Stable Profiles in Simulation-Based Games via Reinforcement Learning and Statistics

by

Mason Wright

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2019

"Life is short,
and Art long;
the crisis fleeting;
experience perilous,
and decision difficult."

Hippocrates *(Francis Adams, trans.)*

Mason Wright
masondw@umich.edu
ORCID iD: 0000-0003-2723-3309

To Josephine*, for her love and support throughout my graduate study*

Acknowledgements

My graduate work would have been less fruitful without the guidance of my advisor, Michael P. Wellman. Mike was recommended to me by previous graduates of his research group, and their high esteem for him matches my own experience. Mike has been a wonderful teacher and manager who helped me to learn about good research practices, and who shows by example how to collaborate effectively with others. I have Mike to thank for much of what I learned in graduate school.

I would like to thank the other committee members, Grant Schoenebeck, Demosthenis Teneketzis, and Jenna Wiens. I have had the pleasure of working together with Demos and Jenna in research projects at Michigan, and I learned a lot from these collaborations. The feedback and advice I received on my thesis proposal have helped me to improve the thrust of my research and hopefully make it more relevant to others.

My fellow Strategic Reasoning Group members helped me in many ways during graduate school, including showing me the ropes of the degree process and providing camaraderie. I would like to thank those I was able to collaborate with directly, including Elaine Wah, Thanh H. Nguyen, and Arunesh Sinha. I learned a great deal by working with each of them. I am also grateful to everyone in the lab who gave me advice, a good example to follow, or friendship over the years, including Erik Brinkman, Bryce Wiedenbeck, Travis Martin, Frank Cheng, Xintong Wang, and Megan Shearer.

I pursued summer internships at Microsoft with the Bing Ads Relevance and Revenue team, where Patrick R. Jordan, a graduate of Mike's lab, was my manager. Patrick gave me a fantastic introduction to applied advertising in industry, besides acting as a management role model. I am also thankful to my internship mentors, Bach Q. Ha and Mohammad Reza Khani, for their patience and support as I ramped up each summer.

A few years before enrolling in graduate school, I had not been sure I wanted to pursue academic research, but my advisor Pratim Sengupta helped me find the path to where I am today. I want to thank Pratim for being my first academic mentor, and for being among those who most influenced my progress as a software developer and researcher.

My family has always encouraged me in my academic career, and without them I would literally not be here now. I thank my parents, Jeff and Karen, along with my sister, Melissa, for their support. I also thank my grandparents for their long-time encouragement of my education.

Finally, thanks to Josephine, who has been beside me throughout the graduate program.

Contents

List of Figures

List of Tables

Abstract

In environments governed by the behavior of strategically interacting agents, game theory provides a way to predict outcomes in counterfactual scenarios, such as new market mechanisms or cybersecurity systems. Simulation-based games allow analysts to reason about settings that are too complex to model analytically with sufficient fidelity. But prior techniques for studying agent behavior in simulation-based games lack theoretical guarantees about the strategic stability of these behaviors.

In this dissertation, I propose a way to measure the likelihood an agent could find a beneficial strategy deviation from a proposed behavior, using a limited number of samples from a distribution over strategies, including a theoretically proven bound. This method employs a provably conservative confidence interval estimator, along with a multiple test correction, to provide its guarantee. I show that the method can reliably find provably stable strategy profiles in an auction game, and in a cybersecurity game from prior literature.

I also present a method for evaluating the stability of strategy profiles learned over a restricted set of strategies, where a strategy profile is an assignment of a strategy to each agent in a game. This method uses reinforcement learning to challenge the learned behavior as a test of its soundness. This study finds that a widely-used trading agent model, the zero-intelligence trader, can be reasonably strategically stable in continuous double auction games, but only if the strategies have their parameters calibrated for the particular game instance.

In addition, I present new applications of empirical game-theoretic analysis (EGTA) to a cybersecurity setting, involving defense against attacker intrusion into a computer system. This work uses iterated deep reinforcement learning to generate more strategically stable attacker and defender strategies, relative to those found in prior work. It also offers empirical insights into how iterated deep reinforcement learning approaches strategic equilibrium, over dozens of rounds.

Chapter 1

Introduction

Many vital real-world domains of study are governed by strategic interactions among agents. Such environments include stock markets, where traders compete to attain their desired holdings of a stock for the best prices, and cybersecurity settings, where attackers and defenders vie for control of computer networks. Analysts model these settings as *games* and use the tools of game theory to reason about the expected outcomes of agents' self-interested behavior.

Real-world settings often involve many agents or many possible strategies, and the payoffs for each agent are determined by a complicated function of the agents' chosen strategies. In such settings, it may be that the only way to determine each agent's payoff is to run a simulator of the game; such game models are known as *simulation-based games*. Analysts can estimate what will happen in a simulation-based game by running a simulator under different mappings from agents to strategies. Analysts use the observed payoff distributions to build a model of the function from strategies to payoffs. Then analysts can use standard game-theoretic methods to determine which agent behaviors are *strategically stable* in the game.

An assignment of a strategy to each agent is strategically stable if it is at or near equilibrium under some solution concept; for example, such an assignment may be called strategically stable if no agent can increase its expected payoff by switching strategies, while every other agent retains its current strategy. Strategically stable assignments of strategies to agents are important in game theory, because they represent rational behaviors for the agents, and consequently justifiable models or predictions of how real agents might behave in such a setting.

One problem in simulation-based game theory is that only a limited number of strategies can be explored by querying the game simulator. Current methods of empirical game-theoretic analysis (EGTA) build a game model for a modestly-sized strategy set, containing perhaps between 10 and 100 strategies, and search for a mapping from players to strategies that is strategically stable over that set [94], [95]. Consider a game of interest, called the *original game* within a study, which may have infinitely many possible strategies; the EGTA approach would be to solve a *restricted game* containing only a finite subset of these strategies. The trouble is that this approach cannot verify the resulting strategies are stable in the original game, where the strategy set is not artificially restricted to just a few options. It would be desirable to have methods for quantifying the stability of strategies with respect to a much larger strategy space. Such methods would help analysts to reason about whether a given strategy is a reasonable model of player behavior in the original game.

1.1 Contributions

In this work, I present approaches for analyzing the stability of strategies in simulation-based games, whose strategy sets may be very large or even infinite. I present methods that allow the analyst to search for stable assignments of strategies to agents in the original game, not merely in a restricted version of that game. I also offer a method to evaluate the stability of an assignment of strategies to agents, with respect to deviating strategies from a very large strategy set.

I present a statistical approach that searches for better assignments of strategies to agents in a game, with the goal of yielding a proven stability property in cases where the process converges within an iteration limit. When it terminates successfully, this method provides a statistical guarantee on the stability of the resulting strategy profile, which I call *probably almost stable*. This method could be especially useful in simulation-based games for which the analyst has restricted agents to using a parameterized class of strategies. The algorithm I propose can search for a strategically stable assignment of strategies to the agents in the game, such that if the algorithm terminates successfully, it is likely that the measure of parameter settings that are beneficial deviations from this assignment is small, under some probability distribution over all strategies in the parameter space.

Another approach uses reinforcement learning (RL), a form of machine learning for maximizing the rewards an agent obtains from its actions, to attempt to find a beneficial strategy deviation against a strategy profile that is presumed to be stable. I use RL in one study as a tool for evaluating the strategic stability of an assignment to agents of strategies from a restricted class, with respect to alternative strategies from a much broader class, such as the set of all strategies representable by a form of tabular policy. In another study, I use RL as a strategy exploration method, while iteratively searching for a stable assignment of strategies to agents in a complex game. In this work, RL is used to find strategies that deviate beneficially from the set of strategies explored already, until RL fails to yield a beneficial deviation, and the process converges. I present experimental evaluations of these methods in simulation-based games representing meaningful, real-world settings, including the continuous double auction and a cyber-defense scenario.

In one application of RL, I use RL to evaluate the strategic stability of a class of strategies, specifically a widely-studied heuristic trading strategy, called *Zero Intelligence*, for the continuous double auction. I consider a setting where zero-intelligence trading agents have been calibrated to the particular game environment, such that no zero-intelligence strategy among the instances for which simulations have been conducted is a beneficial deviation. I then measure how much higher an expected payoff is achieved by an RL agent, when all other agents continue to use the calibrated zero-intelligence strategies. This method seeks to evaluate the strategic stability of an assignment of strategies to agents, with respect to the large class of strategies that could be learned by an RL algorithm. The findings suggest that zero-intelligence strategies are reasonably strategically stable in the continuous double auction, but only if their parameters are calibrated to the particular game instance.

My other application of RL to simulation-based games employs iterated deep RL to seek strategically stable assignments of policies to agents, for complex games that are played sequentially over multiple moves. Deep RL refers to RL combined with deep neural networks, where a neural network is used as a function approximator that predicts the value of actions or states in a game. Iterated deep RL is a process which repeatedly (a) finds an assignment of known strategies to agents, which is strategically stable with respect to other known strategies, and (b) uses deep RL for each agent to search for a beneficial deviation from the current strategy assignments. I show that iterated deep RL can consistently generate beneficially deviating strategies, in a cybersecurity game from prior literature. Moreover, I show that iterated deep RL converges to strategies that are stronger than those suggested in previous work for this cybersecurity game, via mostly steady improvements that gradually diminish from round to round.

1.2 Related work

Many prior works have addressed problems related to those that I take on in this dissertation. Here I provide an overview of related works and where they stand in relation to this work.

1.2.1 Probably almost stable profiles

The probably almost stable guarantee I propose is related to concepts introduced by Bopardikar et al. [7]. Bopardikar et al. propose a method for learning *security policies*, which are policies such that following the policy guarantees an agent will be likely to obtain at least some minimum payoff against an opponent that selects a best-response strategy based on a limited random search. The method proposed in that prior work is intended for zero-sum games, where a minimax solution concept is used, and may not apply directly to general-sum games, which are handled by my probably almost stable method.

The algorithm I propose for searching for probably almost stable profiles is a variant of the empirical game-theoretic analysis (EGTA) process of Wellman [95]. The EGTA method searches for stable profiles in games by finding a Nash equilibrium over a limited strategy set, then iteratively exploring new strategies that might be added to this set. Other authors, such as Sureka and Wurman [77], have also proposed methods for iterated better-response search, similar to my method of searching for probably almost stable profiles. My new methods introduce the theoretical concept of almost-stable and probably almost stable profiles, as well as extending the basic process of EGTA or iterated better-response search, in order to ensure the results have the desired property (if the process terminates successfully).

1.2.2 Continuous double auction agents and reinforcement learning

The continuous double auction market simulator I use, as well as the implementation of the zero-intelligence trading strategy, were derived from prior works by Wah et al. [89], [90]. In those works, the strategies of zero-intelligence agents were selected from a small set of parameter settings, and Nash equilibrium strategy profiles were found for the case where agents were limited to those parameter choices. I extend that prior work by training agents to behave adaptively, depending on the current state of the market, using RL;

the fitness of the trained policies can then be compared to that of zero-intelligence agents with carefully tuned parameters.

Schvartzman and Wellman [71] previously trained reinforcement learners to trade in the continuous double auction, against a variety of agent types from the literature. They showed that an agent trained with RL for a particular market environment could outperform several previously proposed agent strategies, including some that are quite sophisticated. My work differs in goal from Schvartzman and Wellman: while their work attempted to demonstrate that a reinforcement learner could outperform previously proposed strategies in the continuous double auction, I measure the regret, or loss, an agent incurs by adopting a fixed but well-calibrated zero-intelligence strategy, relative to an agent that is free to use an adaptive strategy produced via RL. My work seeks to evaluate the fitness of zero-intelligence for the continuous double auction, by comparing a well-parameterized zero-intelligence agent to a learning agent from a strategy class with much more expressive power.

1.2.3 Iterated deep reinforcement learning in cybersecurity games

Various prior works have proposed or experimented with methods for iteratively learning more strategically stable strategies in complex games. Lanctot et al. [49] introduced Policy Space Response Oracles (PSROs), which alternate between finding an optimal policy over a limited strategy set, based on some solution concept, and using deep RL to explore new deviating strategies. My work differs from Lanctot et al. in the methods used to control the problem of overfitting to the training opponent; in particular, my work proposes pretraining a reinforcement learner to perform well against the current opponent, then fine-tuning against a mixed strategy of previous opponents. Moreover, my work also presents detailed results related to the dynamics of the training process, over dozens of rounds of iterated deep RL and Nash equilibrium solving.

Wang et al. [92] applied iterated deep RL to a green security game, where rangers play an extensive-form game against poachers. Wang et al. introduce a method for encouraging the learned policies to avoid overfitting to a particular opponent, much like my approach, although the methods they propose are different. Moreover, as noted above, my results illustrate the dynamics of changes in strategy during training over many rounds, until convergence is reached after as many as 70 iterations; previous works have generally performed only enough rounds of iteration to demonstrate the efficacy of the procedure, stopping after perhaps 5–20 rounds, even if deep RL continues to learn beneficially deviating strategies.

My work on iterated RL has similarities to prior work by Schvartzman and Wellman [71]. Their paper applied iterated RL to develop stronger trading agent strategies for the continuous double auction. Schvartzman and Wellman did not use neural networks to represent their policies, instead using tabular Q-learning with tile coding. Their procedure converged, being unable to learn another beneficial deviation, after 14 rounds, while with the stronger learning algorithm of deep Q-networks, I observe that as many as 70 rounds are needed in some cases for convergence. There are also considerable differences between these works due to the difference in game environment between an auction among many trading agents and a two-player cybersecurity game.

1.3 Outline of document

In the remainder of this dissertation, I lay out fundamental background knowledge, present three related studies, and sum up the results obtained. Chapter 2 introduces key ideas and notation used in the dissertation, on topics including game theory, reinforcement learning, auction games, and attack-graph games. Chapter 3 presents a study on using a statistical confidence bound and random strategy sampling to analyze the stability of a strategy profile, yielding the probably almost stable guarantee mentioned above; this work also includes applications to the first-price, sealed-bid auction and a cybersecurity game. Chapter 4 presents work on evaluating the stability of strategy profiles using reinforcement learning, with an application to the continuous double auction, as well as the zero-intelligence strategies described above; a paper on this study appeared at AAMAS 2018 [97]. Chapter 5 presents results from applying iterated deep reinforcement learning to a cybersecurity setting, an attack-graph game. Appendix A has supplementary data that would be useful in replicating this work in deep reinforcement learning, and also additional figures giving a more complete picture of the experimental results across different environments. Finally, Chapter 6 discusses the results from the dissertation and concludes.

Chapter 2

Background

In this chapter, I present the foundations of this dissertation in game theory and reinforcement learning, including notation and core concepts. These concepts include game-theory ideas like Nash equilibrium, regret, and empirical game-theoretic analysis (EGTA).

I go on to introduce the main application areas addressed by this work: the continuous double auction (CDA) and the attack-graph game. Next I give a formal definition of each of these game-theoretic domains. I also provide information on the real-world motivation for studying these settings, as well as some conceptual description to supplement the formal definitions and provide an easier path to understanding the various game models.

2.1 Game theory

Game theory is the study of strategic interactions among self-interested agents. By self-interested, I mean that each agent makes an independent decision of how to act, with the goal of optimizing its own expected outcome. As these agents interact, each agent's outcomes may depend on the behavior of the others, giving rise to a strategic game.

I formally describe an N-player *game* as $G = (S, U)$, where $S = (S_1, \ldots, S_N)$ assigns each agent i a *strategy set*, and $U = (U_1, \ldots, U_N)$ assigns each agent a *utility function*. (There is an implicit *agent set* of players indexed by $i \in \{1, \ldots, N\}$.)

I consider *normal form* games in this work. Each agent i can simultaneously, independently select a *pure strategy* $s_i \in S_i$. This results in a *pure-strategy profile* $s = (s_1, \ldots, s_N) \in S_1 \times \cdots \times S_N$. Alternatively, an agent i can select a *mixed strategy*, which is a probability distribution over pure strategies in its strategy set, $\sigma_i \in \Delta_{S_i}$. (By Δ_{S_i}, I mean the set of all probability distributions over set S_i.) When each agent selects its own mixed strategy, the result is a *mixed-strategy profile* $\sigma = (\sigma_1, \ldots, \sigma_N) \in \Delta_{S_1} \times \cdots \times \Delta_{S_N}$. I sometimes discuss the *other-agents profile* in pure strategies, s_{-i}, or in mixed strategies, σ_{-i}, which indicates a strategy for each agent except i.

The utility function of any agent i is $U_i : \prod_j S_j \to \mathbb{R}$, which maps any pure-strategy profile to the expected payoff for agent i when each agent selects the strategy in that profile.

One can compute the expected payoff for any agent i of a mixed-strategy profile σ, given the utility function U_i. This involves taking an expectation of the payoff for i, over all pure-strategy profiles s that are played with positive probability, based on the mixed-strategy profile σ:

$$\begin{aligned} U_i(\sigma) &= \sum_{s \in S_1 \times \cdots \times S_N} \left(\prod_j \sigma_j(s_j) \right) \times U_i(s) \\ &= E_{s \sim \sigma} U_i(s). \end{aligned}$$

In practice, one can speed up this computation by summing over only those pure-strategy profiles s that are in the *support* of mixed-strategy profile σ. The support of a mixed strategy is the set of all pure strategies played with positive probability; similarly, the support of a mixed-strategy profile is the set of all pure-strategy profiles played with positive probability.

A *deviating strategy* or *deviation* from a baseline strategy profile σ is a mixed strategy σ_i' for some agent i that differs from the agent's strategy under σ. Sometimes I use the term "deviation" more narrowly, to indicate a pure strategy s_i' for some agent i, such that s_i' is not in the support of the agent's strategy in baseline strategy profile σ.

To evaluate the reasonableness of a deviation, one needs to compute the expected value of a new pure strategy s_i' for an agent i, when other agents play as in some other-agents mixed-strategy profile σ_{-i}. This can be done by taking the expected payoff for i over all other-agents pure-strategy profiles in the support of σ_{-i}:

$$\begin{aligned} U_i(s_i', \sigma_{-i}) &= \sum_{s_{-i} \in S_{-i}} \left(\prod_{k \neq i} \sigma_k(s_k) \right) \times U_i(s_i', s_{-i}) \\ &= E_{s_{-i} \sim \sigma_{-i}} U_i(s_i', s_{-i}). \end{aligned}$$

Above, the term S_{-i} is a shorthand for $\prod_{j \neq i} S_j$. In addition, I write $U_i(s_i, s_{-i})$ to indicate $U_i(s)$ where $s = (s_i, s_{-i})$.

A strategy s_i' for an agent i is called a *beneficial deviation* from mixed-strategy profile

σ, if $U_i(s'_i, \sigma_{-i}) > U_i(\sigma)$. More specifically, a strategy s'_i for agent i is a *best response* to other-agents profile σ_{-i}, if $s'_i \in \arg\max_{s_i \in S_i} U_i(s_i, \sigma_{-i})$.

In the special case of two-player games, I sometimes discuss *zero-sum games*. A zero-sum game has payoffs that always sum to zero, both in expectation and in each realized outcome. In a two-player, zero-sum game, for all pure-strategy profiles s, $U_1(s) = -U_2(s)$. Games that are not zero-sum are known as *general-sum games*.

The *regret* of a profile σ is defined as the maximum that any agent i can gain in expected payoff by deviating unilaterally to an alternative strategy s'_i. Formally, the regret of σ is

$$\max_{i \in \{1,\ldots,N\}} \max_{s_i \in S_i} U_i(s_i, \sigma_{-i}) - U_i(\sigma).$$

A *Nash equilibrium* of a game is a profile σ where no player i can increase its expected payoff by deviating unilaterally to any alternative strategy s'_i. In other words, a profile σ is a Nash equilibrium if and only if its regret equals zero.

I sometimes discuss the *regret of an agent*, which is the maximum that particular agent i can gain in expectation by deviating unilaterally from σ to a new strategy,

$$\max_{s_i \in S_i} U_i(s_i, \sigma_{-i}) - U_i(\sigma).$$

In settings where there are many available strategies, perhaps infinitely many, it is often convenient to analyze a *restricted game*, where only a subset of those strategies are available. Given a base game $G = (S, U)$, one can define a restricted game $G' = (S', U')$, where for any agent i, $S'_i \subseteq S_i$. The derived utility function U' is identical to the original U, except that it is defined only over the domain where agents play strategies in S'. (The advantage of a restricted game with a small, finite strategy set is that one can always solve for Nash equilibria of such a game.)

In restricted games, I sometimes refer to a policy's *regret with respect to an enlarged strategy set*. For example, suppose there is a base game $G = (S, U)$, its restricted game $G' = (S', U')$, and a profile σ that is a Nash equilibrium in the restricted game. The regret of σ with respect to the base game G is the maximum that any agent i could gain in expectation by deviating to a strategy in S_i. (Note that by definition, $U'_i(\sigma) = U_i(\sigma)$.)

The auction games analyzed in this work are *symmetric games*. By definition, in a symmetric game, all agents i and j have the same strategy set, $S_i = S_j$. In addition, the expected payoff for an agent i playing any strategy s against an other-agents profile σ_{-i} does not depend on the agent's identity: $U_i(s, \sigma_{-i}) = U_j(s, \sigma_{-i})$. In a sense, all agents share a common utility function U. Further, the payoff for an agent i against an other-agents strategy profile σ_{-i} does not depend on *which* other agents play each strategy, but only on *how many* other agents play each strategy. That is, if for two other-agent profiles σ_{-i} and σ'_{-i}, the count of other agents playing each mixed strategy is identical, then $U(s, \sigma_{-i}) = U(s, \sigma'_{-i})$.

Often in symmetric games one searches for *symmetric Nash equilibria*. A profile σ in a symmetric game is a *symmetric profile* if every agent plays the same mixed strategy: for all agents i and j, $\sigma_i = \sigma_j$. A symmetric Nash equilibrium is simply a Nash equilibrium profile σ that is symmetric.

2.2 Empirical game-theoretic analysis

Analysts sometimes study a game where, due to the game's complexity, running a simulator is the only apparent practical way to estimate the expected payoff for each agent of a pure-strategy profile. Examples of this type of game include some stock market games, cybersecurity games, and trading agent competitions. For instance, in a stock market, agents place orders in the market sequentially, which interact via complicated market rules, such that the only way to determine the outcome of multiple agents' interacting strategies is to enact or simulate their outcome in the market. In games like these, analysts often use an environment simulator to determine the payoffs for interacting strategies, in a *simulation-based game* approach. The payoffs of a simulation-based game can be sampled by analysts from a noisy simulator of the underlying utility function, $\mathcal{S} : \prod_i S_i \to \mathbb{R}^N$. The simulator takes any pure-strategy profile s and returns a real-valued payoff for each agent i. These payoffs can be nondeterministic, being sampled from some distribution over $\mathbb{R}^N$ conditioned on the profile s. Naturally, there can be statistical dependencies between the payoffs of the different agents. I do require, however, that each payoff vector be sampled i.i.d. if the game is run repeatedly. The (deterministic) utility function, then, is the expected value of each agent's payoff from the simulator, given a strategy profile.

In the work presented in this dissertation, I search for Nash equilibria of simulation-based games that have noisy payoff samples. The methods for finding such equilibria are part of the field of *empirical game-theoretic analysis*, or EGTA [94], [95]. EGTA research has produced many tools for analyzing simulation-based games, and here I discuss only the components of EGTA that are most important for understanding this work. Foremost among the EGTA-related tools I used that are not explained below, are deviation-preserving reduction [96], a method that makes analysis of games with many similar players more tractable; and EGTAOnline [12], a platform for managing EGTA experiments.

As an example of EGTA in action, suppose there is a simulation-based game $G = (S, U)$ with 3 players, where the payoffs are accessible only by sampling a noisy simulator $\mathcal{S}$. (In practice, it might be necessary to design and program software for the payoff simulator.) Because each agent's strategy set S_i is continuous, it may be useful to analyze a restricted game $G' = (S', U')$ with 10 strategies per agent.

The first step in the EGTA process is to sample payoffs from simulator $\mathcal{S}$ for many pure-strategy profiles s. Because the payoff samples are noisy, the analyst typically collects many samples for $p_s \sim \mathcal{S}(s) \in \mathbb{R}^N$ for each profile s under study. The greater the number of samples collected per profile, the more accurate the resulting model of the underlying utility function is.

After collecting payoff samples for various profiles s, written as $p_s \sim \mathcal{S}(s)$, one can estimate the expected payoff for each agent of profile s, by taking the sample mean $\bar{p}_s$. Combining the estimated expected payoffs for many profiles, one can construct an *empirical game* $G'' = (S', U'')$. In G'', $U''(s) = \bar{p}_s$ by definition. Note that payoffs in the empirical game are known only for that subset of profiles of the restricted game G' where samples have been collected already, and these payoff estimates may change as more samples are collected.

Even if the empirical game G'' has some missing payoffs, it is still often possible in practice to find Nash equilibria of G''. In such a case, the equilibrium profile must be an equilibrium of the restricted game G' also, in the infinite-sample limit where sampling error vanishes. For example, if a profile s has been sampled in empirical game G'', and every unilateral deviation to a pure strategy from s has been sampled as well, then if none of these deviations is beneficial in G'', then s must be a Nash equilibrium of G''. Thus, it can be established that s is a Nash equilibrium of G'', even without sampling any other profiles of G''. If one has collected enough samples of each such profile for the sampling error to be small, one can be almost sure that s is an equilibrium of the restricted game G' as well.

There are many possible methods for determining the order in which to sample different profiles s, but in practice I use a heuristic developed by Erik Brinkman to automate that decision [8]. Essentially, the heuristic performs a gradually expanding search for Nash equilibrium profiles, by sampling only those profiles that represent a unilateral deviation from a current candidate profile, until that candidate has been rejected for not being an equilibrium, or confirmed as an equilibrium of the restricted game.

For example, given a pure-strategy candidate profile s, it is necessary and sufficient to estimate the payoff of s, and of every s' such that for some agent i, $s'_i \neq s_i$, and for all $j \neq i$, $s'_j = s_j$ (that is, s' is some unilateral deviation from s).

The last stage of analyzing the restricted game G' is to search for a Nash equilibrium in the empirical game G'' that has been constructed. If an equilibrium is found, that equilibrium profile σ is returned and the procedure terminates. Otherwise, the search policy can be used to continue exploring new profiles s of G'. In the limit where all profiles have been explored, one is guaranteed to find some Nash equilibrium in a finite-strategy restricted game.

A mixed strategy σ_i for an agent i is called an *equilibrated* strategy if that mixed strategy is equal to the agent's strategy in some Nash equilibrium σ of a game G. For example, at the end of the EGTA process, a Nash equilibrium profile σ in the restricted game must contain an equilibrated strategy for each agent.

2.3 Reinforcement learning

Reinforcement learning (RL) is a domain within machine learning, that considers a framework in which an agent earns rewards based on its actions within an environment; the agent seeks a mapping, from its history of actions and observations to its action set, that maximizes the agent's expected future rewards [78]. I use the RL framework to model the strategic problem for each agent in a strategic setting like a stock market. (Note that in some cases analysts treat the environment as Markovian and learn a mapping from the current state to the action set, instead of from the history to the action set. Moreover, RL techniques can be applied to games by treating one agent at a time as the learner and holding the strategies of the other agents fixed during training only, or by simultaneously training multiple learners.) Formally, in an RL problem, a single agent chooses a *policy* π that maps a history of actions and observations to the next action a of the agent; the agent obtains a reward r_t at each discrete time step t based on the state-action pair (s, a) at that time.

2.3.1 Markov decision processes

Generically, the setting for RL can be formulated as a Markov decision process (MDP), if one assumes that the current state of the environment is fully observable by the agent. (There are a few more assumptions built into the MDP formulation, such as memorylessness, but the model is still fairly general.)

An MDP can be written as a tuple $M = (S, A, T, R, \gamma)$, with a state space S, action space A, transition function T, reward function R, and discount factor γ.

The environment's current status is specified by the *state* $s \in S$, where S is the set of all possible states. The state evolves over discrete time steps $t \in \{1, \ldots, T\}$, with the state at time t written as s_t.

At each time step t, the agent observes the current state s_t and selects an action $a_t \in A$, where A is the action set. This decision is based on a policy $\pi : S \to A$, which the agent must determine before the playout begins.

Given a state-action pair (s, a), the state transitions from s to s' based on a (possibly stochastic) transition function $T : S \times A \to \Delta_S$. The agent also earns a reward $r_t \sim R : S \times A \to \Delta_{\mathbb{R}}$, where $R(s, a)$ is a probability density over the reals with some finite expected value.

The agent's goal is to maximize its expected discounted return, $\sum_{t=0}^{T} \gamma^t r_t$, where $\gamma \in (0, 1]$ is a discount factor for future rewards. The optimal policy, $\pi^* : S \to A$, is any policy that maximizes this objective.

2.3.2 Q-learning

In Q-learning, a classical RL algorithm proposed by Watkins and Dayan [93], the learner finds a value for each state-action pair called $Q(s, a)$, such that in any state s, the optimal policy is to play $\max_a Q(s, a)$. By definition, $Q(s, a)$ is the expected return the agent would earn by playing a immediately in state s and playing optimally thereafter. The Q-learner updates its estimates for these Q-values upon any training experience tuple (s, a, r, s'), of a prior state, action, reward, and successor state, as

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s', a')\big),$$

where $\alpha \in (0, 1)$ is the learning rate, and $\gamma \in (0, 1]$ is the discount factor for future rewards. Note that Q-learning is an *off-policy* training method, unlike alternative RL algorithms like Sarsa [78], meaning that Q-learning learns to estimate the Q-values of the optimal policy, not the current policy being used during training.

A Q-learner acquires training data by exploring strategies adaptively. The learning agent uses an *exploration policy* to select actions during training playouts, resulting in *trajectories* of states and actions $(s_0, a_0, r_0, \ldots)$. An exploration policy maps the agent's current network and state s to a probability distribution over the next action a to take during training. Trajectories provide the training data for updating the estimated Q-values. In this work, I use an *ϵ-greedy* exploration policy for Q-learning. This policy selects the best move $\max_a Q(s, a)$ with probability $(1 - \epsilon)$, according to current Q-value estimates, or selects a uniform random move otherwise.

2.3.3 Partially observable Markov decision processes

To study the decision of a trading agent in a repeated auction, I use a classical model that generalizes the MDP, for the case where the agent cannot directly observe the world state s_t. This model is known as the partially observable Markov decision process, or POMDP.

A POMDP incorporates all the elements of an MDP as described above, in the tuple M. But unlike in an MDP, where the agent directly observes state s_t and has a policy mapping states to actions, the agent in a POMDP observes only noisy signals of the true state. Thus, a POMDP also includes a set Ω of possible observations and a stochastic observation function $O : S \rightarrow \Delta_\Omega$. The POMDP can thus be represented with an augmented version of the MDP tuple, as (M, Ω, O).

In a POMDP, at any time step t, the true state s_t is unobserved. The agent observes a noisy signal $o_t \sim O(s_t)$ and takes action a_t. In principle, the agent could benefit from conditioning its action on the full *history* of previous actions and observations, $H_t = (o_0, a_0, \ldots, o_t)$. Therefore, the agent's policy maps histories to actions, $\pi : H \rightarrow A$, where H is the set of all histories. Note that in some POMDPs, an agent that is limited to observing only a few recent states, instead of the full history, may perform arbitrarily worse than an agent that can condition its actions on the full history, even if both agents play optimally given their limitations.

Some agents in POMDP environments maintain a *belief state*, which is a probability distribution $B \in \Delta_S$ over the set of all underlying states S in the POMDP, representing the agent's belief about the true state. An agent that maintains a belief state must use some mapping from the set of histories, to the set of distributions over states, to generate the belief state, conditional on any history that the agent encounters.

2.3.4 Neural network methods for RL

In many realistic RL environments, the space of observation histories H is so large (possibly uncountably infinite) that a learning agent is likely to encounter many inputs H_t exactly once over the set of all training episodes, and most possible inputs H are not encountered at all during training. A successful RL agent must be able to generalize from observation histories encountered during training, to distinct but similar observation histories encountered at test time. Such generalization has been achieved in many settings through function approximation with neural networks.

For convenience, in the discussion below in this section I treat states s as the inputs to the neural network, instead of histories H or belief states. In settings where the learner does not observe the true state, but must infer a belief probability distribution over possible states based on the full history, one could treat this belief as the state; the description below would then apply as-is. In practice, an analyst might use a recurrent neural network to generate a representation of the full history, in place of a belief estimate over the underlying state; or the analyst could concatenate a few observations to produce the network input, as a way to encode an important part of the history. In this work, I simply concatenated the most recent 3 observations to produce a fixed-size input to each agent's network; experiments showed that this approach worked well, allowing agents to consistently learn beneficial strategy deviations, although it is theoretically possible that a recurrent network may be able to perform even better, by incorporating a representation of the complete history.

A widely-used approach to generalization in RL is to use a parameterized policy, called ϕ_θ, whose output is a differentiable function of its parameters, θ. In the Q-learning setting, the parameterized function ϕ_θ predicts the Q-value of a state-action pair, $Q(s, a)$, or more formally, $\phi_\theta(s, a)$ is a prediction for $Q(s, a)$, for all states s and actions a. The reinforcement learner encounters episodes (s_t, a_t, r_t, s_{t+1}) during training, and it updates the parameterized function ϕ_θ by adjusting the parameters θ' to reduce the prediction error between $\phi_{\theta'}(s_t, a_t)$ and $\hat{Q}_\theta(s_t, a_t)$, where $\hat{Q}_\theta(s_t, a_t)$ is estimated from the trajectory using weights θ as

$$\hat{Q}_\theta(s_t, a_t) = r_t + \gamma \max_{a'} \phi_\theta(s_{t+1}, a').$$

This approach has famously been implemented in deep Q-networks (DQN) [60]. During training, updated network weights θ' are selected by adjusting the weights via stochastic gradient descent, toward minimizing objective $J(\theta')$, the mean-squared error between predicted and empirically observed Q-values:

$$J(\theta') = E_{(s,a)} \left[\left(\phi_{\theta'}(s, a) - \hat{Q}_\theta(s, a) \right)^2 \right].$$

I experimented with a few neural network architectures for the parameterized policy, as applied to a cybersecurity game. Both architectures fall within the DQN framework, where each action a in a finite action set A is assigned an expected value $\phi_\theta(s, a)$, conditional on the vector representation of the current state s; the implicit policy represented by the neural network is to use the action of highest expected value, or $\arg\max_{a' \in A} : \phi_\theta(s, a')$.

The architecture I use most in the deep RL portion of this work is the multilayer perceptron. (I also experimented with a convolutional version of DQN, but found it did not perform as well empirically in the cybersecurity game.) In a multilayer perceptron, there is a vector of hidden nodes, each of which yields a linear function of the input vector and a set of learned scalars, known as weights. Each hidden node's output y is passed through a nonlinearity, such as the rectified linear unit, $\max(0, y)$. An output node for each possible action generates a linear combination of the hidden units' rectified outputs. Finally, the max over output nodes is selected to choose the action to take. Networks such as the multilayer perceptron, and more complex variants that may be easier to train in challenging settings, can be easily handled through libraries such as TensorFlow. Frameworks for neural network training also provide efficient algorithms for updating network weights during gradient descent, which may yield faster convergence than a naive algorithm, due to variance reduction and careful tuning of step sizes. Reinforcement-learning specific libraries like OpenAI Baselines [21] offer example code for popular deep RL methods, including DQN.

2.3.5 Iterated best-response methods

One approach that has been successfully applied to searching for strong mixed strategies in complex games with large strategy spaces is the *iterated best-response* method. In this method, the analyst alternates between: (a) using some solution concept or other rule to select a mixed-strategy profile over the strategies known so far, and (b) searching for a pure strategy for each agent that deviates beneficially from this profile. (Recall that a pure strategy s'_i is a beneficial deviation for agent i from a mixed strategy profile σ, if it yields a higher expected payoff for i than playing as in σ_i, when all other agents continue playing as in σ_{-i}.)

In two-player games, iterated best response is known as the *double-oracle* method. (The essential concept of the double-oracle method can be easily generalized to n-player games, although the name would no longer be appropriate.) The process begins with a subset S' of strategies for players 1 and 2, called S'_1 and S'_2, among which all expected payoffs have been evaluated. The analyst alternates between: (a) finding a mixed Nash equilibrium profile over the strategies in S', and (b) using an *oracle* for each player to find that player's best-response pure strategy to the current profile. The oracle is a function that takes the current profile and returns the best-response pure strategy for a player, using any method. At each round, the new best response for each agent i is added to the set S'_i of strategies for that agent, and the payoff of this strategy is measured against all opponent strategies in the selected subset S'. The process converges when each agent's best response is already in S', at which point the mixed-strategy profile must be a Nash equilibrium of the game. This approach was introduced by McMahan et al. [56], and has been applied to many security game settings [40].

The double-oracle approach can be generalized as the Policy Space Response Oracle (PSRO) method [49], where instead of the profile selection rule being Nash equilibrium, any other rule can be substituted. For example, if one uses the uniform distribution over strategies in selected subset S', the result is the *fictitious play* method, which selects pure strategies to optimize performance against the mean of previous opponents. As another example, if one uses the most recently added strategies only, the result is the *iterated best-response* method, which selects pure strategies to optimize performance against the opponent's previous selection. Unlike double-oracle and fictitious play, the iterated best-response method tends to enter limit cycles in some settings, and may not converge to a fixed point. (A classical example of this failure mode is the matching pennies game, where iterated best response may cause both players to swap strategies forever.)

A potential problem with double-oracle strategy selection is that it may select pure strategies that are overfit to current opponents, unless the method is modified to encourage generalization [49]. For example, neural networks trained in the strategy exploration phase of the double-oracle method may not adequately exploit strategies that are not part of the current equilibrium profile, because those strategies were not seen during training. Lanctot et al. [49] show that neural networks trained under PSRO may have degraded performance against opponent networks that were trained in the same fashion, but with a different random seed, thus having only arbitrary differences from the opponents encountered during training. One method from prior work that could help address this problem is *opponent sampling* [6], where randomly selected previous opponents are used for training, instead of only the current best opponent. Along similar lines, Lanctot et al. suggest using certain *meta-strategy solvers* such as *projected replicator dynamics*, which place a lower bound on the weight of any known strategy in the opponent mixed strategy used for training, and which encourage changes in mixed strategy to be small during each training stage.

2.3.6 Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL) refers to settings where more than one agent is simultaneously trained to increase its expected reward in a game, and each learner's payoff is affected by the behavior of the others. Unlike ordinary RL settings with only one learning agent, MARL presents each agent with a non-stationary objective function, because whenever the other agents update their behavior, the expected reward from any action could be changed as a result. These non-stationary rewards can cause training to enter limit cycles, instead of converging to a local fixed point where all agents may receive greater expected reward [4], [27].

Various approaches [4], [33] have been proposed to improve the convergence of MARL training. Among these is Learning with Opponent Learning Awareness, or LOLA [27], in which each agent maintains a model of its opponent's strategy, predicts how the opponent will update its strategy, and adjusts its own strategy to improve performance against the predicted future opponent. This approach, a type of second-order method, may help avoid limit cycles where both agents fail to anticipate each other's adaptations.

2.4 Application setting: First-price sealed-bid auction

In Chapter 3, the *first-price sealed-bid auction* (FPSB) is used as an example game. This section introduces key concepts of this game as background. I consider here the special case with two agents bidding in the auction.

The FPSB auction represents a scenario where multiple bidders attempt to win a single item, while paying as little as possible. The auction is called *first-price* because the winning agent must pay the amount of its bid to the auction house. It is called *sealed-bid* because all agents place their bids in one shot, with no way to observe the bids of the other agents. The winner is the agent with the highest bid, with ties broken uniformly at random.

Formally, in the version of the two-player FPSB game considered here, each of agent 1 and agent 2 has a type $v_i \in [0, 1]$. The type represents the agent's value for the item in the auction, also known as their maximum willingness to pay. An agent's type is known to the agent but hidden from the other agent. Each agent's type is drawn i.i.d. from a publicly known distribution, $U(0, 1)$ (i.e., the unit uniform distribution).

Each agent simultaneously submits its bid, $b_i \in [0, 1]$, to the auctioneer. If the bids are distinct, the agent of higher bid wins the item and pays its bid, earning payoff $v_i - b_i$. The other agent does not win the item but pays nothing, earning payoff 0. If the bids tie, each agent earns an expected payoff of $(v_i - b_i)/2$, due to having probability $1/2$ of winning the item, otherwise getting nothing.

This work considers a commonly-studied restricted form of this game, where the only available pure strategy is to commit to bidding a fixed fraction of one's value. In other words, each agent i must commit, before learning its type, to playing some factor $c_i \in [0, 1]$, such that its bid will be $b_i = c_i v_i$. It is known from prior work that the unique mixed-strategy Nash equilibrium is for each agent to play $c = \frac{1}{2}$ [24], [42].

2.5 Application setting: Continuous double auction

A core application area of this work is the *continuous double auction* (CDA), a market mechanism that is used in substantially all financial markets, including stock markets like the New York Stock Exchange [19], [29], and accounting for trillions of dollars in trading annually [63]. This mechanism is called continuous, because a transaction is triggered at any moment when a new, matching order arrives to the market. It is called a double auction, because multiple trading agents compete on both the buy and sell sides of the auction [28]. The CDA model I use is fundamentally the same as that in prior works by Wah et al. [89], [90]

2.5.1 Market model

Agents in the CDA model considered here act only by submitting *limit orders* to the market. Each limit order L has a type $\psi \in \{B, A\}$, where B means an order to buy, and A means an order to sell. A limit order also has a price $p \in \mathbb{J}_+$, which must be an integer. It also has an indicator of which agent $i \in \{1, \ldots, N\}$ placed the order. Finally, a limit order has a timestamp $t \in \mathbb{J}_+$ indicating the time when it reached the market, which is also an integer value, representing perhaps the millisecond in which an order arrived. (This work allows only unit-quantity orders, which seek to trade a single share of stock.) Therefore, any limit order can be represented as a 4-tuple $L = (\psi, p, i, t)$.

The market keeps a *limit-order book* at all times, which ranks the orders of each type in priority order. A separate rank-order list is maintained for buy and sell orders. Buy (sell) orders are ranked primarily from high to low price (low to high price), secondarily by time, with earlier orders first. If multiple orders have the same type, price, and arrival time, then they are ranked arbitrarily.

At any moment, the *bid* is the current price of the best buy order in the limit-order book, and the *ask* is the price of the best sell order in the book. If there is no buy (sell) order in the book, the bid (ask) is undefined.

When a new limit order arrives at the market, the market mechanism checks if the order can transact with an order in the book. If the new order L is a buy (sell) order, the mechanism examines the best sell (buy) order in the limit-order book; if there is none, there will be no transaction. Otherwise, the two orders' prices are compared. If the new buy (sell) order's price is at least as high (as low) as the best opposite-type order's price in the book, then there will be a transaction.

In case the new order does not transact, it is inserted into the limit-order book according to its type, price, and arrival time.

If the new order does transact, then it is not inserted in the limit-order book, and its matching order from the book is removed. The agent i of the buy order receives 1 unit of stock from the agent j of the sell order. Simultaneously, the buyer makes a cash payment to the seller. The value of the payment equals the price of the order that had been in the limit-order book (not the price of the order that newly arrived).

In the CDA model considered here, there is a single security being traded, which has an evolving *fundamental value* common to all agents. The fundamental value at time t, r_t, is a signal of the reward each agent will receive at the end of the game, time T, for each unit of stock they hold in their inventory. The fundamental value evolves as a mean-reverting random walk:

$$r_{t+1} \leftarrow \max\Big(0, \kappa\bar{r} + (1-\kappa)r_t + u_t\Big),$$

where $\kappa \in (0,1)$ is a constant mean-reversion parameter, and $u_t \sim \mathcal{N}(0, \sigma_s^2)$ is an independent draw from a Gaussian noise distribution. The long-run mean, $\bar{r}$ is a publicly-known constant to which the fundamental value tends to revert.

Each agent in the CDA model examined here has a type corresponding to the agent's *private value* for each unit of stock it could obtain. An agent's type is represented as a vector θ of length $2M$, where M is the maximum number of shares an agent can own or owe. Each agent is assigned weakly diminishing marginal returns for owning more units; in other words, for all $i > j$, $\theta_i \leq \theta_j$. Conceptually, when an agent buys one share and goes from holding k to $k+1$ shares, the agent receives an incremental value of θ_{k+1}.

At the end of the simulation, time step T, each agent receives as its reward the sum of its cash holdings, the fundamental value of its stock inventory, and the private value of its stock inventory. The private value of the agent's stock inventory, $v(m_i, \theta_i)$, equals 0 if $m_i = 0$. Otherwise, if $m_i < 0$, then $v(m_i, \theta_i) = \sum_{k=-m_i+1}^{0} -\theta_{i,k}$; if $m_i > 0$, then $v(m_i, \theta_i) = \sum_{k=1}^{m_i} \theta_{i,k}$. In summary, if agent i has accrued a net cash value c_i from its transactions, and it holds m_i units of stock, the total reward earned by agent i will be $c_i + r_T m_i + v(m_i, \theta_i)$. Note that c_i and m_i can be positive or negative.

When an agent arrives at the market at some time t in this model, the agent observes the current fundamental value r_t. Using public knowledge of the long-run mean $\bar{r}$, final time step T, and mean-reversion parameter κ, the agent can compute the expectation of the final fundamental value as of time t:

$$E_t(r_T) = \left(1 - (1-\kappa)^{T-t}\right)\bar{r} + (1-\kappa)^{T-t} r_t. \tag{2.1}$$

The agent can then use this value as a guide for what price to demand when it places an order.

An agent can compute the expected value of the next unit gained when it arrives at time t. This is simply $E_t(r_T) + \theta_{k+1}$, where the agent currently holds k items. Similarly, the agent can compute the expected value of the next unit lost as $E_t(r_t) - \theta_k$.

The *surplus demanded* by a buyer (seller) is defined to be the agent's expected final value for the next unit gained minus the order price (order price minus the agent's expected final value for the next unit lost). For example, if a buyer has an expected final value for the next unit gained of 105, and the buyer places an order to buy for 101, then the buyer is demanding a surplus of $105 - 101 = 4$.

2.5.2 Zero-intelligence agents

The zero-intelligence (ZI) strategy has been used as a trading agent model in many prior works. Gode and Sunder introduced the ZI strategy to show how agents using a simple policy could rapidly converge to efficient prices in a continuous double auction [30]. Since then, the ZI strategy has been used to model trading agent behavior by many researchers, thanks to its ability to capture stylized facts about real market outcomes [25], [52], [58].

The zero-intelligence strategy is so called because of its simplicity: The agent simply estimates the true value of the security being traded, then places an order that demands a surplus drawn from some uniform distribution.

This work employs a version of the ZI strategy identical to that introduced by Wah et al. [90] The strategy has three parameters, $\underline{d}, \overline{d}, \eta \in (0, 1]$, with $\underline{d} \leq \overline{d}$. Each time the agent arrives at the market, it observes the current fundamental value r_t and time remaining

$(T-t)$, from which it computes the expected final fundamental value $E_t(r_T)$, as in Equation 2.1. In the market model used here, the agent is uniformly randomly assigned to be a buyer or seller by the market, independently each time it arrives. The agent randomly samples its desired surplus $g \sim U(\underline{d}, \overline{d})$. The agent then checks whether, if it sold (bought) at the current bid (ask) price, it would obtain at least η fraction of its desired surplus g; if so, the agent places a limit order at the bid (ask) price, which will transact immediately (unless, perhaps, another agent places a same-type order in the same time step). Otherwise, a seller (or buyer) then computes its limit-order price as $p = E_t(r_T) + g$ (or $p = E_t(r_T) - g$). The agent then places a limit order at the computed price p.

2.6 Application setting: Attack-graph games

Recent years have seen growing work on security games, which represent attackers and defenders struggling for control of some vulnerable resource. Cybersecurity games are a special case where the resource is a computer system, such as those represented in *attack graphs*, a popular cybersecurity model with a broad research literature [17], [23], [46]. *Attack-graph games* augment an attack-graph model with action spaces and utility functions for attacker and defender agents, yielding a strategic game.

This dissertation focuses on an attack-graph type from recent works in control theory and game theory [57], [62]. Conceptually, an attack graph represents a computer system as a set of nodes and directed links. Each node stands for a binary state variable of the computer system, such as whether an attacker has root access to a certain computer, or whether a particular software vulnerability has been exploited. As such, each node is always in one of two states, *active* or *inactive*, and it can transition from one state to the other and back via a discrete-time process.

There are two agents in an attack-graph game: the attacker and defender. The attacker agent seeks to activate nodes designated as *goal nodes*, for which the attacker receives a reward in each time step they remain active. The defender seeks to prevent the attacker from activating goal nodes, because it suffers a penalty for each time step when they are active. The attacker acts by "attacking," meaning attempting to activate, nodes that are children of active nodes in the graph; each node has a node-specific cost to attack. The defender acts by "defending" any nodes it chooses, which causes those nodes to become inactive, but with a node-specific cost.

The sections below provide an overview of the attack-graph game. For a complete description of the game's details, please refer to the paper by Nguyen et al. [62]

2.6.1 Attack graph definition

An attack graph is a directed acyclic graph $G = (V, E)$. Each vertex $v \in V$ is endowed with special properties. It has an initial state $s_0^v \in \{0, 1\}$, where 1 indicates it is active, 0 inactive. It has a type $\theta^v \in \{\wedge, \vee\}$, where $\wedge$ means that all parents of the node must be active before it can become active, and $\vee$ means that at least one parent must be active before it can become active. Each $\wedge$-type node is endowed with an activation probability p^v, meaning that if it is attacked when all its parents are active, it will become active with probability p^v.

Each edge $e = (u, v) \in E$ can have properties as well. If the edge is directed to an $\vee$-type node, it has an activation probability p^e, meaning that if the attacker uses this edge to attack child v when parent u is active, with probability p^e, the child will become active.

2.6.2 Attack-graph game definition

An attack-graph game has an associated attack graph G, along with several other components, as follows.

The game has a duration in discrete time steps of $\mathcal{T}$. In each time step, attacker and defender simultaneously take an action, and the graph transitions to a new activation state for each node.

Some nodes $v \in V$ are goal nodes, and these are endowed with a reward r_a for the attacker and penalty r_d for the defender, which is accrued in each time step when they are active.

Each node is endowed with a cost to defend c_d, which the defender must pay for each time step when it defends that node. Similarly, each $\wedge$-type node and edge to an $\vee$-type node is endowed with a cost to attack, c_a.

There is an observation function O that maps the graph's activation state S_t to the defender's noisy observation O_t. For each node v, an independent signal O_τ^v in $\{0, 1\}$ of the current activation state S_τ^v is generated, according to two fixed, publicly-known values, $\Pr(o = 1 \mid s = 1)$ and $\Pr(o = 1 \mid s = 0)$. (The attacker gets to observe the true activation state of each node.)

2.6.3 State updates and payoffs

An attack-graph game proceeds over a series of $\mathcal{T}$ time steps. At the beginning of each time step t, the attacker observes the true activation state of all nodes S_τ, while the defender observes a noisy sample O_τ of this state. Attacker and defender simultaneously select their actions, where the defender action is a subset of all nodes, and the attacker action is a subset of all $\wedge$-type nodes and edges to $\vee$-type nodes.

The environment transitions to new state $S_{\tau+1}$, by the following rules. Any node that was not acted on remains in the same state. Any node the defender acts on will become inactive. A node acted on by the attacker alone might become active, if all required parents had been active at the beginning of the time step (i.e, any parent for an $\vee$-type child, all parents for an $\wedge$-type child.) An $\wedge$-type child v becomes active with probability p^v. An $\vee$-type child has an independent probability p^e of becoming active via each in-edge e that is attacked.

The total payoff of the game for the attacker is the sum over all time steps, of the reward for active goal nodes minus the cost of attacked nodes and edges. Likewise, the total payoff for the defender is the sum over time steps, of the penalty for active goal nodes minus the cost of defended nodes.

Chapter 3

Probably Almost Stable Strategy Profiles in Simulation-Based Games

3.1 Introduction

In studies of real-world environments with strategic agents, analysts often use game-theoretic equilibrium to predict outcomes of agent interactions. Many such games are intractable to solve exactly, due to factors like the large number of strategies and lack of explicit payoff specification. Analysts may cope by considering a *restricted game*, where agents are limited to a relatively small enumerated set of strategies. Payoffs over these strategies can be estimated by simulation, inducing a restricted game model that can be solved for equilibrium. This approach has been applied to areas including models of securities markets [90], social dilemmas [50], space debris removal [45], and credit networks [14].

Such restricted-game studies are informative, but they leave questions about the relevance of solutions found with respect to the original, unrestricted game. In particular, there has been no way to quantify the likelihood that the restricted-game results hold in the original game, even approximately. Indeed, it is expected that the equilibria found in the restricted games have beneficial strategy deviations in the original game. It would be useful to have some formal characterization of stability of restricted-game solutions in the original game, in terms of the difficulty of finding beneficial deviations.

I introduce a search algorithm for stable profiles in simulation-based games. This algorithm guarantees that when it terminates successfully, there is high statistical confidence the strategy profile returned is *almost stable*, meaning only a small measure of other strategies can be beneficial deviations. (That is, only a small fraction of trials will yield beneficial deviations, for some stochastic strategy search function.) This is called a *probably almost stable* guarantee. The degree of statistical confidence and acceptable measure of beneficial deviations can be tuned as desired.

The procedure uses iterated better-response search, also known as the double-oracle method [56], to find beneficial deviations. Better-response dynamics, where players iteratively add beneficial deviations to their strategy sets, has been shown to converge to low-regret profiles in various game types [31], [77]. The procedure also incorporates a statistical confidence interval estimation method for determining whether the current profile meets the stability guarantee or further search is needed. The procedure design is completed by a multiple-test correction for avoiding excessive false positives caused by the sequential-testing nature of the algorithm. Optionally, one can use a black-box optimization procedure like simulated annealing with the procedure to achieve stronger guarantees on the difficulty of finding beneficial deviations, at a cost of increased computation during strategy exploration.

These contributions are a novel algorithm for generating strategy profiles in large, simulation-based games with a provable stability guarantee, without requiring that most strategies be examined; and experiments demonstrating the efficacy of that algorithm. I give a formal definition of the probably almost stable property and a simple proof of correctness. I derive the time complexity of the algorithms, in terms of the desired tightness of its statistical guarantee. Then I evaluate the algorithm on two games: the first-price sealed-bid auction (FPSB), and a cybersecurity game on an attack graph [67]. I show that in both settings, the algorithm yields a high frequency of true positives (i.e., profiles found that have acceptably low true beneficial deviation measure), while controlling false positives in accordance with the guarantee. When I incorporate simulated annealing as an improved strategy search method, the frequency of true positives of the new method increases dramatically.

3.2 Related work

My approach was inspired in part by work of Bopardikar et al. [7], which defines a search algorithm for *security policies* in two-player, zero-sum simultaneous games. That paper defines a security policy as a mixed strategy and associated payoff for player 1, such that if player 1 uses that strategy, and player 2 plays the best response it finds during a limited random search over deviating strategies, player 1 will obtain at least the given payoff with high probability. I follow this work in reasoning statistically from one player's exploration to derive probabilistic bounds on what the other player can find. However, since I do not assume the game is zero-sum, I cannot (at least in this way) provide guarantees about security value. Rather, I pursue a different probabilistic stability property.

Several works have studied sequential search procedures for Nash equilibrium (NE) in games with many strategies. McMahan et al. [56] introduced the *double-oracle method*, a procedure for iteratively solving two-player, zero-sum games with large strategy sets, using an oracle per agent that best-responds to the equilibrium strategy of the opponent in the current restricted game. Sureka and Wurman [77] combined best-response dynamics with a tabu list to seek pure-strategy NE, for combinatorial auctions. Goldberg [31] found theoretical convergence rates for a better-response dynamics based on randomized local search in a load-balancing game. Schvartzman and Wellman [71] used reinforcement learning to add better-response strategies to a restricted game, finding an NE each time before alternating back to reinforcement learning. Jordan et al. [42] compared strategy exploration procedures in convergence rate to low-regret profiles in the FPSB auction. Recently, Lanctot et al. [49] proposed a generalized iterative method motivated by advances in deep learning from self-play in games.

The approach I take relies heavily on a strategy exploration function, which is used to model a game player's effort to find a beneficial deviation from a strategic equilibrium over a restricted strategy set. As such, the choice of strategy exploration function has a large effect on how meaningful the statistical guarantee will be; a stronger exploration method will produce more credible evidence that a proposed strategy profile is an approximate equilibrium. For this reason, I perform experiments with two black-box optimization methods: both the naive method of simple random search over a bounded strategy space, and simulated annealing, a classical method that is known to converge to a global optimum with a sufficiently slow annealing schedule. Recent works on black-box optimization have suggested that more complex methods (e.g., Gaussian process bandits) perform better in some problem settings, depending on the number of samples allowed, but with varying results across different problems [32]. I chose to proceed with simulated annealing as a stronger strategy exploration method, as it is well-known, easy to implement, and sufficient to produce noticeably better results than simple random search within the general method proposed here. I believe that if a different, perhaps stronger, black-box optimizer were substituted for simulated annealing, the results would be broadly similar to those I obtained, as simulated annealing already performs quite well at finding beneficial deviations in our settings.

3.3 Problem setting

The methods shown here are presented for two-player games, with the suggestion that the extension to n players is natural. Let $G = (S, U)$ be a two-player general-sum normal-form game, with strategy sets $S = (S_1, S_2)$ and utility functions $U = (U_1, U_2)$. $U_i(s)$ assigns player i a real-valued payoff for pure-strategy profile $s \in S_1 \times S_2$.

G is a *simulation-based* game, meaning the agents have no direct specification of U, but rather limited access to a *payoff oracle* O. Upon query for a pure-strategy profile s, the oracle responds with $O(s) = (U_1(s), U_2(s))$. The oracle may be said to *simulate* payoffs from the game.

A mixed-strategy profile $\sigma = (\sigma_1, \sigma_2)$ assigns each player a probability distribution over its strategy set. By definition, a mixed-strategy profile σ is a Nash equilibrium of a two-player game, if and only if for all $i \in \{1, 2\}$, $s_i \in S_i$, $E_{s_{-i} \sim \sigma_{-i}} U_i(s_i, s_{-i}) \leq E_{s \sim \sigma} U_i(s)$, where s_{-i} is a pure strategy of the other player, and $\sigma_{-i} \in \Delta_{S_{-i}}$ is the mixed strategy of the other player. More generally, σ is called an $\mathcal{E}$-Nash equilibrium, for $\mathcal{E} \geq 0$, if the most any player can gain by unilaterally deviating is no more than $\mathcal{E}$: $E_{s_{-i} \sim \sigma_{-i}} U_i(s_i, s_{-i}) \leq E_{s \sim \sigma} U_i(s) + \mathcal{E}$. The term *$\mathcal{E}$-beneficial deviation* is used to mean any strategy that yields an improved payoff of at least $\mathcal{E}$, as a unilateral deviation from the current equilibrium profile.

Assume there is an efficient means of finding a mixed-strategy Nash equilibrium (MSNE) in restricted games with sufficiently small strategy sets—perhaps containing a few dozen strategies per agent. It is known by Nash's theorem that some mixed-strategy Nash equilibrium must exist in any such game. Solvers implemented in packages such as Gambit can find sample MSNE reliably and efficiently in many practical problems [55].

Assume the strategy sets S_1 and S_2 are too large to explore exhaustively, possibly infinite. One can extract a *restricted game* G' from G, written as $G' \subset G$, by restricting players to $S' = (S'_1, S'_2)$, where $S'_1 \subseteq S_1$ and $S'_2 \subseteq S_2$. One must similarly restrict the utility function U' of G' to S', such that U' yields the same result as U where their domains overlap. If restricted game G' has $|S'_1|$ and $|S'_2|$ sufficiently small, by assumption it will be feasible to find some MSNE of G'.

To model strategy exploration, define a probability distribution D_1 over S_1 and probability distribution D_2 over S_2. Each distribution has full support, that is, for $i \in \{1, 2\}$ and all $s \in S_i$, $D_i(s) > 0$. Assume that agents select strategies to evaluate by sampling from these distributions in an i.i.d. manner. (For example, in this study, I will perform experiments where D_i is defined implicitly by either simple random search or simulated annealing, over continuous strategy spaces.)

3.4 Probably almost stable profile search

Suppose player 1 has found an $\mathcal{E}$-MSNE, $\sigma = (\sigma_1, \sigma_2)$, in a restricted game $G' \subset G$. It is possible that player 2 might, through a limited random search, find some strategy s_2' that player 1 has not considered, from $G \setminus G'$, such that $U_2(\sigma_1, s_2') > U_2(\sigma) + \mathcal{E}$—that is, an $\mathcal{E}$-beneficial deviation for player 2. We would like to show that, if player 2 samples a pure strategy from distribution D_2, the probability of obtaining an $\mathcal{E}$-beneficial deviation is no greater than ϵ, with $0 < \epsilon \ll 1$ and $\mathcal{E} \geq 0$.

Definition 1. *A strategy profile σ is ϵ-almost stable for player 1 with respect to D_2 and $\mathcal{E}$, if*

$$\Pr\big(U_2(\sigma_1, s_2') > U_2(\sigma) + \mathcal{E}\big) \leq \epsilon, \text{for } s_2' \sim D_2.$$

Because it is not feasible to examine all strategies in the game, we must settle for a probabilistic stability guarantee. The best that we can do is to show that profile σ is almost stable with high probability, say at least $(1 - \delta)$, with $0 < \delta \ll 1$. If a profile σ is accepted as almost stable by a statistical test that has a false positive rate of at most δ for any input, we say that σ is *probably almost stable*. Let $p \equiv \Pr(U_2(\sigma_1, s_2') > U_2(\sigma) + \mathcal{E})$.

Definition 2. *A strategy profile σ is probably ϵ-almost stable for player 1 with respect to D_2, $\mathcal{E}$, and δ if it is accepted by a statistical hypothesis test T, such that whenever $p > \epsilon$,*

$$\Pr\big(T(\sigma) = \textit{accept}\big) \leq \delta,$$

with respect to the randomness in the statistical test.

One might like to model the opponent as drawing M i.i.d. samples from D_2, seeking a probably almost stable guarantee on the likelihood of *any* sample being an $\mathcal{E}$-beneficial deviation. This can be achieved by adjusting the deviation probability, as $\epsilon \leftarrow 1 - (1 - \epsilon)^{1/M}$. However, the adjustment may dramatically increase the sample complexity required to ensure the guarantee.

3.4.1 Using the Clopper-Pearson confidence interval

The Clopper-Pearson confidence interval is a classical statistical tool; the summary below (for the upper bound only) is based on the original paper [16]. This method will be used to derive a bound on the probability of any sampled deviation being $\mathcal{E}$-beneficial, based on the observed frequency in a sample. The bound is conservative, even when no successes are observed [1], [10].

Given a binomial random variable of unknown probability p, suppose one observes X successes in N trials. The Clopper-Pearson procedure, given a desired confidence level $\delta \in (0, 1]$ and sample count N, gives an upper bound $\bar{p}_N^{\delta}(X)$ on p, such that for any $p \in (0, 1]$, the risk of error (i.e., $\Pr(\bar{p}_N^{\delta}(X) < p)$) is at most δ.

For the desired δ and sample count N, and for any p, let $\underline{x}_N^{\delta}(p)$ be the greatest integer in $\{-1, \dots, N-1\}$ such that $\Pr(X \leq \underline{x}_N^{\delta}(p) \mid p) \leq \delta$. For any $X \in \{0, \dots, N-1\}$, let $\bar{p}_N^{\delta}(X)$ be the least value $p' \in [0, 1]$ such that $\underline{x}_N^{\delta}(p') = X$. Define $\bar{p}_N^{\delta}(N) = 1$. The Clopper-Pearson upper bound on p is simply $\bar{p}_N^{\delta}(X)$.

It has been shown [16] that given $\delta \in (0, 1]$ and $N \geq 1$, for all $p \in [0, 1]$,

$$\Pr\big(\bar{p}_N^{\delta}(X) < p\big) \leq \delta. \tag{3.1}$$

In this procedure, select a sample count N as small as possible such that if no successes are observed in N trials, the Clopper-Pearson upper bound, $\bar{p}_N^{\delta}(0)$, will be at most ϵ. As Thulin [85] notes, in the case where $X = 0$ trials are successful, one can compute the upper bound of the Clopper-Pearson interval as $\bar{p}_N^{\delta}(0) = 1 - \delta^{1/N}$. Rewriting to solve for the required number of trials to guarantee $\bar{p}_N^{\delta}(0) \leq \epsilon$, one will find:

$$N = \left\lceil \frac{\log(\delta)}{\log(1 - \epsilon)} \right\rceil. \tag{3.2}$$

In case fewer false negatives are desired (i.e., fewer stable profiles rejected), the procedure could be modified to select N such that one or more successes are allowed in N trials, with the general approach being otherwise unchanged.

3.4.2 Hypothesis test for probably almost stable profiles

This section presents a simple hypothesis test for evaluating whether a mixed-strategy profile $\sigma = (\sigma_1, \sigma_2)$ is probably almost stable for player 1 with given parameters. (To guarantee that neither player is likely to find an $\mathcal{E}$-beneficial deviation by sampling, one could simply run Algorithm 1 with each player in turn as focal agent.) If $p > \epsilon$, (i.e., random draws are too likely to be $\mathcal{E}$-beneficial deviations), this test will reject profile σ with probability at least $(1 - \delta)$.

Algorithm 1 PAS-Single

```
Require: $\epsilon \in (0,1), \delta \in (0,1), \sigma \in \Delta_{S_1} \times \Delta_{S'_2}, D_2 \in \Delta_{S_2}, U_2 : \Delta_{S_1} \times \Delta_{S_2} \to \mathbb{R}, \mathcal{E} \geq 0$
1: $N \leftarrow \arg\min_{\ell \in \{1,\dots\}} : \bar{p}^{\delta}_{\ell}(0) \leq \epsilon$
2: for $N$ trials do
3:     Sample a player-2 pure strategy $s_2 \sim D_2$
4:     if $U_2(\sigma_1, s_2) > U_2(\sigma) + \mathcal{E}$ then
5:         return $(reject, s_2)$
6: return $(accept, \perp)$
```

In presenting Algorithm 1 the notation $\bar{p}^{\delta}_{\ell}(X)$ is used to mean the upper bound on p where ℓ is the binomial sample count, and δ the desired confidence level. The hypothesis will be rejected, that profile σ is probably almost stable, if a single $\mathcal{E}$-beneficial deviation is found in N trials, and otherwise the hypothesis is accepted. The algorithm also returns the pure strategy s_2 that $\mathcal{E}$-beneficially deviated, or $\perp$ if none was found. Note that $\sigma \in \Delta_{S_1} \times \Delta_{S'_2}$, indicating that player 2 plays a mixture of only a small (finite) subset of its full strategy set with positive probability.

Proposition 1. *For input $(\epsilon, \delta, \sigma, D_2, U_2, \mathcal{E})$, if Algorithm 1 returns* accept, *then σ is probably ϵ-almost stable at confidence level δ for distribution D_2 and $\mathcal{E}$.*

Proof. We want to show that $\Pr(T(\sigma) = accept \mid p > \epsilon) \leq \delta$. Consider any σ such that $p > \epsilon$. Let T represent Algorithm 1. $T(\sigma) = accept$ only if success count $X = 0$ in N trials. By construction, $\bar{p}^{\delta}_{N}(0) \leq \epsilon$. Thus, if $T(\sigma) = accept$ and $p > \epsilon$, $\bar{p}^{\delta}_{N}(X) < p$. By the Clopper-Pearson bound (3.1), $\Pr(\bar{p}^{\delta}_{N}(X) < p) \leq \delta$. □

Proposition 1 shows that the Clopper-Pearson estimator limits the false positive rate of Algorithm 1 to δ, even if $p \approx \epsilon$. One drawback of the estimator is that it produces a high false negative rate when $p \approx \epsilon$. Specifically, the probability of a false negative in Algorithm 1, when $p \leq \epsilon$, is $1 - (1 - p)^N$. This is simply the probability that at least one sampled deviation will be $\mathcal{E}$-beneficial. In the worst case where $p = \epsilon$, the false negative rate equals $(1 - \delta)$. Since many profiles σ that should be accepted by Algorithm 1 have $p \ll \epsilon$, the false negative rate tends to be much lower than $(1 - \delta)$ on average.

3.5 Combined better-response search and sequential hypothesis test

Algorithm 1 can evaluate whether a given profile σ is probably almost stable. This can be employed within a broader method to *search* for such a profile. Suppose Algorithm 1 returns $(\textit{reject}, s_2)$ for profile σ. One can append the pure strategy s_2 that beneficially deviates from σ to the old restricted strategy set S', to form enlarged strategy set S''. Next, I query the payoff oracle O for the payoffs of every new pure strategy profile in the enlarged restricted game G''. I can then use a game solver to find some MSNE σ'' of G''. Finally, I test profile σ'' to determine whether it is probably almost stable in original game G.

If the new profile σ'' is found to be probably almost stable in original game G, the procedure stops with a successful result. Otherwise, it adds the beneficially deviating pure strategy to the restricted strategy set and repeat, until either some predetermined number of iterations is reached, and the procedure fails, or it succeeds and terminates. Note that this procedure cannot guarantee probably almost stable results, due to the multiple comparisons problem, unless the confidence level δ at each iteration is reduced compared to Algorithm 1.

Algorithm 2 combines a better-response dynamics search for stable profiles with a sequential hypothesis testing procedure that accounts for multiple comparisons. The goal is to limit the familywise error rate (FWER) over all hypothesis tests in the sequence to a given confidence level δ. This means that for any game G, initial restricted strategy set S'_2, deviation probability ϵ, confidence level δ, equilibrium candidate σ, distribution D_2, and deviation tolerance $\mathcal{E}$, the probability of returning $(\textit{accept}, \sigma')$, when the true probability of beneficial deviation from σ' under D_2, $\mathcal{E}$, is greater than ϵ, is at most δ.

Note that function $\textit{solve}(S_1, S'_2, U)$ in Algorithm 2 returns any MSNE of the restricted game on strategy set (S_1, S'_2).

Algorithm 2 uses a Bonferroni correction for multiple comparisons testing, to cap the familywise error rate at δ. K is the maximum number of strategy exploration rounds to perform. $\vec{\alpha} \in \mathbb{R}^K_+$ is an α-spending vector, indicating the confidence level to be used in each iteration, where the confidence levels sum to δ. By the union bound, the probability of a false positive in each round k is at most α_k, the probability of any false positive is at most $\sum \alpha_k = \delta$. Therefore, because Algorithm 1 guarantees probably almost stable profiles when it returns *accept*, it follows that Algorithm 2 does as well. (Note that the Bonferroni

Algorithm 2 PAS-Sequential

```
Require: $\epsilon \in (0,1), \delta \in (0,1), \sigma \in \Delta_{S_1} \times \Delta_{S'_2}, D_2 \in \Delta_{S_2}, U : \Delta_{S_1} \times \Delta_{S_2} \to \mathbb{R}^2, \mathcal{E} \geq 0, K \in \{1, \dots\}, \vec{\alpha} \in \mathbb{R}^K_+ : \sum \alpha_k = \delta, S'_2 \subset S_2$
 1: $\sigma' \leftarrow \sigma$, the current restricted-game MSNE
 2: for iteration $k \in \{1, \dots, K\}$ do
 3:     $N_k \leftarrow \arg\min_{\ell \in \{1, \dots\}} : \bar{p}^{\alpha_k}_{\ell}(0) \leq \epsilon$
 4:     $s \leftarrow \perp$, the $\mathcal{E}$-beneficial deviation
 5:     for $N_k$ trials do
 6:         Sample a player-2 pure strategy $s_2 \sim D_2$
 7:         if $U_2(\sigma'_1, s_2) > U_2(\sigma') + \mathcal{E}$ then
 8:             $s \leftarrow s_2$
 9:             break
10:     if $s = \perp$ then
11:         return $(accept, \sigma')$
12:     else
13:         $S'_2 \leftarrow S'_2 \cup \{s\}$
14:         $\sigma' \leftarrow solve(S_1, S'_2, U)$, an MSNE for the new player-2 strategy set
15: return $(reject, s)$
```

correction used here is conservative, and as a result may lead to more frequent false negatives than some other corrections for multiple tests; other methods, such as Dunnett's test, could be substituted, trading off the PAS guarantee for fewer false negatives. I use Bonferroni here because it allows the theoretical guarantees to be as strong as possible.) Algorithm 2 can make this further guarantee, compared to the single-round Algorithm 1, because it uses a larger number of tests N_k per round to guarantee $\bar{p}^{\alpha_k}_{N_k}(0) \leq \epsilon$.

Proposition 2. *For any input $(\epsilon, \delta, \sigma, D_2, U, \mathcal{E}, K, \vec{\alpha}, S'_2)$, if Algorithm 2 returns $(accept, \sigma')$, σ' is probably ϵ-almost stable at confidence level δ for distribution D_2 and $\mathcal{E}$.*

Proof. For each of up to K rounds of Algorithm 2, let $R^k(\sigma^k) \in \{accept, reject\}$ be the result of round k, on current profile σ^k. For any profile σ'', let $\Pr(R^k(\sigma'') = accept)$ be the acceptance probability of the test in Algorithm 2 for round k, given α_k. For any profile σ'', let $p(\sigma'')$ equal the $\mathcal{E}$-beneficial deviation probability under distribution D_2. Let us define hypothesis test $T(\sigma'')$ as the test that, for any profile σ'', returns *accept* if any round k of Algorithm 2 returns $(accept, \sigma'')$, and *reject* otherwise.

We want to show that for any profile σ'', $\Pr(T(\sigma'') = accept \mid p(\sigma'') > \epsilon) \leq \delta$.

By the union bound,

$$\Pr\big(T(\sigma'') = accept\big) \leq \sum_{k=1}^{K} \Pr\big(R^k(\sigma'') = accept\big).$$

By construction, in each round k of Algorithm 2, for any profile σ'', if $p(\sigma'') > \epsilon$, then $\Pr(R^k(\sigma'') = accept) \leq \alpha_k$. This is because each round of Algorithm 2 essentially implements Algorithm 1 with measure tolerance ϵ and confidence level α_k. Thus, by the union bound, if $p(\sigma'') > \epsilon$, $\Pr(T(\sigma'') = accept) \leq \sum_{k=1}^{K} \alpha_k = \delta$. □

Corollary 3. *The probability Algorithm 2 returns a tuple $(accept, \sigma')$ such that $p(\sigma') > \epsilon$ is at most δ.*

3.5.1 Asymptotic performance of Algorithm 1

The order of growth of time and space required by Algorithm 1 is modest. The algorithm takes inputs ϵ, the allowed deviation probability, and δ, the confidence level; the algorithm's runtime can be represented by N, the worst-case sample count. The space complexity of Algorithm 1 is constant, because the algorithm merely needs to track the expected payoff of the best deviation found so far and how many deviations have been sampled.

The time complexity can be conveniently expressed as a function of $\frac{1}{\epsilon}$, the expected samples per $\mathcal{E}$-beneficial deviation, and $\frac{1}{\delta}$, the expected trials before a false positive. Algorithm 1 has runtime N that is $O(\frac{1}{\epsilon})$ and $O(\log \frac{1}{\delta})$. The bound for $\frac{1}{\delta}$ is trivial, based on (3.2). We can derive the bound for $\frac{1}{\epsilon}$ by letting $y = \frac{1}{\epsilon}$ and noting that an upper bound on N is proportional to $f(y) = -(\log \frac{y-1}{y})^{-1}$; we then take the derivative of f with respect to y, and show that its limit is 1 as y approaches infinity. This shows that N grows with y proportionally to a linear function in the limit. Thus, the order of growth in the strategy count N that the analyst must sample is proportionate to the desired bound on the number of samples an agent is expected to require to find an $\mathcal{E}$-beneficial deviation, $\frac{1}{\epsilon}$.

For a numerical example, with $\epsilon = 0.01$ and $\delta = 0.1$, $N = 230$ samples are needed in the worst case. If instead $\epsilon = 10^{-5}$, $N = 230{,}258$ samples would be needed. Note that $\frac{N}{\epsilon}$ is roughly constant for a given δ, so the analyst's computational needs scale evenly with the agent's power to explore.

3.5.2 Asymptotic performance of Algorithm 2

The worst-case time required to run Algorithm 2 is $N \times K$, where N is the maximum sample count per round, and K is the maximum round count. The time complexity of Algorithm 2, like that of Algorithm 1, is $O(\frac{1}{\epsilon})$ and $O(\log \frac{1}{\delta})$. Moreover, the time complexity $N \times K$ is $O(K \log K)$ with respect to the round count. To see this, observe that there is an $O(K)$ factor due to the K term in $N \times K$ and an $O(\log K)$ factor due to the division of δ by K in (3.2) for finding N, when we divide the δ budget into K equal parts for the various rounds.

For instance, if $\epsilon = 0.01$, $\delta = 0.1$, and $K = 3$, Algorithm 2 will require $N \times K = 1{,}017$ samples in the worst case. Even if K were set as high as 1,000, the worst-case sample count would be 917,000; with this many strategies, the bottleneck would likely be the Nash equilibrium solver, not in our algorithm. The sample complexity $N \times K$ grows proportionally to $\frac{1}{\epsilon}$.

3.6 Experiments

I perform experiments on two classes of two-player, general-sum, normal form games from the game theory literature. I study the first-price sealed-bid auction (FPSB) and a cybersecurity game played on an attack graph, from recent literature [62]. In the FPSB game, I consider a parameterized strategy, where players bid a constant fraction of their value for the item being sold. This family of strategies includes equilibria of the full game, and has been adopted for analytical convenience in prior work on FPSB auctions [24], [69].

Each experiment begins with a restricted strategy set. I execute the single-pass Algorithm 1 or sequential Algorithm 2, and analyze the results in terms of frequency of true and false positives and negatives, as well as the acceptance rate conditional on true beneficial deviation probability p (as estimated via sampling). I also repeat the sequential experiments with both strategy exploration methods under consideration: simple random search and simulated annealing.

I aim to test whether the Algorithms 1 and 2 empirically satisfy their theoretical guarantees, even for values of $p \approx \epsilon$. In addition, I aim to evaluate whether the frequency of true positives of the algorithms are reasonably high, such that the algorithms are likely to be useful for finding stable profiles, instead of rejecting almost all profiles due to excessive conservatism. The results also demonstrate the performance of these algorithms when a relatively sophisticated search process is used to seek beneficial deviating strategies.

3.6.1 First-price sealed-bid auction game

In the two-player FPSB auction game, one item is auctioned to the higher bidder of player 1 and player 2, and the winner pays its bid. The version of the game used here allows each player i to choose a bid fraction $c_i \in [0, 1]$ (before learning the player's value for the item). Each player is then assigned a private value v_i for the item, drawn i.i.d. from $U(0, 1)$, and bids $c_i v_i$. The higher bidder wins the item, earning utility $v_i(1 - c_i)$; the other player earns zero utility.

The unique Nash equilibrium in 2-player FPSB is for player i to play the pure strategy $c_i = \frac{1}{2}$ [24], [42]. In a restricted FPSB auction, allowing only mixed strategies over bid factors $S' = \{c_1, \ldots, c_q\}$, the equilibrium solution is less simple. Also, $c = \frac{1}{2}$ may not necessarily be a beneficial deviation from a Nash equilibrium of such a restricted game. This leads to interesting complications in the iterated better-response dynamics of such restricted games.

Here I summarize analytical tools used for the FPSB auction. Suppose that player 2 plays pure strategy c_2, and it is desired to evaluate the expected payoff for player 1 of playing pure strategy c_1. This utility is given by:

$$\mathbb{E}_{v_1, v_2 \sim U(0,1)}(U_1(c_1, c_2)) = \begin{cases} 0 & \text{if } c_1 = 0 \\ \frac{1-c_1}{2} & \text{else if } c_2 = 0 \\ \frac{(1-c_1)c_1}{3c_2} & \text{else if } c_1 \leq c_2 \\ (1-c_1)\frac{3c_1^2 - c_2^2}{6c_1^2} & \text{otherwise.} \end{cases} \tag{3.3}$$

Using this equation, one can determine whether c_1 is a beneficial deviation from any finite mixed strategy for player 2. Moreover, one can immediately give a minimal example of a restricted game where $c = \frac{1}{2}$ is not a beneficial deviation. Consider the restricted game where the only legal strategy is $c_a = \frac{1}{3}$; in the unique Nash equilibrium, the expected payoff is $\frac{2}{9} = \frac{24}{108}$. A player deviating to play $c_b = \frac{1}{2}$ would receive expected payoff $\frac{23}{108}$, which is lower.

For the experiments on the FPSB auction, I begin each restricted game with a shared set S' of 10 pure strategies c available for the players, selected i.i.d. from $U(0, 1)$. One can use (3.3) to build the payoff matrix for all pairs of pure strategies in S'. Then one can use Gambit's implementation of the extreme points method [54] to find MSNEs of the resulting game, and select uniformly among them to set σ' for the next iteration of Algorithm 2.

3.6.2 Cybersecurity game

I also examine an attack-graph cybersecurity game from a recent study [62]. In brief, the game represents an adversarial but not zero-sum interaction between an attacker and a defender, modeled using Bayesian attack graphs [57]. The defender agent in our experiments varies along three parameters, adjusting how many nodes are typically defended, and how randomly or greedily to act. The game is simulation-based, meaning that profile payoffs are estimated by sampling the results of a simulator. This game serves as a realistic application domain for the PAS algorithms, as it is a simulation-based game where it is costly to obtain payoff samples, and where exact solutions are unknown.

3.6.3 Simulated annealing

In some experiments, I use the classical method of simulated annealing as an improved alternative to simple random search for strategy exploration. Here I briefly describe how the implementation of simulated annealing used here is configured.

One can model the strategy space for the deviating agent as $S_2' \in [0,1]^d$, where $d = 1$ for FPSB and $d = 3$ for the cybersecurity game. (Strategy vectors in this space can be mapped to and from the original strategy space S_2.) Some round count κ of strategies is examined in each run of simulated annealing, using $\kappa = 50$ for FPSB, and due to computational limitations, $\kappa = 5$ in the cybersecurity game.

Each run of simulated annealing begins with an i.i.d. random sample over S_2', in the experiments a uniform random sample; it also has an i.i.d. random seed. From there, I employ a truncated Gaussian as the distribution for sampling a neighbor of the current strategy vector. That is, for each dimension in d, I use an independent Gaussian draw centered on the current strategy value, with some given variance, using rejection sampling to ensure the result is in $[0,1]$. I use a variance of 0.003 for the FPSB game, 0.03 for the cybersecurity game. (These variances were chosen by trial and error in a pilot study, with a lower variance being used in the FPSB game because the higher round count κ allows even low-variance samples enough time to reach good parameters. Higher variances make it easier for simulated annealing to explore the whole space in search of a global optimum, while lower variances are better for finely adjusting parameters near an optimum. I made a modest effort to tune these parameters, trying perhaps 3–5 alternatives before choosing the best one in each setting.)

For the temperature schedule, I anneal the temperature τ linearly over the number of samples analyzed so far, from a maximum of 1.0 in FPSB or 15.0 in the cybersecurity game, to zero. At each step, I update the current strategy vector to the new one being analyzed, if: (a) the new strategy has higher expected payoff, or (b) with probability $\exp((u' - u)/\tau)$, where u' is the new strategy's expected payoff, and u is the current strategy's expected payoff. The procedure returns the parameters that yield the highest expected payoff, not necessarily the final parameters settled on by the search process.

3.7 Results

3.7.1 FPSB auction

I consider the FPSB auction with 10 randomly generated initial strategies, maximum deviation probability allowed $\epsilon = 0.05$, and FWER $\delta = 0.1$. I begin by considering the special case where $\mathcal{E} = 0$, so any payoff improvement is sufficient for an $\mathcal{E}$-beneficial deviation. Over these FPSB games, the mean probability of a sampled strategy being a beneficial deviation from the initial MSNE was 10.4%, with a median of 7.4%. Therefore, in the majority of these FPSB games, the initial restricted game's MSNE is not almost stable, where $\epsilon = 0.05$; more strategies would have to be added to the restricted game to produce an almost-stable profile.

I used a high value of $\epsilon = 0.05$ in the experiments that empirically validate the algorithm's accuracy, because a higher ϵ is more economical, and the value of ϵ should not affect the algorithm's correctness. But the runtime of Algorithm 2 grows only linearly in $\frac{1}{\epsilon}$, such that even if I had used $\epsilon = 10^{-4}$, for example, these experiments would remain computationally feasible. Moreover, I used a low value of $K = 3$ because this is appeared to be the lowest K that would clearly show the need for multiple-tests correction, and a higher K would not yield dramatically more convincing results. The sample count $N \times K$ required by Algorithm 2 grows as $O(K \log K)$, so a larger iteration limit like $K = 10$ would be easily manageable. (Indeed, I also conducted a follow-up experiment with $\epsilon = 0.0005$ and $K = 12$.) Higher iteration counts K might make it easier to find stable profiles in some environments but did not appear necessary in most trials for the settings of these experiments.

As shown in Figure 3.1, the empirical accept rate of Algorithm 1 is less than δ for all true beneficial deviation probabilities $p > \epsilon$. This is the guarantee that is ensured, when certifying profiles as probably almost stable. The plot is based on 400 randomly-generated FPSB games, for which the PAS-single algorithm was run 100 times per game. Overall, the PAS-single algorithm accepted a profile as stable in 13.7% of cases, with frequencies

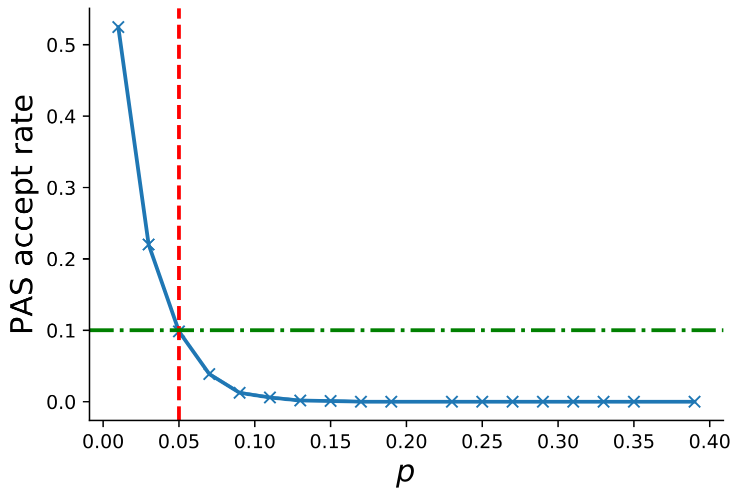

FIGURE 3.1: Accept rate in FPSB of Algorithm 1 vs. probability p of beneficial deviation. Horizontal line is $\delta = 0.1$, vertical line $\epsilon = 0.05$. Accept rate is below δ for all $p > \epsilon$.

as follows: true positive 12.6%, true negative 64.9%, false positive 1.1%, false negative 21.4%. All of our results are summarized in Table 3.1.

Next, I demonstrate the necessity of controlling for multiple comparisons. I show what happens when running the sequential Algorithm 2 with a maximum of $K = 3$ iterations, on the same FPSB game, but without adjusting the α used per iteration by a factor of $\frac{1}{K}$ relative to the value what would be used in single-pass Algorithm 1. Specifically, let $\alpha_i = \delta$ for all rounds, essentially repeating the single-pass Algorithm 1 up to 3 times. Figure 3.2 (left) shows that when the true beneficial deviation probability p is slightly greater than ϵ, this sequential procedure will incorrectly accept the profile as almost stable, more than δ fraction of the time. Therefore, the procedure fails to guarantee probably almost stable results when it accepts. This procedure achieves a false positive rate of 3.5%, which is below δ. But for $p \approx \epsilon$, the procedure fails to guarantee an acceptably low false positive rate.

Finally, I show that Algorithm 2 with $\sum \alpha_i = \delta$ corrects the multiple comparisons problem. Again, one can run sequential Algorithm 2 for up to $K = 3$ iterations, but with $\alpha_i = \frac{\delta}{3}$ in each round. Figure 3.2 (right) shows that when the true beneficial deviation probability p is greater than ϵ, the accept probability is less than δ, meaning that the probably almost stable guarantee is satisfied when Algorithm 2 accepts. The PAS-sequential procedure accepts a profile as almost stable in 40.0% of trials, much better than the 13.7% that was achieved by PAS-single with the same ϵ and δ values. Overall, PAS-sequential produces these frequencies: true positive 39.1%, true negative 24.0%, false positive 0.8%,

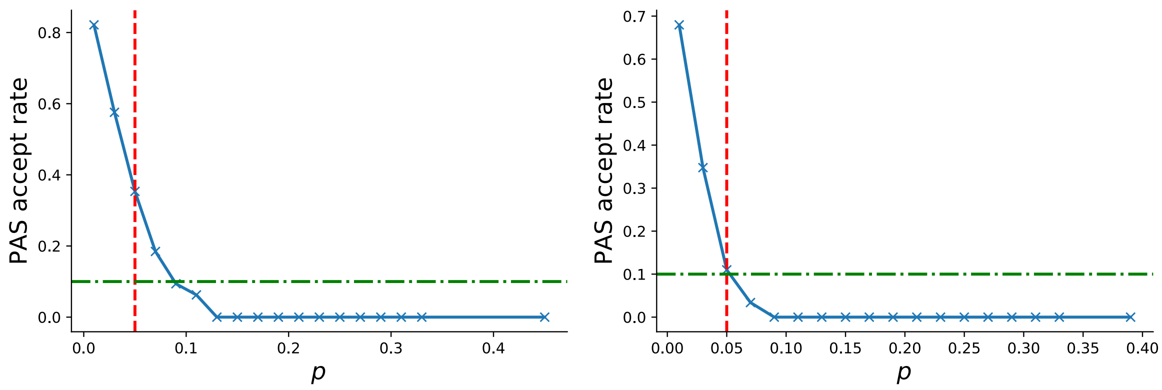

FIGURE 3.2: Accept rate in FPSB of Algorithm 2 vs. probability p of finding a beneficial deviation, with up to $K = 3$ rounds. Left: without multiple comparisons control ($\alpha_i = \delta$). Right: with control ($\alpha_i = \frac{\delta}{3}$). Horizontal line is $\delta = 0.1$; vertical line $\epsilon = 0.05$.

false negative 36.1%. (These rates are computed over only the final iteration's profile from each trial, not including the profiles from earlier iterations within a trial.) On average, PAS-sequential terminated after 2.3 of a possible 3 iterations of strategy search, using a median of 3 iterations.

Notice that PAS-sequential is more successful at finding a stable profile than PAS-single: PAS-single yielded a frequency of true positives of only 13.7%, while the 3-round version of PAS-sequential yielded a frequency of true positives of 39.1%. This is likely because of the effect previously mentioned, where better-response dynamics tends to produce increasingly stable profiles.

Figure 3.3 shows that the probability of finding a beneficial deviation from the current restricted-game MSNE σ empirically decreases in each round of Algorithm 2. In a typical run, the initial strategy set does not yield an almost-stable MSNE at $\epsilon = 0.05$. With each round of the algorithm, however, the distribution of beneficial deviation probabilities shifts to the left, as desired.

To show that Algorithm 2 is feasible with much lower ϵ and higher K, I ran a follow-up experiment with $\epsilon = 0.0005$, $K = 12$, and other settings as before. Over 400 trials, I achieved the following frequencies: true positive 38.5%, true negative 18.7%, false positive 0.0%, false negative 42.8%. The mean rounds before termination was 10.91, of a maximum 12 possible.

Figure 3.4 shows the acceptance rate of Algorithm 2 as a function of the true beneficial deviation probability p, when $\epsilon = 0.0005$, $\delta = 0.1$, $\mathcal{E} = 0.0$, and $K = 12$. Observe that, as desired, for $p > 0.0005$, the acceptance rate is below 0.1. The figure is based on 400 trials.

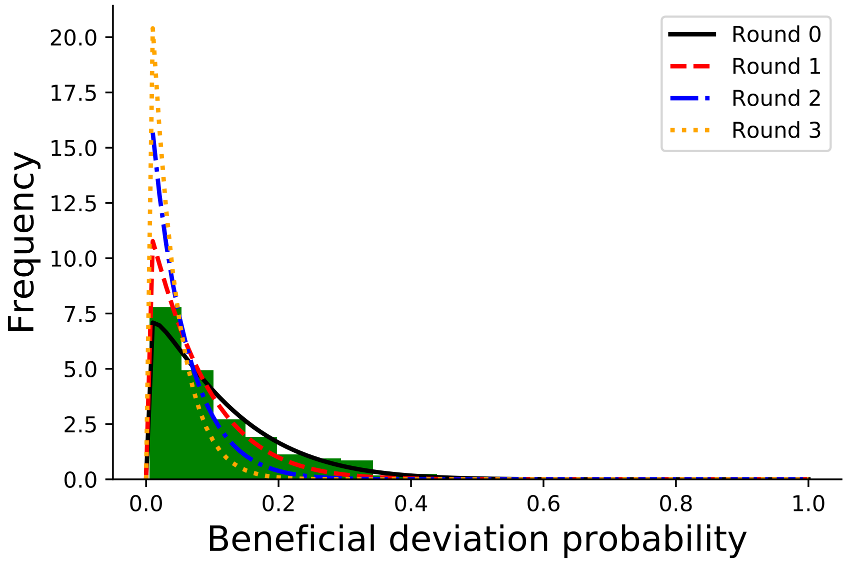

FIGURE 3.3: Empirical distribution in FPSB of deviation probability; random strategies $\sim D_2 = U(0,1)$, vs. MSNE after K rounds of Algorithm 2. Histogram shows Round 0.

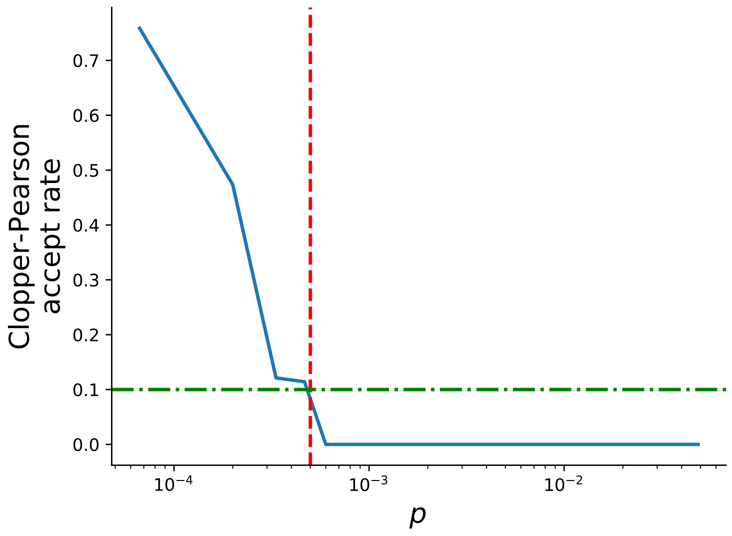

FIGURE 3.4: Accept rate in FPSB of Algorithm 2 vs. probability p of finding a beneficial deviation, where $\alpha_i = \frac{\delta}{12}$ for up to $K = 12$ rounds. Horizontal line is $\delta = 0.1$; vertical line $\epsilon = 0.0005$. X-axis has log scale.

This experiment took considerably longer to complete than the others, due to the larger number of candidate deviations to evaluate per round, and due to the more challenging equilibrium-finding problems, with their many similar pure strategies. Note, however, that the asymptotic performance of Algorithms 1 and 2 is reasonably efficient in theory, based on the rate of growth in runtime with respect to the allowed deviation probability ϵ and confidence level δ. Specifically, the growth in the runtime of Algorithm 1 is linear in $\frac{1}{\epsilon}$ and logarithmic in $\frac{1}{\delta}$. The sequential Algorithm 2 adds only an additional $K \log K$ factor to runtime, where K is the maximum number of rounds.

	ϵ	K	TP	TN	FP	FN
FPSB/Alg. 1/rand.	0.05	-	12.6%	64.9%	1.1%	21.4%
FPSB/Alg. 2/rand.	0.05	3	39.1%	24.0%	0.8%	36.1%
FPSB/Alg. 2/rand.	5e-4	12	38.5%	18.7%	0.0%	42.8%
FPSB/Alg. 2/s. a.	0.05	4	95.5%	4.5%	0.0%	0.0%
Cyber./Alg. 2/rand.	0.05	3	26.0%	23.4%	1.0%	49.6%
Cyber./Alg. 2/s. a.	0.05	10	93.0%	1.0%	0.0%	6.0%

TABLE 3.1: Resulting frequencies of True Positive, True Negative, False Positive, and False Negative.

FPSB with simulated annealing

Here I present results for the FPSB auction, when simulated annealing over $\kappa = 50$ steps is used instead of simple random search as the strategy exploration method. In this experiment, I use a similar configuration to the above, with 10 strategies initially available, $\epsilon = 0.05$, and $\delta = 0.1$.

However, because simulated annealing is much better at finding beneficial deviations than simple random search, I increased the maximum strategies added, K, to 4. I also used a nonzero $\mathcal{E}$ of 0.0001.

Over 400 sampled games with different, randomly-generated initial strategy sets, the PAS-sequential procedure returned an *accept* result (i.e., a supposedly stable profile) in 382 cases, or with frequency 0.955. To evaluate the ground-truth likelihood that simulated annealing would find an $\mathcal{E}$-beneficial deviation from each returned profile, I ran 200 independent trials of simulated annealing. In every trial, the follow-up test frequency of finding $\mathcal{E}$-beneficial deviations was below δ for positive results and above δ for negative results, yielding a frequencies of true positives of 95.5% and frequency of true negatives of 4.5%. These results demonstrate that in a simple domain like the two-player FPSB auction, the iterated profile search method combined with simulated annealing can perform remarkably well. (Note that the reasons for the nearly-ideal performance include the positive $\mathcal{E}$ tolerance, interacting with the smooth payoff function of the FPSB game, and the rounding of sampled strategies to 6 decimal places.)

Accounting for modeling error in distribution D_2

Consider the case where the distribution D_2 that an opponent will use to sample deviating strategies is not equal to the assumed distribution, which I will call $\hat{D}_2$, that is used to construct the probably almost stable guarantee. In the general case where no restriction is placed upon the differences between D_2 and $\hat{D}_2$, the true likelihood of a beneficial deviation being found by sampling D_2 could be arbitrarily high, such as in the case where D_2 places all probability on the set of beneficial deviations.

However, as an empirical test of the effect of error in modeling D_2 with $\hat{D}_2$, I conducted an experiment in the FPSB setting where the true D_2 used to sample deviations is the uniform distribution over parameters in $\mathbb{R}^3$, but the distribution $\hat{D}_2$ used to construct the probably almost stable guarantee is the simulated annealing process described above. One would expect that simulated annealing would yield a higher frequency of beneficial deviations than uniform sampling (i.e., simple random search); thus, if simulated annealing is used to find probably almost stable profiles, while the uniform distribution is used as the ground truth distribution from which to sample, the probably almost stable guarantee should be met even more comfortably than expected.

Indeed, the results indicate that when the true sampling distribution for strategy parameters is uniform, the outcome frequencies produced by PAS-sequential shift to: 0.995 true positive, 0.005 false positive; from 0.955 true positive, 0.045 false positive when the true distribution is generated by simulated annealing. Over all the final profiles returned by PAS-sequential, the mean probability of beneficial deviation by simulated annealing was 0.022, but only 0.002 for uniform sampling. This empirically shows that in cases where the assumed sampling distribution $\hat{D}_2$ has at least as high a probability of finding beneficial deviations as the ground truth distribution D_2, the PAS algorithms appear to uphold their guarantees.

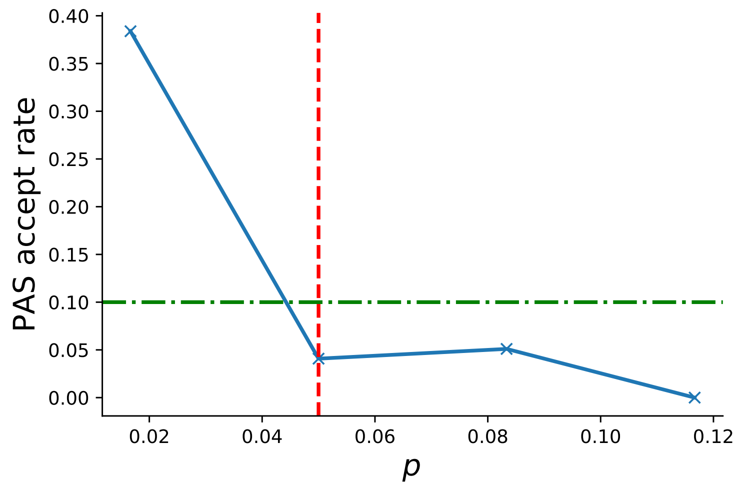

FIGURE 3.5: Accept rate in cybersecurity game of Algorithm 2 vs. true probability p of finding a beneficial deviation, where $\alpha_i = \frac{\delta}{3}$ for up to $K = 3$ rounds. Horizontal line is $\delta = 0.1$; vertical line $\epsilon = 0.05$. As desired, accept rate is below δ for $p > \epsilon$.

3.7.2 Cybersecurity game

In the cybersecurity game, I begin with a fixed set of 10 strategies for the attacker and 12 for the defender. The attacker plays a mixture over this strategy set, while the defender explores randomly sampled alternative strategies. The defender's distribution D_2 may be viewed as sampling parameterizations $(a, b, c) \in \mathbb{R}^3$ for a heuristic strategy that takes 3 parameters, where each parameter is drawn i.i.d. from $U(0, 1)$. I verified empirically that under the initial strategy set, the beneficial deviation probability is greater than $\epsilon = 0.05$, so the initial restricted game's MSNE is *not* almost stable. As before, let $\delta = 0.1$. Begin again with the special case where $\mathcal{E} = 0$, so all payoff improvements count as $\mathcal{E}$-beneficial deviations.

Figure 3.5 shows the accept rate of Algorithm 2 versus an estimate of the true beneficial deviation probability, in the cybersecurity game. In this setting, I estimate the true beneficial deviation probability for the final profile σ when Algorithm 2 terminates, by randomly sampling 400 deviations from D_2 and finding the sample mean payoff of each deviation over 250 simulations. I collected data from 700 independent runs of Algorithm 2, requiring about 400 hours of computation time, the majority of which was used to estimate the true beneficial deviation probability of each terminal profile σ.

Note that as shown in Figure 3.5, for all $p > \epsilon$, Algorithm 2 produces an acceptance rate less than δ, as desired. The figure shows a less smooth curve than for FPSB, with an acceptance rate with $p \approx \epsilon$ markedly below δ, perhaps because the estimation procedure for p did not obtain enough samples to generate accurate estimates of the true beneficial deviation probability. It may be that the Bonferroni correction is too conservative in this setting, which is likely if the hypothesis tests' ground-truth results are positively correlated. In the cybersecurity game, Algorithm 2 yields frequencies of true positive 26.0%, true negative 23.4%, false positive 1.0%, false negative 49.6%. Algorithm 2 terminated after a mean 2.89, median 3 rounds, of up to 3 possible.

Cybersecurity game with simulated annealing

In the cybersecurity game, we test simulated annealing of $\kappa = 5$ steps as the strategy exploration method. The experiments begin with the same initial strategy set as the cybersecurity game study with simple random search. As before, we set $\epsilon = 0.05$ and $\delta = 0.1$. We use maximum round count $K = 10$. Due to computational constraints, we reduce the sample count for each payoff estimate to 100, and sample only 100 strategy deviations via simulated annealing to estimate the ground-truth probability of beneficial deviation. To reduce the occurrence of spurious deviations being found, due to the lower sample count used for payoff estimation, we increase the $\mathcal{E}$ threshold to 1.3 in this experiment. Each run of the experiment consumed between 40 and 120 hours on one Intel Xeon CPU, depending on the number of rounds required.

We have performed 97 runs of Algorithm 2 in the cybersecurity environment with simulated annealing. The mean number of stages before convergence was 4.4 out of the maximum possible of 10. The fraction of runs that returned *accept* was 0.93. All accepting runs appear to be true positives, with a mean estimated ground-truth success probability of 0.009. However, most runs that returned *reject* appear to be false negatives: 0.06 fraction of all results were false negatives, and 0.01 were true negatives. (One *reject* run was terminated by the server because it exceeded the allotted wall time of 100 hours, after rejecting but before the ground-truth deviation probability had been estimated. As a result, I could not measure whether the result was a true negative or false negative.) More encouragingly, the mean estimated ground-truth success probability of runs that returned *reject* was 0.037, much higher than for the runs that returned *accept*. Similarly to the FPSB environment, we observe that using a stronger strategy exploration distribution, instead of simple random search, leads to a higher frequency of true positives rate for Algorithm 2.

3.8 Discussion

This chapter introduced a strategic stability property called *probably almost stable*, and showed it can be efficiently verified in large simulation-based games. I presented a hypothesis test for the property and a sequential search method for probably almost stable profiles. I derived the asymptotic time and space complexity of these algorithms, in terms of the tightness of the desired statistical bounds. I empirically demonstrated the efficacy of the new techniques in the FPSB auction and a cybersecurity game, and showed how the frequency of true positives of results can be improved by using simulated annealing as the strategy search method.

I stated the probably almost stable guarantee from the perspective of player 1 in a two-player game, but the concept can be easily extended to n-player games, or to considering all players' deviation probabilities simultaneously. For example, for a profile σ in an n-player game, the analyst could run Algorithm 1 independently for each player i, sampling pure strategies from D_i, and accept σ as probably almost stable only if each run of the algorithm accepts.

The experiments used high values of ϵ and δ (i.e., 0.05 and 0.1), relative to what analysts might want to apply in practice. For example, an analyst might consider a profile almost stable only if $p \leq 10^{-4}$, such that even a resourceful agent would be unlikely to find a beneficial deviation. I did not use lower ϵ or δ because the computational time required increases with the number of pure strategies available (which must increase to reduce p and ϵ), or with the accuracy desired in acceptance probability estimates (which must increase as δ decreases).

The probably almost stable concept does not provide any bound on the *regret* of a profile σ, which is the maximum any agent could gain by deviating unilaterally from σ. This limitation is unavoidable by methods that sample payoffs for only a subset of strategies, however, because there could exist a strategy not yet sampled with arbitrarily high payoff for an agent i.

In future work, one could try tuning the algorithms to increase the frequency of true positives, subject to the probably almost stable guarantee. One might alter Algorithm 1 or 2, exchanging computational time for more true positives, by setting N such that $k > 0$ deviations or fewer lead to acceptance. One could also tune Algorithm 2 by adjusting the number of rounds or the confidence-spending vector $\vec{\alpha}$. For any setting, there is likely some ideal round count, such that there are enough iterations of better-response dynamics to yield stable profiles, but the confidence budget per round α_k is not too low. Moreover, assigning more confidence budget to later rounds (where stable profiles are more likely) may yield more true positives than a constant budget.

Chapter 4

Evaluating the Stability of Non-Adaptive Trading in Continuous Double Auctions

4.1 Introduction

The continuous double auction (CDA) is the preeminent security trading mechanism, accounting for trillions of dollars in transactions annually [63]. In a CDA, buyers and sellers submit orders to the market, and any order that crosses the best-priced prior order of opposite type *clears*, producing a trade. Due to this mechanism's prevalence, many prior works have attempted to characterize stylized facts of CDA market outcomes, based on simulation or analysis of rule-based traders in action [11], [13], [43], [89]. The literature has also seen a progression of works, each presenting a novel policy for CDA trading agents and experimental evidence comparing it beneficially to less sophisticated policies from earlier papers [15], [29], [71], [83], [84], [87].

Prior studies of heuristic strategies contribute to a strategic understanding of the CDA mechanism, but basic questions remain unanswered. For instance, if one equilibrates strategies over a class of simple heuristic policies, what is the regret of this solution—that is, how much further gain is available by going beyond this class? In particular, one way to refine a strategy is to condition its actions on additional features, adapting its behavior by taking account of more state information. It would be helpful to know whether agents can benefit significantly by adopting more complex, adaptive policies, particularly as analysis of larger strategy spaces may be difficult or costly. Recent work suggests the outcomes of economic simulation studies can be biased by which learning modality agents use, and RL is a useful tool for exploring how the learning environment affects equilibrium results [50].

I present a systematic experimental study of the CDA, in which I derive trading policies via RL and empirical game-theoretic analysis (EGTA). This work uses as a baseline trading heuristic the Zero Intelligence (ZI) strategy, which has minimal capacity to adapt to market state. The version of ZI used here has a few parameters, which are tuned via EGTA to find approximate Nash-equilibrium mixtures. Against these policies I train more sophisticated trading policies through Q-learning [93]. I conduct a statistically rigorous analysis of the benefit of conditioning a policy on market state, relative to using the simpler baseline policy, ZI. Results suggest the equilibrated non-adaptive CDA policies leave positive, but surprisingly modest, room for gain through conditioning on market state.

I also present the most detailed analysis to date on the nature of automatically learned policies for trading in CDAs. For example, I use regression trees as suggested by Schvartzman and Wellman [71]. I then classify conditions where learned policies demand more or less surplus from trade than the ZI mixed-strategy baseline.

Also, I present simple analytical arguments on the conditions when conditioning a CDA trading policy on market state can be helpful. A further empirical analysis shows that market state indicates to a trading agent, most critically, the likelihood at which an order at some price will be executed. The most useful signals for the agent appear to be the recent order history and the current bid and ask.

4.2 Related work

Several recent works in computational economics have employed the simple trading strategy of Zero Intelligence (ZI) in CDA market models. A ZI trader is parameterized with a minimum and maximum surplus to demand from each trade, and it sets each order price based on a uniform random draw from this range. ZI was introduced by Gode and Sunder [30], to demonstrate how a CDA market's allocative efficiency approaches its optimum, even if all traders use such a simple strategy. The ZI policy model in various forms has been popular among experimental and analytical researchers alike, for its simplicity and ability to capture stylized facts of real markets or fit real-world financial data [25], [52], [58]. Several recent works have employed ZI traders in models of financial markets or prediction markets [13], [52], [88], [90].

It is clear that ZI is not an optimal trading strategy. Cliff and Bruten [15] showed that ZI tends to yield efficient allocations only if agents' aggregate supply and demand curves have equal slopes, and the authors proposed one of many strategic improvements on ZI, known as ZI Plus (ZIP). ZIP and other ZI successors such as GD and GDX [29], [83], [84] and AA [20], [87] adjust the surplus demanded by the agent during a run. Studies have shown such policies to be beneficial deviations from a single, fixed ZI policy [84], [91]. Even stronger policies have been derived by RL, to deviate beneficially from a mixed strategy of GDX agents [71]. These studies have the limitation that they compare a new policy against a single, uncalibrated parameterization of ZI, which may be a straw man form of the ZI agent. The more relevant comparison, I would argue, is to equilibrated ZI mixtures rather than to arbitrary ZI instances.

The prior work most similar to this one is a study by Schvartzman and Wellman [71], which used RL to derive the strongest-yet trading strategy in a particular CDA setting. In that work, authors used Q-learning to derive a policy that deviates beneficially from other agents using a fixed ZI strategy, and showed their learned policies also deviate successfully from the strategies ZIP and GDX. This work builds on the methods of Schvartzman and Wellman [71] to serve a different goal. I employ RL in an attempt to characterize when and how CDA traders can benefit from conditioning their actions on market state. I compare equilibrated ZI mixed strategies to (approximate) best responses learned via RL, to measure the strategic effectiveness of calibrated ZI relative to more complex policies. In addition, I analyze the relative importance of features for learning proposed in prior work, through regression over experimentally learned policies.

4.3 Research contributions

I investigate how and to what extent a trading policy in the CDA can improve by conditioning on market state, relative to a calibrated non-adaptive policy. I provide insight into the tradeoffs of using a non-adaptive policy (ZI) to model trading behavior in a CDA, and the relative importance of features for an adaptive trading agent.

My main contributions are as follows:

- I evaluate the strategic stability of calibrated ZI policies against (approximate) best responses from Q-learning. Some surplus is lost by playing a calibrated ZI policy instead of adaptive policies, although the amount is small compared to the surplus lost by using non-equilibrated ZI parameters.

- I analyze the nature of adaptive policies that outperform ZI.

- I provide intuition for when an adaptive agent can deviate beneficially from a ZI policy baseline, and lend insight into the trade-offs facing such an agent.
- The findings suggest equilibrating over many ZI policies yields a profile where no alternative ZI policy can earn significantly greater surplus, but an adaptive policy can still earn a small positive amount more. The common practice of using equilibrated ZIs as an approximation of efficient behavior is likely acceptable.

4.4 CDA market model

This study is based on a CDA market model, similar to those of prior studies by Wah and Wellman [89], [90]. The market has a single security and many trading agents. The security's value to an agent is the sum of the agent's private value for the good (drawn from some random distribution), and the fundamental value, which evolves by a stochastic process. Agents trade the security with one another via the CDA mechanism, by submitting limit orders to the market. Each agent can submit an order only in time steps when it *arrives* at the market, as determined by a random (exponential) inter-arrival time process; at each arrival, an agent is independently randomly assigned to buy or sell. Each agent's payoff from the CDA game is defined as the final fundamental value of its inventory, plus the cumulative private value of its inventory, plus its final cash holdings.

The market model has 17 trading agents, comprising 16 background traders and one market maker (MM). The MM maintains a *ladder* of buy and sell orders separated from the expected final fundamental value by a fixed spread, updated each time it arrives. The background traders act according to parameterized forms of the Zero Intelligence (ZI) policy.

4.4.1 Zero Intelligence

ZI is a simple strategy for CDA trading that can converge to efficient prices and allocations in many settings [30]. The variant of ZI employed here, introduced by Wah et al. [90], has three parameters: $\underline{d}$, $\overline{d}$, and $\eta \in (0, 1]$. At each arrival, a ZI agent places a limit order that demands a surplus equal to a random draw from $\mathcal{U}(\underline{d}, \overline{d})$. The exception is if the agent would earn at least η fraction of its randomly drawn surplus goal at the current quote; in that case, the agent opportunistically places an executable order at the quote instead.

4.4.2 Market model description

All 17 agents arrive at the market with independent inter-arrival times, drawn from an exponential distribution with rate λ_{BG} for background traders, λ_{MM} for the market maker. Let $\lambda_{BG} = 0.012$ and $\lambda_{MM} = 0.05$, with a game duration of $T = 2000$ time steps. Hence, each background trader arrives roughly every 83 time steps in expectation, the market maker every 20 time steps.

The fundamental value evolves as a mean-reverting random walk with zero-mean Gaussian noise and long-run mean μ.[1] At each time step, the fundamental value is updated, $r_t \leftarrow \kappa\mu + (1-\kappa) \times r_{t-1} + \mathcal{N}(0, \sigma_s^2)$, where σ_s^2 is the shock variance to public value, and κ is the mean reversion parameter. Throughout this study, I use $\kappa = 0.01$ and $\sigma_s^2 = 20000$. Given the observed fundamental at time t, the expected terminal fundamental value is

$$\hat{r}_t = \left(1 - (1-\kappa)^{T-t}\right)\mu + (1-\kappa)^{T-t} r_t.$$

ZI and MM agents use this estimate in determining their order prices.

Each background trader is assigned a private value vector at initialization, which lists the value of each additional security unit, given a current inventory. This vector has length 20, because the agent is restricted to hold (or owe) no more than 10 units. The vector is derived by sampling 20 values from $\mathcal{N}(0, \sigma_p^2)$, where σ_p^2 is the private value variance; the samples are sorted in non-increasing order, so that each agent's demand decreases with inventory. This study uses $\sigma_p^2 = 2 \times 10^7$.

When the market maker arrives, it cancels its existing orders and places a new ladder of orders, with 100 rungs each above and below the expected final fundamental. The ladder has orders centered around $\hat{r}_t$, at sell prices of $\hat{r}_t + 256 + 100 \times i$ and buy prices of $\hat{r}_t - 256 - 100i$, for $i \in \{0, \dots, 99\}$.

When a background trader playing a ZI strategy arrives at the market, it cancels its previous order and is assigned with equal probability to place a buy or sell order. Suppose the agent has ZI parameters $\underline{d}, \overline{d}, \eta$. The agent computes the surplus it would obtain by trading immediately at the quote, which for a buyer is $\hat{r}_t + v_{i+1} - A$, or for a seller is $B - (\hat{r}_t + v_i)$, where A is the ask, B is the bid, and v_i is the agent's private value of unit i of inventory. The agent compares this surplus to ηs, where s is a random draw from $\mathcal{U}(\underline{d}, \overline{d})$. If the agent can obtain enough surplus, it transacts immediately. Otherwise, it places an

[1] I take $\mu = 10^5$. The specific level does not matter, if the evolving fundamental has negligible probability of hitting the zero lower bound.

order demanding the surplus goal s: either a buy order at $\hat{r}_t + v_{i+1} - s$ or a sell order at $\hat{r}_t + v_i + s$.

Each agent earns a payoff equal to the final fundamental value of its stock inventory, plus the cumulative private value of its stock inventory, plus its final cash holdings.

4.5 Reinforcement learning methods

To investigate how much adaptive policies can increase payoffs relative to a suitably calibrated ZI policy, as well as the nature of beneficially deviating policies, one needs a method to search for beneficial deviations in a large search space. I tested several RL approaches, including Sarsa [78], Sarsa-λ (i.e., with eligibility traces), and POMCP [76], before settling on a variant of *Q-learning* that empirically worked well in this setting. I found that Sarsa variants performed almost as well as Q-learning, but POMCP took much longer to run and performed worse; there was a significant gain by using Q-learning instead of an alternative. I first fix the policies of all but one agent, which converts the trading game into a decision problem for the one strategic agent, then apply Q-learning.

Q-learning is a classical RL algorithm that provably converges to an optimal policy in finite Markov decision processes (MDPs) with bounded rewards, assuming a suitable learning rate sequence is used [93]. It works as follows. The learning agent progresses through a sequence of observing states s, getting rewards r, and taking actions a. The agent maintains an estimate of the Q-value of each state-action pair, $Q(s, a)$, which represents the expected value of taking action a in state s and playing optimally thereafter. On experiencing the sequence (s, a, r, s'), the agent performs a Q-learning value update,

$$Q(s,a) \leftarrow (1 - \alpha(s,a))Q(s,a) + \alpha(s,a) \left(r + \gamma \max_{a'} Q(s', a') \right),$$

where $\alpha(s, a) \in (0, 1)$ is the learning rate, and γ is the discount factor for future values. I set $\gamma = 0.9$ for learning, as a regularizer, but decay γ toward 1 as learning progresses; these parameter values were chosen based on pilot experiments, and as suggested by reports in the literature that similar values had worked well before. However, the results did not appear extremely sensitive to the γ parameter setting. (The underlying game has no discounting.) I set $\alpha(s, a)$ to the reciprocal of the number of (s, a) observations to this point.

The learning agents employed in this study use the following features in their state observations.

- P, the profit that would be obtained by trading immediately at the current price quote.
- V, the private value of the next unit to be traded.
- O, the *Omega ratio*, estimated at recent trade prices, of the price X with respect to a threshold k defined at the next unit's valuation,

$$\frac{\mathbb{E}(X - k \mid X > k)\,\Pr(X > k)}{\mathbb{E}(k - X \mid X < k)\,\Pr(X < k)}.$$

- A, whether the action assigned to the player is buy or sell.
- D, the duration in time steps since the most recent trade.

Note that ZI agents in the market setting are randomly assigned a role of buyer or seller at each arrival at the market, and for a fair comparison of the agent's abilities with ZI, the RL agents must be assigned their roles as well; therefore, A is part of the agent's observation space, not the action space.

To discretize the observations as is needed for Q-learning, I employ a *tile coding* system with a single tiling [71], [72]. That is, I set thresholds for the numerical features (P, O, V, and D), dividing each into three buckets, using threshold values chosen empirically in pilot simulations to provide evenly distributed observations over buckets. This system was chosen based on pilot experiments, in which a variety of tile configurations was used, including as many as nine buckets per feature; the setup with three buckets per feature performed the best, based on the average payoff of the learned deviating policies. The Q-learning training results were quite sensitive to the number of buckets used, as using more buckets seemed to require more training steps in order to learn strong policies, due to the lower frequency of samples falling into each bucket. It is possible that a function approximation approach, such as deep Q-networks, could outperform tile coding in this setting, but I have not investigated this question. Moreover, it is possible that the Q-learner's performance could be improved by including the full history of observations and actions in the agent's input, instead of only the current observation, because the environment is not completely memoryless. However, in this environment the benefit of memory to a lone RL agent appears likely to be small, because the ZI agents have no memory, and the only effect of history on their behavior is through changes in their stock inventory.

4.6 ZI regret study

I designed an experiment to measure the regret of equilibrated static policies (ZI) with respect to either novel ZIs or an approximate best response over adaptive policies, derived via RL. As a sanity check, I wanted to show that the automated RL process could consistently find policies that outperformed the ZI baseline, as found in prior work [71].

With a consistently effective learning process in hand, I sought to measure the strategic stability of equilibrated ZI mixed strategies with respect to approximate best responses derived via Q-learning. (By an *equilibrated* ZI mixed strategy, I mean a probability distribution over ZI strategy parameters exhibiting negligible empirical regret, relative to a fixed set of other ZI strategies.) I expected an arbitrarily chosen ZI pure strategy would have high regret with respect to a Q-learner or to other ZI strategies. More important, I hypothesized that as the set of ZI strategies is increased in size, the regret of the equilibrated mixed strategy with respect to either other ZI policies or a Q-learner will tend to diminish. The regret with respect to the other ZI policies necessarily approaches zero in the limit, but I expected there would remain a small positive regret with respect to a reinforcement learner. This regret represents the value of conditioning actions on market state in the CDA. If the measured regret is indeed small, this is evidence supporting the use of equilibrated ZI traders as a reasonable agent model.

4.6.1 Definitions

This chapter uses many standard terms from game theory, defined here for completeness. A *policy* or *pure strategy* is a mapping from an agent's set of observation states to the (possibly stochastic) action the agent will take in each state. A *mixed strategy* is a probability distribution over pure strategies. A *profile* is an assignment of a strategy (pure or mixed) to each agent. A *symmetric* profile assigns the same strategy to each agent. A *Nash equilibrium* (NE) is a profile such that no agent can achieve a higher expected payoff by unilaterally deviating from its assigned strategy to any alternative strategy. The *regret* of a profile is the maximum over agents, of the maximum gain in expected payoff the agent can obtain, by deviating to any alternative strategy. By definition, a Nash equilibrium has zero regret.

In this work, a profile is called an *equilibrium over strategy set* $\mathcal{S}$, if all pure strategies played with positive probability are in $\mathcal{S}$, and no agent can achieve higher expected payoff by deviating to a strategy in $\mathcal{S}$. A profile is said to have *regret* x *with respect to strategy set* $\mathcal{S}'$, if x is the maximum any agent gains in expectation by deviating to a strategy in $\mathcal{S}'$.

4.6.2 Experiment design

To measure the strategic stability of calibrated ZI strategies, I began with a set of 10 ZI policies, selected heuristically for high fitness and broad coverage of the space of viable policies. I generated random subsets of the base strategy set, of several sizes, and used empirical game-theoretic analysis (EGTA) to find one or more symmetric Nash equilibria in each subset.[2] Next I challenged each ZI equilibrium strategy, by training a Q-learner against other agents playing that mixed strategy. I also challenged each distinct equilibrium strategy with each of the 10 pure ZI strategies, to evaluate the regret with respect to the base strategy set.

ZI strategy set

The 10 ZI strategies $(\underline{d}, \overline{d}, \eta)$ used in this study are as follows:

$$(0,450,0.5),(0,600,0.5),(90,110,0.5),(140,160,0.5),\\ (190,210,0.5),(280,320,0.5),(380,420,0.5),(380,420,1),\\ (460,540,0.5),(950,1050,0.5).$$

Henceforth, a pure strategy will be written as, for example, `280_320_.5`. A mixed strategy will be written as a series of ordered pairs of pure strategies and their probabilities, such as (`280_320_.5` $\times$ 0.1, `380_420_.5` $\times$ 0.9).

From this base set of 10 strategies, I randomly selected subsets of sizes two, five, or eight. (These subset sizes were chosen, somewhat arbitrarily, to provide a representation of diverse sizes between one and the full set of ten, without requiring as much computation time as training RL models against equilibria from all 11 possible subset sizes. I believe the results are not sensitive to the exact subset sizes chosen.) Strategy subsets were selected uniformly randomly, rejecting duplicates. I used 30 distinct subsets of each size, in addition to the 10 subsets of one ZI strategy each, as well as the base set containing all 10 strategies. Overall, I conducted parallel experiments on 101 ZI strategy sets: 10 of size 1; 30 each of sizes 2, 5, and 8; and 1 of size 10.

[2]Because the games used here are finite and symmetric, symmetric NE necessarily exist. I numerically find approximate symmetric equilibria with negligible regret.

EGTA methods

I used the methods of EGTA to find NE over each subset of ZI strategies. The essential EGTA process has been described at length in many earlier studies [88], [90], so this section presents only an overview. EGTA uses simulation to estimate the expected payoff for each agent in a strategy profile, and explores a space of profiles to identify approximate equilibria.

To test whether a mixed-strategy profile is an equilibrium, EGTA obtains payoff samples of each pure-strategy profile in the support of the mixed strategy (called the *subgame* of the support), as well as each pure-strategy profile where a single agent deviates to any other pure strategy. For example, if the 16 players in this game play strategies A and B with positive probability, it is necessary to sample payoffs of $i \in \{0, \ldots, 16\}$ agents playing A and the rest playing B; I then compute the expected payoff of the mixed strategy. Next, I would compute the payoffs for corresponding profiles where one agent deviates to any other pure strategy.

The sample count required grows rapidly in the number of agents and strategies in support. To make this process tractable, I employ the *deviation-preserving reduction* (DPR) technique of Wiedenbeck and Wellman [96], approximating the game of 16 players with a related 4-player game. I construct a reduced game's payoff table by running simulations of the full, 16-player game as follows. To estimate the payoff for a particular player in a 4-player reduced-game profile, I let that player control 1 agent in the 16-player simulation, while each of the other 3 players in the reduced game controls 5 agents in the full simulation.

I search for NE through a fully automated procedure that begins by testing whether each pure strategy in self-play is an equilibrium. The process then goes on to test equilibria over pairs of strategies, based on beneficial deviations found from the self-play profiles. Exploration continues, extending support size as necessary based on deviations found outside the current support. The process completes when an approximate NE is found with empirical regret less than a numerical tolerance, and all equilibrium candidates up to a current support size have been confirmed or refuted. For a given subgame, replicator dynamics are used [81] to search for a symmetric NE over those strategies.

Pure-strategy regret measurement

I set out to accurately measure the regret of each equilibrium found over a subset of ZI strategies, with respect to the base strategy set. This value serves as an empirical signal of how strategically stable a ZI mixed strategy is, with respect to the universe of all ZI strategies, if it is believed that the base strategy set is sufficiently large and varied.

To estimate the regret of a mixed strategy M with respect to one of the 10 ZI strategies in the base strategy set, I run simulations where all but one agent plays a strategy sampled independently from M, but one agent deviates to that pure strategy. I explored the 10 pure strategies as if they were arms of a multi-armed bandit, seeking an upper bound on the regret with respect to the best arm. Initially, I sampled each strategy in turn, for 50,000 simulations each. Thereafter, following each batch of 2500 simulations, I sampled a bootstrap distribution from the set of all payoffs of the current deviating strategy and selected the pure strategy with the highest 95th percentile for the mean payoff as the deviation to sample next. Thus, I obtained low-variance estimates for the upper confidence bound on those pure strategies that appeared to have a significant chance of being beneficial deviations. I terminated the process when either the greatest upper confidence bound became lower than the expected payoff of the equilibrium policy, or a total of 800,000 simulations had been taken.

This pure-strategy regret measurement procedure lets us evaluate the strategic stability of supposed NE, in case of approximation error caused by player reduction. It is possible for a mixed strategy to be an exact Nash equilibrium in the *reduced game* of the DPR approximation, but not an equilibrium in the original game, because the original game includes payoffs where non-round numbers of agents play each strategy. The regret measurement procedure employs the original, 16-player game, so it can sample payoffs where any number of background traders from 0 to 16 adopts each strategy in the equilibrium support. In this way, I empirically test whether the reduced-game NE are actual equilibria in the original game, and if not, how large their regret may be.

Q-learning regret measurement

I aimed to measure the regret of each equilibrium over a subset of ZI strategies, against the best adaptive policy derived in a large policy space by Q-learning. This provides a lower bound on how much better an adaptive agent can perform in the CDA than an agent that plays a fixed policy (ZI).

I conducted Q-learning against each equilibrium ZI strategy as described above. In summary, I performed a single run of Q-learning against each equilibrium, of 10^6 playouts. The exploration policy was ϵ-greedy, with $\epsilon = 0.1$. Q-learning was slightly modified, based on empirical observations of the most useful tricks in this setting: I truncated reward observations to ± 3000, used an artificial discount factor of 0.9 that decays to 1.0

with increasing iterations, used hand-tuned thresholds in each feature for observation bucketing, and used early stopping. More specifically, to tune the thresholds for bucketing each feature, I simply estimated the thresholds between quantiles, using a pilot run with an equilibrium profile of ZI agents. For three buckets per dimension, I chose boundaries that split the training data into tertiles, or three equal-probability regions. Setting these thresholds appropriately is vital to the training process, as training is most effective when samples are distributed roughly evenly across buckets. To perform early stopping, I recorded the current policy at regular intervals during the later rounds of training, then conducted independent testing to evaluate which of the recorded policies yields the highest expected payoff; only this policy would be returned.

To measure the expected payoff of a policy from RL, I run the simulator with one agent playing the learned policy, and all others playing the baseline mixed strategy. I conduct at least 2×10^5 simulations per learned policy, and use the bootstrap to derive a confidence interval for the mean payoff. I then compare this payoff to the expected payoff of the baseline policy.

Note that the policy space used by the Q-learner is a strict superset of the ZI policies in the base strategy set. The Q-learner builds a table that maps each observable state to a ZI policy selected from the base strategy set of 10. This means that the Q-learner can in theory deviate at least as successfully against any opponent profile as the best fixed response from the base strategy set. To see this, observe that the Q-learner could generate a policy that maps every observation to the ZI best response. However, a Q-learner may not always match or outperform the ZI best response, due to insufficient time to converge, or problems converging in a POMDP.

4.7 Results

In the ZI regret study, the automated learning method consistently found policies of greater expected value than the equilibrated ZI baselines. However, the learned policies achieved only slightly greater payoff than those ZI equilibria that were derived from large sets of ZI pure strategies. The results suggest that there is a small but consistent advantage to conditioning actions on state in the CDA environment, relative to playing a well-calibrated mixed strategy of ZI policies. This benefit of an adaptive policy is small compared to the difference in expected payoff between a well-calibrated ZI strategy and a poorly chosen one.

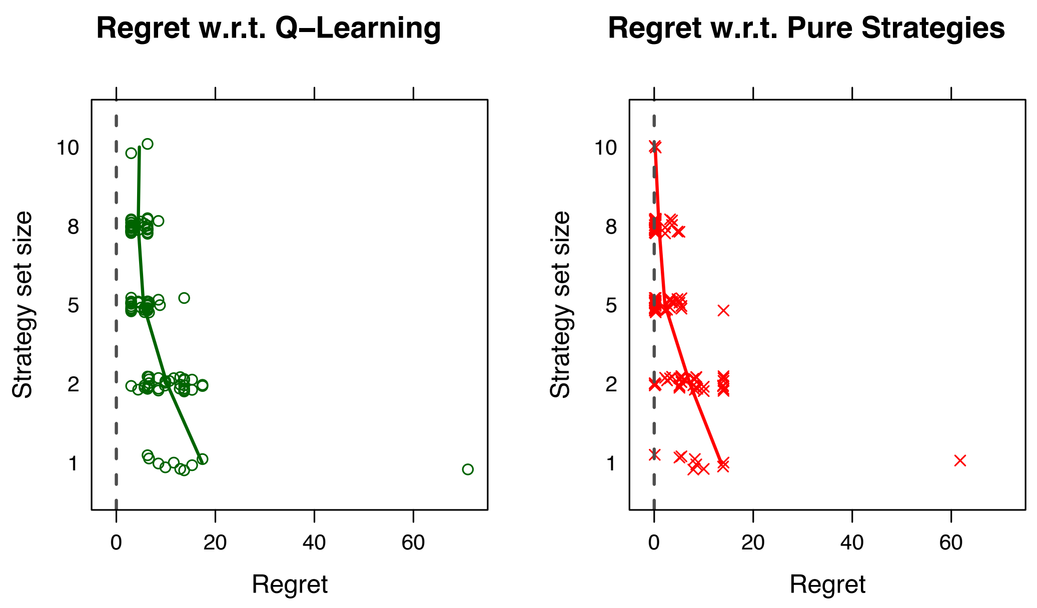

FIGURE 4.1: Regret of equilibria over ZI policy subsets. Each row comprises equilibria over ZI policy subsets of a given size. Left: regret w.r.t. Q-learning response; right: regret w.r.t. best-response ZI policy of full policy set. Solid lines present row means.

4.7.1 Studying effects of ZI strategy set size

As the set of ZI strategies available to EGTA is augmented, the regret of the equilibrium mixed strategy over that set decreases, both with respect to the base set of ZI strategies, and to the adaptive strategy response produced by Q-learning. The regret with respect to other ZI strategies empirically is almost always lower than the regret with respect to adaptive strategies; or in other words, adaptive policies almost always achieve greater benefit in deviating from the ZI baseline than an alternative ZI strategy does.

In Fig. 4.1, I present the mean regret of each ZI subset's NE, with respect to Q-learning (left) and with respect to the best-response ZI pure strategy (right). For example, row 5 displays a marker for each equilibrium in each of the 30 ZI strategy subsets of size 5 that were randomly selected. In any row, each equilibrium is plotted with multiplicity equal to the number of strategy subsets of the appropriate size in which it occurs. A line is used to plot the mean regret of these equilibria for each strategy subset size.

Note in Fig. 4.1 how the regret of ZI equilibria grows smaller on average as the number of strategies equilibrated over increases from 1 to 10. This trend holds for regret with respect to Q-learning and with respect to the best-response ZI policy. The only exception to this trend is the small increase, from 4.4 to 4.7, in the mean regret with respect to Q-learning, from subset size 8 to size 10; this reversal may be due to noise in payoff sampling or the like. For a sense of scale in these payoff differences, note that in the two Nash equilibria found over the base strategy set, the expected payoffs per background trader were 461.8 and 462.6.

This trend of ZI equilibrium regret growing smaller with increasing ZI strategy set size is supported by statistical hypothesis testing via the unpaired t-test. In these tests, I count each equilibrium's regret with a multiplicity equal to the number of strategy subsets in which it appears, similarly to the plot in Fig. 4.1. In the case of regret with respect to Q-learning, I find weak evidence (below statistical significance at 0.05 level) that the regret for size-one subsets is greater than size-two ($p = 0.14$), and strong evidence that regret for size-two is greater than size-five ($p = 10^{-8}$), and size-five is greater than size-eight ($p = 0.02$). In the case of regret with respect to ZI deviations, I find very similar hypothesis test results.

Note that in regards to Fig. 4.1, as the subset of ZI strategies equilibrated over is augmented, the regret of the ZI equilibrium with respect to the base strategy set approaches zero. This regret must be zero when the full strategy set is included, for a Nash equilibrium over the base strategy set cannot have any beneficial deviations within that set. For any subset of k strategies, drawn from a base set of N strategies, the likelihood of a zero-regret subset being selected is simply the likelihood of drawing a superset of the support of any Nash equilibrium of the base set. (The support of a Nash equilibrium is defined as the set of all pure strategies it uses with positive probability.)

Finally, observe how in Fig. 4.1 the regret of ZI equilibria with respect to Q-learning is always strictly positive, even as the number of ZI strategies equilibrated over becomes large. Indeed, the smallest regret of a ZI equilibrium with respect to Q-learning I find is 3.0, and the smallest mean regret for a subset size is 4.4, corresponding to strategy subsets of size 8. This suggests that there is a persistent benefit to adaptive policies, such as those derived by RL in this study, relative to mixtures of ZI policies, even as those mixtures are calibrated over many parameterizations.

Fig. 4.2 presents for each equilibrium the difference in regret between the response derived by Q-learning and the pure-strategy best response from the base strategy set. Each row corresponds to equilibria over subsets of ZI strategies of a certain size. Each equilibrium is plotted with a multiplicity equal to the number of strategy subsets in which it occurs.

Note that in almost all cases, the automated Q-learning procedure achieves more lift

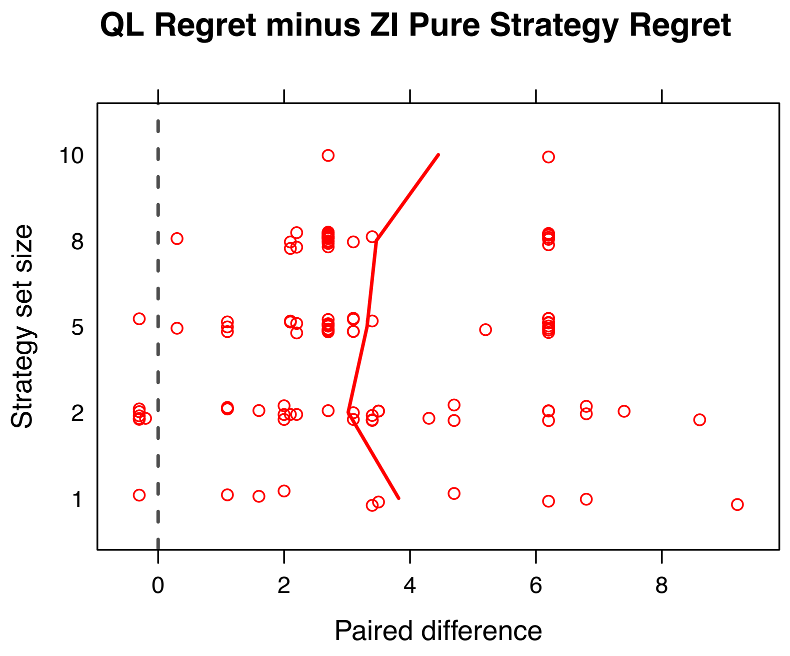

FIGURE 4.2: Paired difference in regret (w.r.t. Q-learning or ZI) for each equilibrium, in ZI policy subsets of various sizes. Regret w.r.t. ZI is subtracted from regret w.r.t. the Q-learning response. Each row comprises equilibria over ZI policy subsets of the given size. Solid lines present row means.

in payoff over the baseline than the ZI best response. In a few cases, it does not, possibly due to insufficient iterations for Q-learning to converge, or the instability of Q-learning in the surface MDP of a POMDP. The increase in payoff improvement of Q-learning over ZI ranges from 3.0 to 4.4, over the various subset sizes, as shown by the solid line. These differences are statistically significant, based on paired t-tests, for subset sizes 1, 2, 5, and 8 ($p = 0.001$, 10^{-8}, 10^{-11}, and 10^{-13}, respectively). It is interesting that the lift of adaptive policies from Q-learning, relative to a non-adaptive ZI best response, appears roughly constant, even as the number of ZI strategies used for equilibration increases. This suggests a lingering benefit from conditioning actions on state, even against non-adaptive agents with carefully tuned parameters, providing a payoff gain of approximately 3.5. (It is conceivable that an RL-trained policy that takes the full history as input could achieve an even higher gain, but because ZIs are memoryless, I expect that the benefit to a deviating RL agent of memory—perhaps encoded by a recurrent network—would be small.)

Deviation payoffs of an example equilibrium

Here I present the range of payoffs for an agent unilaterally deviating from an equilibrium over ZI strategies, to pure strategies in the base strategy set. I will take as an example the

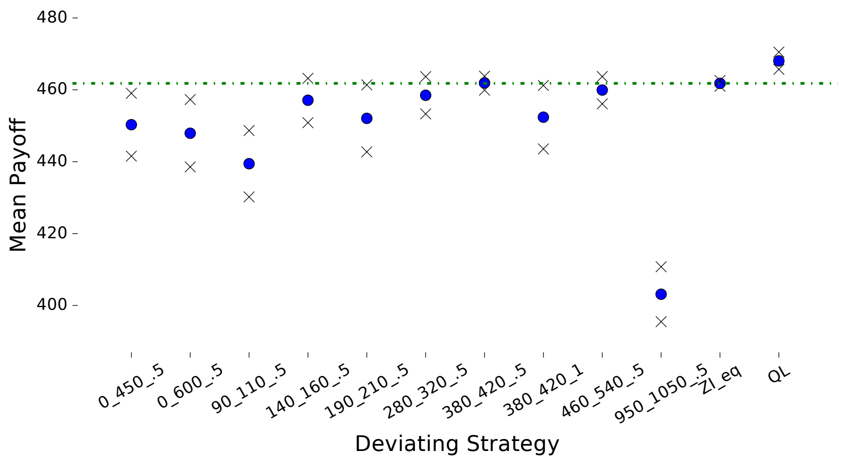

FIGURE 4.3: Empirical payoffs per deviation from an equilibrium of base strategy set. Each column shows a deviating strategy: blue dot is sample mean payoff, crosses are 95% bootstrap confidence interval. `ZI_eq` is the equilibrium strategy (`380_420_.5`). `QL` is the learned response. Dotted line is equilibrium payoff.

pure-strategy Nash equilibrium over the base strategy set, where all background traders play `380_420_.5`.

In Fig. 4.3, I present the empirical distribution of payoffs for each unilateral deviation from this equilibrium profile. The crosses in each column indicate a 95% confidence interval for the mean, derived via bootstrap sampling. Because more samples are automatically collected for pure-strategy deviations that have higher upper confidence bounds, as in the UCB-1 method of sampling [3], some confidence intervals are narrower than others.

Note how all ZI strategies except `380_420_.5` have sample mean payoffs lower than the equilibrium payoff, as expected. The equilibrium action of `380_420_.5` has a mean payoff on resampling almost exactly equal to the independent estimate for the equilibrium payoff, shown as `ZI_eq`. This indicates there have likely been enough samples collected per policy that sampling error is under control.

Finally, observe that the expected payoff of the policy derived from Q-learning (`QL`) is much higher than the equilibrium payoff or any pure-strategy deviation, at 468.1 over an equilibrium payoff of 461.8. This gain from deviating to an adaptive policy, however, is smaller than the loss that would occur by deviating to several inferior ZI pure strategies. Thus, in this example one can see that the payoff gain from conditioning on state as this Q-learner does is moderate, relative to the benefit of choosing a well-calibrated ZI strategy instead of an arbitrary one.

Q-learning performance against ZI strategy sets

For all 20 supposed equilibria over ZI strategies, including the 10 pure strategies and 10 mixed strategies, Q-learning successfully discovered a beneficial deviation over the larger space of adaptive policies. The most common number of training iterations for the best policy was 10^6, which was the end of the training cycle. In several cases, however, an intermediate policy, corresponding to an earlier stopping time, produced a higher expected payoff.

In order to confirm that a learned policy had a payoff significantly greater than the equilibrium baseline, I played back the apparent best policy from a training run for 10^6 total playouts. I then took a bootstrap 95% confidence interval about the sample mean payoff, and in each case the lower bound thus obtained was greater than the mean payoff of the baseline profile.

Summary of effects of ZI strategy set size

In this series of experiments, the automated RL process consistently yielded a beneficial deviation, even against ZI strategies that were equilibrated over the full base strategy set. However, the regret of equilibrated ZI policies grew lower, as the number of strategies used for equilibration was increased. The lift of an adaptive policy from Q-learning, relative to a ZI best response, appears to be almost constant on average, even as the strategic stability of the baseline ZI mixed strategy is increased. This benefit from policy adaptation is positive, but reasonably small, relative to the differences in payoffs between the deviating ZI policies tested.

4.7.2 Intuition behind successful policy adaptation

In the model CDA environment, only a few aspects of market state are not known to a background agent. In fact, the only useful aspects of state assumed to be unknown to the agent when it arrives at the market are: (a) the pure strategy drawn by each other agent from a publicly known symmetric mixed strategy; (b) the private value vector of each other agent; (c) the inventory of each other agent; and (d) any orders beyond the best bid and ask in the limit-order book. There are other unobserved aspects of state, such as previous fundamental values and agent arrival times, but because the fundamental value and arrival processes are memoryless, these state features are of no value in choosing an action.

A policy in the CDA is said to *condition on state*, if the probability distribution over surplus demanded differs, based on the history of the agent's observations and actions. If a policy does not condition on state, the agent has the same probability distribution over surplus demanded, regardless of the observed market state.

An agent's observations are useful only to the extent that they provide information about the private value vectors, inventories, or strategy assignments of other traders, or the hidden part of the limit-order book. An agent in the model used here can benefit through observations, only if its observations help it to predict the orders of the other agents, which are the only environment events that affect its payoff besides the final fundamental value. The agent can be intuitively viewed as attempting to predict the likelihood its order will transact, as a function of the surplus demanded. Many strategy proposals in the literature, such as ZI Plus and GD, attempt to maximize expected revenue based on such a belief over the demand curve. Similarly, I expected that in the analysis of learned policies, I would find the agent to condition its actions on state in a way that appears to demand more surplus in favorable conditions.

4.7.3 Analysis of learned policies

Let us take as a running example the pure ZI strategy equilibrium over the base strategy set, where all agents play `380_420_.5`. This example is chosen because it is an equilibrium over all the base strategies, so a Q-learner deviating successfully from it is making an improvement over any of its component pure strategies. Thus, it is an example of the benefit of policy adaptation over a fixed policy.

To study the range of successful adaptive deviations from `380_420_.5`, I performed 10 runs of Q-learning, selecting the best policy from each run. I analyzed the 10 resulting policies together, to find what they have in common to explain how they improve on the base strategies they are composed of. Note that learning was conducted on a dataset containing 10 policies produced by independent training runs, where each policy comprises 54 entries, corresponding to 3 buckets along dimensions for (P, V, O), and 2 categories for A, namely buy and sell. Thus, there were 540 total state-action pair entries to examine. In Fig. 4.4, I present the distribution of mean surplus demanded by the adaptive agent, over the 10 policies derived by Q-learning against `380_420_.5`. *Mean surplus demanded* is used to mean $\frac{\underline{d}+\overline{d}}{2}$, based on the parameters $\underline{d}$ and $\overline{d}$ of a ZI action. The Q-learner tends to demand slightly less surplus than the baseline ZI agents: 379 on average, compared to 400 for the ZI, and it demands strictly less surplus than the ZI in 56% of its arrivals. It demands strictly more mean surplus than the others, perhaps opportunistically, in 25%

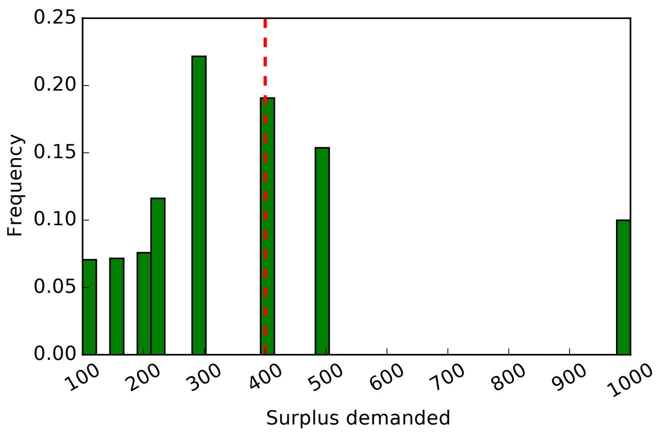

FIGURE 4.4: Histogram of the mean surplus demanded by 10 Q-learning derived policies, deviating from 380_420_.5. Each state-action pair is weighted by the state's occurrence frequency. The mean surplus demanded by the equilibrium baseline policy is shown by the dotted red line.

of arrivals, and the same amount 19% of the time. (In all of these figures, I weight each state-action pair of a learned policy by the state's occurrence frequency.)

Machine learning for policy analysis

It has been noted that tabular policies are difficult to understand intuitively [71]. To draw useful insights into the adaptive policies for CDA trading from Q-learning, I use machine learning techniques that can help to summarize complicated policies.

Here I analyze the set of policies from Q-learning, against a baseline strategy with parameters 380_420_.5, as above, but without the feature D, which did not appear useful in a greedy feature elimination study (not shown). I use regression and classification methods to summarize or draw insight from the set of policies learned in this setting, over multiple independent runs of Q-learning. (The greedy feature elimination study was conducted by training Q-learning agents, in multiple independent training runs, with each feature in turn omitted from the inputs; the feature leading to the smallest mean drop in performance when omitted would then be removed.)

My first effort was to predict the mean surplus demanded by the learner's action in a given market state, where mean surplus demanded is defined as above, $\frac{\underline{d}+\overline{d}}{2}$. Market state is determined by the four features P, V, O, and A. Recall that A, the action type in $\{buy, sell\}$, is a binary feature, while the other features are partitioned into a low, medium, or high bucket. I encode the binary feature as 1 for buy, 0 for sell. I encode each other feature as 0 for low, 1 for medium, 2 for high.

Least squares regression yields a fairly poor fit, with an MAE of 159.7. (I weight each state-action pair according to its state's occurrence frequency, for both learning and evaluation.) The learned coefficients are $P = 74$, $O = 43$, $A = 29$, $V = 4$. This agrees with the results of the greedy feature elimination study (not shown), that P is by far the most critical feature for learning, and O is also important. However, it suggests that V, the private value of the next unit traded, has little linear effect on surplus demanded.

Due to the poor linear fit, I tested decision tree regression, splitting to reduce MSE, with a maximum depth of 3. I found normalized feature importances of $P = 0.58$, $O = 0.34$, $A = 0.01$, $V = 0.07$. Again, P is the most important feature, with O second, though now V is third. These results further support the idea that P is the most useful feature for an adaptive agent in the CDA.

Beyond relative feature importances, these results give insight into the directional effect of each feature on surplus demanded. It appears, from the high positive coefficient in linear regression, that a high P indicates that the agent can earn a large surplus by trading immediately, so it may be an opportune time to demand more surplus than usual.

Similar to the decision tree regression study, I performed decision tree classification, classifying the mean surplus demanded by each state's action as less than, equal to, or greater than the mean surplus demand of the equilibrium ZI (in this case, 400). Classifier splits are chosen based on the Gini coefficient. I achieved a weighted zero-one loss of 0.34 with a decision tree of depth 3. The relative feature importances in this tree were almost identical to those of the decision tree regressor.

A simple, depth-2 decision tree classifier predicts the adaptive agent will demand less surplus than the baseline, if neither P nor O is high. Only with high P (immediate surplus available) or high O (recent transaction prices) will the agent demand higher surplus than the baseline. Deeper decision trees yield higher accuracy, but their rule sets do not appear as interpretable in this case.

Summary of learned policy analysis

I discussed the intuition behind what market information is most useful to trading agents in the market model used here, as well as how agents trade off between the likelihood an order will transact and the surplus demanded. I showed that against a running example ZI equilibrium profile, successful adaptive agents typically demand slightly less surplus than the baseline agents, but occasionally demand much more. Adaptive agents benefit most from conditioning on the features P (immediate surplus available), O (signal of recent transaction prices), and V (private value of next unit traded). Adaptive agents tend to demand more surplus when P or O is high.

4.8 Conclusion

I studied the extent to which adaptive policies yield greater payoffs than non-adaptive, ZI policies at equilibrium in the CDA. I thus addressed whether a calibrated ZI strategy profile is a reasonable model for strategic behavior in the CDA.

The findings suggest traders can benefit from conditioning actions on state in the CDA, even against an equilibrated ZI profile. But the size of the regret of an equilibrated ZI profile, with respect to an adaptive deviating strategy, appears to be small, especially when ZI is equilibrated over many parameterizations.

Further, I provided insight into how a strategy that deviates from ZI can condition on market state to achieve greater surplus. It seems the most useful state features for adaptive CDA traders are the immediate surplus available, the recent history of transaction prices (as encoded in the Omega ratio O), and the private value of the next unit traded. These signals can help the agent to determine how to trade off the likelihood of an order transacting against the amount of surplus requested. In particular, adaptive agents appear to benefit from demanding greater surplus when the current bid or ask is favorable, otherwise demanding less surplus to increase the likelihood of trading at all.

Chapter 5

Iterated Deep Reinforcement Learning in Cybersecurity Games

5.1 Introduction

Cyber-attacks on computer systems are an increasingly serious problem for large organizations. Attackers can infiltrate a system through a security vulnerability and progress through the system, gaining new capabilities before eventually reaching target data or services. The interaction between such an attacker and a defender seeking to protect its system can be represented formally as an *attack-graph game* [47], [67].

In brief, attack-graph games represent a computer system's attack surface as a directed acyclic graph, where each node is a capability that an attacker could have, and each edge is a vulnerability an attacker can use to achieve a new capability. Prior works have used attack-graph games to study how to defend a system against attacker intrusion [22], [23], [57], [61], [62].

Previous studies of the attack-graph game considered here have assumed the attacker will either follow a simple fixed policy [57], such that the defender's choice becomes an optimal control problem; or that attacker and defender will each select strategies from a predetermined, finite list, such that the agents might be expected to play a Nash equilibrium strategy of the resulting game [62]. These studies leave open the possibility that novel strategies may be beneficial deviations. If such deviations exist, then the results of these analyses do not represent how rational agents would behave in the attack-graph game.

This study applies a method called *double-oracle EGTA* (DO-EGTA) for deriving approximate equilibrium strategies in complex games, using deep RL as an (approximate) best-response oracle to iteratively refine an equilibrium over strategies for which payoffs are known. This approach is a variant of the double-oracle method [40], [56] with a deep RL oracle; it is also similar to the approach taken by Wang et al. [92] to learning strategies for green security games. I present results of applying this iterative technique to two instances of the attack-graph game studied by Nguyen et al. [62], using dozens of rounds of deep RL per game to find an equilibrium.

DO-EGTA searches for a strategic equilibrium in games where there is no known method of analytically finding a solution or provably converging to one efficiently. It begins with a (possibly empty) set of heuristic strategies for attacker and defender, which are adopted from prior work [62]. It alternates between (a) solving for a Nash equilibrium over the strategies developed so far, and (b) using deep RL as a best-response oracle to add a beneficial deviation to each agent's strategy set, until no better deviation can be found. DO-EGTA returns the final equilibrium mixed strategy profile as the solution of the game.

When I began experiments with DO-EGTA, I found successive training rounds sometimes appeared unstable, in that a pure strategy would appear with high weight in one round's Nash equilibrium, then drop from subsequent equilibria, only to reappear later. This instability intuitively seems harmful to the training process, as it could cause thrashing, in the sense that agents in successive rounds are not being trained against known, robust strategies that will be present in later equilibria. This and other issues of training instability are known to occur in multi-agent reinforcement learning (MARL) settings, due to the non-stationary training objective for each agent, and to non-transitive win-loss relationships among pure strategies [4], [27].

In an effort to produce more stable iterative training and better final strategies, this work introduces *history-aware double-oracle EGTA* (HADO-EGTA), an extension of DO-EGTA with a modified strategy exploration function. HADO-EGTA works by training each new RL strategy first against the current equilibrium opponent, then fine-tuning against a mixed strategy over previous equilibrium opponents. The resulting deep neural network (DNN) is recorded periodically during training, and the best version is selected based on mean performance against the current and previous opponents, subject to being a beneficial deviation against the current opponent. (Mean performance is measured for each of several interim networks produced during training, via independent sampling of payoffs.) I find that HADO-EGTA tends to produce more consistent strategy improvements per iteration, with stronger final mixed strategies than DO-EGTA.

I find that deep RL can successfully deviate from the best previously proposed strategies for the attack-graph game, which included hand-designed applications of particle filtering and tree search [62]. I employ deep RL instead of classical RL, such as tabular Q-learning, because the state space of the attack-graph game environment is complex, including attributes of many nodes in a graph, and as such does not appear to lend itself naturally to a tabular approach. Iterated deep RL eventually learns defender strategies that appear to strike a reasonable balance between protecting the most valuable goal nodes and avoiding costly actions. In some cases, the attacker or defender retain the most effective of the original hand-designed strategies as some fraction of their mixed strategy, even after many rounds of deep RL strategy addition, although they tend to favor the more recently-developed deep RL strategies. In order to apply deep RL algorithms to attack-graph games, this work employs a novel trick for handling combinatorial action spaces over the graph, called *greedy action set building*, which I find highly effective.

One drawback of iterated deep RL methods like DO-EGTA is they can take many iterations to converge, and the number of iterations required both has high variance and is and difficult to predict a priori. I report the ranges of round counts used in repeat trials of the same game environment and algorithm, which can range from, for example, 32–82 for the same setting with different random seeds. A related limitation is that deep RL training is itself noisy, with different random seeds sometimes causing success or failure in training [36]; this could lead to an iterated deep RL procedure terminating, even though the training method would have been able to learn a beneficial deviation with a different random seed. (I discuss options for dealing with this issue in the paper, including duplicate training runs with different seeds.) To provide insight into the anytime performance of DO-EGTA (i.e., when training is stopped before convergence), I present results on the regret of each training round's equilibrium mixed strategy, with respect to the final mixed strategy after training converges. These results indicate that after the first half of the total rounds are complete, the regret of stopping training early empirically tends to be low and decreasing.

There appears to be a dynamic between attacker and defender, where first the attacker learns a highly beneficial deviating strategy, then the defender responds by learning a successful counter to the new attack; this pattern is repeated occasionally as training proceeds over many rounds. Eventually, the attacker becomes unlikely to learn dramatically successful new attacks and simply relies on a mixed strategy over the best of the strategies learned earlier. Then the defender gradually learns better responses to these known attacks.

The key contributions of this work are:

- results from convergence to equilibrium over dozens of rounds of iterated deep RL, in attack-graph games from prior literature;
- a trick for RL in combinatorial action spaces, called greedy action set building;
- a novel method called HADO-EGTA for iterated deep RL that fine-tunes each agent against a mixed strategy over previous opponents, yielding stronger strategies than the basic approach in some settings;
- empirical results on the convergence behavior and anytime performance of iterated deep RL.

The rest of the chapter is structured as follows. First I present related work, including papers on attack-graph games and on iteratively solving complex games. I summarize the attack-graph game, referring to previous work for a complete description. Next I present the method for iterative game analysis, including aspects of EGTA, double oracle, deep Q-networks (DQN), and solving for equilibria in two-player strategic games. I also present a novel approach to strategy exploration for iterated deep RL, called HADO-EGTA. I show results from iteratively searching for and finding equilibria in two variants of the attack-graph game. These results establish that the proposed methods empirically converge reliably to more strategically stable strategies.

5.2 Game-theoretic preliminaries

The strategic interaction between attacker and defender is modeled as a *two-player game* $G = (S, U)$, where the attacker and defender simultaneously select one *pure strategy* each from finite strategy sets S_a or S_d. Initially, the attacker strategy set S_a is the set of available attacker heuristics taken from prior work [62], and similarly for the defender in S_d. When the two players simultaneously select a strategy *profile* $(s_a, s_d) \in S_a \times S_d$, rewards for attacker and defender are stochastically sampled according to the random properties of the game, which may be represented by the stochastic function U, as $U_d : S_a \times S_d \to \mathbb{R}$ for defender reward, and similarly with U_a for attacker reward. These payoff samples are generated by a game simulator.

More generally, attacker and defender can play *mixed strategies* σ_a and σ_d that are probability mass functions over the players' finite strategy sets. Players are playing a *Nash equilibrium* of the game, if they play a mixed strategy profile $\sigma = (\sigma_a, \sigma_d)$ such that for any deviating strategy s_a or s_d, the player's expected payoff when using that pure strategy while the opponent plays as in σ is no better than its expected payoff under σ. That is, σ is a Nash equilibrium if and only if $U_a(s_a, \sigma_d) \leq U_a(\sigma_a, \sigma_d)$, for all $s_a \in S_a$ (and similarly for the defender).

The *regret* of a mixed-strategy profile σ, or $\rho(\sigma)$, is the maximum any agent i can gain in expectation by unilaterally deviating from its mixed strategy in σ to an alternative strategy s_i. I define the regret of a mixed strategy σ_i with respect to a Nash-equilibrium profile σ^*, or $\rho(\sigma_i, \sigma^*)$, as the expected payoff that agent i would lose by unilaterally deviating from σ^* to σ_i:

$$\rho(\sigma_i, \sigma^*) := U_i(\sigma^*) - U_i(\sigma_i, \sigma^*_{-i}).$$

5.3 Related work

5.3.1 Attack-graph games

Attack graphs represent a vulnerable computer system as a directed acyclic graph, where nodes represent capabilities or privileges an attacker could have, and edges represent exploits or other actions that could allow the attacker to escalate its capabilities. Attack graphs were introduced by Philips and Swiler [67] and have been widely studied ever since. Attack graphs are used in the field to model vulnerabilities of computer systems, and automated attack-graph generation software has been produced, both academic [38], [66], [73] and commercial [2], [39].

An attack graph has *goal nodes* that represent the ultimate goals of an attacker, such as root access to a server. Other nodes in the graph might represent capabilities such as the ability to log in to a particular server, or read or write access to a file. Examples of vulnerabilities that might be represented by edges in the graph include bugs in FTP or SSH servers, buffer overflow vulnerabilities, unsafe security policies on servers or firewalls, SQL injection, password guessing, or remote code execution [18], [47], [68].

Attack-graph games are strategic games between an attacker and defender in an attack graph, where the attacker gains utility by activating (i.e., taking control of) nodes, for which the defender is penalized. The game proceeds over a series of discrete time steps, as the attacker attempts to progress into the graph from root nodes, and the defender attempts to stop the attacker.

Recent works [22], [23], [61] have searched for optimal attacker and defender strategies in attack-graph games. Miehling et al. introduce the form of attack graph studied in this work [57]. They present methods to solve for the best defense policy, formulated as an optimal control problem, assuming that the attacker follows a known, fixed strategy.

Nguyen et al. [62] extend the problem to a two-player game, where both attacker and defender can select among various strategies for attacking or defending graph nodes. The authors propose several heuristic strategies, including some sophisticated ones based on particle filtering, for the attacker and defender. They employ the methods of EGTA [95] to estimate the payoff of all pairs of strategies in some set and solve for equilibria among them.

I extend this work by allowing attacker and defender to use strategies beyond a hand-designed set, including any strategy that can be found by a deep reinforcement learner. I apply deep RL to expand the agents' strategy sets and arrive at more stable equilibrium behaviors.

5.3.2 Deep reinforcement learning

Deep RL has famously succeeded in learning superhuman strategies for extensive-form strategy games like Go, chess, and shogi [74], [75], which may be seen as a development building upon earlier successes in training neural networks with RL to play extensive-form games like backgammon [82]. A DNN and associated learning algorithm for deep RL known as a deep Q-network (DQN) [60] has been shown to learn strong policies for playing Atari games using only pixel input, based on the Q-learning method [93] of Watkins and Dayan. Deep RL has also been applied with success to problems (like the security game shown here) that involve some or all of imperfect information, multiple players with asymmetric roles, and non-zero-sum payoffs – from cooperative-competitive games [79], to video games like Super Smash Bros. [26], Dota 2 [65], and Doom [48].

In the security domain, Kamra et al. [44] trained a DNN to learn policies for a zero-sum game between a ranger and poachers, where agents could act in continuous space. Wang et al. [92] used an iterated deep RL procedure similar to mine to develop new strategies in a different game of rangers and poachers. And in the moving-target defense community, Venkatesan et al. [86] demonstrated a system using reinforcement learning (without DNNs) to determine the placement of honeypots and attack detectors, defending a simulated computer system against persistent attackers.

Since the original work on DQN, various new deep RL algorithms have also produced solid results, such as asynchronous advantage actor critic [59], trust region policy optimization [70], and Rainbow [37]. After comparing DQN and an actor-critic method, I found that DQN is consistently successful (and competitive with alternatives) at learning good policies in the attack-graph setting, so I proceeded with this method alone.

The problem of conducting deep RL training over combinatorial action spaces has been addressed in some prior work, such as by He et al. [35], who trained a deep RL agent to recommend a subset of news articles to a user from a larger candidate set. He at al. experimented with multiple methods for handling this combinatorial problem where the set size K to select was fixed, such as: (a) representing each news item as an embedding vector, concatenating K vectors, and predicting the value of the set; and (b) estimating the value of each news item independently (based on its embedding vector), and selecting the K highest-value items.

5.3.3 Double-oracle and empirical game-theoretic methods

Several prior works have explored double-oracle methods for solving complex games. Briefly, a double-oracle method solves a two-player game iteratively, by first solving the game where only a finite set of strategies may be used, and then using an *oracle* for each player to derive the best-response strategy against the current equilibrium opponent. Eventually, no better response will exist, and the method will have converged to an equilibrium.

The double-oracle method was introduced by McMahan et al. [56], who applied it to a robot path-planning game, using specialized algorithms as oracles. In the security field, Jain et al. [40] developed a well-known application of double oracle, using a mixed integer-linear program (MILP) to compute best responses for attacker and defender agents in a checkpoint-placement game.

In the absence of an explicit game model, one can employ a simulator and learn best responses through RL. The general method of solving game models from simulation has been termed *empirical game-theoretic analysis* (EGTA) [94]. Schvartzman and Wellman [71] combined RL with EGTA, using tabular Q-learning with tile coding to learn better-response policies in a dynamic trading game; a similar approach using Q-learning without DNNs was taken in another recent work [97].

Lanctot et al. [49] presented a generalized version of double-oracle, called *policy-space* *response oracles* (PSRO), designed for RL approaches to empirical games. They demonstrated the effectiveness of this approach with deep RL methods on gridworld games. Recently, Wang et al. [92] presented work on applying DQN as the oracle, for a zero-sum security game. They found that employing double oracle with DQN may take many rounds and a lengthy computation time to converge, which matches my experience, based on many more iterations of training.

5.3.4 Multi-agent reinforcement learning

The proposed training procedure for the agents' strategies is a form of MARL, because I concurrently train attacker and defender agents, each with its own action space and utility function. The difficulty is that each agent's training objective is non-stationary, which can cause unstable training progress, or (theoretically) even an endless cycle where training fails to converge to a fixed point [4], [27].

The DO-EGTA approach builds on the framework of double oracle, but I propose an extension with a novel strategy exploration rule, called HADO-EGTA, that seeks to improve training stability by yielding intermediate results that perform more strongly against an opponent's former strategies, than what would result from the basic double-oracle approach of DO-EGTA. As suggested by Lanctot et al. [49], double-oracle methods can produce DNNs that are overfit to the current opponent mixed strategy and do not sufficiently exploit weaker opponents, unless the approach is modified to encourage agents to learn strategies that generalize well.

The method of fine-tuning agents to perform well against former opponents, while deviating beneficially against the current opponent, is related to some techniques proposed in recent work by others. Lanctot et al. [49] suggested new meta-strategy solvers such as *projected replicator dynamics*, as extensions to iterated deep RL with double oracle. Their projected replicator dynamics method forces each agent to include every pure strategy trained so far in its mixed strategy with at least some minimum probability, ensuring that during training any agent will likely observe every pure strategy of its opponent. Moreover, projected replicator dynamics forces the mixed strategy of each agent to change by only a small step size from one round to the next, which may prevent the problem of thrashing, in which an agent trained in one round fails to learn what the previous agent did, due to drastic changes in the training opponent's mixed strategy. Our approach of first training against the current equilibrium opponent, then fine-tuning against a decaying weighted average of previous equilibrium opponents, is arguably an improvement on projected replicator dynamics, as follows. This method, like that of Lanctot et al., trains each agent against a wide variety of opponent pure strategies, while encouraging smooth changes in training opponent from round to round, but also: (a) allows the current mixed strategy to adapt arbitrarily quickly when strong new pure strategies are introduced, by not using a limited step size; and (b) enables an agent to completely exclude weak pure strategies from its mixed strategy. Opponent sampling [6], where training is conducted against a randomly sampled old opponent instead of the current opponent DNN, has been reported to speed up training of competitive agents in simulated physics games (perhaps by providing a form of curriculum learning where neither training opponent can improve too quickly), as well as improving the robustness of the trained agents to unseen opponents. The authors experiment with refining opponent sampling, by randomly sampling opponent strategies from only the most recent fraction of training episodes, such as the most recent half, which they find sometimes improves results. Our proposed method differs from opponent sampling in that instead of merely training against a uniform distribution over previously trained opponent pure strategies, we first train against the current equilibrium opponent in an attempt to ensure a beneficial deviation, then fine-tune against a decaying weighted mean of previous rounds' equilibrium opponents. Our approach thus focuses training effort on learning to perform well against those opponent strategies that appear most strategically stable, while ignoring irrelevant opponent strategies. Wang et al. [92] propose a similar approach to encouraging generalization, in which an agent is trained against a weighted average of the current equilibrium opponent mixed strategy and the uniform distribution over opponent pure strategies. Foerster et al. [27] propose Learning with Opponent Learning Awareness (LOLA), a deep RL learning rule intended to stabilize MARL training, that models the opponent's policy, predicts the opponent's policy update, and updates one's own policy in anticipation of the opponent's update. Grnarova et al. [33] introduce a method for training the two competing policies in a generative adversarial network (GAN), which is designed to improve training stability, using the online learning algorithm *follow the regularized leader* (FTRL), and returning a mixed strategy over the resulting networks.

In earlier work on stabilizing EGTA, Jordan et al. [42] found that augmenting the set of available pure strategies with arbitrary beneficial deviations from the current Nash equilibrium could lead to faster convergence to fixed points than only adding best responses. This suggests that some objective other than maximizing payoff against the current opponent's mixed strategy could be better for training DNN policies in games.

5.3.5 Evaluation of game-playing agents or training methods

Evaluating the quality of learned policies in games can be difficult, due to the possibility of non-transitive relationships among several agents [5], [80], also known as rock-paper-scissors relationships. There may not exist a scalar score per agent, such that agents with higher scores are expected to outperform agents with lower ones. Similarly, there is no single metric that fully captures the advantages and disadvantages between training methods for agents in games.

The comparisons made here between training procedures for deep RL policies are focused on the performance of each at Nash equilibrium, as in the NE-regret ranking method previously used by Jordan et al. [41]. This approach was also recently advocated for by Balduzzi et al. [5], who argue that it is reasonable to evaluate an agent against the Nash equilibrium opponent mixed strategy rather than against a uniform mixed strategy over all opponents developed so far.

I also report results on round-robin tournaments among the final mixed strategies yielded by the different training methods, as suggested as a method of evaluating trained GANs by Olsson et al. [64]. The performance of the final mixed strategy from one training run against mixed strategies from other training runs also offers a measure of how well a training method generalizes to unseen opponents. It has been noted previously that deep RL agents tend to overfit to the opponents seen in training, and perform worse during testing even against agents generated by a different run of the same procedure [49].

All the results are reported based on multiple independent training runs for each method and game, as Henderson et al. [36] suggest is necessary to overcome the randomness inherent in deep RL training.

5.4 Game Description

Here I summarize the attack-graph game considered in this work; see the original paper on the game [62] for a complete description.

5.4.1 Attack-graph model

The game takes place on a *Bayesian attack graph*, of the type defined by Miehling et al. [57]. This is a directed acyclic graph $\mathcal{G} = (V, E)$, where vertices $v \in V$ represent capabilities the attacker could have, and edges $e \in E$ are exploits that can be used to gain capabilities. Note that when the attacker has a capability, the associated node is called *active*; otherwise, nodes are called *inactive*.

An *attack-graph game* is defined by a Bayesian attack graph endowed with additional specifications. The game evolves over a finite number of time steps $\mathcal{T}$ fixed in advance. At each time step τ, the state of the graph is simply which nodes are active (i.e., attacker-controlled), indicated by $s_\tau(v) \in \{0, 1\}$. The defender receives only a noisy observation $O_\tau(v) \in \{0, 1\}$ of whether each node is active, based on publicly known probabilities $P_v(o = 1 \mid s = 1)$ and $P_v(o = 1 \mid s = 0)$. Positive observations are known as *alerts*. (These observations are generated independently.)

Attacker and defender act simultaneously at each time step τ. The defender's action space is the power set of nodes V, meaning the defender can choose to defend any subset of the nodes.

The attacker action space is defined as follows. Some nodes in the graph, including all root nodes, are called $\wedge$-nodes (*and* nodes), because they can be attacked only if all their parent nodes are active. Other nodes are called $\vee$-nodes (*or* nodes), because they can be attacked when at least one parent is active, via any edge or edges from an active parent. The attacker can select any or all edges from active parents to use in attacking an $\vee$-node.

At each time step, the game state is updated as follows. The defender's actions override attacker actions, such that any node v that is defended will become inactive. Otherwise, active nodes remain active; an $\wedge$-node v that is attacked becomes active with probability $P(v)$, and any $\vee$-node that is attacked becomes active with probability based on the combination of the (independent) edge attack success probability $P(e)$ of each incoming edge that is attacked.

Each goal node v has an attacker reward $r_a(v)$ and defender penalty $r_d(v)$ that will be assessed in each time step the node is active. Each item an agent can act on also has a cost: $c_d(v)$ for all nodes defended, $c_a(v)$ for all $\wedge$-nodes attacked, and $c_a(e)$ for all edges to $\vee$-nodes attacked. There is a discount factor $\eta \in (0, 1]$ for future rewards.

The defender's loss at a time step is the cost of its action (total cost of nodes defended), plus the penalty for active goal nodes after that step. The defender's total payoff is the negated, exponentially weighted sum of costs over time, with discount factor η. The attacker's total payoff is likewise the exponentially weighted sum (with discount factor η) of payoffs over time, in this case the reward for active goal nodes, minus cost of attacks pursued.

5.4.2 Heuristic strategies

Prior work by Nguyen et al. [62] proposed sophisticated heuristic strategies for the attack-graph game, some of which use a particle filtering approach to search for good actions to take.

Attacker heuristic strategies

Several heuristic strategies were proposed for the attacker. In the *uniform* strategy, the attacker selects a uniform random set of nodes and edges to attack, of a certain size.

The *value propagation attacker* strategy (aVP) is more complex. This attacker estimates the value of attacking each node or edge, by propagating rewards backward from goal nodes to their ancestors in the graph. The computation also accounts for the costs and success probabilities of attacks. This strategy has a fixed number of items to be attacked, and it randomly selects the nodes and edges to attack based on a softmax over the computed value of each item that can be attacked.

The *random walk attacker* strategy (aRW) assigns a value to each node or edge differently. This method samples a random process that progresses forward from the currently active nodes toward goal nodes. A value is assigned to each goal node based on the probability with which the random walks activate it, and the costs and rewards incurred. The strategy uses a heuristic to select the next items to attack, based on which nodes or edges frequently occurred in paths toward the highest-value goal nodes.

Defender heuristic strategies

Multiple defender heuristics were proposed as well, including several naive baseline strategies. In the *goal-only* defender strategy, the defender defends a fixed number of goal nodes in each time step, sampled from a softmax distribution of the sum of the defender penalty for active nodes and the defender cost to defend each goal node. In the *root-only* defender strategy, the defender defends a fixed number of root nodes, chosen uniformly randomly. The *min-cut* defender strategy defends a fixed number of nodes in the min-cut set of the graph (separating goal nodes from root nodes), selected uniformly randomly. The *uniform* defender strategy defends a fixed number of nodes, chosen uniformly randomly.

More complex strategies for the defender share a Bayesian belief vector over the current game state, where a game state is the set of active nodes. This belief vector is updated using a particle filter, based on a fixed assumption about the attacker strategy being used.

In the *value propagation defender* strategy (dVP), similarly to aVP, the defender assigns a value to each node by propagating the penalty for active nodes and the cost of defense backward from each goal node. The defender defends a fixed fraction of those nodes the attacker is believed to be able to attack next, based on the defender's belief state. A heuristic is used to determine the set of nodes to defend, accounting for the estimated value of each node. Because the defender can simulate attacker actions assuming an attacker strategy of aVP or aRW, this defender strategy type is fully written as either dVP_aVP or dVP_aRW.

In the *random walk defender* strategy (dRW), like in aRW, the defender simulates a series of future attacker actions by sampling a random process. Each node is assigned a value based on the penalty of the goal nodes activated by this process, as well as the defender costs of blocking the attacker paths. The defender greedily adds nodes to a set to defend, until adding another node would not increase the expected value of the defense set. As with dVP, this defender strategy can assume the attacker uses aVP or aRW, leading to a full name of dRW_aVP or dRW_aRW.

5.5 Methods

5.5.1 Empirical game-theoretic analysis

In a *simulation-based game*, no explicit formula is known for the expected payoff between any two strategies, but one can sample payoffs from their distribution by running a game simulator. The attack-graph game is an example of a simulation-based game.

In *empirical game-theoretic analysis* (EGTA) for a two-player game, the procedure begins with a finite set of strategies for each player [95]. Next payoffs are sampled among all pairs of strategies, in order to estimate the expected payoff of all strategy profiles. A Nash equilibrium solver can be used to find the equilibrium solution over the game comprising this strategy set.

The strategy set is then augmented with beneficially deviating strategies, if any can be found. Another round of payoff samples is generated, using the new strategies, and a new equilibrium is found. This process continues until no beneficial deviations are easily obtained, and the process is considered to have converged.

5.5.2 Attack-graph game instances

As in the previous work of Nguyen et al., I study attack-graph games based on two kinds of graph topology: *random graphs* and *separate-layers graphs*. In summary, random graphs are constructed similarly to Erdős-Rényi graphs, where the node and edge count are fixed and edges are added uniformly randomly. Separate-layers graphs are built with a given number of layers, where each layer has a fraction (less than one) of the node and edge count of the previous layer, and edges exist only from nodes of one layer to the next layer.

The games used here are built on two randomly-generated graph instances, one of them separate-layers with 29 nodes (s_{29}), the other random with 30 nodes (r_{30}). In game s_{29}, there are 3 layers, 7 goal nodes with values in $[15, 45]$, and 89 edges. In game r_{30}, there are 6 goal nodes with values in $[10, 20]$, and 100 edges. The topology of the graph in game s_{29} is presented in Figure 5.1, and for game r_{30} in Figure 5.2. In both games, all non-root nodes are $\vee$-nodes, corresponding to the 0%$\wedge$ condition from Nguyen et al. Also in both games, the *low-noise* condition from Nguyen et al. is used, where active nodes have alert probability in $(0.8, 1)$, and inactive nodes have alert probability in $(0, 0.2)$.

Each game proceeds over $\mathcal{T} = 10$ time steps, with payoff discount factor $\eta = 0.9$.

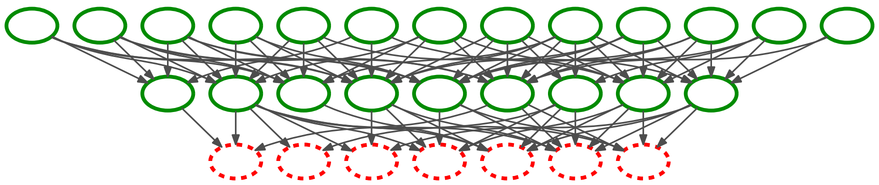

FIGURE 5.1: Topology of the separate-layers graph from game s_{29}. Goal nodes are shown with dashed red outlines, non-goal nodes with solid green.

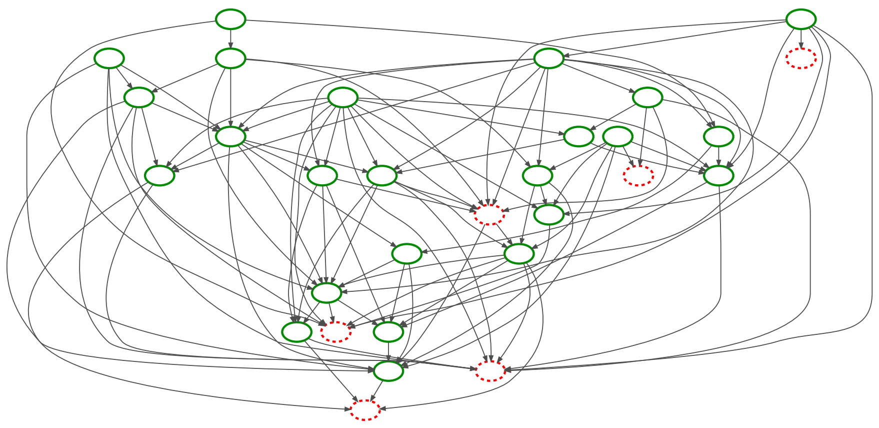

FIGURE 5.2: Topology of the random graph from game r_{30}. Goal nodes are shown with dashed red outlines, non-goal nodes with solid green.

5.5.3 Heuristic strategies

In both games, I use the same set of heuristic strategies at the beginning of the iterative best response process. For the defender, there are 50 heuristic strategies, including multiple parameterizations of the general strategy types defined above. There are 4 goal only, 2 min-cut, 2 root only, 2 uniform, 4 dRW_aRW, 4 dVP_aRW, 16 dRW_aVP, and 16 dVP_aVP strategies. For the attacker, there are 8 heuristic strategies: 2 uniform, 2 aRW, and 4 aVP.

5.5.4 Deep Q-networks

I employ the deep Q-network (DQN) deep reinforcement learning model popularized by Mnih et al. [60], in the Double DQN variant form introduced by van Hasselt et al. [34] The implementation is derived from the version developed by OpenAI as part of the OpenAI Baselines library [21]. I have modified the library implementation, to tune the number of episodes used to determine if the current DNN is better than the previous best, and to implement HADO-EGTA pre-training and fine-tuning.

DQN works by taking as input a vector representing the agent's current observation of the state of the environment, filtering this vector through one or more convolutional or linear layers and rectifier nonlinearities (ReLUs), and finally yielding a regression estimate of the value of taking each action in the current state. Double DQN extends DQN by training two sets of network weights, with the goal of reducing the bias of state-action value estimates that could result from the max operator of the DQN update step [34]. I experimented with various architectures for the DNN, before settling on a multilayer perceptron with two hidden, fully-connected layers of 256 neurons each. Other architectures I tested in pilot experiments included two fully-connected layers of 128 neurons each; one hidden layer of 256 neurons; and a convolutional network with two convolutional layers, each of which used 32 or 64 filters per layer, convolved along one dimension corresponding to the nodes of the attack graph. There were large differences in performance between different network architectures, indicating that architecture selection is a vital part of hyperparameter tuning for DQNs in this setting. The structure of the DQN, in the example of the attacker strategy for game r_{30}, is shown in Figure 5.3.

The DNN is trained by playing many games against a fixed opponent. Episodes consisting of the current observation, the action taken, the reward, and the next observation are added to a replay memory. Batches of episodes are sampled uniformly from the replay memory to update the DNN's weights. Weights in the DNN are updated to reduce the error between the DNN's estimate of the value of taking the given action in the current

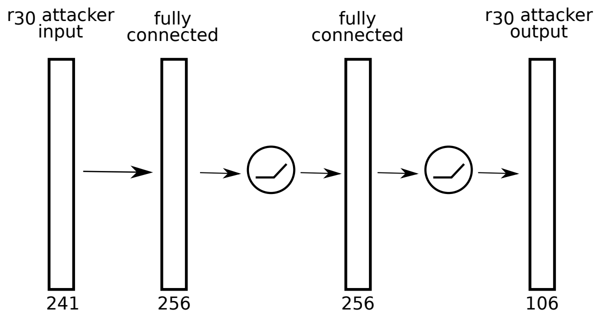

FIGURE 5.3: Architecture of the deep Q-network used in this study. The input and output dimensions shown are for the attacker DNN in game r_{30}.

state, and the observed value. The agent explores during training by selecting actions with an ϵ-greedy policy, taking the action estimated to have the highest value with probability $(1 - \epsilon)$, otherwise taking a uniform random action.

Additional details of the network architecture and training hyperparameters used are presented in Table A.1 in Appendix A.

5.5.5 Game representation for deep RL

One key challenge in this work was the design of an effective representation of the game's observation and action spaces for deep RL. The game has partial observability for the defender, because the defender does not directly observe which nodes are active, so the defender should have the ability to condition its actions on the history of previous observations, not only the current observation. The action space is huge for attacker and defender, because the defender can choose to defend any subset of the nodes, leading to 2^{29} possible actions in game s_{29}, and the attacker can choose to attack any subset of the $\wedge$-nodes and edges to $\vee$-nodes for which the parents are active.

A key technique to make the exponential action space tractable is to let each deep RL agent add items to be attacked or defended, one at a time, to an *attack set* or *defense set*. Taking the defender DNN for example, initially in each time step the defense set is empty, and the defender DNN can either add one node to the set or *pass*. Eventually, when the defender DNN passes, the game proceeds with the defender acting on whichever nodes were in the defense set. Thus, a deep RL agent's DNN may be called multiple times during one time step, to determine the set of items to be acted on, depending on the DNN's outputs. (Note that the heuristic attacker and defender agents, unlike the deep RL agents, select their sets to act on all at once, instead of one item at a time.) Our method differs from those proposed by He et al., as it allows for interactions between the values of items to include in the subset (unlike the approach that learns an independent value per item), while also scaling up better to large sets than the approach that takes a concatenation of the selected subset as input.

To encourage deep RL agents' DNNs to add appropriate items to their sets, I impose rules that cause undesirable actions to be treated as a pass. The defender DNN's action is counted as a pass, whenever it selects a node that is already present in the defense set. The attacker DNN's action is counted as a pass, whenever it selects a node or edge already in the attack set, and whenever it selects a node or edge whose preconditions are not satisfied. And to encourage agents' DNNs to add items to their sets greedily, beginning with the most valuable, I randomly force the DNN to pass with probability 0.1 in each update where at least one item is already in the action set. (The parameter value of 0.1 was selected based on intuition, without experiments testing alternative values; it might be possible to achieve better performance by tuning this value better, or by decaying it toward 0 during training.)

The action selection procedure of the deep RL defender is presented more formally as Algorithm 3. Here the current defender DNN input vector is *defObs*, the defender DNN is ϕ_{def}, $rand()$ samples independent draws from $U(0, 1)$, and the returned *defenseSet* is the set of nodes to defend. The defender DNN's choice n can represent a node to add to the defense set, or *pass*.

Algorithm 3 Deep RL defender's greedy action set building

```
Require: $defObs$
  $defenseSet \leftarrow \emptyset$
  do
      $n \leftarrow \phi_{def}(defObs, defenseSet)$
      $isDup \leftarrow n \in defenseSet$
      if $\neg isDup \wedge n \neq pass$ then
          $defenseSet \leftarrow defenseSet + \{n\}$
  while $\neg isDup \wedge n \neq pass \wedge rand() > 0.1$
  return $defenseSet$
```

Algorithm 4 shows the procedure for generating the deep RL attacker's action. The attacker DNN ϕ_{att} selects one choice at a time x based on the current attacker DNN input vector *attObs*. This selection can represent *pass*, or an $\wedge$-node or edge to $\vee$-node, to add to the *attackSet*. The choice is legal only if it is to *pass*, or if any parent nodes sufficient for the attack, called $pre(x)$, are in the active node set A (i.e., the set of nodes controlled by attacker).

Algorithm 4 Deep RL attacker's greedy action set building

```
Require: $attObs$
  $attackSet \leftarrow \emptyset$
  do
      $x \leftarrow \phi_{att}(attObs, attackSet)$
      $isDup \leftarrow x \in attackSet$
      $isLegal \leftarrow x = pass \vee pre(x) \subseteq A$
      if $\neg isDup \wedge isLegal \wedge x \neq pass$ then
          $attackSet \leftarrow attackSet + \{x\}$
  while $\neg isDup \wedge isLegal \wedge x \neq pass \wedge rand() > 0.1$
  return $attackSet$
```

The input vector for the attacker DNN includes only information from the current observation of the attacker. The attacker can see which nodes are active, so it may have less need than the defender to condition its actions on additional game history. For each node, an observation bit indicates whether that node is active or not. For each $\wedge$-node and edge to $\vee$-node, one bit indicates whether the item is reasonable to attack, in the sense that its preconditions are active but the target is not. Another bit indicates whether that item is currently in the attack set or not. Finally, one vector element indicates how many time steps are left in the game. In total, the attacker DNN input vector has 234 elements in game s_{29} and 241 in game r_{30}. The attacker DNN's input vector is summarized in Table 5.1, where N is the node count, $N_{\wedge}$ is the $\wedge$-node count, and $E_{\vee}$ is the edge to $\vee$-node count.

(Note that heuristic attacker and defender agents in the attack-graph game are allowed unlimited memory of the game's history, but the input vectors are limited to a fixed history depth for the DNN attacker and defender. I fix the history length so it can be used as input to a DQN; if a recurrent network were used instead, the full history could be used by the learning agents.)

The defender DNN's input vector includes information from the defender's most recent 3 observations, filling in with zeros if fewer than 3 exist. (I chose to use a fixed input depth of 3 observations, based on pilot experiments showing that this was sufficient to consistently learn beneficially deviating strategies. It is possible that a different observation depth might yield similar or better performance, or that still better outcomes could be produced by a recurrent network that represents the full history of the game.) For each node and each time step, a bit indicates whether an attacker alert was observed, and with

Attacker		Defender	
Feature	Entry count	Feature	Entry count
isActive	N	*hadAlert*	hN
canAttack	$N_\wedge + E_\vee$	*wasDefended*	hN
inAttackSet	$N_\wedge + E_\vee$	*inDefenseSet*	N
timeStepsLeft	1	*timeStepsLeft*	N

TABLE 5.1: Input vectors for attacker and defender DNN agents, with the number of entries in each.

another bit whether the node was defended. Another bit shows whether each node is currently in the defense set. Finally, a vector element for each node indicates how many time steps are left in the simulation (repeating by the number of nodes for symmetry, in case convolutional layers are used, convolving over the N nodes). The defender DNN's input vector has length 232 in game s_{29} and 240 in game r_{30}. Table 5.1 summarizes the defender DNN's input vector, where h is the history length, which is set to 3.

The action space of the attacker DNN has one action for each $\wedge$-node and each edge to an $\vee$-node, plus one action to pass. This sums to 103 actions in game s_{29} (which has 13 $\wedge$-nodes and 89 edges to $\vee$-nodes) and 106 actions in game r_{30} (which has 5 $\wedge$-nodes and 100 edges to $\vee$-nodes). The defender action space has one action per node to defend and one to pass. This leads to 30 actions in game s_{29} and 31 actions in game r_{30}.

The implementation of the attack-graph game logic is based directly on the code from Nguyen et al. [62] The game code is in Java, and the deep learning code uses Python, so the Py4J library is used for accessing Java Virtual Machine objects from Python, bridging between the learning library and game logic. I have implemented an OpenAI Gym environment [9] as a wrapper around each version of the attack-graph game, which allows the game to be used easily for training and testing by existing deep RL code.

5.5.6 DO-EGTA

In the double-oracle approach to solving two-player games, the procedure alternates between solving for an equilibrium over the strategies for which payoffs are known, and attempting to find beneficially deviating strategies from the current equilibrium. This method theoretically converges to an equilibrium over all possible strategies.

Each round of DO-EGTA begins with computing a Nash equilibrium over the currently strategy set for attacker and defender. This includes all heuristic strategies, along with any beneficially deviating DQN strategies that have been learned. I use the Gambit library [55] of game solvers to search for Nash equilibria. I find that for two-player, general-sum games like ours, the *linear complementarity method* known as *gambit-lcp* runs faster than alternatives; this method is based on the Lemke-Howson algorithm [51]. When the algorithm produces more than one Nash equilibrium, the method simply proceeds with the first equilibrium returned.

I use DQN to search for a beneficial deviation for the attacker and defender. The iterated best response process is considered to have converged when there is a round of training in which both the attacker and defender fail to generate a beneficial deviation in the number of training steps allowed.

In DO-EGTA, I train the DNNs for 700,000 time steps (but 1,000,000 for the defender in game r_{30}), with a learning rate of 5×10^{-5} and an artificial discount factor on future rewards of 0.99. During training, I anneal the exploration rate ϵ linearly from 1 to 0.03 over the first half of DO-EGTA training steps, holding it constant at 0.03 afterward.

Pilot experiments showed that the hyper-parameters used here—including number of training steps, learning rate, and exploration schedule—consistently produce beneficial deviations. Moreover, I have successfully generated 20 or more rounds of beneficial deviations in each game. Based on that experience, this procedure does not appear prone toward false-positive convergence. Note that the final payoff achieved by deep RL training is sensitive to the choice of random seeds [36].

One could easily reduce the likelihood of spurious convergence of this double-oracle method, by requiring multiple trials of DQN to fail (perhaps 3–5) for attacker and defender in the same round, before considering the method converged. In other words, instead of stopping after a training round in which both attacker and defender training produce strategies that are not beneficial deviations, one could repeat both attacker and defender training in such cases, up to some maximum count such as 3 attempts each, stopping only if these repeated attempts all fail to yield a beneficial deviation.

Definition of DO-EGTA

Here I formally define the double oracle EGTA (DO-EGTA) procedure. Let the set of initial heuristic strategies for attacker and defender, which could be empty, be $S^H = (S_a^H, S_d^H)$. In each round $t = (0, 1, \ldots)$ of the iterative procedure, the current strategy sets are $S_{a,t}$ and $S_{d,t}$. Let $\sigma_t = (\sigma_{a,t}, \sigma_{d,t})$ be any mixed-Nash equilibrium over these strategy sets under the game's utility function $U = (U_a, U_d)$. Let $g()$ be any deep RL algorithm, such as DQN, that can optimize over an objective function like U_a against an opponent mixed strategy like $\sigma_{d,t}$, returning a new pure strategy such as $\delta_{a,t}$. Let $\nu()$ be a Nash equilibrium solver, which returns any mixed-Nash equilibrium, given a utility function and finite strategy sets. Note that the utility function $U()$ may not be analytically defined; DO-EGTA requires only a simulation-based means of sampling payoffs for any strategy profile. Then the DO-EGTA procedure is shown in algorithm 5.

Algorithm 5 DO-EGTA iterated deep RL method

```
Require: $U, S^H, g(), \nu()$
  $t \leftarrow 0; \quad S_{a,t} \leftarrow S_a^H; \quad S_{d,t} \leftarrow S_d^H$
  $\sigma_t \leftarrow \nu(U, S_{a,t}, S_{d,t})$
  do
      $t \leftarrow t + 1$
      $\delta_{a,t} \leftarrow g(U_a, \sigma_{d,t-1}); \quad \delta_{d,t} \leftarrow g(U_d, \sigma_{a,t-1})$
      if $U_a(\delta_{a,t}, \sigma_{d,t-1}) > U_a(\sigma_{a,t-1}, \sigma_{d,t-1})$ then
          $S_{a,t} \leftarrow S_{a,t-1} + \{\delta_{a,t}\}$
      if $U_d(\delta_{d,t}, \sigma_{a,t-1}) > U_a(\sigma_{d,t-1}, \sigma_{a,t-1})$ then
          $S_{d,t} \leftarrow S_{d,t-1} + \{\delta_{d,t}\}$
      $\sigma_t \leftarrow \nu(U, S_{a,t}, S_{d,t})$
  while $S_{a,t} \neq S_{a,t-1} \vee S_{d,t} \neq S_{d,t-1}$
  return $\sigma_t, S_{a,t}, S_{d,t}$
```

Note that algorithm 5 is a variation of PSROs [49], as well as the double-oracle method. Distinguishing features of DO-EGTA are: (a) it begins with a set of heuristic strategies S^H, instead of random strategies; (b) if a proposed new strategy $\delta_{a,t}$ or $\delta_{d,t}$ is not a beneficial deviation, it is not added to the game's strategy set; and (c) payoffs from the players' utility functions U may be sampled from a simulator, rather than computed analytically.

5.5.7 HADO-EGTA

History-aware double-oracle EGTA (HADO-EGTA) is a DO-EGTA variant with a new strategy exploration procedure. Recall that DO-EGTA strategy exploration simply seeks a best response for each agent i to the current equilibrium opponent mixed strategy $\sigma_{-i,t-1}$:

$$\delta_{i,t} := \arg\max_{s_i} U_i(s_i, \sigma_{-i,t-1}).$$

Instead, HADO-EGTA balances deviating beneficially against the current equilibrium opponent, with exploiting previous opponents. HADO-EGTA pre-trains each DNN strategy against the current equilibrium opponent, then fine-tunes the DNN against a mixed strategy over current and previous opponents. During fine-tuning, interim DNN weights are periodically recorded, and the weights are selected that yield the highest payoff against a mixed strategy over current and previous opponents, subject to being a beneficial deviation against the current opponent. More specifically, I record the current DNN weights at 4 time points: at the end of pre-training, and after $\frac{1}{3}$, $\frac{2}{3}$, and all fine-tuning training steps are complete. The expected payoff of each recorded network is estimated based on an independent sample of game outcomes against either the current equilibrium opponent or a mixed strategy over current and previous opponents.

HADO-EGTA begins by training against the current equilibrium opponent, rather than against the mixed strategy over past equilibrium opponents, because it is of primary importance that the resulting strategy be a beneficial deviation from the current equilibrium; if the result is not a beneficial deviation from the current equilibrium, the result is not useful, even if it performs well against non-equilibrium opponent strategies.

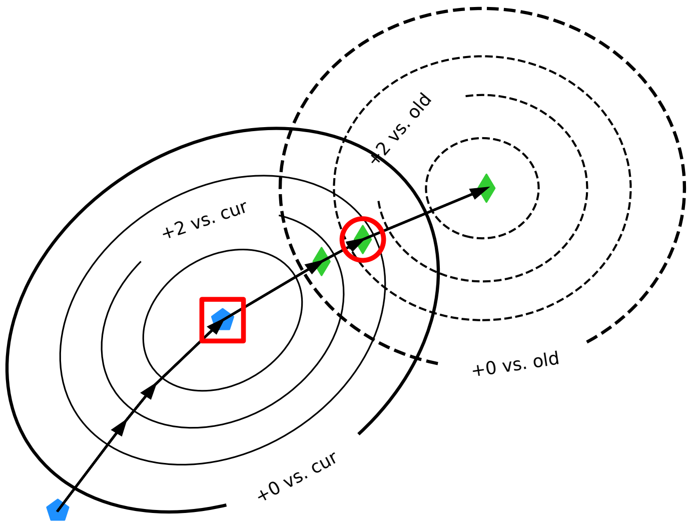

FIGURE 5.4: Conceptual diagram of HADO-EGTA. Solid contours show payoff w.r.t. current opponent, dashed contours w.r.t. old opponents. Blue pentagons show HADO-EGTA pre-training (or DO-EGTA training), producing result in red square. Green diamonds show interim results of HADO-EGTA fine-tuning, with selected result in red circle.

Figure 5.4 shows a diagram of the intuition behind HADO-EGTA strategy exploration. In HADO-EGTA pre-training (or DO-EGTA), a network is trained to maximize payoff against the current equilibrium opponent, shown as training the blue pentagon to find the optimum of the solid contour lines. Observe that it may be possible to increase payoffs against previous opponents, indicated by dashed contour lines, while maintaining a beneficial deviation against the current opponent (i.e., staying within the bold, solid contour). HADO-EGTA fine-tunes the strategy to maximize payoff against previous opponents, and returns a strategy that deviates beneficially against the current opponent while also exploiting old opponents, represented in the figure by the highlighted green diamond.

More formally, the HADO-EGTA strategy exploration procedure, written as $h()$, takes as input: an agent's utility function such as U_a; a count of DNNs to record $\kappa \in \{2, \ldots\}$; a decay factor $\gamma \in (0, 1]$ for weighting old opponents; a weighting $\alpha \in [0, 1]$ between payoffs against the current and old opponents; and the history of opponent equilibrium strategies $(\sigma_{d,0}, \ldots, \sigma_{d,t-1})$. It also requires a deep RL method for pre-training like the $g()$ of algorithm 5, and a deep RL fine-tuning method, called $g'()$, that proceeds for a limited number of steps from a pre-trained DNN. This complete HADO-EGTA procedure $h()$ can take the place of $g()$ as the strategy-exploration routine in the DO-EGTA algorithm.

I present HADO-EGTA from the attacker's perspective as Algorithm 6. For each agent i, HADO-EGTA strategy exploration seeks a beneficial deviation that also exploits old opponents:

$$\delta_{i,t} := \arg\max_{s_i} \alpha U_i(s_i, \sigma_{-i,t-1}) + (1-\alpha) U_i(s_i, \overline{\sigma}_{-i}),$$
$$\text{s.t. } U_i(s_i, \sigma_{-i,t-1}) > U_i(\sigma_{t-1}).$$

The defender's strategy exploration procedure is analogous to Algorithm 6. Note that if no interim DNN is a beneficial deviation, the procedure returns null, and no new strategy will be added to the agent's strategy set in the current iteration.

Note that depending on the parameter settings, HADO-EGTA allows considerable choice of which opponents a strategy is fine-tuned against. HADO-EGTA can fine-tune against a uniform mixed strategy over all previous opponent equilibrium strategies, like in fictitious play, if $\gamma = 1$. Or HADO-EGTA can fine-tune against only the current equilibrium opponent, like in the double-oracle method, if $\gamma \approx 0$.

Moreover, adjusting α allows a user to make HADO-EGTA favor more beneficial deviations against the current opponent, or more exploitation of previous opponents. HADO-EGTA can select only for maximal payoff against the current opponent if $\alpha = 1$, while still reaping the benefits of multiple interim strategy options $\delta^k_{a,t}$. Or HADO-EGTA can select

Algorithm 6 HADO-EGTA attacker strategy exploration rule

```
Require: $U_a, \kappa, \gamma, \alpha, g(), g'(), (\sigma_{d,0}, \ldots, \sigma_{d,t-1})$
  $\overline{\sigma}_d \leftarrow \left(\sum_{\psi=0}^{t-1} \gamma^{t-1-\psi}\right)^{-1} \sum_{\psi=0}^{t-1} \gamma^{t-1-\psi} \sigma_{d,\psi}$
  $k \leftarrow 0$
  $\delta_{a,t}^{k} \leftarrow g(U_a, \sigma_{d,t-1})$
  while $k < \kappa$ do
      $k \leftarrow k + 1$
      $\delta_{a,t}^{k} \leftarrow g'(U_a, \overline{\sigma}_d, \delta_{a,t}^{k-1})$
  $\delta_{a,t}^{*} \leftarrow \arg\max_{\delta_{a,t}^{k'}} \alpha U_a(\delta_{a,t}^{k'}, \sigma_{d,t-1}) + (1-\alpha) U_a(\delta_{a,t}^{k'}, \overline{\sigma}_d)$,
  s.t. $U_a(\delta_{a,t}^{k'}, \sigma_{d,t-1}) > U_a(\sigma_{a,t-1}, \sigma_{d,t-1})$ [or $\emptyset$ if none].
  return $\delta_{a,t}^{*}$
```

for the strategy that best exploits previous opponents, subject to being a beneficial deviation against the current opponent, if $\alpha = 0$.

In the experiments with HADO-EGTA, I used $\kappa = 4$ strategies to select among, including the pre-trained strategy. I discounted old strategies with factor $\gamma = 0.7$. I set $\alpha = \frac{2}{7}$. (This makes the total weight of the current opponent equal $\frac{1}{2}$ in the limit as round count approaches infinity, or $\frac{2}{7} \times 1 + \frac{5}{7} \times \frac{3}{10}$.)

In HADO-EGTA, I pre-train the DNNs for 700,000 steps against the current equilibrium opponent (but 1,000,000 for the defender in game r_{30}), before fine-tuning for 400,000 against a mixed strategy over previous equilibrium opponents, recording $\kappa = 4$ interim strategies to select from (including the pre-trained strategy). For the exploration rate in HADO-EGTA, I linearly anneal ϵ from 1.0 to 0.03 over the first half of pre-training steps, then hold it at 0.03 for the remainder of pre-training. I then linearly anneal ϵ from 0.3 to 0.03 over the first half of fine-tuning steps, holding it at 0.03 for the remainder of fine-tuning.

5.6 Results

5.6.1 Effectiveness of deep RL oracles

I find that DQN consistently learns beneficial deviations from the current strategies' Nash equilibrium over many rounds of DO-EGTA, only at times failing to generate beneficial deviations after several rounds of training have already been completed. In all, I have conducted 2 runs of HADO-EGTA and 3 of DO-EGTA in each environment (s_{29} or r_{30}), of which 9 runs have converged, after between 21 and 70 rounds. The remaining run has an elapsed round count of 82.

Figure 5.5 shows the expected payoffs for attacker and defender equilibrium strategies and learned deviations, in an example run of DO-EGTA in game r_{30}. Here one can see that the defender fails to deviate beneficially in rounds $(16, 19, 23)$, while the attacker fails in seven of the later rounds. Moreover, the largest deviation gains tend to occur in earlier training rounds, such as the attacker gain of 73 in round 0. In this run of DO-EGTA, the process converges after round 23, when neither training process learns a beneficial deviation. (Additional plots of payoffs before and after DNN training, including both games and both training methods, are shown in Appendix A, Figures A.5 and A.6.)

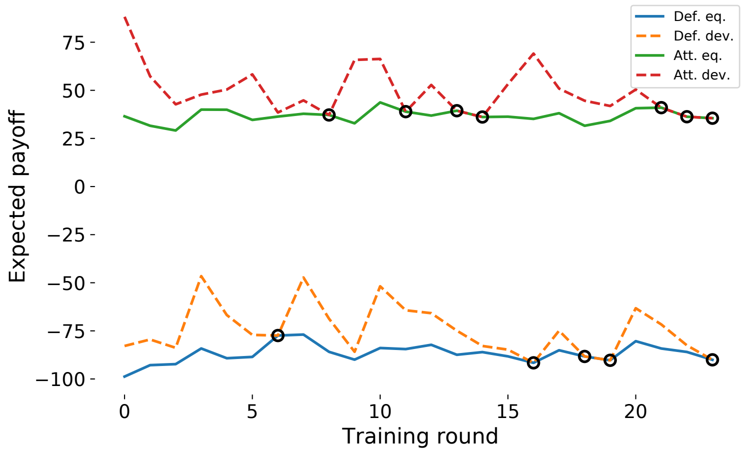

FIGURE 5.5: Expected payoff of attacker and defender, before and after learning a new DNN, in each training round. Results are for game r_{30}, in a single example run of DO-EGTA. Circles indicate rounds with no beneficial deviation. (Deviation payoffs based on fixed opponent mixed strategy.)

Trends in deviation gains from round to round are easier to see in a plot of the difference between the equilibrium payoff and the payoff of the learned deviating strategy, as shown for game r_{30} in figure 5.6. The figure shows the mean payoff gain of each round, over multiple independent runs of DO-EGTA, with standard error of the mean confidence intervals. (Note that in cases where no beneficial deviation was learned, the deviation gain is shown as zero.) This figure more plainly shows that in game r_{30}, deviation gains tend to become smaller as more training rounds are completed, but with large variations from one round to the next. Figure 5.6 also suggests there may be a lagged cross-correlation between the deviation gains for attacker and defender. A positive lagged cross-correlation between gains might be caused by a process where, if either player happens to learn a highly successful new strategy, this gives the opponent an opportunity to achieve a large gain by learning a counter-strategy during the next few training rounds. (Additional plots of gains from DNN training, including both games and both training methods, are shown in Appendix A, Figures A.1 and A.2.)

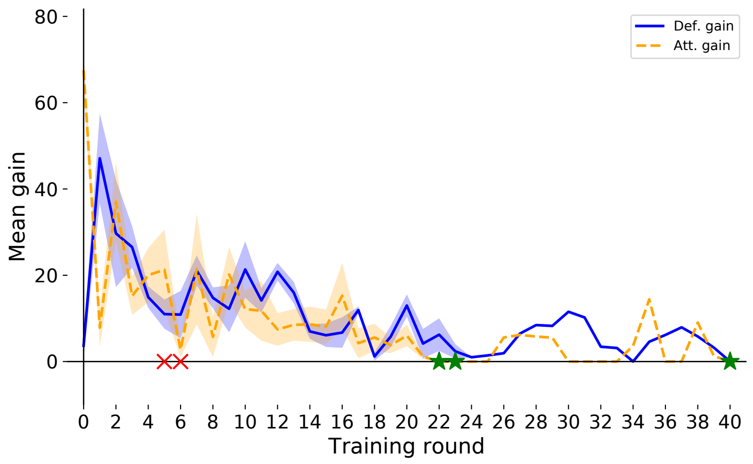

FIGURE 5.6: Payoff gain for new learned strategy, relative to current equilibrium. Results are the mean over all 5 DO-EGTA runs for game r_{30}, shown with standard error of the mean. Green stars show when runs converged, red crosses when runs were stopped early.

One clear result in Figure 5.5 is that the trend in deviation gains from deep RL is not smooth or monotonic. Indeed, the attacker and defender deviation gains seem to be small in most rounds, but with occasional very large gains, even (but infrequently) in later rounds of training.

The anytime performance of DO-EGTA is important, because iterated deep RL takes a long time even with parallelism within each round, the number of rounds to convergence is hard to predict, and the noisy nature of deep RL training can lead to spurious convergence when beneficial deviation is still likely. Figure 5.7 shows the regret of each training round's current equilibrium strategy for attacker and defender, relative to the final equilibrium strategies from converged training. The results shown are mean regrets over two runs of DO-EGTA, in game r_{30}, shown with standard error of the mean confidence intervals. Observe that after about half of the rounds in a typical run have completed (i.e., after round 11), regrets generally decrease over time for each agent and remain lower than the worst regrets of the early rounds. This suggests that there are only mild risks of high regret, in case training stops a few rounds too early due to spurious convergence or time constraints.

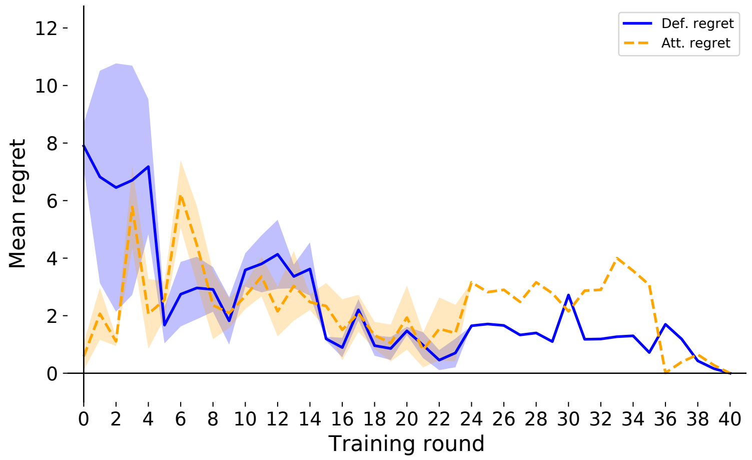

FIGURE 5.7: Regret of current equilibrium strategy, relative to the run's final equilibrium. Results shown are means over independent runs of DO-EGTA in game r_{30}. Shading depicts standard error of the mean.

Figure 5.8 presents the learning curve of an example run of DO-EGTA for each of 21 rounds of training in game r_{30}. Each curve has been normalized as the current gain relative to the previous round's equilibrium expected payoff. Observe that most of the curves showing large gains over the equilibrium payoff are in earlier rounds, which are mapped to purple colors. Even in later training rounds (which are mapped to yellow), the learner steadily improves in expected payoff, especially over the first half of training, before leveling off at a payoff above the equilibrium payoff until the last round. (Appendix A presents learning curves for both training methods and both games, in Figures A.3 and A.4.)

As other researchers have noted [36], the performance of deep RL is noisy and sensitive to arbitrary variations, such as the choice of random seed. I have observed in pilot tests with different random seeds that the DQN implementation used here occasionally achieves very different payoffs from one run to the next, although in many cases payoffs are similar across runs. Another factor that could contribute to non-smooth changes in deviation gain from round to round, is the change in the equilibrium opponent between rounds. When a new strategy is added to a game, it can cause large changes in which other strategies have high weight at Nash equilibrium, leading to a new equilibrium that may be much easier or harder for DQN to exploit. It is possible that the large variation in deviation gain from one round to another is caused by changes in the composition of the Nash equilibria.

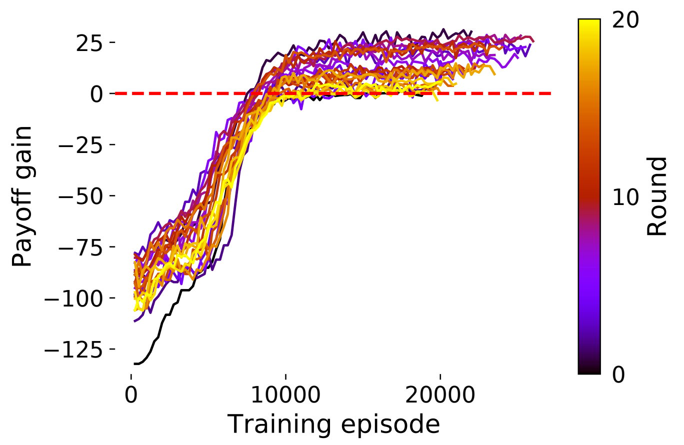

FIGURE 5.8: Defender's learning curve for each round of DO-EGTA training, in an example run for game r_{30}. Curves are shown as gain relative to previous equilibrium. Color map goes from purple in early rounds to yellow in later ones.

5.6.2 HADO-EGTA improvement over DO-EGTA

The principal approach I take to compare the performance of the HADO-EGTA and DO-EGTA methods is to evaluate the strategic stability of each beneficially deviating DNN produced by a run of either method. As suggested in recent work by Balduzzi et al. [5], the best way to evaluate the quality of a strategy in an unsolved game may be to test the performance of that strategy against other agents playing a Nash equilibrium mixed strategy. Following on this lead, I take the union over all beneficially deviating DNNs produced by all runs of HADO-EGTA and DO-EGTA in an environment (e.g., r_{30} or s_{29}), along with all the heuristic strategies, and find Nash equilibria of the resulting *combined game*. One can then measure the quality of any strategy by measuring its regret with respect to the Nash equilibria of the combined game—that is, how much lower the strategy's expected payoff is than the equilibrium strategy's, when the opposing agent uses its equilibrium strategy. The combined games contain many more strategies than the final games from individual runs: in setting s_{29}, 179 attacker and 333 defender strategies, vs. 30–50 and 82–122 in individual runs' final games; in setting r_{30}, 109 attacker and 182 defender strategies, vs. 23–35 and 67–89.

As shown in Figure 5.9, HADO-EGTA produces DNNs with lower regret on average, with respect to Nash equilibria of the combined game, compared to DO-EGTA. Figure 5.9 shows results from the 2 runs of HADO-EGTA and 3 of DO-EGTA carried out in each setting, r_{30} or s_{29}. In the combined games, I found 4 Nash equilibria in setting r_{30} and 1 in s_{29}.

Each HADO-EGTA column has 2 markers per equilibrium, each DO-EGTA column 3 per equilibrium. Across all equilibria, almost all HADO-EGTA runs produced mean regret below any corresponding DO-EGTA run. The magnitude of the HADO-EGTA improvement ranges from roughly a factor of 0.25 to 0.5 for a given condition, taking the mean over results for attacker or defender, in setting r_{30} or s_{29}. Note that the combined games we analyze here for r_{30} contain all DNN strategies, but for s_{29} contain only DNN strategies up to round 73 in the single DO-EGTA run that has not completed. Runs included in this analysis have either converged or completed at least 82 rounds.

When one examines the regret of DNNs relative to the training round in which they are produced, it appears that DNNs trained by both DO-EGTA and HADO-EGTA tend to have lower regret, with respect to the combined game, in later training rounds. However, as shown in the appendix in Figures A.9 and A.10, HADO-EGTA produces DNNs whose regrets decrease more rapidly during the initial training rounds, and remain lower late in training; in contrast, DO-EGTA produces DNNs whose regrets decrease slowly from round to round, in spite of beginning on par with HADO-EGTA. This result makes sense, considering that during early training rounds, there is little history of opponents for HADO-EGTA to train against, so HADO-EGTA should perform similarly to DO-EGTA at first.

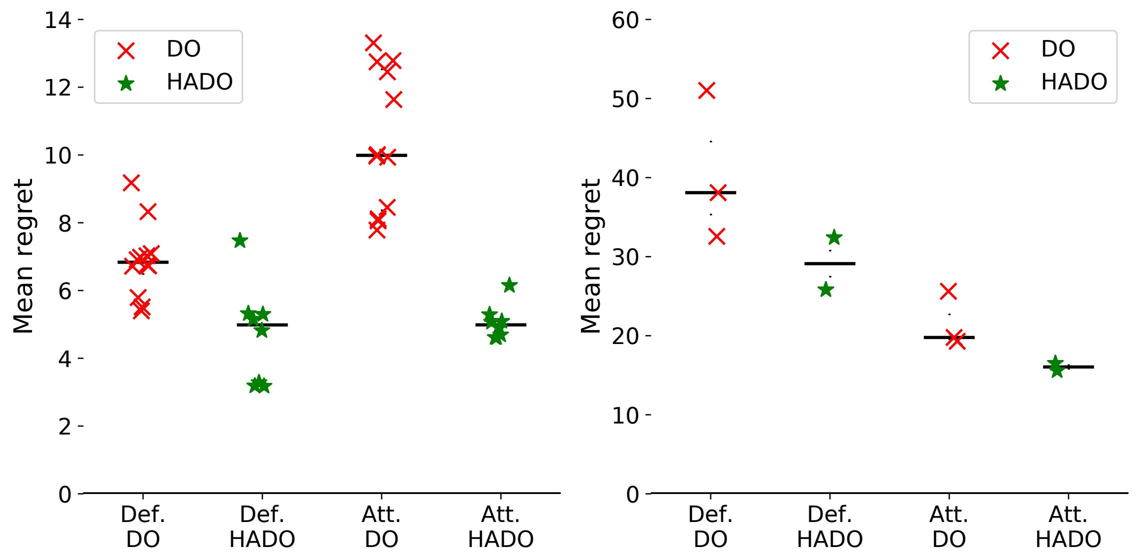

FIGURE 5.9: Mean regret of all beneficially deviating DNNs trained, with respect to Nash equilibria of the combined game, from runs of HADO-EGTA or DO-EGTA. Left: setting r_{30}; right: setting s_{29}. Each marker is based on one run, taking regret w.r.t. to one Nash equilibrium. Bars show median result for each condition.

A distinct approach I take to comparing HADO-EGTA and DO-EGTA performance is to conduct a tournament between the final equilibrium mixed strategies from all converged runs. In the tournament, for a given environment (e.g., r_{30} or s_{29}), each final HADO-EGTA or DO-EGTA equilibrium mixed strategy will meet each final equilibrium mixed strategy of the opposing agent type. For any pair of converged runs, where one is HADO-EGTA and the other DO-EGTA, a run is considered the clear winner if it outperforms the other run in each of 8 payoff comparisons. Specifically, for some run A to win over run B: for both attacker and defender, each run must achieve lower payoff against A than against B; and A must yield higher payoff than B against each opposing run.

For the tournament approach to analysis, I have 8 pairs of converged runs to use, comprising 2 runs each of DO-EGTA and HADO-EGTA in setting s_{29}, as well as all 3 DO-EGTA and two HADO-EGTA runs in setting r_{30}. The HADO-EGTA run is the clear winner in 2 pairs from setting s_{29} and in 1 of the 6 pairs from setting r_{30}, with all 8 payoff comparisons being in favor of the HADO-EGTA mixed strategies. The other 2 matched pairs from setting s_{29} have HADO-EGTA superior in $(4, 6)$ of 8 payoff comparisons. In the other 5 matched pairs from setting r_{30}, HADO-EGTA is superior in only $(3, 5, 6, 7, 7)$ of 8 payoff comparisons respectively, meaning neither equilibrium is clearly stronger. In all, the tournament results constitute modest evidence that HADO-EGTA produces stronger mixed strategies upon convergence than DO-EGTA.

An ancillary finding is that on average, the mixed-strategy Nash equilibria produced during HADO-EGTA training tend to have smaller support than those of DO-EGTA. For example, in setting r_{30}, the mean defender equilibrium strategy support size from HADO-EGTA runs is 4.29, compared to 5.53 from DO-EGTA. As one might expect, there is an increasing trend over successive training rounds in the support size of equilibria, as more high-quality strategies are added to the strategy set. However, for a given round index, HADO-EGTA equilibria tend to have smaller support than those of DO-EGTA, and among runs with similar overall round count, HADO-EGTA runs have lower mean equilibrium support size. This trend could possibly be explained by the way HADO-EGTA trains DNNs against not just the current equilibrium opponent mixed strategy, but also against other opponent strategies. As a result, perhaps when a DNN trained by HADO-EGTA is present at equilibrium, these other opponent strategies are prevented from being in the equilibrium support, because that DNN is able to exploit their weaknesses. Thus, I consider the smaller support of HADO-EGTA equilibria an encouraging sign, that HADO-EGTA is successful at producing DNNs that perform well against a variety of opponent strategies.

Analyzing causes of HADO-EGTA gains

I have shown supporting evidence that HADO-EGTA tends to yield more strategically stable results than DO-EGTA, both in terms of the resulting DNNs' regret with respect to the combined game, and in terms of the final equilibrium mixed strategies' performance in a tournament. Here I consider possible causes of any gains of HADO-EGTA relative to DO-EGTA, and I attempt to determine which explanations are better supported by the evidence.

There are multiple methodological differences between HADO-EGTA and DO-EGTA that could contribute to performance gains of HADO-EGTA:

1. Each round of HADO-EGTA in these experiments adds 400,000 training steps in the retraining phase, to the 700,000 or 1,000,000 training steps used by DO-EGTA.

2. Each round of HADO-EGTA in these experiments selects the best DNN among 4 intermediate training outputs, while DO-EGTA must use the DNN from the end of training.

3. HADO-EGTA fine-tunes each DNN against opponents from equilibria of previous rounds, while DO-EGTA trains only against the current round's equilibrium opponent.

It is important to consider all of these possible explanations, not only Item 3, to provide a clear picture of how HADO-EGTA works without bias in favor of the rationale that motivated the HADO-EGTA approach [53]. In any case, a future researcher could incorporate some of these aspects of HADO-EGTA into the basic DO-EGTA method, such as by increasing the length of each round's training phase as in Item 1, or by recording several interim training outputs and selecting the best as in Item 2.

The strongest method for distinguishing between these possible causes of HADO-EGTA improvements might be an ablation study, but the time and computation requirements of re-running DO-EGTA and HADO-EGTA under various ablations are too great for such an analysis to be included here. Instead, I present an analysis based on data that were collected during the DO-EGTA and HADO-EGTA runs already performed.

To evaluate Item 1, the effect of additional training steps, one can examine whether the expected payoff of the DNN being trained continues to increase substantially in the final steps of DO-EGTA training; if not, it would seem less likely that additional training is a main cause of HADO-EGTA improvements. (Admittedly, it is possible that if the learning rate were annealed toward zero more slowly, as is typical for longer training runs, then larger gains might have been produced in late training steps.) To this end, I examined 3 DO-EGTA training runs for each environment (i.e., s_{29} or r_{30}) and measured the mean difference in payoff over all rounds between the DNN after the end of training and the DNN after only a 0.9-factor of training steps were complete. On average, the gain in payoff for the fully-trained DNN was 3.6 in setting s_{29}, and 0.5 in setting r_{30}. The fraction of rounds in which the fully-trained DNN had higher payoff was 0.64 in setting s_{29}, and 0.57 in setting r_{30}. Thus, the gains in payoff over the last steps of DO-EGTA training were small in magnitude and not consistently positive. It is possible, but not assured, that these gains could have been improved with a different learning rate schedule. These results tend to counter the possibility that Item 1, the additional training steps in each round of HADO-EGTA, is a primary reason for the better performance of HADO-EGTA.

To investigate the impact of Item 2, the selection among 4 interim DNNs by HADO-EGTA, I take two approaches: I measure how much worse each DNN produced by DO-EGTA performs, relative to the best interim performance achieved during its training, as an estimate of how much better performance could have been if training had stopped early at the optimal point. I also check how often HADO-EGTA selects DNNs other than the final one produced during a training round.

Results show that the mean difference in expected payoff between the DNN after all DO-EGTA training steps, and the best interim DNN during the corresponding round of DO-EGTA training, is 3.37 for r_{30} and 8.66 for s_{29}. These gains are fairly modest, indicating that the reduction in performance due to sub-optimal stopping time cannot fully account for the poorer performance of DO-EGTA, relative to HADO-EGTA. Moreover, these estimates of the loss due to selecting the final DNN from training instead of stopping earlier are probably exaggerated, because the payoff estimates recorded during training are based on few sample runs, leading to noisy values and the possibility of bias when taking the maximum.

In the analysis of HADO-EGTA fine-tuning, I find that the final DNN produced in a training run is selected in 0.63 fraction of cases for setting s_{29}, and 0.55 for setting r_{30}. Thus, slightly more than half the time, the DNN produced at the end of fine-tuning is selected. The average increase in expected payoff against the current equilibrium opponent, between the selected DNN and the final fine-tuned one, is 4.61 in setting s_{29} and 1.51 in setting r_{30}. This means that in cases where HADO-EGTA does not select the final DNN from fine-tuning, the selected network has only slightly better performance against the current equilibrium opponent. (Of course, the selected network might have markedly better performance against other opponent strategies.) Hence, the evidence does not support the idea that HADO-EGTA achieves better results than DO-EGTA primarily because HADO-EGTA selects the best among several interim DNNs.

In summary, these analyses provide some evidence that neither Item 1, the extended training period of HADO-EGTA, nor Item 2, the DNN selection component of HADO-EGTA, can fully account for the improved performance of HADO-EGTA relative to DO-EGTA. It appears that the final training steps of each round of DO-EGTA training tend to produce only modest increases in expected payoff. If DO-EGTA training were stopped early in each round at the optimal step, it seems there would typically be only a small increase in the expected payoff of the resulting DNN. HADO-EGTA selects the final fine-tuned DNN the majority of the time. And when HADO-EGTA selects an earlier DNN, that DNN's gain in expected payoff against the current equilibrium opponent, relative to the fully fine-tuned DNN, tends to be small. This leaves the remaining explanation, Item 3, that it is fine-tuning against previous rounds' equilibrium opponents that largely accounts for the improved performance of HADO-EGTA relative to DO-EGTA.

5.6.3 Evolution of equilibrium mixed strategies

The training method used here progresses from an initial equilibrium over only heuristic strategies, through equilibria that can include all beneficially deviating DQN strategies learned so far. I find that as the training round increases, both attacker and defender tend to weight the heuristic strategies less in the equilibrium mixed strategy, and they place increasing weight on the most recently learned DNNs. This suggests progress is made in each training round toward generating more strategically stable strategies.

One can see a fuller picture of how equilibrium behavior changes from round to round, by looking at a heat map, which plots the weight of each DNN in each round's equilibrium. In figure 5.10, for each of 22 training rounds of an example run of DO-EGTA in game r_{30}, I plot the weight in the Nash equilibrium of each defender DNN strategy learned so far, as well as of all heuristic strategies combined (shown as DNN 0). By definition, the full weight of 1 will be on heuristic strategies in round 0. After a later round n, weight of 1 can be spread across heuristics and any DNN up to n. (Heat maps showing attacker and defender mixed strategies for game r_{30}, under DO-EGTA or HADO-EGTA training, are shown in Appendix A, in figures A.7 and A.8.)

In a case where the agent learns dramatically better strategies in each round, one would see almost all weight along the main diagonal of the heat map, where the newest DQN strategies appear. In contrast, if learning plateaued, one would see little weight

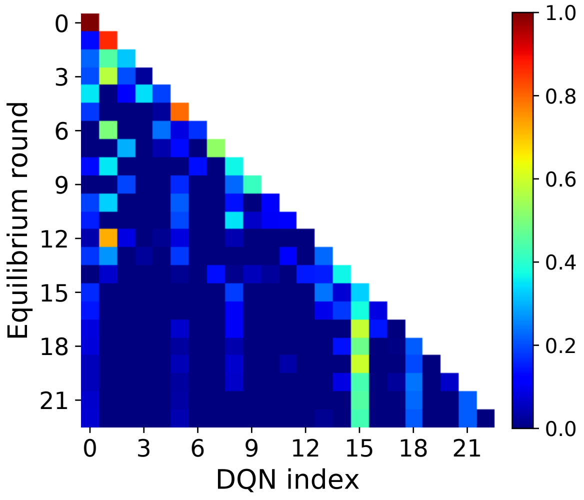

FIGURE 5.10: Heat map of the mixed strategy of the defender after each round of training, as a mixed strategy over DNNs learned in previous rounds, for a single example run of DO-EGTA. Results are for game r_{30}.

on the main diagonal beyond a certain vertical line, and weight would remain on the most successful DNNs or heuristics from earlier rounds. Figure 5.10 seems to show that the defender in game r_{30} was learning successfully until about round 15, placing a great deal of weight at equilibrium on the newest DNNs, although with some weight also on heuristic strategies. After round 16, the defender began using the epoch 15 DNN heavily at equilibrium, and not placing as much weight on new DNNs. This pattern is typical of DO-EGTA training runs. One could learn more about the nature of the round-15 defender DNN strategy by observing its actions in sample game playouts and comparing them to those of other strategies, but I have not conducted this analysis.

I find that when the Nash equilibria contain heuristic strategies, they almost exclusively contain the more sophisticated heuristics previously found to be successful by Nguyen et al. [62], not the naive baselines. In fact, only one equilibrium that was found, over 57 rounds in these two games, included a naive baseline heuristic strategy. Specifically, one equilibrium in game s_{29} included the attacker's *uniform* strategy, with a weight of 0.02. This confirms that the complex strategies previously proposed are more strategically stable than the naive baseline strategies. The superiority of the sophisticated heuristics to simple baseline strategies makes it more noteworthy that DQN consistently generates beneficial deviations from the heuristics and has greater weight in Nash equilibria.

Only a few distinct heuristic strategies were present in any of the equilibria. In game

s_{29}, I found for the attacker 1 aRW and 3 aVP strategies in some equilibrium, and for the defender, 5 dRW_aRW and 6 dRW_aVP parameterizations. In game r_{30}, I found for the attacker 1 aRW and 2 aVP strategies in some equilibrium, and for the defender, 5 dRW_aRW and 2 dRW_aVP parameterizations.

5.6.4 Time requirements of DO-EGTA

I developed a completely automated system to run iterated deep RL experiments, so no time is wasted between the completion of one stage and the beginning of the next. This system runs several steps during each round, including: (1) calling *gambit-lcp* to find a Nash equilibrium over the current strategy set; (2) training a DQN defender and attacker to seek beneficial deviations; (3) sampling the new defender's and attacker's expected payoff; (4) for any player's DNN that deviated beneficially, sampling the payoff of that DNN against all opponent strategies. The system stops when both attacker and defender DQNs fail to deviate beneficially in the same round.

The most time-consuming parts of this system are training the attacker and defender DNNs, and sampling the payoff of a new DNN against all opponents. Because the number of opponents increase in each round, sampling the payoff against all opponents takes an increasing amount of time in later rounds.

I ran the experiments shown in this work on servers running Intel Xeon CPUs with 2.0–3.7 GHz clock speed, at least 4 GB RAM available, and using either Ubuntu 16 or CentOS 7. The DNNs were trained with CPU only, using TensorFlow version 1.5. I believe that training on GPU might have sped up the process only slightly, because a key bottleneck was in running the Java game simulator to sample training episodes, which is CPU-bound. I have not, however, tested training on GPU to evaluate possible gains in training speed. This problem could be fixed by running the game simulator on multiple processes in parallel, asynchronously from the DNN updates.

In my experience, DQN training takes 6–12 hours for the defender and 13–21 hours for the attacker (simultaneously), depending on how long the opponent's mixed strategy takes to generate moves. Computing the payoffs of the new attacker and defender DNNs against all opponent strategies requires about 6–15 hours, with an increasing trend as the number of learned strategies rises. I sample the payoff of 400 random games, for each pair of a new DQN strategy and an opponent strategy, to find the sample mean payoffs with sufficiently low standard error.

In total, each round of DO-EGTA takes about 21–32 hours, depending on the equilibrium mixed strategies, which server is used, and resource contention with other experiments. Therefore, a complete run of 20–30 rounds might require about 25–40 days to complete, assuming iteration times increase as strategies are added.

The HADO-EGTA method requires more time per iteration than DO-EGTA (unless pre-training iterations are decreased), because it adds a fine-tuning step, and another step for evaluating the payoff of several learned DNNs to select the best one. In this study, each iteration of HADO-EGTA took from 21–51 hours, compared to 21–32 hours for DO-EGTA. In addition, HADO-EGTA tends to require more rounds before converging, due to increased training effort per round and the multiple opportunities to find a beneficial deviation within each training round. A complete run of HADO-EGTA might be expected to take about 30% more days than a run of DO-EGTA.

There are several opportunities to reduce the time per training round. One could use an asynchronous training algorithm like asynchronous advantage actor-critic [59] instead of DQN, allowing one to more easily parallelize the generation of game simulation episodes. And one could parallelize the generation of payoff samples for new strategies against all opponents, running multiple copies of the simulator in different processes. If one could parallelize the generation of training game episodes, one might then take better advantage of faster GPU updates to the DNN being trained.

5.7 Discussion

Based on the experience in this study, DO-EGTA consistently finds strategic equilibria of complex cybersecurity games, with greater stability than equilibria over previously-developed heuristics alone. This study applies such techniques to a dynamic attack-graph game and finds that deep RL can repeatedly learn new attack and defense strategies that deviate beneficially from equilibria over the set of strategies developed previously. Because the learning method searches over a larger strategy space than prior work, I expect that the solutions found represent more accurate models of rational attacker and defender behavior.

The experimental results indicate that the history-aware method, HADO-EGTA, tends to produce more strategically stable strategies than the naive approach, DO-EGTA. However, there are multiple possible explanations for why HADO-EGTA performs better, which could be the subject of future work. The strongest method for distinguishing between these possible causes of HADO-EGTA improvements might be an ablation study.

One could run DO-EGTA with an increased number of training steps to match those of HADO-EGTA, eliminating the difference in training time per round. Another experiment could run HADO-EGTA with $\kappa = 2$, such that there was only one fine-tuned DNN to select, reducing the difference in the number of DNNs available to select from per round. Finally, an experiment could run HADO-EGTA with $\gamma \approx 0$, such that the opponent for fine-tuning was approximately the same as the pre-training opponent, eliminating the difference in training opponents. The difficulty with this approach is that it would be computationally expensive and require a long time to run such experiments, each of which is inherently impossible to parallelize due to the sequential nature of HADO-EGTA.

There seems to be a trade-off in game-theoretic analysis between the expressive power of the agents' strategy sets and the computation required to complete the study. The time and computation needed to simulate all pairs of heuristic strategies in this study was just a small fraction of what was needed to train attacker and defender DQNs for 20 or more rounds. The payoff gains achieved by the DQN agents are considerable, but it is desirable to increase the parallelism of the pipeline to reduce the start-to-finish time required. I hope that by generating training episodes on multiple threads in parallel, I could reduce the time per round to less than one day.

One difficulty with iterated deep RL methods such as DO-EGTA is that it is difficult to predict how many rounds will be required for convergence a priori, and this round count sometimes has high variance between runs. Fortunately, the anytime performance appears to be fairly good after the first half of the expected run count has been completed, in that the current equilibrium strategies empirically tend to have low regret with respect to the final equilibrium after training converges. Another difficulty is that deep RL sometimes fails to learn effectively with some random seems, which could lead to spurious convergence. This problem could be remedied by requiring multiple failures of deep RL (with different random seeds) before considering a run converged, but in the experience of this study, spurious convergence was not a problem, with good hyperparameters for the DNNs and training algorithm.

There are many challenges to applying these methods to cybersecurity domains in general, such as designing a suitable input representation for the DNN, designing the mapping from DNN output to game actions, tuning hyperparameters of the DNN and training procedure, and adjusting the size of the game itself to suit limitations of current deep RL. (For example, it may be too difficult for DQN to learn a game with millions of state elements or actions.) Designing the DNN's input representation requires both providing sufficient information for the DNN to make good decisions, and not giving irrelevant inputs or using intractably many input dimensions. More concretely, this chapter dealt with DNN inputs and outputs that map to nodes on a graph by omitting the graph's structure from the input, such that a distinct DNN must be trained for each graph, and by training the DNN to output one node or edge to add to an action set, instead of indicating an entire set of items on which to act. Despite these challenges in application, deep RL is a powerful tool for discovering strong strategies or evaluating weaknesses in hand-coded ones, with great potential to improve the analysis of cybersecurity games.

Chapter 6

Conclusion

In this dissertation, I have addressed the problem of how to validate or improve the quality of strategies and equilibria produced in studies of strategic games. This work is relevant to many real-world domains that are modeled as games in the literature, and I have demonstrated applications to financial markets and cybersecurity settings. The tools of game theory and reinforcement learning are widely used to analyze how self-interested agents might behave in such domains. But in sufficiently complex games, there may be no known (practical) way to analytically find optimal, rational strategies for agents to use. The tools of simulation-based game theory, including empirical game-theoretic analysis, as well as strategy exploration methods like deep reinforcement learning, have been used successfully in prior work to find strong strategies in many diverse games. My work in this dissertation offers new methods for evaluating the stability of strategy profiles, or for searching for more strategically stable strategy profiles, with applicability to complex, simulation-based games.

6.1 Discussion

Chapter 3 concerns the problem, in games with intractably large (or infinite) strategy sets, of evaluating the strategic stability of a profile. Often in studies of simulation-based games with large strategy sets, analysts will find a profile that is known to be a Nash equilibrium over a restricted game, where only small, finite subset of strategies is allowed. But the question remains, how robust this profile might be to an agent that deviates by playing an alternative strategy outside the known subset. I presented a statistical method, which samples strategies from a probability distribution over the game's full strategy set and checks whether each is a beneficial deviation. If any sampled strategy is a beneficial deviation, it can be added to the set of known strategies, a new Nash equilibrium found, and the process repeated until no deviation is found or a maximum number of trials have elapsed. I showed that when this procedure converges successfully, it yields a theoretical guarantee on the probability of an opponent finding a beneficial deviation using the given probability distribution, and I demonstrated the efficacy of the procedure in an auction game and a cybersecurity game.

In Chapter 4 I presented a method for evaluating the strategic stability of profiles that fall within a restricted subset of a simulation-based game, using reinforcement learning to measure how much better a deviating strategy from the unrestricted game can perform. In cases where an analyst studies a game by finding a Nash equilibrium over a restricted, finite subset of strategies, my method challenges this equilibrium by automatically learning beneficially deviating strategies within a much larger strategy space. If the best deviating strategies produce only a small gain in payoff relative to the equilibria of the restricted strategy set, one may conclude that the original equilibria are reasonably strategically stable. I presented an application of this method to a simulated trading agent game, where equilibria had been found among a finite set of parameterizations of a simple trading agent design. Results indicated that a widely studied trading agent design, known as the zero-intelligence trader, can be reasonably stable, when its parameters are chosen through an equilibration process to be well-suited to the particular game instance.

The work in Chapter 5 presents an application of iterated deep reinforcement learning to a cybersecurity scenario, known as an attack-graph game. Prior works have proposed strategies and solution methods for attack-graph games, which are widely used to model the interaction between a defender and an attacker trying to progressively gain control of a computer system. My approach uses deep RL as a better-response oracle, in a procedure similar to the double-oracle method or policy space response oracles, which can iteratively learn stronger mixed strategies in a complex game. I demonstrated the progress of this method in two versions of the attack-graph game, over dozens of rounds of strategy learning. I also proposed a more sophisticated strategy exploration method, taking advantage of the history of opponent strategies during training, which empirically tends to produce better-performing final mixed strategies, more consistent improvements from learning, and sparser equilibrium mixed strategies. Both our basic method and our more complex method consistently produce beneficially deviating strategies in the first several rounds of training, and steadily converge to more stable equilibrium profiles over many training rounds.

6.2 Future work

There are many questions and problems raised by this work that would be interesting to pursue in the future. Regarding the work of Chapter 3 on probably almost stable profiles, it would be desirable to experiment with more powerful search algorithms, instead of simple random search or a brief run of simulated annealing. For example, one could try other black-box optimization algorithms from recent literature, or even deep RL, as the search algorithm for beneficially deviating policies. One challenge with such experiments is that they will likely be computationally demanding, considering that the policy search method must run many times at each stage of our algorithm. It is likely that careful tuning of the problem size and other parameters would be necessary to render such an experiment feasible, if many trials are to be conducted, especially if experiments are run in simulation-based games with costly payoff samples.

Respecting Chapter 4 on the stability of equilibrated zero-intelligence strategy profiles, future work could use deep RL, instead of tabular Q-learning, in an attempt to produce larger payoff gains against other agents using zero-intelligence strategies. It is conceivable with an appropriate DNN architecture, and a suitable input representation of market state, deep RL could learn to achieve meaningfully better performance than zero-intelligence traders in our market model, even if the zero-intelligence traders had been equilibrated over many parameterized forms. A study investigating this possibility could add to our understanding of the continuous double auction and the nature of rational trading strategies.

Chapter 5 raises many questions for future work. Deep RL successfully learned new strategies that beneficially deviated from equilibria over hand-coded ones, but the nature of these strategies remains opaque. It would be good to understand the behavior of these learned strategies better, along several dimensions, including why they produce higher payoffs, how they represent information about the game state, what their goals are, and what weaknesses they have. It would also be interesting to more thoroughly investigate the performance of my proposed methods for learning stronger mixed strategies, which have been shown to work in an attack-graph game, but have not been subjected to a full ablation study or to comparisons with alternative approaches on a broad variety of games. Such thorough evaluation is currently computationally expensive for deep RL, particularly for the kind of iterated deep RL that was performed here. One would need considerable resources to experiment on ablated versions of an algorithm, or duplicated runs across many game types, over dozens of rounds of iterated deep RL.

Appendix

Appendix A

Supplemental Data on Deep Reinforcement Learning

A.1 Deep RL hyperparameters

r_{30} attacker inputs	241
r_{30} defender inputs	240
s_{29} attacker inputs	234
s_{29} defender inputs	232
hidden layers	2 FC layers, size 256
activations	ReLU per hidden layer
r_{30} attacker outputs	106
r_{30} defender outputs	31
s_{29} attacker outputs	103
s_{29} defender outputs	30
learning rate	5×10^{-5}
training discount factor	0.99
batch size	32
ϵ in DO-EGTA	linear 1.0 to 0.03 in first half, then 0.03
ϵ in HADO-EGTA pre.	linear 1.0 to 0.03 in first half, then 0.03
ϵ in HADO-EGTA fine.	linear 0.3 to 0.03 in first half, then 0.03
train steps in DO-EGTA	700k (1m for r_{30} defender)
train steps in HADO-EGTA pre.	700k (1m for r_{30} defender)
train steps in HADO-EGTA fine.	400k
κ in HADO-EGTA	4
γ in HADO-EGTA	0.7
α in HADO-EGTA	$\frac{2}{7}$
DQN options	no prioritized replay; no param noise

TABLE A.1: Deep RL hyperparameters

A.2 Payoff gains from deep RL by round

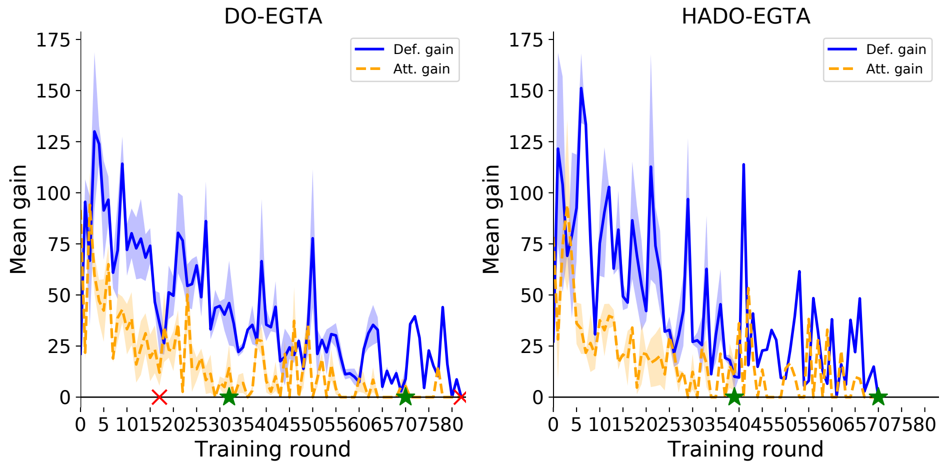

FIGURE A.1: Payoff gain for new learned strategy, relative to the current equilibrium. Results shown are means over independent runs of DO-EGTA (4) or HADO-EGTA (2) in game s_{29}. Shading depicts standard error of the mean. Green stars show when runs converged, red crosses when they were stopped early.

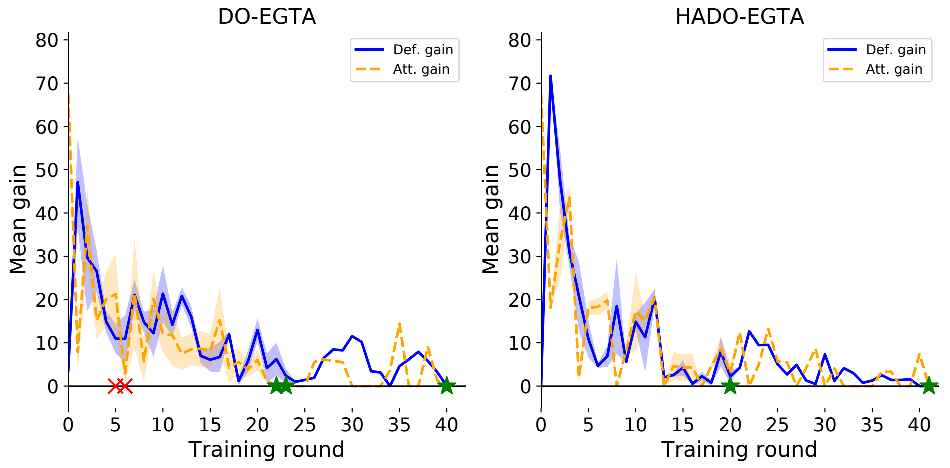

FIGURE A.2: Payoff gain for new learned strategy, relative to the current equilibrium. Results shown are means over independent runs of DO-EGTA (5) or HADO-EGTA (2) in game r_{30}. Shading depicts standard error of the mean. Green stars show when runs converged, red crosses when they were stopped early.

A.3 Example learning curves of DO-EGTA runs

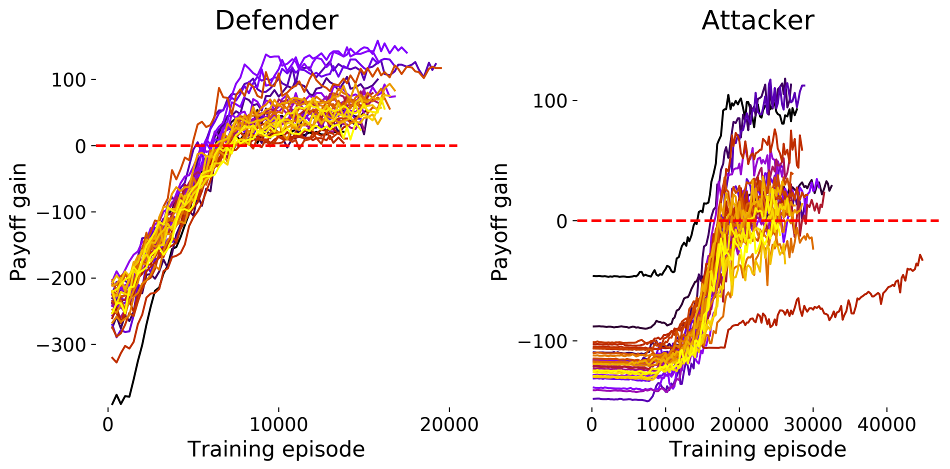

FIGURE A.3: Learning curve of each round of DO-EGTA training, for defender (left) or attacker (right), in an example run for game s_{29}. Curves are shown as gain relative to previous equilibrium. Color map goes from purple in early rounds to yellow in later ones.

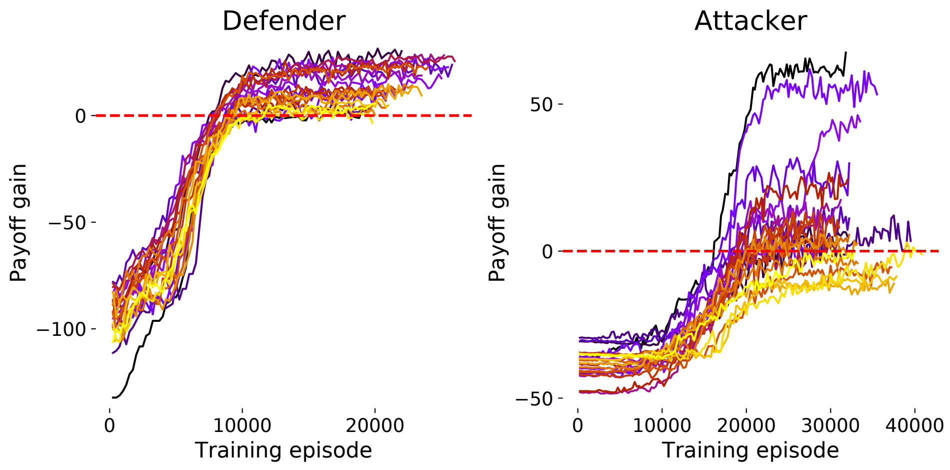

FIGURE A.4: Learning curve of each round of DO-EGTA training, for defender (left) or attacker (right), in an example run for game r_{30}. Curves are shown as gain relative to previous equilibrium. Color map goes from purple in early rounds to yellow in later ones.

A.4 Payoffs of equilibria and deviations

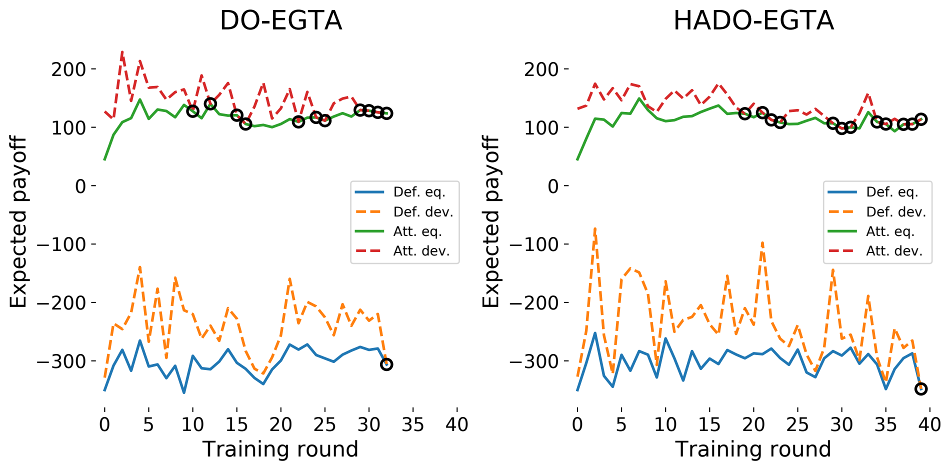

FIGURE A.5: Expected payoff of each equilibrium mixed strategy, and each agent's corresponding deep RL deviating strategy, in game s_{29}. Circles indicate rounds with no beneficial deviation. Results are for one example run of DO-EGTA or HADO-EGTA.

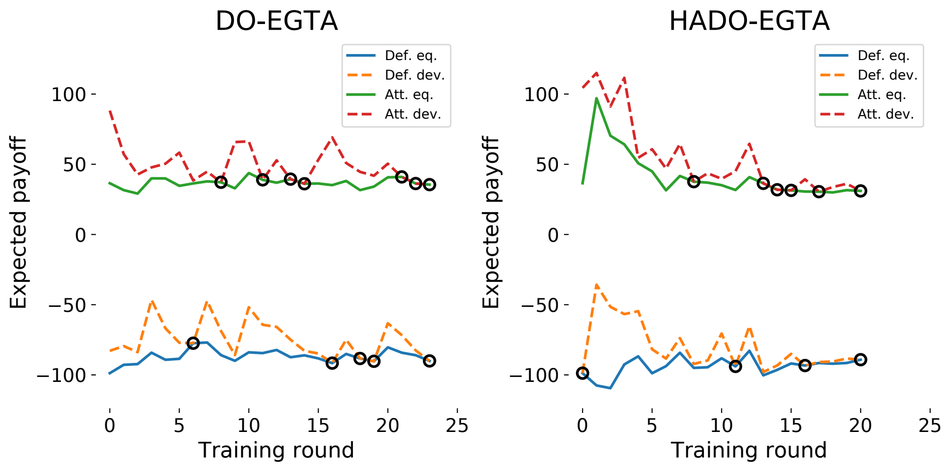

FIGURE A.6: Expected payoff of each equilibrium mixed strategy, and each agent's corresponding deep RL deviating strategy, in game r_{30}. Circles indicate rounds with no beneficial deviation. Results are for one example run of DO-EGTA or HADO-EGTA.

A.5 Equilibrium strategies by round

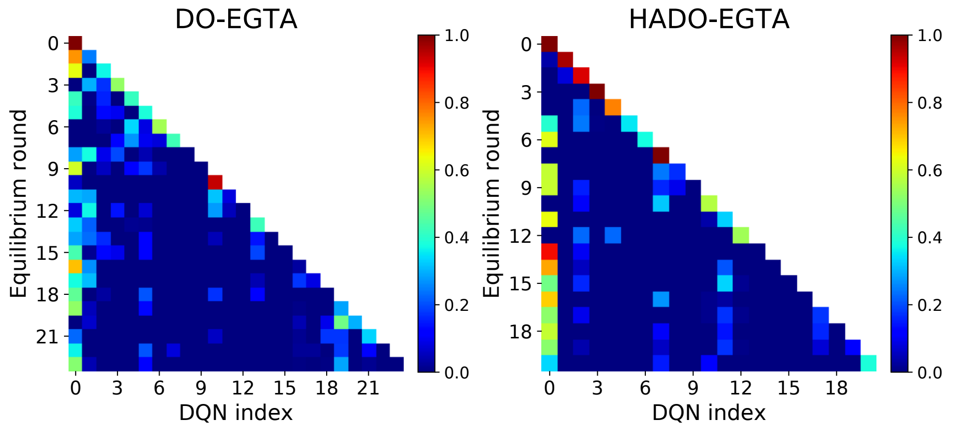

FIGURE A.7: Heat map of the equilibrium attacker mixed strategy by round. Results are for a single run of DO-EGTA or HADO-EGTA, in game r_{30}. Network column 0 represents heuristic strategies.

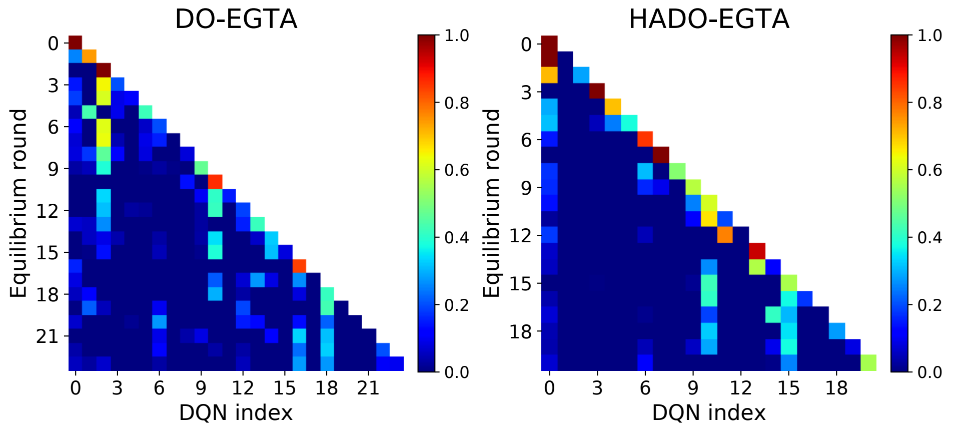

FIGURE A.8: Heat map of the equilibrium defender mixed strategy by round. Results are for a single run of DO-EGTA or HADO-EGTA, in game r_{30}. Network column 0 represents heuristic strategies.

A.6 DNN regrets by round

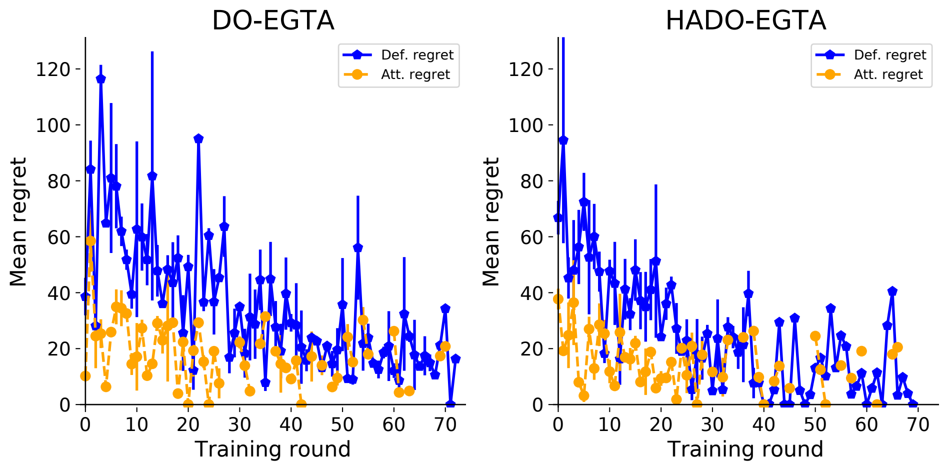

FIGURE A.9: Mean over training runs of DO-EGTA or HADO-EGTA, of the regret of each round's DNNs, with respect to a Nash equilibrium of the combined game for s_{29}. Gaps appear where no beneficially deviating network was produced. Error bars show standard error of the mean.

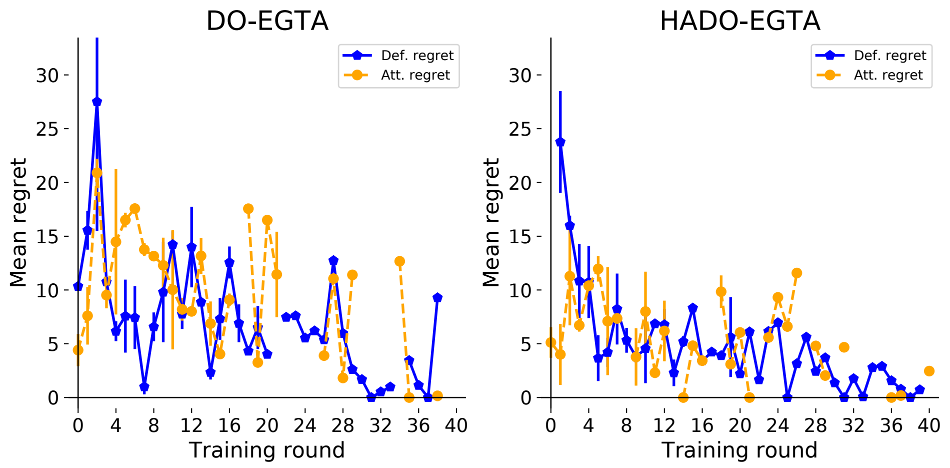

FIGURE A.10: Mean over training runs of DO-EGTA or HADO-EGTA, of the regret of each round's DNNs, with respect to a Nash equilibrium of the combined game for r_{30}. Gaps appear where no beneficially deviating network was produced. Error bars show standard error of the mean.

Bibliography

[1] A. Agresti and B. A. Coull, "Approximate is better than 'exact' for interval estimation of binomial proportions", *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998.

[2] Amenaza. (2018). SecurITree, [Online]. Available: `http://www.amenaza.com/` (visited on 10/12/2018).

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[4] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games", in *35th International Conference on Machine Learning*, 2018.

[5] D. Balduzzi, K. Tuyls, J. Perolat, and T. Graepel, "Re-evaluating evaluation", in *32nd Conference on Neural Information Processing Systems*, 2018.

[6] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition", in *6th International Conference on Learning Representations*, 2018.

[7] S. D. Bopardikar, A. Borri, J. P. Hespanha, M. Prandini, and M. D. Di Benedetto, "Randomized sampling for large zero-sum games", *Automatica*, vol. 49, no. 5, pp. 1184–1194, 2013.

[8] E. Brinkman and M. P. Wellman, "Shading and efficiency in limit-order markets", *Algorithmic Game Theory Workshop at IJCAI*, 2016.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym", *arXiv:1606.01540*, 2016.

[10] L. D. Brown, T. T. Cai, and A. DasGupta, "Interval estimation for a binomial proportion", *Statistical Science*, vol. 16, no. 2, pp. 101–133, 2001.

[11] E. Budish, P. Cramton, and J. Shim, "The high-frequency trading arms race: Frequent batch auctions as a market design response", *Quarterly Journal of Economics*, vol. 130, no. 4, pp. 1547–1621, 2015.

[12] B.-A. Cassell and M. P. Wellman, "EGTAOnline: An experiment manager for simulation-based game studies", in *Multi-Agent Based Simulation XIII*, ser. Lecture Notes in Artificial Intelligence, vol. 7838, Springer, 2013.

[13] M. Chakraborty, S. Das, and J. Peabody, "Price evolution in a continuous double auction prediction market with a scoring-rule based market maker", in *29th AAAI Conference on Artificial Intelligence*, 2015, pp. 835–841.

[14] F. Cheng, J. Liu, K. Amin, and M. P. Wellman, "Strategic payment routing in financial credit networks", in *17th ACM Conference on Economics and Computation*, 2016, pp. 721–738.

[15] D. Cliff and J. Bruten, "Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets", HP Laboratories, Tech. Rep., 1997.

[16] C. J. Clopper and E. S. Pearson, "The use of confidence or fiducial limits illustrated in the case of the binomial", *Biometrika*, vol. 26, no. 4, pp. 404–413, 1934.

[17] M. Dacier and Y. Deswarte, "Privilege graph: An extension to the typed access matrix model", in *European Symposium on Research in Computer Security*, 1994, pp. 319–334.

[18] M. Dacier, Y. Deswarte, and M. Kaâniche, "Models and tools for quantitative assessment of operational security", in *Information Systems Security*, 1996, pp. 177–186.

[19] R. Das, J. E. Hanson, J. O. Kephart, and G. Tesauro, "Agent-human interactions in the continuous double auction", in *17th International Joint Conference on Artificial Intelligence*, 2001, pp. 1169–1178.

[20] M. De Luca and D. Cliff, "Human-agent auction interactions: Adaptive-aggressive agents dominate", in *22nd International Joint Conference on Artificial Intelligence*, vol. 22, 2011, pp. 178–185.

[21] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "OpenAI Baselines", *GitHub repository*, 2017.

[22] K. Durkota, V. Lisỳ, B. Bošanskỳ, and C. Kiekintveld, "Approximate solutions for attack graph games with imperfect information", in *Sixth International Conference on Decision and Game Theory for Security*, 2015, pp. 228–249.

[23] K. Durkota, V. Lisỳ, B. Bosanskỳ, and C. Kiekintveld, "Optimal network security hardening using attack graph games", in *24th International Joint Conference on Artificial Intelligence*, 2015, pp. 526–532.

[24] R. Engelbrecht-Wiggans and E. Katok, “Regret and feedback information in first-price sealed-bid auctions”, *Management Science*, vol. 54, no. 4, pp. 808–819, 2008.

[25] J. D. Farmer, P. Patelli, and I. I. Zovko, “The predictive power of zero intelligence in financial markets”, *Proceedings of the National Academy of Sciences*, vol. 102, pp. 2254–2259, 2005.

[26] V. Firoiu, W. F. Whitney, and J. B. Tenenbaum, “Beating the world’s best at Super Smash Bros. with deep reinforcement learning”, *arXiv:1702.06230*, 2017.

[27] J. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch, “Learning with opponent-learning awareness”, in *17th International Conference on Autonomous Agents and Multiagent Systems*, 2018, pp. 122–130.

[28] D. Friedman, “A simple testable model of double auction markets”, *Journal of Economic Behavior & Organization*, vol. 15, no. 1, pp. 47–70, 1991.

[29] S. Gjerstad and J. Dickhaut, “Price formation in double auctions”, *Games & Economic Behavior*, vol. 22, no. 1, pp. 1–29, 1998.

[30] D. K. Gode and S. Sunder, “Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality”, *Journal of Political Economy*, vol. 101, no. 1, pp. 119–137, 1993.

[31] P. W. Goldberg, “Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game”, in *23rd ACM Symposium on Principles of Distributed Computing*, 2004, pp. 131–140.

[32] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D Sculley, “Google Vizier: A service for black-box optimization”, in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1487–1495.

[33] P. Grnarova, K. Y. Levy, A. Lucchi, T. Hofmann, and A. Krause, “An online learning approach to generative adversarial networks”, in *6th International Conference on Learning Representations*, 2018.

[34] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning”, in *30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[35] J. He, M. Ostendorf, X. He, J. Chen, J. Gao, L. Li, and L. Deng, “Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads”, in *Conference on Empirical Methods in Natural Language Processing*, Austin, Texas, 2016, pp. 1838–1848.

[36] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters”, in *32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 3207–3214.

[37] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning”, in *32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 3215–3222.

[38] K. Ingols, R. Lippmann, and K. Piwowarski, “Practical attack graph generation for network defense”, in *Computer Security Applications Conference*, IEEE, 2006, pp. 121–130.

[39] Isograph. (2018). AttackTree+, [Online]. Available: `http://www.isograph-software.com/atpover.htm` (visited on 10/12/2018).

[40] M. Jain, D. Korzhyk, O. Vaněk, V. Conitzer, M. Pěchouček, and M. Tambe, “A double oracle algorithm for zero-sum security games on graphs”, *10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 327–334, 2011.

[41] P. R. Jordan, C. Kiekintveld, and M. P. Wellman, “Empirical game-theoretic analysis of the TAC supply chain game”, in *6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 1188–1195.

[42] P. R. Jordan, L. J. Schvartzman, and M. P. Wellman, “Strategy exploration in empirical games”, in *9th International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 1131–1138.

[43] B. Jovanovic and A. J. Menkveld, “Middlemen in limit order markets”, *SSRN preprint: https://ssrn.com/abstract=1624329*, 2016.

[44] N. Kamra, U. Gupta, F. Fang, Y. Liu, and M. Tambe, “Policy learning for continuous space security games using neural networks”, in *32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 1103–1112.

[45] R. Klima, D. Bloembergen, R. Savani, K. Tuyls, D. Hennes, and D. Izzo, “Space debris removal: A game theoretic analysis”, *Games*, vol. 7, no. 3, 20:1–20:18, 2016.

[46] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, “Attack–defense trees”, *Journal of Logic and Computation*, vol. 24, no. 1, pp. 55–87, 2014.

[47] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, “DAG-based attack and defense modeling: Don’t miss the forest for the attack trees”, *Computer Science Review*, vol. 13, pp. 1–38, 2014.

[48] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning", in *31st AAAI Conference on Artificial Intelligence*, 2017, pp. 2140–2146.

[49] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, J. Perolat, D. Silver, T. Graepel, *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning", in *Advances in Neural Information Processing Systems*, 2017, pp. 4190–4203.

[50] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas", in *16th International Conference on Autonomous Agents and Multiagent Systems*, 2017, pp. 464–473.

[51] C. E. Lemke and J. T. Howson Jr., "Equilibrium points of bimatrix games", *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.

[52] Z. Li and S. Das, "An agent-based model of competition between financial exchanges: Can frequent call mechanisms drive trade away from CDAs?", in *15th International Conference on Autonomous Agents and Multiagent Systems*, 2016, pp. 50–58.

[53] Z. C. Lipton and J. Steinhardt, "Troubling trends in machine learning scholarship", *arXiv:1807.03341*, 2018.

[54] O. L. Mangasarian, "Equilibrium points of bimatrix games", *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 4, pp. 778–780, 1964.

[55] R. D. McKelvey, A. M. McLennan, and T. L. Turocy, "Gambit: Software tools for game theory", 2006.

[56] H. B. McMahan, G. J. Gordon, and A. Blum, "Planning in the presence of cost functions controlled by an adversary", in *20th International Conference on Machine Learning*, 2003, pp. 536–543.

[57] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal defense policies for partially observable spreading processes on Bayesian attack graphs", in *Second ACM Workshop on Moving Target Defense*, 2015, pp. 67–76.

[58] S. Mike and J. D. Farmer, "An empirical behavioral model of liquidity and volatility", *Journal of Economic Dynamics and Control*, vol. 32, no. 1, pp. 200–234, 2008.

[59] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", in *33rd International Conference on Machine Learning*, 2016, pp. 1928–1937.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[61] A. K. Nandi, H. R. Medal, and S. Vadlamani, "Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model", *Computers & Operations Research*, vol. 75, pp. 118–131, 2016.

[62] T. H. Nguyen, M. Wright, M. P. Wellman, and S. Singh, "Multistage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis", *Security and Communication Networks*, vol. 2018, 2018.

[63] NYSE. (2017). NYSE group volume records - top 10 years, [Online]. Available: `http://www.nyxdata.com/nysedata/asp/factbook/viewer_edition.asp` (visited on 10/01/2017).

[64] C. Olsson, S. Bhupatiraju, T. Brown, A. Odena, and I. Goodfellow, "Skill rating for generative models", *arXiv:1808.04888*, 2018.

[65] OpenAI. (2018). OpenAI Five, [Online]. Available: `https://blog.openai.com/openai-five/` (visited on 06/25/2018).

[66] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer", in *14th USENIX Security Symposium*, 2005.

[67] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis", in *Workshop on New Security Paradigms*, 1998, pp. 71–79.

[68] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs", *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[69] D. M. Reeves, "Generating trading agent strategies: Analytic and empirical methods for infinite and large games", PhD thesis, University of Michigan, 2005.

[70] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *32nd International Conference on Machine Learning*, 2015, pp. 1889–1897.

[71] L. J. Schvartzman and M. P. Wellman, "Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning", in *8th International Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 249–256.

[72] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice", in *International Symposium on Abstraction, Reformulation, and Approximation*, 2005, pp. 194–205.

[73] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs", in *IEEE Symposium on Security and Privacy*, IEEE, 2002.

[74] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play", *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[75] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of Go without human knowledge", *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[76] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs", in *Advances in Neural Information Processing Systems*, 2010, pp. 2164–2172.

[77] A. Sureka and P. R. Wurman, "Using tabu best-response search to find pure strategy Nash equilibria in normal form games", in *4th International Conference on Autonomous Agents and Multiagent Systems*, 2005, pp. 1023–1029.

[78] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[79] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning", *PLOS One*, vol. 12, no. 4, 2017.

[80] A. Tavares, H. Azpurua, A. Santos, and L. Chaimowicz, "Rock, paper, StarCraft: Strategy selection in real-time strategy games", in *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016, pp. 93–99.

[81] P. D. Taylor and L. B. Jonker, "Evolutionary stable strategies and game dynamics", *Mathematical Biosciences*, vol. 40, no. 1–2, pp. 145–156, 1978.

[82] G. Tesauro, "Temporal difference learning and TD-Gammon", *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[83] G. Tesauro and J. L. Bredin, "Strategic sequential bidding in auctions using dynamic programming", in *1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002, pp. 591–598.

[84] G. Tesauro and R. Das, "High-performance bidding agents for the continuous double auction", in *3rd ACM Conference on Electronic Commerce*, 2001, pp. 206–209.

[85] M. Thulin, "The cost of using exact confidence intervals for a binomial proportion", *Electronic Journal of Statistics*, vol. 8, no. 1, pp. 817–840, 2014.

[86] S. Venkatesan, M. Albanese, A. Shah, R. Ganesan, and S. Jajodia, "Detecting stealthy botnets in a resource-constrained environment using reinforcement learning", in *Fourth ACM Workshop on Moving Target Defense*, 2017, pp. 75–85.

[87] P. Vytelingum, D. Cliff, and N. R. Jennings, "Strategic bidding in continuous double auctions", *Artificial Intelligence*, vol. 172, no. 14, pp. 1700–1729, 2008.

[88] E. Wah, S. Lahaie, and D. M. Pennock, "An empirical game-theoretic analysis of price discovery in prediction markets", in *25th International Joint Conference on Artificial Intelligence*, 2016, pp. 510–516.

[89] E. Wah and M. P. Wellman, "Latency arbitrage, market fragmentation, and efficiency: A two-market model", in *14th ACM Conference on Electronic Commerce*, 2013, pp. 855–872.

[90] E. Wah, M. Wright, and M. P. Wellman, "Welfare effects of market making in continuous double auctions", *Journal of Artificial Intelligence Research*, vol. 59, pp. 613–650, 2017.

[91] W. E. Walsh, R. Das, G. Tesauro, and J. O. Kephart, "Analyzing complex strategic interactions in multi-agent systems", in *AAAI-02 Workshop on Game-Theoretic and Decision-Theoretic Agents*, 2002, pp. 109–118.

[92] Y. Wang, Z. R. Shi, L. Yu, Y. Wu, R. Singh, L. Joppa, and F. Fang, "Deep reinforcement learning for green security games with real-time information", in *33rd AAAI Conference on Artificial Intelligence*, 2019.

[93] C. J.C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.

[94] M. P. Wellman, "Methods for empirical game-theoretic analysis (extended abstract)", in *Twenty-first National Conference on Artificial Intelligence*, 2006, pp. 1552–1556.

[95] ——, "Putting the agent in agent-based modeling", *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 6, pp. 1175–1189, 2016.

[96] B. Wiedenbeck and M. P. Wellman, "Scaling simulation-based game analysis through deviation-preserving reduction", in *11th International Conference on Autonomous Agents and Multiagent Systems*, 2012, pp. 931–938.

[97] M. Wright and M. P. Wellman, "Evaluating the stability of non-adaptive trading in continuous double auctions", in *17th International Conference on Autonomous Agents and Multiagent Systems*, 2018, pp. 614–622.