, while that for the randomly-trained defender and the SUQR-trained defender were -11.442 and -8.3561 respectively. Therefore, the Double Oracle defender performed approximately thrice as well as a SUQR-trained defender, and almost four times as well as the randomly-trained defender. This relative increase in performance of the Double Oracle defender, compared to smaller games, incorporates the increased number of attacks, but also suggests that the iterative process of the Double Oracle framework provides better results for more extensive games. Further experimentation would be required to test this theory.

7.3 Large Game

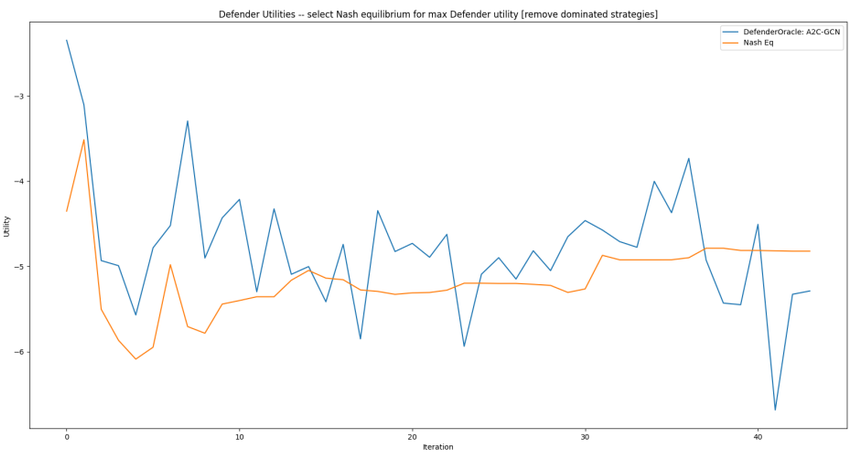
For the large game setting, there are 30 targets with 15 time steps, and 15 defender resources.

In this section, we first show the plots of how the attacker oracle and the defender oracle converge their training utilities towards the Nash Equilibrium utilities to derive an optimal solution for the Double Oracle algorithm. Then, we compare the optimal defender solution with the network trained against a uniformly randomizing attacker and the network trained against a SUQR attacker by showing their utilities playing against an optimal attacker.

7.3.1 Double Oracle



(a) Attacker Oracle



(b) Defender Oracle

Figure 5: Oracle Convergence for 30-target Games

For the large game setting, the Double Oracle algorithm took 44 iterations to converge, with a running time of approximately 18.4 hours.

7.3.2 Model Comparison

In the following plot, **MIX** labels the defender strategy returned by the Double Oracle algorithm, **RAND** labels the defender strategy trained against a uniformly randomizing attacker, and **SUQR** labels the defender strategy trained against a SUQR attacker.

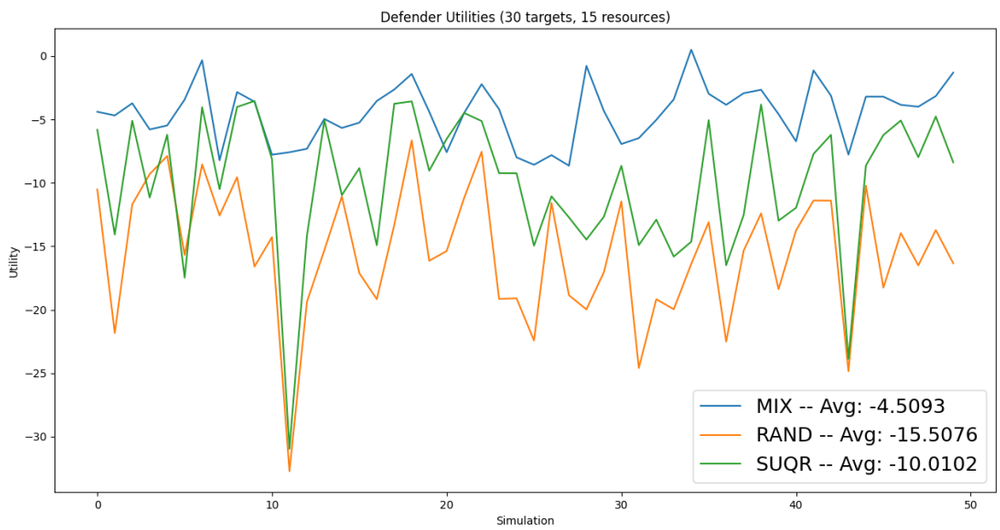


Figure 6: Comparison of Models against an Optimal Attacker for 30-target Games

As shown in the average utilities in Figure 6, the defender strategy returned by the Double Oracle algorithm outperforms the baselines. Across 50 simulations, the average utility of the Double Oracle defender was -4.5093, while that for the randomly-trained defender and the SUQR-trained defender were -15.5076 and -10.0102 respectively. Therefore, the Double Oracle defender performed approximately twice as well as a SUQR-trained defender, and about thrice as well as the randomly-trained defender. The performance of the Double Oracle defender, relative to the other models, is similar to that in the small game setting. This plateauing of performance increase suggests that while the Double Oracle framework is scalable, there is a cutoff point at which the benefits of the Double Oracle process will not outweigh training a defender with a random or SUQR-based attacker. Further experimentation would be required to test this theory.

CHAPTER 8

8 CONCLUSION AND FUTURE WORK

In this paper, we looked into the problem of computing an optimal strategy for the defender in sequential Stackelberg Security Games. Specifically, we tackled the issue of handling exponential action spaces by combining three approaches. First, we used marginal probabilities to represent defender actions instead of probabilities across all possible actions, which aided in reducing the computational intractability for the defender action space. We then leveraged the benefits of the Advantage Actor-Critic (A2C) framework (Mnih et al., 2016), which generates both the value function of the game as well as its corresponding best response policy through reinforcement learning principles. Wrapped around these two approaches was the Double Oracle algorithm (Jain et al., 2011), which allowed us to iteratively search through best response policies for the attacker and defender by playing them against each other during each iteration, ultimately finding optimal equilibrium solutions for each player.

The results of the experiments showed that the defender strategy returned by the Double Oracle outperformed the proposed baselines. However, more experimentation is needed to confirm how the increase of performance is related to specific aspects of our main contributions.

Furthermore, using marginal probabilities to represent defender actions does not allow the imposition of nonlinear constraints on defender resources. For example, in real-life scenarios, certain resources (e.g. a security officer and a medic) could be required to be located at neighboring targets for emergency purposes. Thus, the model would need to enforce as well as learn how to generate valid actions according to those constraints. As yet, our A2C model is unable to do this. Future work using Generative Adversarial Networks (GANs) and Normalizing Flow models could be done to explore this scenario further. Otherwise, we are pleased to report promising results from using the methods proposed in this paper to tackle exponential action space for sequential Stackelberg Security Games.

9 REFERENCES CITED

Fang, F., Nguyen, T. H., Pickles, R., Lam, W. Y., Clements, G. R., An, B., Singh, A., Tambe, M., and Lemieux, A. (2017). Deploying paws: Field optimization of the protection assistant for wildlife security. *Proceedings of the Twenty-Eighth AAAI Conference on Innovative Applications*.

Jain, M., Korzhyk, D., Vanek, O., Conitzer, V., Pechoucek, M., and Tambe, M. (2011). A double oracle algorithm for zero-sum security games on graphs. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*.

Karwowski, J. and Mandziuk, J. (2020). Double-oracle sampling method for stackelberg equilibrium approximation in general-sum extensive-form games. *Decision and Game Theory for Security, GameSec 2020, Lecture Notes in Computer Science*.

Li, R., Wang, S., Zhu, F., and Huang, J. (2018). Adaptive graph convolutional networks. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

McMahan, H. B., Gordon, G. J., and Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. *Proceedings of the Twentieth International Conference on Machine Learning*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*.

Nguyen, T. H., Yadav, A., Bosansky, B., and Liang, Y. (2019). Tackling sequential attacks in security games. *Decision and Game Theory for Security - 10th International Conference, GameSec 2019, Proceedings*.

Nguyen, T. H., Yang, R., Azaria, A., Kraus, S., and Tambe, M. (2013). Analyzing the effectiveness of adversary modeling in security games. *Proeedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Pita, J., Jain, M., Ordonez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., and Kraus, S. (2009). Using game theory for los angeles airport security. *Association for the Advancement of Artificial Intelligence*.

Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., and Meyer, G. (2012). Protect: A deployed game theoretic system to protect the ports of the united states. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*.

Sinha, A., Fang, F., An, B., Kiekintveld, C., and Tambe, M. (2018). Stackelberg security games: Looking beyond a decade of success. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction, 2nd edition*. MIT Press.

Tomasek, P., Bosansky, B., and Nguyen, T. H. (2020). Using one-sided partially observable stochastic games for solving zero-sum security games with sequential attacks. *Decision and Game Theory for Security, GameSec 2020, Lecture Notes in Computer Science*.

Tsai, J., Rathi, S., Kiekintveld, C., Ordonez, F., and Tambe, M. (2009). Iris - a tool for strategic security allocation in transportation networks. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*.