Artificial Intelligence Algorithms for Large Economic and Computer Games

by

Zun Li

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
2024

(Generated by DALL·E 3)

Zun Li

lizun@umich.edu

ORCID iD: 0000-0003-1748-0883

DEDICATION

This thesis is dedicated to my beloved grandfather, Baoshen Li, whose love, optimism, and integrity have nurtured me in countless ways.

ACKNOWLEDGMENTS

I am blessed with a marvelous Ph.D. journey paved with many arduous trials and fortunate turns. And yet, all the accomplishments in this dissertation cannot be achieved without the help of others. I am indebted to the following great minds; it is with them I can propel myself forward to the furthest places I could ever imagine.

First and foremost, I would like to thank my Ph.D. advisor, Prof. Michael P. Wellman. His infinite patience and integrity incubated a benevolent environment and mentorship. His ideas and visions, on the other hand, always enlightened me to a higher level of understanding of science, truth, and the world. From the intellectual debates with him on both technical and non-technical problems, I learned to be a philosophical and strategic thinker. I am grateful to be one of Mike's students, and I am sure the things I learned from him will benefit me for my whole life.

The second person I want to thank most is my internship mentor at DeepMind, Dr. Marc Lanctot. Marc's patience, optimism, humor, passion, generosity, and wisdom are just some of the reasons why I appreciate this delightful mentorship and friendship. He had helped me resolve those very difficult technical challenges even before the internship started, during which he taught me the most professional skills in the field and guided me into the deepest regions of research. From him, I learned a spirit of pursuing the truth and values of dealing with the world. I look forward to working with him and learning from him again in the future.

I am also grateful for other members of my dissertation committee, whom I am fortunate to have been interacting with since my first year. Thank Eugene for teaching me the correct meaning of extensive-form games, subgame perfect equilibrium, and other knowledge, and I always admired his intellectual insights; thank Mingyan for organizing the weekly game theory meetings in my first year and providing feedback, with which I laid a solid foundation on multi-agent research; thank Satinder for offering a wonderful reinforcement learning course in my first year, where I started to contemplate those most important concepts in machine learning and AI very early.

I want to thank my co-authors on the MURI project: Feiran Jia, Shahin Jabbari, and Aditya Mate. My excellent collaborators during my internship at DeepMind: Luke Marris, Ian Gemp, Kevin McKee, Daniel Hennes, Paul Muller, Kate Larson, and Yoram Bachrach. I had spent an unforgettable time at Alberta with the following admired: Neil Burch, Nolan Bard, Kevin Waugh, Michael Bowling, Finbarr Timbers, Josh Davidson, and Martin Klissarov. I am also grateful for the other people I met at Google: Florin Constantin, Ying Lu, Shuang Yang, and Jichen Yang.

I have been surrounded by so many amazing labmates at the Strategic Reasoning Group, both present and past: Xintong Wang, Yongzhao Wang, Qiurui Ma, Mason Wright, Frank Cheng, Megan Shearer, Mithun Chakraborty, Arunesh Sinha, Steven Jecmen, Max Smith, Madelyn Gatchel, Christine Konicki, Katherine Mayo, Austin Nguyen, and Chris Mascioli. I am also very lucky to have interacted with many legends from this group: Junling Hu, Bill Walsh, Pete Wurman, Shih-Fen Cheng, and Bryce Wiedenbeck. Their insights are very valuable in the intermediate steps of my Ph.D..

Going through a Ph.D. life can be painful, but over the years spending time with friends is always the best remedy. I would like to thank some of the people who I have been maintaining friendships with from an early age and who keep supporting me during these years: Junyan Qi, Junjie Fan, Yifei Wang, Kunyan Han, Yucheng Shi and Jiawen Ma. I am grateful to have spent a wonderful time on the badminton courts of UMich with the following interesting spirits: Xun Xu, Chengcheng Li, Chen Liang, Chenrui Hu, Jiaheng Xie, Zhen Zhong, Jiajia Fan, Weiling Wen, and many others. Thanks to my Ph.D. peers for supporting me and always giving me helpful suggestions: Jiacheng Ma, Fan Lai, Dingyu Wang, Qingzhao Zhang, Tianhan Gao, and Ying Yang. And thanks to my former roommate Zetong Guan, for sharing many ordinary but important daily life with me, and taking care of me when I was ill.

Lastly, I would like to thank my parents Qunying Gu and Guocheng Li. Thanks for supporting me and staying with me during all these crisis moments. I love you forever.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE

ABSTRACT

Contemporary artificial intelligence algorithms (search, graphical models, machine learning, etc.) have achieved great success in a variety of practical domains. This thesis particularly considers their application to the equilibrium analysis of multiagent systems. Specifically, I study the following subject: *how a structured combination of modern artificial intelligence methods facilitates strategic reasoning focusing on equilibrium concepts on multiagent systems of diverse domains, especially those without tractable and analytical description*.

After laying out the technical foundations, I present four research works to illustrate the theme. The first three follow the chronological order in which most game theory textbooks are organized: the most basic normal-form games are first studied, then games with incomplete information, and then dynamical games with imperfect information. The difference here, though, is that my approaches are more from a computational perspective using practical AI methods, instead of deriving the exact mathematical solutions. First, I demonstrate how supervised learning and unsupervised learning techniques can be utilized under a model-based structure learning framework to facilitate equilibrium computation in many-player normal-form games. This method can scale to games with hundreds of players. Second, I show how a particular class of policy search algorithms being well-studied in deep reinforcement learning can be employed in generic frameworks to solve many-player games of incomplete information. The pure equilibria computation method can recover classic analytical solutions in simple auction games. And both the pure and mixed equilibria methods scale to games with high-dimensional type space and action space. Third, I develop a general-purpose multi-agent algorithm that combines an AlphaZero-styled tree-search and a population-based RL training loop, for general-sum extensive-form games with large imperfect information. Using this algorithm, a game-playing bot is built and can achieve comparable social welfare with humans as when humans trade with themselves in a class of negotiation game. In the last part, instead of focusing on *solving* a particular game, I consider the problem of *evaluating* different interactive AI algorithms by using a meta-game analysis framework. A variety of game-theoretic properties of model-free, model-based, self-play, and population-based multi-agent reinforcement learning algorithms are uncovered.

CHAPTER 1

Introduction

"Every man takes the limits of his own field of vision for the limits of the world."

Arthur Schopenhauer, *Studies in Pessimism: The Essays*

The field of Artificial intelligence (AI) has shown great success in a wide range of applications, and as a result, it has become one of the most popular research areas in computer science. This is more pronounced since the emergence of modern deep learning [Goodfellow et al., 2016] and reinforcement learning [Sutton and Barto, 2018] techniques, which now supersede classical methods on domains including natural language processing [Mikolov et al., 2013] and computer vision [Krizhevsky et al., 2012]. With the ultimate ambition of these AI research fields being reaching artificial general intelligence (AGI), [1] people's efforts toward this ends are most prominent in perhaps the oldest challenge — what McCarthy [1997] called the "Drosophila of AI" [Omidshafiei et al., 2020] — building strong, or even superhuman-level game-playing AI. Historical breakthroughs of AI were often delivered by progression in this thread: people had already made Grandmaster AI in chess [Campbell et al., 2002], Go [Silver et al., 2016], poker [Brown and Sandholm, 2018], real-time strategy video games [Vinyals et al., 2019, Berner et al., 2019], automobile racing video games [Wurman et al., 2022], Diplomacy [FAIR et al., 2022], and Stratego [Perolat et al., 2022].

The classical setup of an AI problem centers around the concept of an *agent* [Russell and Norvig, 2020], which is broadly defined as an entity that interacts with an environment and makes decisions given its perception inputs and its limited computational resources. In other words, *every agent takes the limits of its own field of vision for the limits of the world.* To produce *machina economica* [Parkes and Wellman, 2015], or agents that make perfect decisions, we need to evaluate the *rationality*[2] of an agent's behavior as a metric of intelligence. When a system involves multi-

[1]AGI encompasses various definitions, one of which being a computational entity that can "achieve a variety of goals, and carry out a variety of tasks, in a variety of different contexts and environments" [Goertzel, 2014].

[2]Rationality also has different meanings across different domains; the current most well-accepted one in AI [Russell, 2016] is the principle of maximizing expected utility.

ple autonomous agents, classical decision-theoretic approaches to single-agent settings may fail to produce a rational outcome, partially due to the non-stationarity of the environment [Hernandez-Leal et al., 2019]. This issue is also reflected in classical microeconomics [Mas-Colell et al., 1995] where multiple decision makers are interacting in a market. Each of these market participants is rendered an analytical description of its utility or payoff and needs to consider other players' strategic effects.

The presence of other agents immediately makes individual utility optimization a wrong objective, which drives agent designers to pursue equilibrium notions [Nash, 1950b] as a more reasonable solution alternative. In fact, game-theoretic reasoning had been proved effective in building successful game-playing AI. The superhuman chess program DeepBlue [Campbell et al., 2002] was designed based on backward induction in dynamic games. The counterfactual regret minimization algorithm [Zinkevich et al., 2008] originating from Blackwell's approachability game [Blackwell, 1956] now occupies the center routine of every superhuman Poker bot [Moravčík et al., 2017, Brown and Sandholm, 2018], and had been shown successful even in a multiplayer, general-sum setting [Brown and Sandholm, 2019] where a theoretical guarantee is missing. As another example, incorporating a competitive market equilibrium [Arrow, 1951] computation process boosted the performance of a trading agent in a complex market-based environment [Cheng et al., 2005].

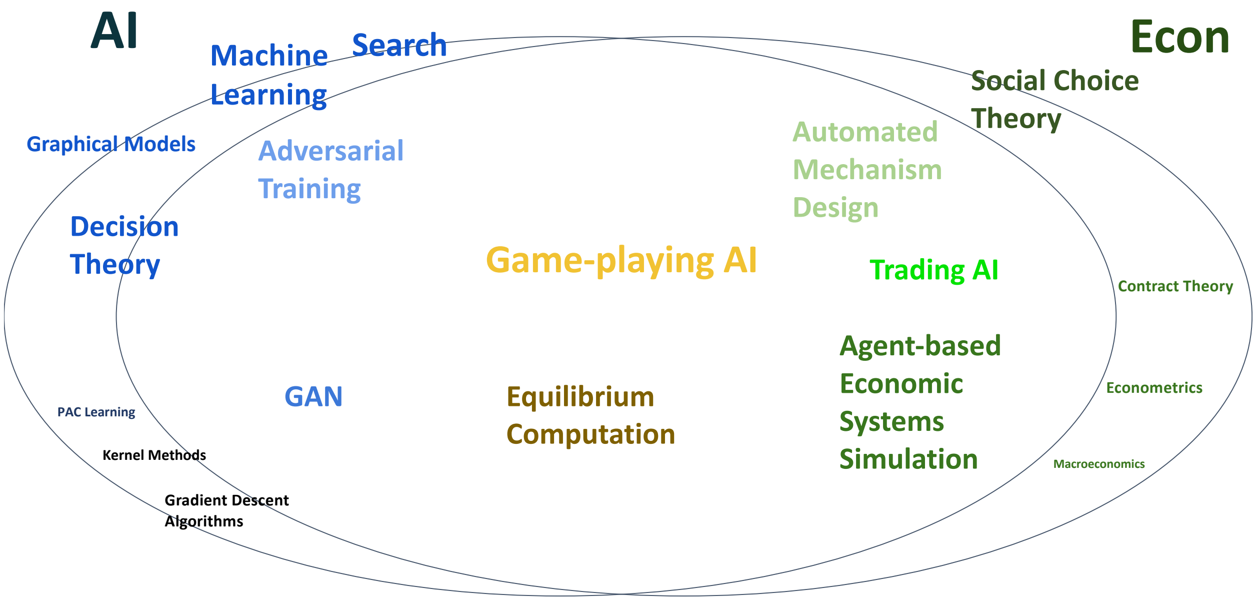


Figure 1.1: A conceptual diagram about the relation between AI and economics.

Today the interplay between AI algorithms and economic principles is exhibited far beyond game-playing AI. I here provide a diagram in Figure 1.1 to illustrate my view of the relation between these two domains. The left circle encloses algorithms and problems studied by AI researchers, while the right surrounds those that economists often consider. I want to qualitatively capture the proportions of a concept's AI and economic elements by placing the concept at a specific diagram position. There are two different frontiers: artificial intelligence algorithms, which include search algorithms, machine learning, graphical models, etc., and economic methodologies, such as social choice theory and econometrics.

I consider the center of this diagram to be game-playing AI since most of the current breakthroughs are benefited equivalently from methodologies from both domains. However, we have also witnessed cases where principles from one domain assist the other and complement the desired goals. For example, in recent years, game-theoretic principles have inspired the ideas of the generative adversarial network (GAN) [Goodfellow et al., 2014] and adversarial training [Vorobeychik and Kantarcioglu, 2018] of deep neural networks. Analogously, today AI algorithms also boost computational methods for economic applications: seminal examples include mechanism design via deep learning [Dütting et al., 2019] and learning algorithms in economic systems [Fischer and Krauss, 2018].

My thesis focuses on the center and the regions near the economic frontier of this diagram. Specifically, I study the following subject: *how a structured combination of modern artificial intelligence methods facilitates strategic reasoning centering around equilibrium concepts on complex multiagent systems of diverse domains, especially those without analytical or tractable description.* I next clarify my usage of these terms.

By "a structured combination" I mean that I am not limiting myself to a particular category of AI algorithms; I hope tools from diverse AI areas can be assembled properly and interacting with each other under a high-level algorithmic paradigm. One example is the *generalized policy iteration* (GPI) adopted in DeepMind's go-playing agent AlphaZero [Silver et al., 2017]. Here the method of self-play deep reinforcement learning plays the *policy evaluation* part, where the process of Monte Carlo tree search (MCTS, detailed in Section 2.2.4.5) serves the *policy improvement* part of the paradigm. Another example is the *double oracle framework* [McMahan et al., 2003] for solving games with large strategy space which was later generalized to the *policy space response oracle* (PSRO) [Lanctot et al., 2017] in multiagent RL settings. PSRO is also dichotomized into one component called *meta-strategy solver* and another one called *best-response oracle*. Studying which AI algorithms to instantiate these building blocks in different applications has been a popular research topic today [Muller et al., 2020, Liu et al., 2021].

By "modern AI methods" I refer to contemporary popular AI algorithms or those which are rapidly evolving and being researched when I am working on this thesis. The algorithms are typically opposed to the classical ones. For example, a "modern" version of the tabular-based Q-learning algorithm of reinforcement learning is the deep Q-network [Mnih et al., 2015] that uses function approximators. Another example is the MCTSnets [Tamar et al., 2016] which uses deep neural networks to conduct approximate planning computation by abstracting MCTS in a differentiable architecture. These modern versions may or may not require a theoretic guarantee but are expected to perform well empirically.

By "strategic reasoning" I mean game-theoretic analysis of multiagent systems. The primary focus would be to develop efficient equilibrium computation algorithms; however other explorations such as devising a principled statistical analysis of a sampled game model [Wiedenbeck et al., 2014] or an evaluation metric of strategic interactions [Balduzzi et al., 2018b] also fall into my consideration.

By "multiagent systems of diverse domains" my goal is to discover methods that are generalizable beyond ad hoc application scenarios and effective enough in terms of dealing with large-scale multiagent environments.

By "without analytical or tractable description" I would like to emphasize that in many applications of interest, there may not be available analytical description of a game model, but only a procedural call or an interface that guarantees simulation access to data samples. This falls into the category of empirical game-theoretic analysis [Wellman, 2006, Tuyls et al., 2020]. For example in the game of Go, the strategy space consists of all mappings from a board state to a distribution over actions, which is too enormous to enumerate. Therefore, in this case, direct game-theoretic reasoning is impossible. But knowing the rules of the game helps one create a simulator that represents the game and conducts a central computation process using simulation data. Another example includes devising trading strategies in the financial market through simulation. In cases when simulation becomes expensive, however, the goal may be switched to producing quality outcomes given the limited sample budget. I aim to develop methods that can handle this kind of scenario.

Therefore, in this thesis, I study the effects of artificial intelligence algorithms on strategic reasoning over complex multiagent systems. In this narrative, the protagonist is a *game analyst* who aims to conduct game-theoretic analysis on a domain of interest. The analyst has access to some *game knowledge*, which can either be a fully analytical description of the game as in the classical setting, or only black-box simulation access as in the modern setting, or a mix of both. It may employ various AI algorithms to facilitate the computation procedure. Each of these AI subprocedures may resort to the game knowledge part and control the query distribution of the black-box inputs, selecting the primary region of the game knowledge it wants to acquire.

This thesis comprises four distinct works. Specifically,

- In Chapter 3, I study the problem of solving many-player normal-form games by using supervised learning and unsupervised learning. The classical approach first represents the whole game as a tensor and directly plugs into existing game-solvers that exploit its given structure. My proposed method adopts a model-based learning approach. I let the analyst iteratively learn the game structure and build a concise representation through machine learning techniques, and then use the equilibrium results of these empirical game models as approximate solutions of the ground-truth game. This work was presented at the Thirty-Fourth AAAI Conference on Artificial Intelligence [Li and Wellman, 2020].

- In Chapter 4, I study the problem of solving many-player games with incomplete information by using deep learning and evolutionary search. The classical approach is to analytically solve for a fixed point of a differential equation of some best-response mapping. However, due to the lack of analytical descriptions in general settings, I represent the game as a black-box oracle, use neural networks to represent the strategy space, and adopt evolutionary search as an efficient optimization procedure. I employ all these techniques as different components in frameworks designed to solve pure and mixed equilibria. This work was presented at the Thirty-Fifth AAAI Conference on Artificial Intelligence [Li and Wellman, 2021b].
- In Chapter 5, I study the problem of solving general-sum games with imperfect information, by combining AlphaZero-styled game-tree search algorithm and a population-based RL method. The classical approach is to conduct one or multiple full traversals of the game-tree and apply backward-induction-typed reasoning. However, it may not scale for games with complex information structures and general-sum elements. To handle games with large imperfect information, I augment the AlphaZero-styled MCTS with a deep generative model at the root of the search tree that represents a belief state. Furthermore instead of using self-play based training, which may usually only work for two-player zero-sum games, I adopt policy-space response oracle (PSRO) as an outer-loop training algorithm. I evaluate my method on several classes of negotiation games, and discover one agent can achieve human-level performance.
- In Chapter 6, I study the problem of *evaluating* different interactive AI algorithms represented by deep multi-agent reinforcement learning methods in general-sum environments using a meta-game analysis framework, where each AI algorithm is framed as a *meta-strategy* in a *meta-game*. The major difference of this work from the previous three is that instead of solving a game for a particular solution concept, I am evaluating the performance of one algorithm by considering its *interactive* performances with other algorithms. By bootstrapping combinations of random seeds and constructing multiple empirical game instances, I construct confidence intervals of different kinds of game-theoretic statistics, and make several interesting observations among state-of-the-art deep MARL algorithms.

I provide in Chapter 2 the basic technical terms and mathematical languages that I will be using throughout this thesis. This will be a coarse survey of computational game theory and multiagent systems. I then finally draw the conclusion and present several future directions in Chapter 7.

CHAPTER 2

Background

In this chapter, I develop technical foundations to provide basic mathematical languages for this thesis. The content focuses on introducing standard decision-making frameworks.

2.1 Game Theory for Multiagent Decision Making

2.1.1 Normal Form Games

2.1.1.1 Definitions

Game theory provides a basic mathematical framework for modeling multiagent scenarios. I particularly concentrate on non-cooperative game theory. Normal-form game representation is the most standard representation of a game. A normal form game (NFG) consists of $(\mathscr{N}, \mathscr{S}, \mathscr{U})$. Here $\mathscr{N} = \{1, \dots, N\}$ is the set of agents and $\mathscr{S} = \times_{n=1}^{N} \mathscr{S}_n$ is the set of joint strategies, where $\mathscr{S}_n$ is the strategy space of agent n. $\mathscr{U} = \{u_1, \dots, u_N\}$ is the set of utility functions. In a play of the game, agent n chooses a strategy $s_n \in \mathscr{S}_n$, and receives its utility value according to its payoff function $u_n : \mathscr{S} \to \mathbb{R}$.

The standard solution concept for NFGs is *Nash equilibrium*, which I present as follows.

Definition 2.1.1 (ϵ-Best response set)**.** Given other-players' strategies s_{-n}, player n's ϵ-best response set is $\mathit{BR}_n^{\epsilon}(s_{-n}) = \{s_n \in \mathscr{S}_n \mid \forall s'_n \in \mathscr{S}_n, u_n(s_n, s_{-n}) + \epsilon \geqslant u_n(s'_n, s_{-n})\}$. The function $u_n(\cdot, s_{-n})$ is called the *deviation payoff function* of n given s_{-n}.

Definition 2.1.2 (Regret/Exploitability)**.** For a joint strategy profile $\boldsymbol{s} = (s_1, \dots, s_N)$, the regret of player n is $\textsc{Regret}_n(\boldsymbol{s}) = \max_{s'_n} u_n(s'_n, s_{-n}) - u_n(s_n, s_{-n})$. The regret of $\boldsymbol{s}$ is $\textsc{Regret}(\boldsymbol{s}) = \max_n \textsc{Regret}_n(\boldsymbol{s})$. In two-player zero-sum games, regret is also called exploitability.

Definition 2.1.3 (ϵ-Nash equilibrium)**.** A joint strategy profile $s^* = (s_1^*, \dots, s_N^*) \in \mathscr{S}$ is an ϵ-Nash equilibrium if $\forall n$, $s_n^* \in \mathit{BR}_n^{\epsilon}(s_{-n}^*)$.

Generally speaking, an NE characterizes the stability of a joint profile. In an ϵ-NE, the incentive of an agent to deviate unilaterally is bounded by ϵ. Then an ϵ-Nash equilibrium is a profile with regret less than or equal to ϵ.

The most basic class of games is finite games with few strategies. An N-player, M-strategy game generally needs a tensor of order $O(NM^N)$ to record all the information about utility values. In terms of finding an equilibrium, however, for many such games, e.g., rock-paper-scissors, a pure Nash equilibrium (a Nash equilibrium defined on this discrete strategy space) may not exist. In such cases, mixed-Nash equilibrium [Reny and Robson, 2004] is usually considered, where now each strategy space $\mathscr{S}_n$ is augmented to a probability simplex $\Delta(\mathscr{S}_n)$. It has been proved [Nash, 1950b] that for finite-strategy games a mixed Nash always exists. Furthermore, for any *non-degenerate games* there is always an odd number of mixed equilibria [Wilson, 1971]. An interesting question is to consider the hardness of computing a mixed Nash equilibrium of a finite game. However, as it had been shown [Daskalakis et al., 2009] that computing a mixed Nash is hard for multiplayer general-sum games. Next, I briefly survey classical algorithms for solving finite-strategy games.

2.1.1.2 Algorithms

I mainly refer to [McKelvey and McLennan, 1996] as an excellent survey of algorithms for computing Nash equilibrium for finite-strategy games. For two-player games, the most famous algorithm is the Lemke-Howson algorithm [Lemke and Howson, 1964] that formulated equilibrium computation as a linear complementary programming problem. It is generalized to the simplicial subdivision algorithm [Rosenmüller, 1971], but practically it is hard to scale to many-player cases. Researchers developed more advanced algorithms over the years including function minimization [McKelvey, 1998], global Newton method [Govindan and Wilson, 2003, 2004], support enumeration approach [Porter et al., 2008], and the more recent exclusion method [Berg and Sandholm, 2017]. A recent work [Gemp et al., 2021] uses a sampled-based approach to approximate the global Newton method and achieve very impressive performances in many-player games. A useful open-source package is Gambit [McKelvey et al., 2006] which implements several Nash-computation algorithms for finite games. In my experience, the global Newton method usually performs the best scalability and efficiency. For two-player finite games, computing an optimal Nash equilibrium can be achieved by solving a mixed integer programming [Sandholm et al., 2005].

2.1.2 A Universe of Games

Before I delve into other classes of games, I here provide a systematic view to unfold the major complexities of a game into different dimensions. Here we identify three different dimensions that

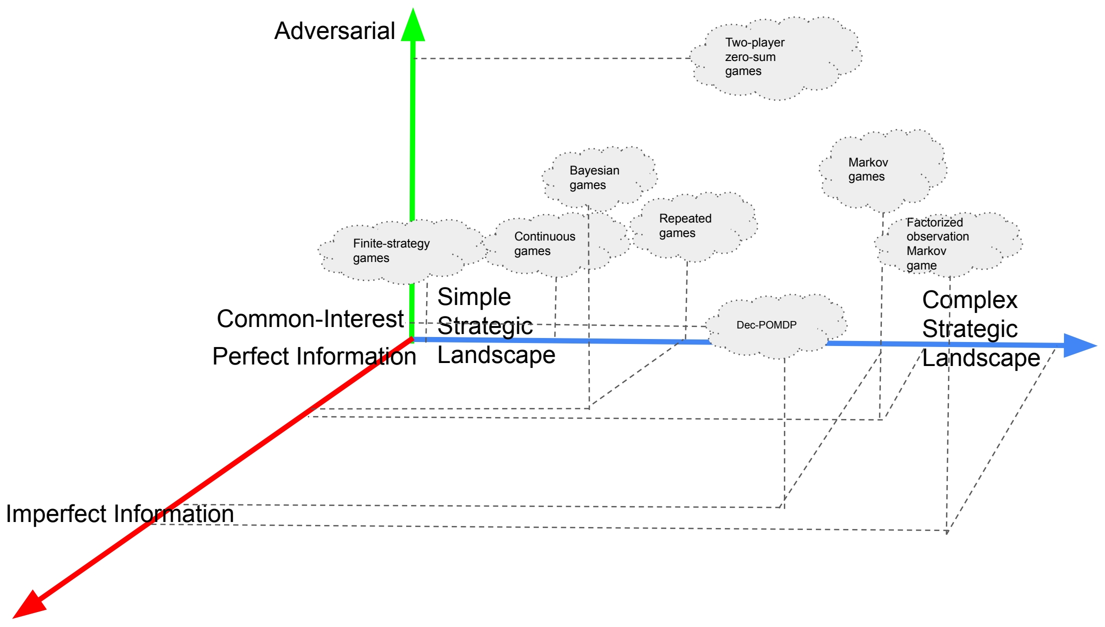


Figure 2.1: A Universe of Games. Each class of game is placed at a specific coordinate consisting of three dimensions: the degree of imperfect information, the degree of conflicts in interest among the players, and the complexity of strategy space.

I consider are the most representative. These are (1) the structural complexity of strategy space, (2) the degree of conflicts in interests, and (3) the degree of imperfect information. I provide a visualization in Figure 2.1. Here each class of game is placed at a specific coordinate of this game space, where each dimension reflects its specific properties.

I next briefly introduce my reasons for this view and develop my ideas in more detail in later sections.

Complexity of strategy space. This first dimension reflects the mathematical structure of the strategy sets. The simplest ones are unstructured finite strategy sets. Incorporating more structure in the strategy spaces may add up to the complexity of a game. If the strategies are combinatorial objects, such as subsets of a finite set, permutations, or graphs, then we may call such games combinatorial games. If the strategies are real vectors, which may provide local gradient information in utilities, we may call such games continuous games. Furthermore, in many games we may encounter functional strategy space. For example, in an auction, a strategy is a function mapping from private values to bids. A functional strategy space may be combinatorially complex or even infinite-dimensional. The most complex class of strategy space that is implementable computationally is state machines — or functions with states. A state may summarize critical information through history for decision-making purposes. For example, in Poker, a strategy is a mapping from all historical observations to a distribution over actions at the current round. However, people will devise more succinct state information for computational purposes.

The degree of conflicts in interests. The second dimension mainly considers the correlation among players' outcomes in a strategic play. At one extreme, a collaborative game consists of agents with one common objective. All agents may share equivalently the same outcome values in all scenarios. Hence, a collaborative game is strategically equivalent to a single optimization problem but may be implemented in a decentralized fashion. At the other extreme, a two-player zero-sum game models a completely adversarial environment. Here one must hurt the other to improve its outcomes in the game. Many well-known games fall into this category, such as chess and the game of Go.

The degree of imperfect information. The final important dimension is the degree of imperfect information (elaborated in Section 2.2.4.1). By definition, imperfect information refers to any unobserved events that took place from the point of view of an acting player when it is making a decision. For example in a financial market, imperfect information may consist of other traders' private information or historical trading behaviors. The imperfect information structure typically reflects the hardness of utilizing historical observations effectively. A special case of imperfect information is incomplete information, which is the hidden parameter of the opponents in their utility functions. For example, in Poker, the private cards of one's opponents constitute incomplete information.

Remark. I disclaim here that I am providing an intuitive conceptual viewpoint rather than a rigorous mathematical taxonomy of games. For example, I defined a complexity notion of a strategy space mainly based on its extrinsic description as a mathematical object. However, practically speaking, whether I call a strategy space is complex or not should depend on its intrinsic game-theoretic properties. For example, a fully collaborative, transitive game with continuous multidimensional strategy space can be much easier to solve than even a 2-player 5-strategy general-sum game with large intransitive components (i.e., a rock-paper-scissors-like structure) [Balduzzi et al., 2019]. I admit my layout here is a coarse description, but I hope the picture conveys an intuitive meaning of a game complexity notion.

2.1.3 Collaborative and Potential Games

2.1.3.1 Definitions

The next classes of interesting games within the above game universe I am considering is collaborative and potential games [Monderer and Shapley, 1996b]. Essentially speaking, these games are not quite "games" since all agents are collaborating on maximizing the same objective function but in a decentralized manner. But still, these models constitute a large domain of multiagent applications, e.g., congestion games [Rosenthal, 1973].

Definition 2.1.4 (Collaborative Games)**.** In collaborative games, $\forall n, u_n = u$ for some function $u : \mathscr{S} \to \mathbb{R}$.

Definition 2.1.5 (Potential Games)**.** In potential games, there exists a potential function $\Phi : \mathscr{S} \to \mathbb{R}$, such that $\forall i, \Phi(s_i, s_{-i}) - \Phi(s'_i, s_{-i}) = l_i(u_i(s_i, s_{-i}) - u_i(s'_i, s_{-i}))$ for some $l_i > 0$.

2.1.3.2 Algorithms

Since in collaborative or potential games, the players are essentially maximizing the same objective function, it can be observed that a Nash equilibrium corresponds to a local maximum of the potential function. Then starting from an arbitrary strategy profile $\boldsymbol{s}_0$, every time a player makes a move (while fixing the others) that increases its utility, the potential function also increases. This inspires an algorithm that computes a Nash equilibrium called iterative best response (IBR). IBR selects one player while fixing other players' strategies in each timestep and chooses the strategy that maximizes this player's current utility. The algorithm terminates until we cannot find one such player. The algorithm, however, may take an exponential number of timesteps to converge.

2.1.4 Zero-Sum Games

2.1.4.1 Definitions

Zero-sum games model perfectly adversarial scenarios [Morgenstern and Von Neumann, 1953]. In two-player zero-sum games, $u_1 + u_2 = 0$. A combination of zero-sum games and collaborative games are (N_1, N_2) teamed zero-sum games, which w.l.o.g. I let $u_1 = u_2 = \ldots = u_{N_1-1} = u_{N_1} = -u$, and $u_{N_1+1} = u_{N_1+2} = \ldots = u_{N_1+N_2-1} = u_{N_1+N_2} = u$. Interestingly, though zero-sum games are strategically opposite to collaborative games, in many contexts, specific properties can be proved only for these two classes of games, or teamed zero-sum games (defined below). Examples include convergence of fictitious play [Monderer and Shapley, 1996a, Robinson, 1951, Sela, 1999] (although it holds for general two-player two-action games [Miyasawa, 1961]), Nash-Q learning [Hu and Wellman, 1998, Littman, 2001],and existential properties of equilibria in Stackelberg Stochastic games [Vorobeychik and Singh, 2012] and continuous-action games [Mazumdar et al., 2020].

Many of the games we are familiar with, such as the game of Go, Poker, Dota II, are essentially two-player zero-sum.

For two-player zero-sum games, for convenience, I assume player 1 is minimizing u (called the min player) while player 2 is trying to maximize u (called max player).

2.1.4.2 Minimax/Maximin

For two-player zero-sum games, another two equilibrium notions that are related to but not necessarily the same as Nash equilibrium are minimax and maximin equilibrium.

Definition 2.1.6 (Minimax/Maximin Equilibrium)**.** For a two-player zero-sum game, a profile $s^* = (s_1^*, s_2^*)$ is a minimax equilibrium if $s_1^* = \arg\min_{s_1} u(s_1, s_2^*(s_1))$, where $s_2^*(s_1) = \arg\max_{s_2} u(s_1, s_2)$ The maximin equilibrium is similarly defined, where we just exchange the positions of minimum and maximum operators.

The key to minimax or maximin equilibrium is that due to this nested structure, the decision made by player 2 actually can be dependent on player 1's strategy. This effectively transforms the original game into a new game where the new strategy set of player 1 is $\mathscr{S}_1' = \mathscr{S}_1$ while the new strategy set of player 2 $\mathscr{S}_2' = \{s_2' : \mathscr{S}_1 \to \mathscr{S}_2\}$.

This implicitly defines an order of play: player 1 first takes a move observable by player 2, then player 2 makes a move that takes this information as input.

When such minimax/maximin equilibrium is achievable, we call the corresponding value of u as the minimax/maximin value of the game. Furthermore, for a class of two-player zero-sum games including finite-strategy ones, we have the following results.

Theorem 2.1.7 ([Sion, 1958])**.** *In a two-player zero-sum game where u is upper semicontinuous and quasi-concave on $\mathscr{S}_2$ for every $s_1 \in \mathscr{S}_1$, and lower semicontinuous and quasi-convex on $\mathscr{S}_1$ for every $s_2 \in \mathscr{S}_2$, the minimax value equals to the maximin value of the game. We call this number the value of the game. Furthermore, in this case, the equilibrium path of the minimax equilibrium and the maximin equilibrium is equivalent to Nash equilibrium.*

Here by "equilibrium path" I refer to s_1^* and $s_2^*(s_1^*)$ resulting from a play when everyone follows the equilibrium and the sequential order.

These conclusions also imply a useful property for two-player zero-sum games.

Corollary 2.1.8 (Interchangability of Nash)**.** *For the class of two-player zero-sum games I just considered, for two different Nash equilibria (s_1^*, s_2^*) and (s_1^{**}, s_2^{**}), then (s_1^*, s_2^{**}) and (s_1^{**}, s_2^*) are also Nash equilibria.*

2.1.4.3 Min-Max/Max-Min in Machine Learning

I here provide a side note about two-player zero-sum games. In recent years in the domain of machine learning, especially deep learning, many research problems exhibit a common min-max optimization structure. These problems are effectively two-player zero-sum games with high-dimensional continuous action spaces and non-convex payoffs. Examples include generative adversarial networks (GAN) [Goodfellow et al., 2014] and adversarial machine learning [Vorobeychik and Kantarcioglu, 2018].

2.1.4.4 Algorithms

For finite-strategy two-player zero-sum games, the problem can be formulated as linear programming [Raghavan, 1994]. Another method is to use an iterative regret-minimization algorithm [Schapire and Freund, 2013], whose empirical average strategy across iterations converges to a Nash.

2.1.5 General-Sum Stackelberg Games

2.1.5.1 Definitions

We can generalize the two-player nested structure in the previous section into any two-player general-sum games. This class of games is called Stackelberg games [Von Stackelberg, 1952]. In a Stackelberg game, player 1 is called the leader while player 2 is called the follower. In a play of a game, the leader first *commit to* a strategy by making it common knowledge, and then the follower chooses its strategy as a function of player 1's strategy. In general Stackelberg games, the player's payoffs can be general-sum. The corresponding equilibrium notion $(s_1^*, s_2^*(\cdot))$ is called Stackelberg equilibrium.

In fact, we can transform any two-player game into a Stackelberg game, by re-defining the second player's strategy space as $\mathscr{S}_1 \to \mathscr{S}_2$. This allows us to composite a variety of interesting game classes. For example, we can define stochastic Stackelberg games [Vorobeychik and Singh, 2012, Letchford et al., 2012] by allowing the second player's policy in a Markov game (detailed in Section 2.2.2) to depend on the first player's, which induces an intertemporal interaction. A similar combination with extensive-form games (detailed in Section 2.2.4.2) was presented in [Letchford and Conitzer, 2010]. Pay attention to that if the leader's strategy is a mixed strategy, then normally the follower's strategy is *a function of this mixed strategy*, instead of *a function of the stochastic realization of this mixed strategy* [Conitzer, 2016]. Normally the players may achieve a higher payoff in a Stackelberg equilibrium than in a Nash or correlated equilibrium [Von Stengel and Zamir, 2010].

2.1.5.2 Algorithms

For finite-strategy Stackelberg games, a standard algorithm is to formulate it via a set of linear programs [Conitzer and Sandholm, 2006].

2.1.6 Graphical Games

2.1.6.1 Definitions

In this section, I consider a particularly interesting class of structured games called graphical games [Kearns et al., 2001]. In a graphical game, the interaction structure of the game is described by a graphical model, where each vertex of the graph represents a player. Denote $\mathscr{N}(n)$ as player n's neighborhoods (including n itself). Then player n's utility only depends on the strategies chosen by players in $\mathscr{N}(n)$. Graphical games are also called network games [Jackson, 2010] studied by sociologists.

Graphical games capture the strategic scenarios of many real-world applications. For finite-strategy games, it also reduces the representational complexity of the game matrix from $O(NM^N)$ to $O(NM^\kappa)$, where N and M are the numbers of players and strategies, and κ is the maximum size of the neighborhood in the game.

2.1.6.2 Algorithms

Since a graphical game has a succinct representation, an efficient equilibrium computation method that could exploit such a structure was developed. Example includes homotopy method [Blum et al., 2006] and hybrid refinement [Vickrey and Koller, 2002].

2.1.7 (Role-)Symmetric Games

2.1.7.1 Definitions

Another structured class of games is games with player symmetry. More concretely, in a symmetric game [Cheng et al., 2004] everyone shares the same strategy set $\forall n, \mathscr{S}_n = \mathscr{S}_0$. And everyone shares the same utility functional form: $\forall n, u_n = u$. Due to the symmetry of the game, we can find that one's utility effectively only depends on the chosen action and the action distribution of other players. A symmetric Nash equilibrium is an equilibrium profile where everyone adopts the same strategy.

A generalization of symmetric games is role-symmetry games, where agents are partitioned into different roles. Players of the same role are symmetric while players across different roles are asymmetric. The utility of a player depends on its role attribute, its strategy, and the strategy distribution of each role.

A corresponding solution for role-symmetric games is role-symmetric Nash, where agents of the same role choose the same strategy.

2.1.7.2 Algorithms

Replicator dynamics [Cheng et al., 2004] is an algorithm originated with an updated rule from evolutionary game theory (detailed in Section 2.1.8) that solves for a symmetric Nash. It generally follows the update rule $\dot{\sigma}(s) = \sigma(s)(u(s,\sigma) - u(\sigma,\sigma))$. Here $u(s,\sigma)$ is the pure strategy payoff of s while the others are choosing a mixed strategy σ, and $u(\sigma)$ is the expected payoff when everyone is choosing σ. The idea is that it should increase the probability of choosing s when its payoff is significantly greater than the average payoff of the population. Notice that generally there is no last-iteration guarantee for replicator dynamics. But it has been shown by Hofbauer et al. [2009] that replicator dynamics has average-iteration convergence in special cases like two-player zero-sum games and potential games.

Function minimization [McKelvey and McLennan, 1996] is another method that directly optimizes an objective function where the global minimum corresponds to a Nash.

2.1.7.3 Action-graph games

Action-graph games (AGG) [Jiang et al., 2011] representation further incorporates *action dependency* into symmetric games. Specifically, in an AGG there is a graphical model where each vertex corresponds to an action. Then players choosing a particular action depends on the numbers of players choosing this action and all actions that are neighbors to this action. Efficient equilibrium computation methods were also devised for AGGs [Jiang et al., 2011].

2.1.7.4 Two-Player, Two-Strategy Symmetric Games

Being perhaps the most basic version of symmetric games, two-player two-strategy symmetric games can be represented by the following payoff tensor

$$\begin{bmatrix} R,R & S,T \\ T,S & P,P \end{bmatrix}.$$

Here the entry (a,b) at the (i,j)-th entry means the resulting payoffs when the row player plays i and the column player plays j is a for the row player and b for the column player. Different relative orders among R, S, T, P render different strategic properties of the game. W.l.o.g. I assume $R > P$. Here are some of the games:

1. If $T > R$ and $S > P$, $(S, T), (T, S)$ are the only equilibria. Furthermore, if $T > R > S \geqslant P$ then it is called Hawk–dove. The only stable outcome emerges when one chooses T (Hawk) and another S (dove).

2. If $R > T$ and $P > S$, then $(R, R), (P, P)$ are the only equilibrium. Furthermore, $R > T \geqslant P > S$, then it is called Stag hunt [Skyrms, 2004]. Here (R, R) is called the Stag equilibrium, and (P, P) is called the Hare equilibrium. These are the only equilibria of the game. The meaning of this game is that if players can cooperate they can achieve a greater stable outcome.

3. $R > T$ and $S > P$: (R, R) the only Nash. Furthermore when $T > R > P > S$: it is called prisoners' dilemma [Axelrod and Hamilton, 1981]. The meaning of this game is that the only stable rational outcome (R, R) is Pareto dominated by (P, P), which suggests cooperation is not rational. But this may also be changed in, for example, a repeated game version of the game [Axelrod and Hamilton, 1981] (more in Section 2.4.6.1).

Although the previous game-theoretic properties are analyzed in finite-strategy games, a similar structure can be detected in much more complex games such as multiagent RL [Leibo et al., 2017].

2.1.8 Evolutionary Game Theory

Besides the mainstream game theory studied by economists, there is also another branch called evolutionary game theory (EGT) [Smith, 1982] that is more exposed to biologists, physicists, and sociologists. EGT adopts the idea of Darwin's natural selection to study interactions among strategies with individual anonymity. The interactions are usually represented by some continuous differential dynamics. In an EGT narrative, each strategy is regarded as a gene, and a distribution of strategies among anonymous players (usually represented as a mixed strategy) is called a *population*. Due to player anonymity, EGT essentially considers role-symmetric games (in Section 2.1.7), or in an infinite-player limit called K-population game [Sandholm, 2010]. For single-population games, another formulation is called the *random matching* process: each time an evolutionary game is constructed by randomly matching two strategies in the population according to the current population mixture and making it a two-player symmetric game.

The payoff of an individual choosing a certain strategy against a certain population state (a role-symmetric mixed strategy of other players) is called its *fitness*, and strategies of low fitness values are expected to be *extinct*, i.e., occupy lower and lower mass in a solution through the dynamics. Therefore there are many connections between EGT and role-symmetric games, where many methods transfer between these two domains. For economists and computer scientists, it is the tools where EGT analyzes the strategic interactions that are really insightful, including the dynamics and equilibrium notions [Friedman, 1991]. We refer to the survey by Bloembergen et al. [2015] for a comprehensive overview of EGT and AI in general.

2.1.8.1 Evolutionary Stable Strategies

A solution concept originated from EGT is evolutionary stable strategies (ESS), which can be regarded as a refinement of Nash. Suppose now the evolutionary game is represented by $u_k(s_k, \boldsymbol{\sigma})$ which is the payoff (fitness) of strategy s_k of population k when the whole population is in the state $\boldsymbol{\sigma}$. The state $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K)$ is often represented by the role-symmetric mixed strategies [1] adopted by all K populations.

A population state $\boldsymbol{\sigma}$ is an ESS, if $\forall k, \forall s_k \in \textsc{Supp}(\sigma_k), \forall s'_k \notin \textsc{Supp}(\sigma_k)$, if (1) $u_k(s_k, \boldsymbol{\sigma}) > u_k(s'_k, \boldsymbol{\sigma})$ or (2) $u_k(s_k, \boldsymbol{\sigma}) = u_k(s'_k, \boldsymbol{\sigma})$ and $u_k(s_k, (1-\epsilon)\boldsymbol{\sigma} + \epsilon\boldsymbol{\sigma}|_{\sigma_k \to s'_k}) > u_k(s'_k, (1-\epsilon)\boldsymbol{\sigma} + \epsilon\boldsymbol{\sigma}|_{\sigma_k \to s'_k})$, for $\epsilon > 0$ sufficiently small. Here $\textsc{Supp}(\sigma_k)$ is the support of σ_k. $(1-\epsilon)x + \epsilon y$ means we choose strategy x with probability $(1-\epsilon)$ otherwise y. $\boldsymbol{\sigma}|_{\sigma_k \to s'_k}$ means the σ_k component of $\boldsymbol{\sigma}$ is replaced by the pure strategy s'_k. The interpretation of ESS is as follows. Condition (1) and the first equation of condition (2) are Nash conditions. For the second part of the condition (2), it basically means the mutant s'_k is just doing less well in a new polluted environment created by their invasion with an ϵ probability.

Some interesting properties of ESS [Gintis, 2009] in two-player games: (1) there cannot be two ESS-s with exactly the same support, or one contains the other. (2) finite games can only have a finite number of ESS (3) if there is a completely mixed ESS, then it is the unique Nash of the game. For common-payoff symmetry games, there are additional properties such as the mean payoff increases monotonically in replicator dynamics. Interestingly, many of these properties cannot be extended to symmetric games with more players [Broom et al., 1997, Bukowski and Miekisz, 2004, Plan, 2023].

For a dynamics F satisfying certain regularity (such as it always boosts strategy with higher fitness values), it can be shown that generally [Friedman, 1991] $\textit{ESS}(u) \subseteq \textit{LASP}(F) \subseteq \textit{NE}(u) \subseteq \textit{FIX}(F)$. Where $\textit{ESS}(u), \textit{NE}(u)$ are the sets of ESS, NE of the game described by u, while $\textit{LASP}(F), \textit{FIX}(F)$ are the sets of locally asymptotically stable fixed points and fixed points of F.

[1] However generally in a K-population game u_k does not have to be multilinear w.r.p. to $\boldsymbol{\sigma}$, as was supposed to be true for a finite-player K-role symmetric game.

2.2 Decision Making under Sequentiality and Uncertainty

2.2.1 Markovian Decision Process and Reinforcement Learning

Markovian Decision Process (MDP) is a standard model for sequential decision-making and will serve as a building block for my development of multiagent decision-making theories. MDP essentially models an interaction process between an agent and an environment. I consider discrete-time MDP, where an episode terminates in a finite or countable number of timesteps.

More concretely, an MDP consists of $(\mathcal{S}, \mathscr{A}, \mathcal{P}, \gamma, r)$. Here $\mathcal{S}$ is the set of environment states, $\mathscr{A}(s)$ returns the set of actions available at state s. $\mathcal{P}(s' \mid a, s)$ is the transition kernel, which is the probability of transiting from state s to s' in the next timestep if the agent takes action a in the current timestep. $r(s, a, s') \in \mathbb{R}$ is the corresponding intermediate reward the agent receives after taking the action.

A policy $\pi : \mathcal{S}^* \to \Delta(\mathscr{A})$ specifies a randomized selection over actions given arbitrarily histories. Here $\mathcal{S}^* = \bigcup_{n \in \mathbb{N}} \bigtimes_{t=0}^{n} \mathcal{S}$ is the set of all possible state history. I denote the space of all such policies as $\mathscr{S}$. I usually consider a policy $\pi : \mathcal{S} \to \Delta(\mathscr{A})$ only depends on the current state, which we call a stationary policy.

Given an agent policy π, coupled with $\mathcal{P}$ and a distribution over the initial states $\mathcal{P}_0$, the interaction between the agent and the environment renders a distribution over the trajectory $\tau = (s_0, a_0, s_1, a_1, s_2, \dots)$ as $\mathcal{P}(\tau) = \mathcal{P}_0(s_0) \prod_{t=0}^{\infty} \pi(a_t \mid s_t) \mathcal{P}(s_{t+1} \mid s_t, a_t)$. [2] The corresponding accumulated return is $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})$. Where here $\gamma < 1$ is a discount factor. There are two possible interpretations of γ: (1) it indicates a setting where the trajectory "could have terminated" in each timestep with probability $(1 - \gamma)$ [Shapley, 1953], and (2) it is reflected by an interest rate and a deflation rate notion from economics where immediate rewards are valued more than the future ones. Nevertheless, in practice, γ is often selected as a model hyperparameter instead of reflecting some real-world notions. For example, people may adopt different γ-s for training and test time just as a means for regularization [Jiang et al., 2015].

The goal of the agent is to select a policy such that the expected accumulated return $\mathbb{E}_\tau R(\tau)$ is maximized. The policy that maximizes this objective is called the optimal policy.

Given a policy π, it is helpful to have two mathematical objects to analyze an MDP. The state value function $V_\pi(s) : \mathcal{S} \to \mathbb{R}$ returns the expected accumulated return of applying π from a state onward. The state-action value function $Q_\pi(s, a) : \mathcal{S} \times \mathscr{A} \to \mathbb{R}$ returns the expected accumulated return by first applying an action a in a state s, and then applying π onward in the next timestep. Both V and Q effectively evaluate the advantages of being in a state s or applying an action a in s

[2]This is called infinite-horizon discounted-MDP. Another formulation assumes a trajectory ends in a finite number of rounds T and the agent is trying to maximize $\sum_{t=0}^{T} r(s_t, a_t, s_{t+1})$. These are different choices of models. I adopt this one for convenience.

to reach a higher return.

If the agent knows the environment models $\mathcal{P}, r$, then solving for an optimal policy is called planning. If the agent does not know such information and only has black-box access to the environmental interfaces (i.e., the agent only has stochastic access to data (s, a, s', r)), then solving for an optimal policy is called reinforcement learning [3].

2.2.1.1 Value Iteration and Policy Iteration

For discounted MDP processes, if $\mathcal{P}$ and r are given, there are two seminal algorithms called value iteration and policy iteration for solving optimal policies. In value iteration, the algorithms maintain a V table and iteratively update it via the following equation

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}(s' \mid s, a)(r(s, a, s') + \gamma V^i(s')) \tag{2.1}$$

where i indexes the iteration number.

By contrast, the policy iteration (PI) paradigm maintains two mathematical objects that one iteratively updates itself and improves upon the other. The vanilla version of PI consists of two components policy evaluation and policy improvement.

The policy evaluation operator can be regarded as a function *PE* : $\mathscr{S} \rightarrow \mathbb{R}^{|\mathcal{S}|}$(or $\mathbb{R}^{|\mathcal{S}||\mathscr{A}|}$) that returns an estimation of the state(-action) value function for a given policy π.

One standard iterative algorithm that achieves this purpose is via the following update equation:

$$Q_\pi^{i+1}(s, a) \leftarrow \sum_{s'} \mathcal{P}(s' \mid s, a)(r(s, a, s') + \gamma \sum_{a'} \pi(a' \mid s') Q_\pi^i(s', a')) \tag{2.2}$$

until convergence.

Given the output of the policy evaluation, policy improvement finds an improvement direction on the policy. Following the vanilla PE procedure we had above, after we get Q_π for a given policy π, the corresponding policy improvement step is

$$\pi(s) \leftarrow \max_a Q_\pi(s, a) \tag{2.3}$$

which selects the action that maximizes the state-action value in each state. Then the whole PI procedure starts from an arbitrary policy and iterates policy evaluation and policy improvement until the policy converges.

[3]Another way to think of RL as a machine learning category is that in RL, the hypothesis being the policy affects the data distribution. By contrast, the data distributions are always fixed in supervised and unsupervised learning. That makes RL, in some sense, much more challenging, given the useful data are only (s, a, s', r).

2.2.1.2 Q-Learning

If the environment setting is unknown, the agent can only learn an optimal policy through interaction data (s, a, s', r).

One of the most famous RL algorithms is Q-learning. In Q-learning, the agent maintains a Q-table and uses some exploration policy (e.g., ϵ-greedy) for interaction with the environment. Upon receiving a data (s, a, s', r), an Q-learning agent uses the following update rule:

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t \left(r + \gamma \max_{a'} Q(s', a') \right) \tag{2.4}$$

where α_t is the learning rate in the current timestep t.

2.2.1.3 Challenges of Reinforcement Learning

I here highlight the critical challenges of RL, some of which are recurring and representative topics in other domains of AI.

The first is called exploitation v.s. exploration tradeoff. Since the agent wants to find the optimal policy, it may want to take effective actions in order to visit those state space regions that it thinks can lead to a high return. This is called *exploitation*, which is to select actions that are optimal given the current knowledge of the world. However, given the limited knowledge of the environment, the agent needs to explore unvisited states to gather useful information, avoiding falling into local optima. One of the technical conditions to guarantee convergence in Q-learning is to ensure the behavior policy visits every state infinitely often. There will be conflicts about exploitation and exploration in most cases, given a limited computational budget.

The second is called credit assignment. In many applications the reward signals are sparse or delayed (e.g., in the Game of Go the only reward is the outcome). This may cause trouble in designing an agent that can do efficient exploration. An agent should attribute proper credits over the states and actions to guide its intermediate moves during the learning process. One of these techniques is to employ *intrinsic reward* [Chentanez et al., 2004] techniques. We refer to Pignatelli et al. [2023] for more discussion on this topic.

The third is called the approximation-estimation error tradeoff. This is sometimes formulated as the bias-variance tradeoff in a general machine learning context. An approximation error generally refers to the distance between the ground-truth optimal model and the best model within the model class that the learning algorithm is considering. For example, I can define a time-delayed version of Q-learning where each time it makes the target as $(r + \gamma r' + \gamma^2 \max_{a''} Q(s'', a''))$. Here r and r' are rewards of the next two timesteps and s'' is the second-next state from s. Theoretically, this target is expected to be closer to the ground-truth optimal Q value in expectation compared with the vanilla target $r + \gamma \max_{a'} Q(s', a')$, because the biases introduced by the Q tables are discounted more. However, this target may exhibit higher variance due to the extra random variable r'. This reflects the estimation error, which generally refers to the hardness of reaching the optimal model within the model class being considered. It is also generally impractical to achieve both low approximation error and low estimation error at the same time. For RL settings, however, people usually are more concerned with high estimation error than high approximation error.

The last challenge I would like to briefly mention is the generalization of RL algorithms. This generally refers to the adaptive capability of an RL agent to diverse test environments. Several approaches to this issue are transfer RL, multi-task RL, and meta-RL. Typically the new test environments may have different state transitions $\mathcal{P}$ but may have the same reward structure.

2.2.2 Markov Games and Multiagent Reinforcement Learning

I here lay out the basics of Markov games and multi-agent reinforcement learning. For a more comprehensive overview of MARL, please refer to the excellent book by [Albrecht et al., 2024].

Markov games (MG, or Stochastic games) generalize MDP into multiagent settings. Specifically, an N-player Markov game consists of $(\mathscr{N}, \mathcal{S}, \mathscr{A}, \mathcal{P}, \gamma, r)$. Here $\mathscr{N} = \{1, 2, \ldots, N\}$ is the set of agents. Each play of the game consists of multiple timesteps. Starting from an initial state of the world s_0 (which may be sampled from an initial distribution), in each timestep t agent n having observed the state s_t chooses an action $a_{t,n} \in \mathscr{A}_n(s_t)$, and the state of the environment transits according to $\mathcal{P}(s_{t+1} \mid, \boldsymbol{a}_t, s_t)$. Then agent n receives its immediate reward $r_n(s_t, \boldsymbol{a}_t, s_{t+1})$.

If the joint actions and states of past timesteps are observable, then the policy of agent n is $\pi_n : (\mathcal{S} \times \mathscr{A})^* \times \mathcal{S} \to \Delta(\mathscr{A}_n)$ which is a mapping from state-action history to a probability distribution over actions. A stationary policy is a policy that only depends on the current state.

Given joint policies $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$ interacting with the environment, we can also define the probability of having a trajectory $\tau = (s_0, \boldsymbol{a}_0, s_1, \boldsymbol{a}_1, s_2, \ldots)$ as

$$\mathcal{P}(\tau) = \mathcal{P}(s_0) \prod_{t=0}^{\infty} \left(\prod_{n=1}^{N} \pi_n(a_{n,t} \mid s_t) \right) \mathcal{P}(s_{t+1} \mid s_t, \boldsymbol{a}_t).$$

The corresponding accumulated return for agent n is $R_n(\tau) = \sum_{t=0}^{\infty} \gamma^t r_n(s_t, a_t, s_{t+1})$. Then actually, we can define the utility of agent n in a Markov game as the expected accumulated return $\mathbb{E}_\tau R_n(\tau)$. Then the game-theoretic concepts I have in Section 2.1 naturally transfer, where now the strategy spaces are spaces of policies.

In a Markov game, I also can define notions of state value function and state-action value function. The value function of agent n given a joint policy $\boldsymbol{\pi}$ is $V_{n,\boldsymbol{\pi}}(s) : \mathcal{S} \to \mathbb{R}$, returns the expected accumulated return of agent n assuming all agents follow the joint policy $\boldsymbol{\pi}$ onward. The Q table for agent n given a joint policy $\boldsymbol{\pi}$ is $Q_{n,\boldsymbol{\pi}}(s, \boldsymbol{a}) : \mathcal{S} \times \mathscr{A} \to \mathbb{R}$ records the expected return of agent n if the agents first choose the joint action $\boldsymbol{a}$ in state s, and then everyone follows the joint policy $\boldsymbol{\pi}$ from the next state.

We can define Nash policies as follows.

Definition 2.2.1 (Nash equilibrium in Markov games)**.** A joint policies $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ is a Nash equilibrium if, $\forall n, s_0, \pi'_n$, $V_{n,\boldsymbol{\pi}}(s_0) \geqslant V_{n,\pi'_n,\boldsymbol{\pi}_{-n}}(s_0)$

The existence of a stationary Nash policy in stochastic games is proved in [Filar and Vrieze, 1997].

Also, if the agents are aware of all the environmental information (including transition functions and other players' rewards), then the problem of solving a Nash policy can be called multiagent planning. Otherwise, it is called multiagent reinforcement learning.

2.2.2.1 Nash Value (Policy) Iteration & Nash-Q Learning

The generalization of value iteration and Q-learning to multiagent settings is straightforward. The main change is to replace the one-step $\max$-operator in the updating equation with a Nash operator.

More concretely, for multiagent planning, the Nash iteration process is

$$\forall n, Q_n^{i+1}(s, \boldsymbol{a}) \leftarrow \sum_{s'} \mathcal{P}(s' \mid s, \boldsymbol{a})(r_n(s, \boldsymbol{a}, s') + \gamma \mathbb{E}_{\boldsymbol{a}' \sim \sigma(s')}[Q_n^i(s', \boldsymbol{a}')]). \tag{2.5}$$

Where $\sigma(s')$ is the Nash equilibrium of the matrix game defined by the Q-tables $Q_1^i(s', \cdot), \ldots, Q_N^i(s', \cdot)$. The Nash policy iteration can be similarly derived.

In Nash-Q [Hu and Wellman, 1998], upon receiving an data point $(s, \boldsymbol{a}, s', \boldsymbol{r})$, it follows the update rule:

$$\forall n, Q_n(s, \boldsymbol{a}) \leftarrow (1 - \alpha_t) Q_n(s, \boldsymbol{a}) + \alpha_t(r_n + \gamma \mathbb{E}_{\boldsymbol{a}' \sim \sigma(s')}[Q_n(s', \boldsymbol{a}')]) \tag{2.6}$$

Where $\sigma(s')$ is again the Nash equilibrium of the matrix game defined by the current Q-tables $Q_1(s', \cdot), \ldots, Q_N(s', \cdot)$. Here the problem is to guarantee convergence we need to make sure every agent chooses the same equilibrium (with some additional required property of this equilibrium). One possible implementation is to create a centralized controller to compute the Q-tables and the Nash and then distribute the equilibrium outcome back to the agents. But the experiments conducted in [Hu and Wellman, 2003] suggest a decentralized implementation could also be effective.

2.2.3 Partially Observable Environments

In single-agent MDPs or multiagent MGs I defined above, I assumed the environments are fully-observable. I.e., the agents can access the full knowledge of the states or states and the previous actions of the other agents. However, in many applications, this information can be partially-observable. I.e., the agents can only observe some information that is only correlated with the actual state. I introduce several standard partially observable environments in this section.

2.2.3.1 Partially Observable Markovian Decision Process

In the partially observable Markovian decision process (POMDP), there exists an observation space $\mathscr{O}$. In every timestep t instead of the actual state s_t, the agent observes an observation o_t according to $\mathcal{P}(o_t \mid s_t)$. Then the agent's policy is generally $\pi : (\mathscr{O})^* \to \Delta(\mathscr{A})$. The agent's goal is still to maximize the accumulated reward by selecting a proper policy.

A common approach to solving POMDP is to transform a POMDP into an MDP by adopting a Bayesian viewpoint. The idea is to resort to the *belief state representation*. During a trajectory, the agent will maintain a probability distribution over the state space as its belief state. After receiving an observation, the belief state transits according to the Bayes' rule using the transition kernels. More formally, the agent maintains a belief $b \in \Delta(\mathcal{S})$. After taking action a_t at timestep t, the agents observes o_{t+1}, and update its belief state as $b_{t+1} \propto b_t \otimes \mathcal{P}(o_{t+1} \mid \cdot, a_t)$. The agent policy is $\pi : \Delta(\mathcal{S}) \to \Delta(\mathscr{A})$. Therefore we transform the original POMDP to an MDP where the new state space is a distribution over the original POMDP state space. Planning algorithms [Kaelbling et al., 1998] and learning algorithm were developed [Jaakkola et al., 1994] for POMDP respectively.

2.2.3.2 Dec-POMDP

The first class of multiagent partially observable environments I would like to introduce is decentralized-POMDP (Dec-POMDP). In an N-player Dec-POMDP environment, at timestep t agent n selects its action $a_{n,t}$, and the state transits according to $\mathcal{P}(s_{t+1} \mid s_t, \boldsymbol{a}_t)$. Then the system generates an observation $\boldsymbol{o}_{t+1}$ according to $\mathcal{P}(\boldsymbol{o}_{t+1} \mid s_{t+1})$. The observation $\boldsymbol{o}_t$ can be further factorized as components $o_{1,t}, o_{2,t}, \ldots, o_{N,t}$, where $o_{n,t}$ is the observation that only agent n receives privately. Denote the set of observations of agent n can receive as $\mathscr{O}_n$, then agent n's policy is $\pi_n : \mathscr{O}_n^* \to \Delta(\mathscr{A}_n)$. Furthermore, all agents receive the same immediate reward value $r(s_t, a_t, s_{t+1})$. Then the goal of every agent is to coordinately find a joint policies $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$, such that the shared accumulated reward $\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})$ is maximized. Since the agents share the same utility value, a Dec-POMDP can be viewed as a collaborative game. I refer to book [Oliehoek and Amato, 2016] as an excellent survey for Dec-POMDP.

2.2.3.3 Factorized-Observation Stochastic Games

Factorized-observation stochastic games (FOSG) [Kovařík et al., 2022] is a recent popular multi-agent model that generalizes Markov games and Dec-POMDP in two senses: (1) the observation

$\boldsymbol{o}_t$ is factorized into both private components $(o_{1,t}, o_{2,t}, \ldots, o_{N,t})$ and a public component $o_{0,t}$. Here $o_{0,t}$ is an observation that is public to all agents, and this is common knowledge. Agent n only receives $(o_{0,t}, o_{n,t})$ at timestep t. (2) Agent n receives $r_n(s_t, \boldsymbol{a}_t, s_{t+1})$ in timestep t whose value does not necessarily equal to other agents' rewards. So these entail that agent n's policy is $\pi_n : (\mathscr{O}_n \times \mathscr{O}_0)^* \to \Delta(\mathscr{A}_n)$, where $\mathscr{O}_n$ is the set of private observations of agent n, $\mathscr{O}_0$ is the set of public observations and $\mathscr{A}_n$ is the set of actions of agent n. FOSG is the most general and structured multiagent model to date that can capture a range of applications, including Poker. If $\forall n, r_n = r$, then FOSG is equivalent to Dec-POMDP with public observations.

Just like we can transform a POMDP into an MDP, we can transform a FOSG into a fully-observable Markov game by adopting a belief representation. I here briefly summarize the key ideas of the transformation, based on [Nayyar et al., 2013, Foerster et al., 2019]. We define a public belief state $B \in \mathscr{O}_0 \times_{n=1}^{N} \Delta(\mathscr{O}_n)$, which consists of the public state and the distributions over the agents' private observations. Then we construct a new fully-observable Markov game where the state space is the space of public belief states, and the action space for agent n is all mappings of $A_n : \mathscr{O}_n \to \mathscr{A}_n$. We here assume the action $A_{t,n}$ taken at timestep t is common knowledge after timestep t, which is more or less assumed in previous literature [Brown et al., 2020]. After taking the joint action $\boldsymbol{A}_t$, the public belief state transits according to Bayes' rule $B_{t+1} \propto \left(\prod_{n=1}^{N} A_n(a_{t,n} \mid \cdot)\right) \otimes B_t$. The reason is that given the agents' current one-step strategies $\boldsymbol{A}$, it actually can gather more information about each agent's private information based on the public observations.

2.2.4 Information Perfectness and Extensive-Form Games

In AI literature, POMDP, Dec-POMDP, and FOSG all aim to capture the notion of observability of the environment state from the agents' viewpoint. In economics and game theory literature, there are several similar notions about the information structure of the multiagent decision-making process, including information completeness and perfectness. However, there has been confusion between the terminologies used in these two domains. Next, I clarify these notions and the relationships between them.

2.2.4.1 Observability and Information Perfectness

In AI literature, full observability means the state of the environment is common knowledge to all agents. Partial observability of an agent means this agent cannot access the true state of the environment but can only infer it through some observable statics that are correlated with it. In game-theoretic literature, however, researchers are more interested in the information structure within the agents' strategic decision processes. Perfect information means anytime an agent is going to make a decision, it is informed of any events that happened before this decision point. Such events include the actions chosen by other players and the stochastic outcomes of the environments (where some people call the non-player part of the environments the "nature player"). Imperfect information means the opposite: some players may not be aware of some predecessor events when they make decisions. Information completeness is a special case of information perfectness: it refers to whether a full characterization of opponents' utilities is known. This includes the utility functional form, parameters, as well as strategy set descriptions. For example, a standard class of games with incomplete information is Bayesian games, where before the game starts, each agent privately receives a private signal called its *type* that serves as a parameter in its utility function.

Here are some conclusions about the above notions. Fully-observable environments do not necessarily imply perfect information. One example is a one-shot simultaneous-move game. Perfect information implies fully-observablility. The contrapositives of the above conclusions also follow directly.

2.2.4.2 Extensive-Form Games

Now I am ready to introduce the final class of game representation I am interested in this work. An extensive-form game representation of an N-agent environment consists of

$$(\mathscr{N}, \mathscr{A}, \mathscr{I}, \mathscr{H}, \pi_0, \mathcal{Z}, \{u_n \mid, n \in \mathscr{N}\}).$$

Here $\mathscr{N} = \{0, 1, \dots N-1, N\}$ is the set of players. Here we augment a special "natural player" indexed by 0 in the player set, whose strategy (defined later) is fixed in the system. Here $\mathscr{A} = \times_{n=0}^{N} \mathscr{A}_n$ and $\mathscr{I} = \times_{n=1}^{N} \mathscr{I}_n$, where $\mathscr{A}_n$ and $\mathscr{I}_n$ are the spaces of actions and decision points of agent n, respectively. The game begins with a decision point of player $n_0 \in \mathscr{N}$. Upon reaching a decision point $i_n \in \mathscr{I}_n$ of agent n, agent n needs to take an action $a_n \in \mathscr{A}_n(i_n)$, where $\mathscr{A}_n(i_n)$ is the set of actions available at decision point i_n. Then the decision point transits deterministically according to $\mathscr{H}(i_n, a_n) = i_{n'}$, and then it is agent n''s decision point. Or $\mathscr{H}(i_n, a_n) = z \in \mathcal{Z}$ reaches a terminal point. When this happens, for every agent n it receives a utility value $u_n : \mathcal{Z} \to \mathbb{R}$. Therefore an EFG is graphically represented as a tree where each vertex corresponds to a decision point while each action is mapped to an edge.

I further elaborate on the expressiveness of EFG in terms of modeling the information structure. Upon reaching a decision point i_n, there is a world history trajectory $\tau(i_n) = (a_1, a_2, \dots)$ which records every action that has been played prior to i_n. Here the subscripts index the order of actions played before i_n. Let $\mathcal{T}(i_n)$ be the set of all such possible world trajectories $\tau(i_n)$, and let $\tau^*(i_n)$ be the actual world state during a play. However, player n may only observe a subsequence of τ. I denote it as $\tau_n(i_n) = (a_{n,1}, a_{n,2}, \dots)$. This $\tau_n(i_n)$ is called the information state of i_n. The model constrains that i_n has a one-to-one correspondence to $\tau_n(i_n)$. Furthermore i_n also has a one-to-one correspondence to a mathematical object called the information set $\{\tau \mid \tau_n(i_n) \subseteq \tau, \tau \in \mathcal{T}(i_n)\}$. That is, the set of possible world histories that are consistent with $\tau_n(i_n)$. For convenience I let i_n denote a decision point, $\tau_n(i_n)$, and its corresponding information set. I let $[\tau]_n$ denote the maximal subsequence of τ consisting of only agent n's actions. Then an EFG is called of perfect recall if $\forall n, i_n \in \mathscr{I}_n, \tau, \tau' \in i_n$, $[\tau]_n = [\tau']_n = [\tau^*(i_n)]_n$ during any play of the game.

Then a player n's pure strategy is $s_n : \mathscr{I}_n \to \mathscr{A}_n$. Its policy or behavioral strategy is $\pi_n : \mathscr{I}_n \to \Delta(\mathscr{A}_n)$. For the natural player 0, it will always be playing a fixed policy π_0. Therefore the natural player can model any stochastic events in the environment.

Compared with models such as FOSG in AI, EFG representation has the following features. The first thing is about the information structure. EFG sequentializes a player order and focuses on the information available of an *acting player* by representing them as observable action trajectories. For EFG of imperfect recall, an information state can be recurring for a player, but generally, this is not true for many sequential games of interest. By contrast, FOSG typically assumes a simultaneous move of the agents at each timestep (though it is straightforward to extend this basic model), and each environment state in principle can be revisited any number of times. FOSG further separates public and private observations, even for those who are not acting. The second difference is about the reward structure. EFG only attributes a utility at the terminal point. FOSG, however, generally can have an intermediate reward structure at any information state.

Bayesian Games Bayesian games [Harsanyi, 1967] is a special class of games where the imperfect information part mainly comes from players' private parameters in their utility functions, or their *types*. The EFG representation of a Bayesian game starts with the natural player. And in each of the following information sets the corresponding acting player can not distinguish its opponents' types. The FOSG representation of a Bayesian game is more straightforward, where now one's individual private observation at the first timestep is just its own type.

2.2.4.3 Subgames

The notion of subgames aims to capture the decomposability of a game model. A subgame normally only captures a partial component of the full game. E.g., some players are restricted to play only a subset of the original strategies space. For sequential games, a subgame typically refers to a game starting at an intermediate decision point of the original game. The classical definition [Selten, 1988, Mas-Colell et al., 1995] of a (proper) subgame in an EFG constraints that it begins with a root information set containing only one possible history. However, people later develop some generalizations [Kreps and Wilson, 1982b, Burch et al., 2014] of EFG subgames in imperfect information games, which is a forest-like data structure being a closure under ancestor relation and information set membership relation. Analogously, in a Markov game or in a FOSG, a subgame means a FOSG begins with some intermediate state of the original game [Maskin and Tirole, 2001]. In general, a subgame notion should be developed as an auxiliary tool to facilitate solution computation.

Next, I briefly introduce three types of tree-search algorithms operated on EFGs of different classes. The unified ideas of these algorithms are: (1) by exploiting the structure of the tree, the computation can be implemented by traverses over the tree (2) by reducing the computational complexity by decomposing the computation into subgames.

2.2.4.4 Min-Max Search

Min-max search with its pruning components [Knuth and Moore, 1975] is designed for turn-based, two-player zero-sum games of perfect information. The idea is we can recursively compute the value of a subgame by propagating the game values of its children, assuming both players are playing optimally. To be more specific, we maintain a game value at each decision point of the tree, which is the payoff of the min-player, assuming each player is playing optimally from that point onward. Starting with the terminal nodes, we back-propagate the values by letting the current acting player always choose the action that leads to the optimal payoff for itself, using the computed values below. This method, in principle, works for any turn-based two-player zero-sum game, including chess and Go. However, due to the large branching factors and depths in many games, people developed techniques such as pruning a subtree by maintaining bounds or conducting a depth-limited search by replacing vertices several steps ahead with some heuristic values. Typically min-max search is conducted completely *offline*, i.e., all the computation is finished before being deployed into real play. The outcome of the min-max search is generally an equilibrium strategy for both players.

2.2.4.5 Monte Carlo Tree Search

For games with large branch factors, a min-max search may be inefficient and it is hard to produce an accurate value estimation. Monte Carlo tree search (MCTS) is another class of tree search algorithms that leverage the power of Monte Carlo simulation to identify the important region of the game tree.

MCTS is typically implemented as an *online* algorithm, which means the strategies are computed incrementally during game-play. Upon reaching a game state, MCTS conducts multiple simulations on the game tree rooted at the current game state. MCTS maintains statistics on the set of *expanded nodes*, which constitutes a *search tree* consisting of decision nodes that were simulated a confident number of times. One of these statistics choice is the UCB score [Kocsis and

Szepesvári, 2006] $Q(s) + C\sqrt{\frac{\log n_{PA}(s)}{n(s)}}$, where $n(s)$ is the number of simulations that were conducted at state s, $PA(s)$ being the parent node of s. Each iteration of the MCTS search consists of four phases: selection, expansion, simulation, and back-propagation. The *selection* phase identifies a path from the root state to a leaf node in the search tree by some criterion, e.g., by always choosing the maximum UCB score. Upon reaching such a leaf node, this node is *expanded* and some of its children are included in the search tree. Then a Monte Carlo simulation is conducted starting from this subgame. The purpose of this *simulation* phase is to have an estimation of the optimal state value efficiently. One way to conduct such a simulation is to employ random policies for both players from this state onward. Then the outcome of this simulation is *back-propagated* to all nodes lying on the path from this newly expanded node to the root. By conducting this four-phase procedure, we may finally come up with a search tree with informative statistics being maintained. Then to select the next move in the real-play, MCTS just outputs an action based on the statistics. For example, choose an action that was simulated most often during the simulation.

The key of MCTS is that given a limited computational budget, the algorithm gradually allocates its computation within regions that are more appealing in terms of finding an optimal play. MCTS is thus suitable for handling games with large branch factors. One of the disadvantages of MCTS is that it is hard to deal with trap states [Browne et al., 2012]. The outcome of MCTS is typically an approximate equilibrium for both players in two-player zero-sum games, and a best-response policy against a given opponent strategy in general cases.

2.2.4.6 Counterfactual Regret Minimization

Counterfactual regret minimization (CFR) [Zinkevich et al., 2008] is a class of tree search algorithms for solving imperfect information games and has been shown effective in solving Poker. The vanilla CFR is an offline algorithm and conducts tree search on the entire game tree. Similar to MCTS, in CFR, the algorithm maintains statistics on the nodes of the game tree, where each node is an information set. CFR iteratively updates the statistics and policies resulting from these statistics. The key idea of CFR is that by having these statistics called *counterfactual regret* at every information set, in each iteration, it conducts regret minimization procedures at every node simultaneously, instead of minimizing the full-game regret directly which may need to enumerate every possible strategy. The final output of CFR is the average policy across all these training iterations, which is theoretically an equilibrium policy.

To be more specific, the algorithm maintains *counterfactual regrets* for each (information state, action) pair. Given the current joint policy $\pi = (\pi_n, \pi_{-n})$, the counterfactual value for (i_a, a) is defined as $Q_c^\pi(i_n, a) = \sum_{\tau \in i_n} \sum_{z \in \mathcal{Z}(\tau)} \pi_{-n}(\tau)\pi(z \mid \tau)u_i(z)$. Here $\mathcal{Z}(\tau)$ is the set of all terminal nodes that are reachable from τ. $\pi_{-n}(\tau)$ is the probability of reaching a possible history τ conditioned on that player n will always choose to reach τ. $\pi(z \mid \tau)$ is the probability of reaching a terminal node z from τ, considering every players' contribution to these probabilities. The counterfactual state value of i_n is defined as $V^\pi(i_n) = \sum_a \pi_n(a \mid i_n) Q^\pi(i_n, a)$. Then the instant counterfactual regret for (i_n, a) is $\text{CFR}\textsc{egret}^\pi(i_n, a) = Q^\pi(i_n, a) - V^\pi(i_n)$. Given π, these quantities can be computed by two traverses of the full game tree. Then by having this regret notion, we can actually run any regret minimization procedure at every information set simultaneously in each iteration of CFR. Each of these regret minimization procedures outputs a strategy on the corresponding information set. It has been proved that the summation of this counterfactual regret is an upper bound of the full-game regret. So by minimizing each counterfactual regret separately, we can minimize the full regret of the game.

2.2.4.7 CFR-Decomposition

Both min-max search and MCTS deal with perfect information games. The nice part about perfect information extensive-form games is that it enables *decomposition*, i.e., computing an optimal strategy within a subgame is irrelevant to other subgames. However, this may not hold anymore for imperfect information. The reason is as follows. In imperfect information EFG, upon reaching information set i_n, agent n needs to reason the probabilities of being in each $\tau \in i_n$ to compute a proper game value of this information state. These probabilities depend on how the opponents played before this decision point. Furthermore, assuming the rationality of the opponents, these probabilities may also depend on what the opponents *could have received in other subgames*. Therefore the optimal play of a subgame may depend on information outside that subgame, and the decomposition methods in perfect information games may not be directly transferred.

I here furthermore briefly introduce the CFR-D [Burch et al., 2014] (D standards for decomposition) variant that had served as the core technique for contemporary master-level Poker bots. The goal of CFR-D is to conduct an efficient depth-limit search and improve upon a previous strategy by enabling subgame decomposition in imperfect information games. The application scenario typically starts with a known strategy π^{bp}, called *blueprint strategies* which is normally a strategy with domain knowledge or an approximate equilibrium strategy from an abstracted version of the game. Then suppose starting from the root, we are only interested in the three lookahead actions from the current state, and let's call this part of a full strategy the *trunk strategy*. Our goal is to find a trunk strategy that improves the trunk part of π^{bp}. Just like in perfect information, the way we conduct such depth-limit search is by truncating future game states with value estimates, where in this case, for CFR, they are counterfactual values. To get accurate estimations of these counterfactual values which assume the players are going to play fairly optimally onward, however, we need to *resolve* the subgames. Since a subgame here may be rooted in some information sets, we compute the posterior probability distribution over the actual histories based on the counterfactual probabilities consisting of only the opponents' probability terms. Then to ensure our resolved subgame strategy combined with the trunk strategy is no more exploitable than the blueprint, we need to offer the opponent an "opt-out" option, which guarantees them can at least receive values when the acting player adopts π^{bp}. This is achieved by creating a data structure called *regret gadget* on the root of the subgame, which creates new actions, some of which abstract the counterfactual values of π^{bp}. Then after solving the newly constructed subgame using CFR, the algorithm only retains the counterfactual values on the root information sets. Then it updates the trunk strategies by running one CFR iteration on the trunk part using these resolved counterfactual values.

2.3 Empirical Game-Theoretic Analysis

Upon reaching this section, I have already outlined several basic decision-theoretic frameworks that researchers have been developing over the years. However in many applications, the game analyst may only have black-box access to an agent-based simulation of the strategic scenarios, but neither a detailed configuration of the underlying environment nor an analytical description of a model.

Empirical game-theoretic analysis (EGTA) [Wellman, 2006, Tuyls et al., 2020] is a set of tools and methodologies that researchers developed over the years that provides principled guidelines to conduct strategic reasoning in games with no tractable analytical solution but only black-box simulation data. The game analyst typically can access an interface of the agent-based simulation, and normally it is capable of selecting the inputs to an interface to acquire the simulation data of interest. The simulation data typically reveal partial information about the multiagent scenarios. In the simplest case, the data may just take the form of (agents' strategies, agents' payoffs). Therefore classical game theory and empirical game theory exhibit a dichotomy in expressing multiagent scenarios: classical game theory adopts a declarative approach that gives a full analytical description of the game model, while empirical game theory, in contrast, focuses on the procedural and algorithmic representation of a game by summarizing it in a black-box interface. Classical game theory is prescriptive while EGTA is descriptive.

To better understand EGTA, I first introduce a few terms.

2.3.1 Black-Box Games and Empirical Games

The first two concepts I would like to introduce are black-box games (also termed simulation-based games) and empirical games, which are also easily confused with each other in many problem settings. Black-box games typically refer to the interfaces where the game analyst can access the agent-based simulations. The input-output structure of the interface is known to the analyst, and the analyst typically is free to choose the inputs to the black box. Since the outcome of the simulation could be stochastic, in some formulation (detailed in Section 4.3.3 of Chapter 4), the analyst can also decide on a random seed as an extra input. Therefore, a black-box game is one representation of the ground-truth multiagent scenario one can look into. Algorithms that work directly with this interface can be called *model-free*.

Empirical games, by contrast, are any kind of analytical game model that the analyst builds using simulation data from the black-box game. Therefore in EGTA, the game analyst typically adopts a *model-based* approach by constructing empirical games and developing an algorithm that works more directly on them.

Empirical games are typically built in the form of an NFG by considering only a subset of heuristic strategies of the entire game and estimating the corresponding payoffs using simulation data. These heuristic strategies are typically identified using domain knowledge of the games or a generic method such as deep reinforcement learning.

2.3.2 Empirical Equilibrium Analysis

In EGTA, once an empirical game is built through someway, the most common strategic reasoning method is to apply equilibrium analysis on the empirical game as a means to understand the full strategic landscape of the full game. Typically the way it works is to first solve for some equilibrium notions of the empirical game (the most common one being Nash) and then investigate the behaviors of the players/strategies under these equilibria. One flaw of this approach is the issue of the equilibrium selection problem: for a multi-player general sum game the Nash equilibrium may not be unique, and which one I eventually arrive at depends largely on the equilibrium computation algorithm and random seed I adopt. One way to alleviate this issue under EGTA is to try to find all the possible equilibria one can compute by running the algorithm multiple times and analyzing all of them.

One application is analyzing the Trading Agent Competition [Wellman et al., 2005]. Here the strategies were ranked according to their properties in empirical equilibria. Another is an investigation into market manipulation [Wang et al., 2021a] where different empirical equilibria outcomes were compared in terms of surplus.

2.3.3 Strategy Exploration and Generation

In this section, I briefly survey how representative strategies can be generated in principled ways.

This first approach is via direct search, possibly equipped with some heuristic evaluation function. This was studied in [Jordan et al., 2008]. When $\mathscr{S}$ is finite or easy to enumerate, the analyst may start from some initial strategy set, and iteratively expand this strategy set by adding new

Algorithm 1 Incremental Strategy Generation

```
Input: Game knowledge oracle $\mathcal{O}$, EVA, IMP procedures
Output: Finite strategy set $S_1, S_2, \ldots, S_N$
  Initialize strategy sets $S_1, S_2, \ldots, S_N$
  Initialize empirical game $\mathcal{G}$
  while Termination Criterion not met do
    $\mathcal{G}, \sigma \leftarrow \textit{EVA}(\mathcal{O}, \mathcal{G}, \sigma, S)$
    for $n = 1, 2 \ldots, N$ do
      $s'_n \leftarrow \textit{IMP}(\mathcal{O}, \mathcal{G}, \sigma, S)$
      $S_n \leftarrow S_n \bigcup \{s'_n\}$
    end for
  end while
```

strategies. Some ways of adding strategies include uniformly sampling a strategy outside the current strategy set, or adding those with high heuristic scores. Such a heuristic function includes the deviation value of a strategy under some equilibrium within the restricted set.

We can actually further generalize the above approaches into a framework called incremental strategy generation, as I outlined in the algorithm diagram 1. This framework was originated from the double-oracle framework [McMahan et al., 2003].

This framework employs two procedures, which I denote as *EVA* and *IMP* that interact with each other and generate effective strategies. I assume there is an oracle $\mathcal{O}$ that gives the algorithms the basic knowledge of the game (typically, it is implemented using the black-box game interface). The purpose of *EVA* is to generate statistics that can guide the direction of adding a new strategy. I further segment the statistics into a game model part $\mathcal{G}$, which is the empirical game constructed, and the non-model part σ. σ is often implemented as a distribution over S, e.g., a mixed Nash on the game defined by S. The purpose of *IMP* is to find a new strategy utilizing all the statistics. Typically *IMP* is implemented as a best response optimization procedure. This framework also resembles the policy iteration diagram I introduced before. [Wang et al., 2021b] deepened this line of work into deep RL settings.

2.3.4 Strategy Evaluation

An application of EGTA is to analyze the strategic interaction structure among the strategies or evaluate some "strength" notion of the strategy pool. For investigating strategic interaction, a replicator dynamics plot [Walsh et al., 2002, Tuyls et al., 2020] was often visualized to show the interplay towards an equilibrium. For multiagent evaluation, one way is to compute the deviation payoff of a strategy at an equilibrium [Balduzzi et al., 2018b]. Another is to compute how much mass does each strategy occupies in some equilibrium solution [Omidshafiei et al., 2019].

2.4 Applications

In this section, I introduce several domains where methods from game-theoretic multiagent systems have practical applications.

2.4.1 Auctions and Mechanism Design

Economists have long studied auctions as a standard class of game models of incomplete information. Furthermore, online advertising auctions [Edelman et al., 2007] have become the main channel for IT companies in deriving their revenue. More specifically, in an auction, players first observe their own valuations of the goods as private information. Then they decide their bids as a function of their valuation. The allocations and prices charged are based on the bids. Normally the bidders who bid the highest obtain the goods. The payment rule, however, varies in different scenarios. Some standard rules include first-price payment or second-highest payment, i.e., the winner pays the highest or second-highest bid in the auction. For a general introduction to auction theory, I refer to two famous textbooks [Milgrom, 2004, Krishna, 2009]. For practical algorithms and bidding agent design, [Cramton et al., 2006] is a comprehensive survey.

Researching on auctions naturally leads to another research area called mechanism design. Generally, different configurations or parameters of the game (e.g., allocation rule or payment rule) would lead to different equilibrium outcomes. However, the game designer would like to search for a game that can maximize some objective function (such as revenue) in an equilibrium state. Furthermore, the designer may expect the bidders to *truthfully report* their private valuations. So the area of mechanism design is about finding such game configuration in a principled way. This has large real-world implications since, in the IT industry, the designer may want to maximize the revenue as much as possible. Classical mechanism design focuses on an analytical treatment [Myerson, 1981], while people today are using computational methods to enhance mechanism design, which is termed *automated mechanism design* [Sandholm, 2003]. One example is to use deep learning for learning optimal auctions [Dütting et al., 2019].

2.4.2 Bargaining

Bargaining, or negotiation, normally refers to an interactive process among multiple entities to reach some commonly agreed outcome for all parties. Bargaining is also normally modeled as games with incomplete information, where the types of players are their respective valuations, goals, or baselines. The game, however, contains both collaborative and competitive elements: the goal is to reach some common outcome that benefits all parties, and it is undoubtedly not a zero-sum game. Typically a bargaining protocol needs multiple rounds of interaction, e.g., offer/counteroffer [Rubinstein, 1982]. It is during this process that a player may gradually learn the private information of the other agents in order to take more effective action in future timesteps. And how useful the communication process will be in terms of achieving satisfying outcomes more or less depends on the alignment of the players' interest [Crawford and Sobel, 1982].

In the field of multiagent systems, researchers have developed various techniques to develop efficient negotiation agents. I refer to surveys by [Rosenschein and Zlotkin, 1994, Faratin et al., 1998, Fatima et al., 2014, Baarslag et al., 2016] for a more comprehensive overview of this domain.

2.4.3 Game-Playing AI

One of the most central applications of game theory, as was signified by several AI breakthroughs, is recreational game-playing AI. In this section I survey the achievements of AI algorithms in different classes of games.

2.4.3.1 Chess

Chess is a two-player zero-sum game of perfect information. It is a turn-based game with a branching factor of about 35 at each decision point.

DeepBlue DeepBlue [Campbell et al., 2002] beat World Chess Champion Garry Kasparov and marked one of the first breakthroughs in the field of gaming AI. There were a lot of engineering tricks used in DeepBlue both on the hardware level and software level, including parallelization and hardware search on silicon. I here briefly summarize the algorithmic ideas. The DeepBlue algorithm consists of three components: move generator, evaluation function, and search control. The move generator adopted a best-first-search-like approach, which ranks all possible moves and selects the one with the highest score. The evaluation function is used for depth-limit-search, and was constructed using a variety of sophisticated domain knowledge and techniques. It can be further decomposed into fast evaluation and slow evaluation. The search algorithm is essentially null-window alpha-beta pruning with quiescence search. An additional component is called dual credit, which is to balance the depth-limit for both players adaptively to constrain the computational expense. For practical implementation, it uses one processor to conduct search on the top nodes where several other processors for the leaf nodes.

2.4.3.2 Go

Go is also a two-player zero-sum game of perfect information. Being different from chess, it has an even more larger state space and branch factor. I here survey the famous Alpha-series Go agent.

AlphaGo AlphaGo [Silver et al., 2016] achieved superhuman-level by combining Monte Carlo tree search and deep reinforcement learning. It beat professional Go player Lee Sedol by 4:1. In AlphaGo it trains three different neural networks. The first is the supervised learning (SL) network, which is trained based on human data. The second is a rollout policy being a condensed version of SL network, which is employed in the simulation phase of the MCTS search (Section 2.2.4.5). The third is the reinforcement learning policy, which also outputs a value network for evaluating a position of the game. The RL policy is initialized with the SL network. During the real-time play, upon reaching a state AlphaGo conducts an MCTS search. In the expansion phase, it uses the SL policy to decide the next node to be expanded. And then it uses the rollout policy to sample a trajectory and get a value in the simulation. The statistics being back-propagated is a linear combination of the outputs by the RL value network and the simulation value. The SL network also serves as the prior parameter in maintaining the UCB statistics. Then to decide the next move, it selects the next-step action that was visited most often during the MCTS search. The RL network is then trained by using the game data resulting from MCTS play.

AlphaZero The successors AlphaGo Zero [Silver et al., 2017] and AlphaZero [Silver et al., 2018] improve upon AlphaGo by purely using self-play RL without human data, and achieves greater performance than AlphaGo. These Zero-version agents also share much more neat algorithmic ideas compared with AlphaGo. The unified paradigm is called *generalized policy iteration*. In AlphaZero, it only trains one RL network, which outputs both the policy and value. Then during the real-time play, upon reaching a state AlphaZero conducts an MCTS search, And it expands and evaluates the leaf nodes in the search tree by an estimated value provided by the value network. The policy head is used to compute the UCB score of each node. Then it trains the RL network by using MCTS data. Then the MCTS serves as the policy improver over the RL policy, while the self-play RL training is the policy evaluator that produces the evaluation of the states during MCTS search.

MuZero Schrittwieser et al. [2020] extended AlphaZero into a model-based method called *MuZero*. What MuZero effectively does is to learn a transition model and a reward model using trajectory data, and then conduct the whole MCTS within the latent state space of the learned model. This method effectively learns an *value equivalence model*.

2.4.3.3 Poker

Poker is a standard game model that incorporates large imperfect information. I here survey several famous agents that were designed for general imperfect information games and empirically showed perform well in Poker.

DeepStack DeepStack [Moravčík et al., 2017] combines the idea of deep value training and a recursive application of CFR-D(Section 2.2.4.7) to build a general-purpose AI bot for two-player zero-sum games with imperfect information. It is an online algorithm. It is perhaps the first that (implicitly) introduces the idea of public belief state (Section 2.2.3.3) into practical AI building. It beats several professional-level players.

The core idea of DeepStack is continual re-solving: everytime the agent needs to take an action it resolves the subgame rooted at the current public state. To achieve this it conducts a depth-limit-search using CFR-D on the current public subtree. The regret gadget of this CFR-D is constructed based on the previous-solution regret values. The leaf nodes in the subtree are truncated by value estimation provided by a deep counterfactual value network. It also employs a sparse lookahead actions to reduce the breadth of each information set. The deep counterfactual network is trained by applying CFR-D on randomly sampled states with a pre-obtained value network at a depth limit.

Then when choosing an action, it just directly samples an action from the current resulting strategy after search.

Libratus Libratus [Brown and Sandholm, 2017, 2018] is the first AI bot that achieves superhuman-level on heads-up no-limit Texas hold'em Poker. Libratus introduces a few new techniques over the classical ones. First, it improves the subgame re-solving techniques in CFR-D by using information from other subgames, which is called *reach subgame solving*. This information however is not computed by directly solving other subgames, but is estimated using the blueprint strategies.

The second is called *nested subgame solving*. Roughly speaking, the algorithm first starts with an abstraction of the original game. Once it sees in real-time an opponent action that was not in the abstraction, it creates a new subgame and then solves it. It also iteratively improves the blueprint strategies by using the online computation.

Pluribus Following Libratus, Plurbius [Brown et al., 2018, Brown and Sandholm, 2019] was soon proposed and became the first superhuman AI bots for six-player heads-up no-limit Texas hold'em Poker. Pluribus, however, employed a different set of techniques from Libratus, as I am going to elaborate as follows. Upon reaching a depth-limit subgame, here is how Pluribus computes a Nash strategy of the acting player within the subgame. First, it assumes the acting player plays a blueprint strategy outside of this depth-limit subgame, serving as an approximate equilibrium. The blueprint is obtained by domain knowledge or solving an abstracted version of the game. Then it selects a set of the other-player's strategies, for estimating values on the leaves of this subgame. One way to select them is via some domain knowledge. The other way is to generate a pool of strategies via a double-oracle (Algorithm 1) like approach. It aims to select a few representative other-player's strategies to produce accurate estimations of the leaf node values. With this information, it suffices to compute an approximate equilibrium strategies in this subgame, e.g., use CFR algorithms.

Therefore during the real-time play, each time the agent needs to make a decision it just applies the above depth-limit solving on the current subgame, and plays according to the resulting strategy.

ReBeL The ReBeL architecture [Brown et al., 2020] combined the idea of AlphaZero, CFR-D, and the public belief state representation I covered in Section 2.2.3.3. Whenever the agent reaches a public belief state β_r, ReBeL starts a search process as follows. First ReBeL constructs a depth-limited subgame tree rooted at β_r, and truncates the leaf nodes β_l with a value estimation $\hat{v}(\beta_l)$. Here $\hat{v}(\beta_l)$ is trying to estimate the value of PBS β_l assuming the players play an approximate equilibrium policy from β_l onward. This $\hat{v}$ provides information-state values for CFR-D via an algebraic relation. In iteration t of CFR-D it produces a strategy π^t within this subgame. For every π^t it can be directly computed or estimate the corresponding PBS value $\hat{v}(\beta_r; \pi^t)$, e.g., either by Monte Carlo sampling or directly traversing over the subgame tree. This continues for T iterations totally, and the algorithm produces the average-iteration policies $\sum_t \pi_t / T$. And the corresponding PBS value of β_r is $\sum_t \hat{v}(\beta_r; \pi^t)/T$. This is added to the training data for $\hat{v}$. Then it randomly sample an iteration t_{sample} from $[1, T]$, and use $\pi^{t_{sample}}$ with some exploration to select a path from β_r to a leaf node β_l. Then it expands β_r and applies the above search process repeats on the depth-limit subtree rooted on β_l, until it reaches a terminal node of the full game. This completes the training process of ReBeL.

During test-time, each time it just plays the policies resulting from search. The goal is to output a Nash in expectation. One implementation is to uniformly sample a policy π^t from 1 to T and play this π^t.

The key idea of ReBeL is to conduct an efficient depth-limit search by learning an accurate PBS value network. The accuracy is achieved by a self-play RL styled training paradigm, so the regions that the agent are exposed to are closer and closer to the rational region. Using deep learning to learn a PBS value network is efficient, and this network can also serve as a heuristic function for the leaf nodes in the depth-limit search tree leveraging algebraic relations between PBS values and information-state values. The search tree is also incrementally growing, just like MCTS.

Student of Games Student of Games (SoG) [Schmid et al., 2023] combines the ideas of AlphaZero and CFR. At the core of SoG is a variant of CFR that is adapted to online search called growing-tree CFR (GT-CFR). GT-CFR consists of two components: the regret update phase and the expansion phase. SoG also adopts a public belief representation of the game and trains a counterfactual value and policy network (CVPN) $(\boldsymbol{v}, \boldsymbol{p}) = f_\theta(\beta)$, where $\boldsymbol{v}$ is the counterfactual values for each information state attached to the current node, and $\boldsymbol{p}$ is the prior policies.

The search tree of GT-CFR consists of public states. Statistics are maintained on information states of the action players. The regret update phase can be implemented by running CFR on the current depth-limited public tree. The leaf nodes are abstracted by the values provided by the CVPN. Then after one regret phase, c simulations are conducted on the public tree. At the start of each simulation, a historical trajectory τ is sampled from the root PBS β_r. Then actions are selected according to a mixture of the current CFR strategy, and a UCT strategy that utilizes the CVPN. Upon encountering a public state that is not in the current public tree, this state is added to the search tree. Then the visited counts are updated through the trajectory to the root node, for the UCT policy. PoG expands multiple actions attached to that new public state.

The outcome resulted from search train the value head policy output from search train the policy head. The data are augmented after one game trajectory. The algorithm also stores the PBSs that were queried during the execution of the algorithm, and selectively and recursively chooses some of these and uses GT-CFR to compute a refined $\boldsymbol{p}$ and $\boldsymbol{v}$.

2.4.3.4 Real-Time Video Games

Starcraft II AlphaStar [Vinyals et al., 2019] achieves master-level performances in Starcraft II. Starcraft is a real-time strategy game with large imperfect information, high-dimensional observation and action space, and a teamed two-player zero-sum game structure. The basic idea of AlphaStar is population-based training. It maintains a pool of RL agents. Each agent is initialized with a supervised learning agent based on human data. During RL training it also incorporates a term that penalizes it from being away from the SL agent. Each training the agent is training against a non-uniform mixture of the agents in the pool. This mixture is computed based on the pairwise winning rates. Each trajectory it samples an opponent based on this mixture, and updates the parameters of our agent. AlphaStar also categorizes different classes of training pools to enhance diversity and robustness of the training process.

Dota II OpenAI Five [Berner et al., 2019] was the first AI agent that defeated world champions in Dota II. Dota II is being similar to Starcraft II. It trains the agent using purely self-play RL using proximal policy optimization (PPO) [Schulman et al., 2017].

Gran Turismo Gran Turismo is a real-time car-racing game which has one of the most fidelity simulators. GT Sophy [Wurman et al., 2022] is a human-level car-racing agent that is trained using quantile regression soft actor-critic, using a dense reward design and a customized application of twin networks.

2.4.3.5 Hanabi

Hanabi is a Dec-POMDP with a public state structure. In this game the players collaborate together to reach a higher common objective. Each player holds a different piece of the world state from the beginning (but no more private observations after that), and needs to efficiently communicate with each other to recover the true state by selecting proper action sequences.

Bayesian action encoder (BAD) BAD [Foerster et al., 2019] was proposed to solve this class of game by combining deep RL and public belief representation. The keys of BAD are: (1) it trains a common network that represents the policy in the public-belief MDP. To be more specific, the input of the network is the public belief and private information, and the output is the action choice. Since in principle, the policy in the public-belief MDP (detailed in Section 2.2.3.3) is a mapping from public-belief state to, a distribution over mappings from private information to actions. The architecture is designed in a way such that it first deterministically incorporates the public belief state, and then a random seed, which resorts to a deterministic neural architecture with inputs of the private and the outputs of the actions. (2) It adopts a mean-field like tractable factorized representation for the public-belief updating process.

CAPI Common-payoff approximate policy iteration [Sokota et al., 2021] adopted a slightly different approach than BAD. It also adopted a public-belief MDP representation for solving common-payoff multiagent games of imperfection information like Hanabi. However, its algorithm adopted a Q-learning/policy iteration like approach, where each time it uses the one-step maximization values and actions for training policies and value functions. While BAD uses policy-gradient methods for training the BAD agent.

SPARTA Another thread is to improve a blueprint policy during online-play via search. The SPARTA algorithm [Lerer et al., 2020] employs search by making an agent simulate the search process for other agents, by assuming a common knowledge of random seed. SPARTA has achieved state-of-the-art performance in Hanabi for self-play based evaluation.

2.4.3.6 Avalon

Avalon is a multiplayer game with hidden role information and partially observable actions. The goal of Avalon is to find out the teammates and opponents through rounds of communication and then reach a higher outcome for the team.

DeepRole DeepRole [Serrino et al., 2019] combines CFR with deep value network and becomes the first superhuman five-player Avalon agent. It employs a depth-limit CFR as an online search method with a posterior belief over roles being maintained and a deductive logic that eliminates action possibilities that are inconsistent with the history. The leaf nodes are truncated by a value network. The value network returns values for the information sets given the proposer's index and role belief. The value network is trained by sampling random trajectory over the belief and using CFR to get the target values. Each time one needs to make a decision, it just uses the search procedure on the current public state, and then acts according to the computed strategy.

2.4.3.7 No-Press Diplomacy

No-Press Diplomacy is a class of multiplayer, general-sum, fully observable Markov games with deterministic transition, combinatorial action space.

DipNet [Paquette et al., 2019] trains a model based on human data using supervised learning. It also has an improved version using self-play RL.

Best Response Policy Iteration In [Anthony et al., 2020] it trains an agent via a PSRO-style paradigm using fictitious play as the meta-strategy solver.

SearchBot SearchBot [Gray et al., 2020] employs online search during the real-time play. More concretely, it uses a supervised learning algorithm to train a blueprint policy by human data. Then each time when SearchBot needs to make a decision, it only considers a one-step lookahead action, and assumes every player plays the blueprint afterward. Then it solves the game defined by the next-step action using regret matching. It also selectively chooses the actions being considered in the support, according to scores output by the blueprint.

DORA DORA [Bakhtin et al., 2021] built superhuman AI without using any human data. It adopts an AlphaZero like training paradigm. It maintains a value network that returns the state value assuming everyone is going to play an equilibrium policy onward. It refines this value network by using a variant of Nash value iteration (Section 2.2.2.1) using deep learning, which is to minimize a loss based on a Bellman-structure of the equilibrium condition. To construct the target values of the loss it solve a game defined by the next-timestep action, using regret matching. The second component it incorporates is a proposal network, which returns action probability given each state. The proposal network is to identify promising actions within the original combinatorial action space. After the proposal network has sampled a few actions as in the support, it uses regret matching to compute an equilibrium in the current one step lookahead game. The proposal network is refined by minimizing its distance from the equilibrium search from regret matching.

To avoid making the proposal network stuck in local regions, the algorithm also employs a double-oracle inner-loop to each time discover novel actions into the support, after the initial equilibrium computation. It first added several local modifications of the actions in the current support as the empirical full strategy space. Then it uses double-oracle algorithm to search for an equilibrium within this empirical strategy space.

Diplodocus and Cicero However, an issue of DORA is that it may produce machine convention that is not interpretable by humans. To achieve good performance playing with humans, the Diplodocus [Bakhtin et al., 2023] agent proposed a new equilibrium computation algorithm within the DORA architecture, called piKL. The idea is to pretrain a supervised learning policy, and use it as a regularization policy. Therefore, the final policy will not be too far away from the human convention. The Cicero [FAIR et al., 2022] proceeded one step further, by combining a pre-trained language model with the RL agent. The language model takes historical dialogues and observations as input, and outputs an intention to the RL policy. Cicero achieved human-level performance in a language version of Diplomacy.

2.4.3.8 Stratego

DeepNash Stratego is a two-player zero-sum game with large imperfect information. The goal is to configure your hidden unit during the deployment phase wisely, such that at the action phase you can eliminate the opponent's unit more strategically. DeepNash [Perolat et al., 2022] is a model-free multi-agent RL algorithm that produces human-level Stratego agents. The underlying algorithm, Regularized Nash dynamics, solves a sequence of regularized games, therefore has a stronger convergence guarantee to Nash equilibrium.

2.4.4 Security

Security is another large-scale application domain of game theory. This includes police allocations in airports, and patrolling scheduling for natural resource protections. Normally the game consists of defenders, who want to protect several resources, and attackers, who aim to attack some resources for malicious reasons. The model usually begins with the defenders decide their strategy given limited protection resources, and then the attackers react. This usually essentially transforms the problem to Stackelberg games with structured utilities and action spaces. For a more detailed cover of the formulation, I refer to [Tambe, 2011, Sinha et al., 2018, Fang et al., 2015].

2.4.5 Finance

Finance is another practical domain that game-theoretic methods can apply. Financial activities naturally involve a lot of strategic behaviors: the trading activities among the traders have well-defined utilities and strategy space, yet the large stochasticity and imperfect information cause a direct analytical treatment intractable. I refer to works [Wellman, 2011, Wright and Wellman, 2018] for more references.

2.4.6 Multiagent Competitions

In my last section about applications, I introduce several multiagent software competitions. Normally the goals of these competitions are to encourage researchers and engineers to contribute their ideas by empirically evaluating the agents' performances.

2.4.6.1 Iterated Prisoner's Dilemma

Iterated prisoner's dilemma was held by Prof. Robert Axelrod of the University of Michigan as a mix of sociology study and computer programming tournament. The goal of holding IPD is to understand why cooperation behaviors emerge from human history, if we assume humans are rational selfish agents. Especially the game of prisoners' dilemma was repeatedly replayed in our lives in different forms: there are many situations where a malicious player who breaks the conventions (the defect strategy) can lead to a large reward for itself at the cost of decreasing the overall social welfare. However making it a repeated game may change its properties drastically.

The tournament was extremely successful. Among the first round of 15 agents submitted, a concise strategy called *tit-for-tat* stands out and obtains the highest ranking. This strategy is actually very simple: you just play what your opponent played in the last iteration. This retaliative strategy turns out to be pretty well. And in fact, it again ranked first in the second round of this tournament where all strategies from the first found were made public. Actually this may illustrate why cooperation emerged during human history: it is because of the repeated nature of the PD that makes such punishment behavior work well. TFT in principle can achieve an optimal outcome with a collaborative agent, and will not be downgraded too much in the face of a defective one. So the success of TFT may also come from the large collaborative elements within the agent pool. I refer to [Axelrod and Hamilton, 1981] for a detailed analysis of this tournament.

2.4.6.2 Trading Agent Competition

Trading agent competition [Wellman et al., 2007] was an annual multiagent competition that asked the participant to design effective agents that can be involved in different market trading behaviors.

The agents typically need to decide the resources to be traded and the prices in a dynamic market environment. The agents are also rendered their own private information. One example is an ad auction game [Jordan and Wellman, 2009] where the agents are supposed to strategically bid in a series of ad auctions.

2.4.6.3 Computer Poker Competition

The Annual Computer Poker Competition had also been a successful multi-agent competition. Participants are encouraged to submit Poker agents for different versions of the game. Over the years the competitions have successfully contributed to the advancement of computer game research, including the development of CFR algorithm.

2.4.6.4 Automated Negotiation Agent Competition

Finally, I briefly introduce the ANAC competition [Jonker et al., 2017]. ANAC consists of different leagues each with a different class of games. A unified theme of these games is to design effective negotiation agents in complex market environments. The bargaining processes are typically variants of Rubinstein's alternating protocol. The agents typically need to learn the hidden information of the opponent's utilities, in order to extract a higher payoff for itself.

CHAPTER 3

Structure Learning for Solving Large Normal-Form Games

Games with many players are difficult to solve or even specify without adopting structural assumptions that enable representation in compact form. Such structure is generally not given and will not hold exactly for particular games of interest. We introduce an iterative structure-learning approach to search for approximate solutions of many-player games, assuming only black-box simulation access to noisy payoff samples. Our first algorithm, K-Roles, exploits *symmetry* by learning a *role assignment* for players of the game through unsupervised learning (clustering) methods. Our second algorithm, G3L, seeks *sparsity* by greedy search over local interactions to learn a *graphical game* model. Both algorithms use supervised learning (regression) to fit payoff values to the learned structures, in compact representations that facilitate equilibrium calculation. We experimentally demonstrate the efficacy of both methods in reaching quality solutions and uncovering hidden structure, on both perfectly and approximately structured game instances.

3.1 Introduction

Many of the real-world multiagent systems we would like to understand strategically involve an enormous number of interacting (or potentially interacting) agents. For example, domains of multiagent research interest—such as ad auctions [Guo et al., 2019], financial markets [Nevmyvaka et al., 2006], traffic routing [Bazzan, 2009], and rumor spreading over social media [Yang et al., 2018]—all encompass (depending on the scope being considered) thousands or millions of participating agents. Straightforward game-theoretic representations of these systems do not scale well: A direct normal-form representation of an N-agent, M-action game is $O(NM^N)$, which is obviously not feasible to construct or reason about directly for even moderately large-scale multiagent systems. In response, AI researchers and others have identified various kinds of regularity that may be exhibited in such systems, particularly invoking some form of homogeneity (*symmetry*) or locality of interaction (*sparsity*) that can be exploited to develop a more compact game representation [Jiang et al., 2011, Kearns et al., 2001]. However, it may not be apparent or easy to specify the exact structure that applies in a given multiagent scenario, and indeed it may be that no pre-identified structural simplification holds exactly for a problem of interest. We therefore investigate in this work the possibility of learning such structure, and moreover doing so in an iterative manner interleaved with game-theoretic reasoning about structured game models as they are developed.

We develop our methods in the framework of empirical game-theoretic analysis (EGTA) [Wellman, 2016, Tuyls et al., 2020]. EGTA assumes as input a representation of the game in terms of a payoff oracle (*e.g.*, a simulator). The game analyst may sample this *simulation-based game* by querying the oracle to obtain data from which to estimate or induce a game model, called the *empirical game*. This framework makes sense for the problem at hand, because specifying simulation models does not suffer from the curse of dimensionality that inhibits scaling explicit game models to large numbers of agents. The challenge is to make effective use of the simulator to gain game-theoretic insights on such large multiagent systems.

Our hypothesis is that the game-learning process can be enhanced by a focus on the structure of agent populations and interactions. For population structure, we appeal to the partition of agents into distinct *roles*. Many applications have obvious role dichotomies: investors and traders in financial markets, commuters and vacationers in road traffic, brand and sales marketers in advertising. More generally, we expect that many multiagent systems will at least roughly support classification into broad roles. For interaction structure, we appeal to locality. For example, on a social network, one is directly influenced (by rumors, innovations, etc.) primarily through one's connections on the network. Both kinds of structure—role-organized symmetry and locality of influence—have been formalized in terms that support compact game representations. Whereas the formal requirements for compact representation may not strictly hold for games of interest, we expect that they will often hold approximately to a useful degree. If so, it is worth trying to identify that useful structure in payoff samples, thereby enabling induction of more compact and sample-efficient game representations, and accordingly supporting simpler and more reliable game-theoretic reasoning.

Our approach draws on a broad variety of machine learning techniques. Inspired by the model-based reinforcement learning framework [Sutton and Barto, 2018, Sec. 8.2], we build our iterative learning-and-solving architecture diagrammed in Figure 3.1. The underlying game is represented by a simulator, $\mathcal{O}$, providing black-box oracle access. The only explicit game descriptors are the sets of agents and actions. Starting with an arbitrary guess solution $\boldsymbol{\sigma}^*$, on each iteration, the method first queries $\mathcal{O}$ in the region of $\boldsymbol{\sigma}^*$, obtaining by this online sampling process a new dataset $\mathcal{D}_{val}$, which is added to the data buffer $\mathcal{D}$. Through offline interaction with $\mathcal{D}$, we then learn and solve a game model to reach the next $\boldsymbol{\sigma}^*$. The learning process encompasses two steps: It first discovers the approximate hidden structure from payoff data, and then enables payoff regression

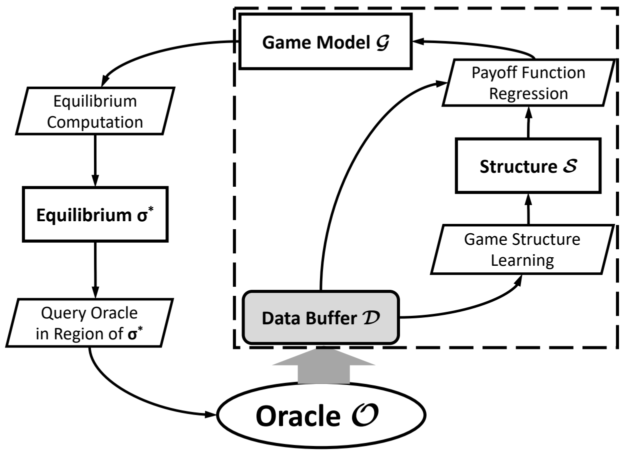


Figure 3.1: Iterative game model learning and solving. The dashed box encompasses the model learning components.

by feeding both data and learned structure to function approximators. The sampling process across iterations is designed to concentrate the data buffer $\mathcal{D}$ around solution candidates, thus prioritizing the quality of generalization on the most relevant regions. The method employs offline computation and storage with the aim of limiting online sample complexity, in service of effective reasoning about large-scale simulation-based games.

We propose two algorithms instantiating our framework: K-Roles for learning role-symmetry, and Greedy Graphical Game Learning (G3L) for learning graphical structure in a game model. We begin by introducing necessary background information in Section 3.2, followed by Section 3.3 reviewing related work on game model learning and solving. Algorithmic details of K-Roles and G3L are covered respectively in Sections 3.4 and 3.5. Section 3.6 presents our evaluation on both perfectly and approximately structured game instances. Conclusion and insights on methods developed are drawn in Section 3.7.

3.2 Preliminaries

3.2.1 Normal Form Games

In an N-player normal form game $\mathscr{G}$, player (or *agent*) $n \in \mathscr{N} = \{1, \ldots, N\}$ chooses its action $a_n \in \mathscr{A}_n$, and receives payoff $u_n(\boldsymbol{a})$ as a function of the agents' joint action or *action profile* $\boldsymbol{a}$.[1] We assume agents share a universal finite action set: $\forall n.\ \mathscr{A}_n = \mathscr{A} = \{1, \ldots, M\}$. Thus, $\boldsymbol{a} \in \mathscr{A}^N$, and for convenience, we write payoff functions in a form $u_n(a_n, \boldsymbol{a}_{-n})$ that separates the subject agent's action and the vector of other-agent actions in distinct arguments. A mixed strategy σ

[1]Actions here may correspond to complex strategies; we refer to action and strategy profiles interchangeably.

is a probability distribution induced over $\mathscr{A}$. The payoff for n under a joint mixed strategy $\boldsymbol{\sigma}$ is $u_n(\boldsymbol{\sigma}) \triangleq \mathbb{E}_{\boldsymbol{a}\sim\boldsymbol{\sigma}}[u_n(a_n, \boldsymbol{a}_{-n})]$. The *deviation payoffs* vector for n under $\boldsymbol{\sigma}$ is $\nabla_{\sigma_n} u_n(\boldsymbol{\sigma}) = \left(\frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,1}}, \ldots, \frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,M}}\right)^{\mathbf{T}}$, where the m^{th} element $\frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,m}} = u_n(m, \boldsymbol{\sigma}_{-n}) \triangleq \mathbb{E}_{\boldsymbol{a_{-n}}\sim\boldsymbol{\sigma}_{-n}}[u_n(m, \boldsymbol{a}_{-n})]$, that is, the payoff of n choosing m while others act according to $\boldsymbol{\sigma}$.

3.2.2 Approximate Nash Equilibrium

Define the *regret* of agent n at $\boldsymbol{\sigma}$ as $\textsc{Regret}_n(\boldsymbol{\sigma}) \triangleq \max_{a_n} u_n(a_n, \boldsymbol{\sigma_{-n}}) - u_n(\boldsymbol{\sigma})$. The overall regret of the game at $\boldsymbol{\sigma}$ is $\textsc{Regret}(\boldsymbol{\sigma}) \triangleq \max_n \textsc{Regret}_n(\boldsymbol{\sigma})$, and if $\textsc{Regret}(\boldsymbol{\sigma}) \leqslant \epsilon$, we call $\boldsymbol{\sigma}$ an *ϵ-Nash equilibrium*. We typically seek solutions to minimize $\textsc{Regret}$, or alternately $\textsc{NashConv}(\boldsymbol{\sigma})$, defined as the aggregate regret over agents: $\sum_n \textsc{Regret}_n(\boldsymbol{\sigma})$ [Lanctot et al., 2017, Srinivasan et al., 2018].

3.2.3 Succinct Games

Without further structural assumptions, the representation complexity of an N-agent, M-action game is $O(NM^N)$, which is generally intractable both for storing a game description and computing its solution. Therefore, extensive work has been directed at identifying *succinct game* representations [Daskalakis et al., 2009].

In an *anonymous game* [Daskalakis and Papadimitriou, 2015], agent n's payoff depends only on its action and how many (or equivalently, what fraction of) agents choose each action: $u_n(a_n, \boldsymbol{a}_{-n}) = u_n(a_n, f_1, \ldots, f_M)$, with f_m counting the frequency of agents choosing m. If furthermore $\forall n.\ u_n = u$, the game is *symmetric*. A *role-symmetric game* generalizes this concept by introducing asymmetry: Agents are partitioned into different *roles*, where agents within the same roles are interchangeable from the view of others. Let $\mathcal{R}(n) \in \{1, \ldots, K\}$ denote the role for agent n. Then the payoff for agent n depends on its action and the action distribution within each role: $u_n(a_n, \boldsymbol{a}_{-n}) = u_n^{\mathcal{R}}(a_n, f_{1,1}, \ldots, f_{1,M}, \ldots, f_{K,M})$, where $f_{k,m}$ is the action frequency of m within role k. For a role-symmetric game, we are typically interested in role-symmetric profiles: $\forall n, n'.\ \mathcal{R}(n) = \mathcal{R}(n') \implies \boldsymbol{\sigma}_n = \boldsymbol{\sigma}'_n$. For finite role-symmetric games, equilibria are guaranteed to exist in role-symmetric profiles [Nash, 1951].

A second category of succinct representations, *graphical games* [Kearns et al., 2001], capture *sparsity* in multiagent interaction. By assuming agent n's payoff depends only on the joint action profile over its neighborhood $\mathcal{N}(n)$ on an interaction graph, $u_n(a_n, \boldsymbol{a}_{-n}) = u_n(a_n, \boldsymbol{a}_{\mathcal{N}(n)})$, the representation complexity is reduced to $O(NM^\kappa)$, where κ is the maximum size of a neighborhood.

3.2.4 Empirical Game Models

The methodology of *empirical game-theoretic analysis* (EGTA) employs simulation or sampling to induce a game model. This approach is called for when it is not feasible to express a game model in analytic form, either due to representation complexity or difficulty of manual specification. Formally, in EGTA the multiagent environment is represented by a *game oracle* $\mathcal{O}$ (*e.g.*, a simulator), which can be queried to generate a dataset $\mathcal{D}$ of action-payoff tuples $(\boldsymbol{a}, \boldsymbol{u})$, where $\boldsymbol{u}$ is either an exact or noisy sample of the payoff vector associated with action profile $\boldsymbol{a}$. A normal-form game model induced from $\mathcal{D}$ is called an *empirical* game.

In most EGTA studies, the dominant cost is that of simulating action profiles (*i.e.*, querying the oracle). Accordingly, several prior works have addressed the problem of controlling the sampling process to maximize analysis value while minimizing query costs [Jordan et al., 2008, Walsh et al., 2003], and have obtained theoretical bounds in some cases [Viqueira et al., 2019, Goldberg and Turchetta, 2017]. Our work can be viewed in this line, distinguished by its focus on identifying structure both to improve generalization and facilitate reasoning.

3.2.5 Game Model Learning

Game model learning in our setting aims to induce a representation of a game, within a specified *hypothesis game space*, from limited payoff data using standard machine learning methods. The hypothesis spaces of interest correspond to succinct game formats where practical structure-exploiting Nash solvers exist. For example, one can apply *replicator dynamics* [Sandholm, 2010, Section 4.3.1] or *function minimization* [McKelvey and McLennan, 1996] to solve a role-symmetric game, and *homotopy method* [Blum et al., 2006] or *hybrid refinement algorithm* [Vickrey and Koller, 2002] to a graphical game. Given a hypothesis space $\mathcal{H}$, one preprocesses data point $(\boldsymbol{a}, \boldsymbol{u})$ to construct the *features* on the raw $\boldsymbol{a}$ according to $\mathcal{H}$. For a role-symmetric game it is to aggregate the action frequency over different roles, thus it can be viewed as *feature extraction*; while for a graphical game it is to eliminate agent dependency, thus it can be interpreted as *feature selection*. One therefore builds an empirical game by learning a mapping from the set of features to the set of payoffs.

3.3 Related Work

Computational Game Theory The idea of using iterated game approximation to find equilibrium dates back to the *homotopy method* [Govindan and Wilson, 2003, Herings and Peeters, 2010] in classic computational game theory. Homotopy method typically starts from a perturbation of the original game model with a trivial solution. By keeping track of the perturbation vector and equilibrium along a homotopy path, it is guaranteed to reach the equilibrium of the origin game. One elegant instantiation is the *iterated polymatrix approximation algorithm* [Govindan and Wilson, 2004, Blum et al., 2006], which approximates the original game as a sequence of polymatrix games and solves them by some efficient subroutine such as the Lemke-Howson algorithm.

Multiagent Simulation for Game Model Learning In simulation-based game model learning, the analyst samples a variety of strategy profiles and receives data in the form of (profile,payoff-vector) for model learning [Vorobeychik et al., 2007].

The only prior work we are aware of that expressly exploits clustering for learning a normal form game model is by Ficici et al. [2008]. Their approach starts by clustering agents via k-means ($k = 2$) according to the average payoff vector in the dataset. They then use linear regression to estimate mixed-strategy payoffs for the clusters. These regressors are in turn used to construct the pure-strategy payoff table of a reduced two-player approximation of the game. They compute a Nash equilibrium of this game, and ascribe the resulting mixed strategies to agents in the respective clusters. Our method for learning role-symmetry structure can be viewed as a variation of theirs that: (1) reduces dimensionality by clustering payoff deviation functions rather than payoff functions, (2) allows for more clusters, and (3) solves the role-symmetric game rather than a reduced game.

There also has been prior work on regression for role-symmetric games, for given role assignments [Wiedenbeck et al., 2018, Sokota et al., 2019]. Duong et al. [2009] and Fearnley et al. [2015] studied algorithms for inducing structure of graphical games.

These works typically assume that training data collection through simulation is fully controlled by the analyst. This makes the setting akin to *active learning* [Settles, 2009]. Thus, the simulation-based approach could also be regarded as a semi-supervised style of game model learning.

Game Model Learning from Behavioral Data Another line of work employs behavioral data for game model learning. Here the data are typically assumed to be generated from approximate equilibria repeatedly played by bounded rational agents. In contrast to the simulation-based approach described above, the observational data here takes the form of actions rather than payoffs.[2] Thus, we classify this style of game model learning as *unsupervised*, and note that it can also be viewed as a multiagent form of inverse reinforcement learning [Ng and Russell, 2000].

The focus of prior work in this area has been to recover underlying game structure. For example, Honorio and Ortiz [2015] adopt a specific generative model, and optimize the fit of key parameters to the available data. Other works also employ diverse optimization techniques to uncover the payoff matrix under a best-response constraint [Kuleshov and Schrijvers, 2015, Waugh et al., 2011,

[2] Gao and Pfeffer [2010] study game learning based both on payoff data and assumed rationality of action choice.

Algorithm 2 K-Roles

```
Input: Hyperparameter $\hat{K}$, Oracle $\mathcal{O}$.
  Initial solution $\boldsymbol{\sigma}^*$, Data buffer $\mathcal{D} = \{\}$
  repeat
    $\mathcal{D}_{val} \leftarrow$ QUERY$(\mathcal{O}, \boldsymbol{\sigma}^*)$
    $\nabla_{\boldsymbol{\sigma}^*}\hat{u} \leftarrow$ DEVEST$(\mathcal{D}_{val})$
    $\mathscr{C} \leftarrow$ DETCLUSTER$\left(\nabla_{\boldsymbol{\sigma}^*}\hat{u}, \mathcal{D}, \mathcal{D}_{val}\right)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{val}$
    $\{\nabla_{\boldsymbol{\sigma}}\mathscr{R}_k\}_{k\in[\hat{K}]} \leftarrow$ FITREGRESSOR$(\mathscr{C}, \mathcal{D})$
    $\boldsymbol{\sigma}^* \leftarrow$ NASHSOLVER$\left(\{\nabla_{\boldsymbol{\sigma}}\mathscr{R}_k\}_{k\in[\hat{K}]}\right)$
  until $\boldsymbol{\sigma}^*$ sufficiently close to equilibrium;
```

Ling et al., 2018, 2019]. Models learned with such techniques have been shown to fit well to some real-world scenarios [Garg and Jaakkola, 2016, 2017].

3.4 K-Roles: Learning Role Symmetry

We now describe a specific algorithm that employs structure learning, under the hypothesis that the game approximately exhibits role symmetry structure. The *K-Roles* algorithm follows the basic template of Figure 3.1, with structure defined by a mapping of players to roles, players within each role treated symmetrically. The method combines unsupervised methods (clustering) for structure learning, with supervised techniques (regression) for payoff estimation. Though the target game is in general not perfectly role-symmetric for $\hat{K} < N$, we may still expect it to exhibit approximate role structure for some reasonable number of roles.

3.4.1 Overview

As shown in Algorithm 2, our approach employs a hyperparameter, $\hat{K}$, denoting the number of roles in the game model. In each iteration the algorithm first augments the payoff dataset by sampling near the current candidate solution. It then estimates deviation payoff vectors for each player (DEVEST in Algorithm 2), and assigns players to roles through an unsupervised clustering method (DETCLUSTER). A new payoff function is then learned by regression (FITREGRESSOR), taking the derived role assignment as a constraint. With role symmetry, the regression essentially entails training payoffs for $\hat{K}$ separate "role agents", each representing its respective role as determined from the clustering operation. Finally, we compute a role-symmetric mixed Nash equilibrium (NASHSOLVER) for the game model at the current iteration.

By imposing role symmetry and a regression model, we reduce the representational complexity of the game from $O(NM^N)$ to $O(M^2\hat{K}^2)$. The reason is that there are totally $\hat{K}$ different utility functions that need to be represented, one for each role. The input of these utility functions would be a strategy counter of length $M\hat{K}$, and the output would be M values for each of them. Therefore for example, for a neural network with fixed intermediate layers and numbers of nodes, it scales with $\hat{K}$ and M quadratically.

3.4.2 Structure Learning

We propose two heuristic agent clustering methods based on well-known partitional clustering algorithms: k-means and hierarchical clustering. We represent agent n by its deviation payoffs $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_n$, where we call its *vector embedding*. Here $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_{n,m}$ is estimated by taking the average of the payoffs when agent n chooses action m in $\mathcal{D}_{val}$. If no data is available for action m, we simply take estimate $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_{n,m}$ as n's average payoff across all actions. The task here is to cluster these agents by similarity of strategic view into $\hat{K}$ roles.

The first method is similar to the one adopted by Ficici et al. [2008]: We directly apply k-means ($k = \hat{K}$) on the vector embeddings to obtain a cluster assignment. Per the k-means procedure, on each iteration we calculate the centroids of each cluster based on the current assignment, then update the clustering by assigning each agent to the cluster to which it is closest based on these centroids.

In the second method we define distance measures between any pair of agents (i, j) and then perform hierarchical clustering. Here we use a weighted L^p-norm between deviation payoffs as the distance metric. Specifically we compute $\|u_i - u_j\|_{\boldsymbol{\sigma},p}$ as

$$\left(\sum_{a_i}\sum_{a_j}\sigma_{i,a_i}\sigma_{j,a_j}|u_i(a_i,\boldsymbol{\sigma}_{-i}) - u_j(a_j,\boldsymbol{\sigma}_{-j})|^p\right)^{1/p},$$

at the latest equilibrium point $\boldsymbol{\sigma} = \boldsymbol{\sigma}^*$ with $p \geqslant 1$. We use $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_{i,a_i}$ to estimate $u_i(a_i, \boldsymbol{\sigma}^*_{-i})$. We adopt the version of agglomerative average linkage clustering: Starting from N singletons, in each iteration for any pair of current clusters we calculate the average inter-cluster distance, and merge the pair that minimizes that until we reach $\hat{K}$ clusters.

In the experiments described below, we employ both clustering methods for K-Roles. First we try hierarchical agent clustering with $p = 2$. If any returned cluster is of size below 20, we discard the result and apply k-means clustering instead.

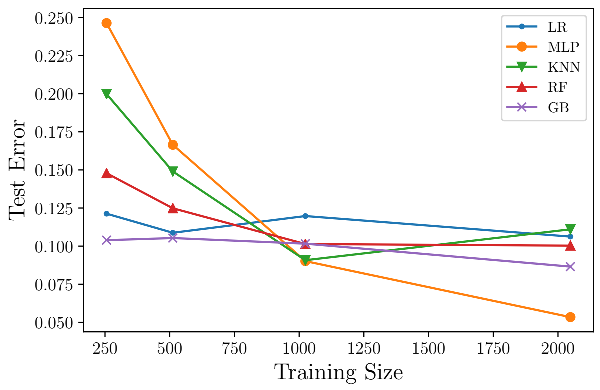


Figure 3.2: Performance of deviation payoff estimations for linear regression (LR), multilayer perceptron (MLP), k-nearest-neighbor (KNN) with $k = 5$, random forest (RF), and gradient boosting (GB). Training data are corrupted with Gaussian noise of variance 0.2^2.

3.4.3 Payoff Function Regression

Given a cluster assignment $\mathscr{C}$ derived from the preceding step, we perform regression (FITREGRESSOR in Algorithm 2) to estimate deviation payoffs for each action of each role agent. Define $\nabla_{\boldsymbol{\sigma}}\mathscr{R}_k = (\nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,1}, \ldots, \nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,M})^{\mathbf{T}}$ as the vector of deviation payoffs to learn for role k, where the m^{th} element is the payoff for an individual of role k playing m while others act according to the role-symmetric strategy $\boldsymbol{\sigma}$. We compute that by aggregating the payoff information according to $\mathscr{C}$.

Specifically, given partition $\mathscr{C} = \{\mathcal{R}_1, \ldots, \mathcal{R}_{\hat{K}}\}$, we organize the data for each role as follows. For a raw data point $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$, we first calculate the action counts (empirical distribution) from $\boldsymbol{a}$ for each role and concatenate them as the feature vector $\boldsymbol{f}$, and then for each agent n of role k we store $(\boldsymbol{f}, (\boldsymbol{u})_n)$ as a data point for training $\nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,(\boldsymbol{a})_n}$. This corresponds to the *point method* proposed by Wiedenbeck et al. [2018] for mixed payoff estimation.

We tried a variety of regression methods; Figure 3.2 plots their performance for a random role-symmetric game with $N = 300, M = 3, K = 3$, trained according to the ground-truth role partition. We sample 500 role-symmetric mixed strategies from a Dirichlet distribution as the test set, and define the test error to be the L2 loss between the regressor predictions and the true deviation payoffs, averaging across all roles and all actions. For each profile we estimate these target deviation payoffs via 1000 samples from the oracle. We find that generally gradient boosting and multilayer perceptron regression methods outperform the others in terms of accuracy and robustness against noise, while the latter generalize better with sufficient amount of training data.

3.4.4 NashSolver

After we have trained the role agents $\{\mathscr{R}_1, \ldots, \mathscr{R}_{\hat{K}}\}$, they naturally define a role-symmetric game, where the action space for each role is an M-dimensional simplex consisting of all possible action distributions within that population. We resort to function minimization [McKelvey and McLennan, 1996] to attain a role-symmetric equilibrium.

Remark 1 The regressed model supports $O(1)$ access to the deviation payoffs of role-symmetric mixed strategies needed for NashSolver as defined here. This avoids the infeasible multiplication and summation over payoff matrices for deviation calculation employed in classical Nash algorithms.

Remark 2 The learned role-symmetric game model represented by differentiable function approximators is a special case of *differentiable game* [Balduzzi et al., 2018a]. Optimization techniques designed for that class could therefore also be applied in an alternative NashSolver.

3.5 G3L: Learning Graphical Structure

3.5.1 Overview

Our second algorithm operates under the hypothesis that the game approximately exhibits graphical dependence structure. *G3L* (Algorithm 3) employs a refinement of the greedy loss minimization approach of Duong et al. [2009] for structure learning. The procedure also resembles score-based structure learning in probabilistic graphical models [Heckerman et al., 1995], and greedy forward feature selection in representation learning [Friedman et al., 2001, Sec. 3.3.2].

3.5.2 Structure Learning

For each iteration we maintain an estimated neighborhood set $\hat{\mathcal{N}}(n)$ for each agent n. All such $\hat{\mathcal{N}}(n)$ would define a directed graph for a graphical game. Initialized as $\{n\}$, we perform a sequence of local searches on $\hat{\mathcal{N}}(n)$ until convergence: Each time we either add a new neighbor or delete an old one from $\hat{\mathcal{N}}(n)$, such that the training loss (described next subsection) would decrease the most. Furthermore to control model complexity for efficient deviation computation during equilibrium calculation, we employ a regularizer $\hat{\kappa}$ to constrain the maximum neighborhood size.

Algorithm 3 Greedy Graphical Game Learning

```
Input: Hyperparameter $\hat{\kappa}$, Oracle $\mathcal{O}$.
  Initial solution $\boldsymbol{\sigma}^*$, Data buffer $\mathcal{D} = \{\}$
  repeat
    $\mathcal{D}_{val} \leftarrow$ QUERY$(\mathcal{O}, \boldsymbol{\sigma}^*)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{val}$
    for $n \in \mathcal{N}$ do
      $\hat{\mathcal{N}}(n) = \{n\}$
      repeat
        $\hat{\mathcal{N}}(n) \leftarrow \underset{\substack{S:|S \triangle \hat{\mathcal{N}}(n)| \leqslant 1 \\ |S| \leqslant \hat{\kappa}}}{\arg\min} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{a},\boldsymbol{u}) \in \mathcal{D}} L(\hat{u}_n(S, \boldsymbol{a}), (\boldsymbol{u})_n)$
      until $\hat{\mathcal{N}}(n)$ converged;
    end
    $\boldsymbol{\sigma}^* \leftarrow$ NASHSOLVER$\left(\left\{\hat{u}_n(\hat{\mathcal{N}}(n)\right\}_{n \in \mathcal{N}}\right)$
  until $\boldsymbol{\sigma}^*$ sufficiently close to equilibrium;
```

3.5.3 Payoff Function Regression

For a given profile $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ of the learned graphical game, we set $\hat{u}_n$ to the average of all $(\boldsymbol{u})_n$, such that $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$ and $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ is contained in $\boldsymbol{a}$. If $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ does not appear in any profile of $\mathcal{D}$, we just set $\hat{u}_n$ to the average payoff of n choosing a_n in $\mathcal{D}$. The training loss for a given $\hat{\mathcal{N}}(n)$ is defined as the average of L2 loss between payoff prediction $\hat{u}_n(\hat{\mathcal{N}}(n), \boldsymbol{a})$ and target payoff $(\boldsymbol{u})_n$ for all $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$.

We plot in Figure 3.3 the performance of greedy forward learning on a random graphical game, for different regularizer $\hat{\kappa}$. We sample 500 pure strategy profiles according to uniform mixed strategy as the test set, and define the test error as the L2 loss between the regressor predictions and these ground truth pure strategy payoffs. We find that in general the hyperparameter $\hat{\kappa}$ trade-offs prediction accuracy for sample complexity: A bigger $\hat{\kappa}$ represents a richer model class while requires more data to make good estimations.

3.5.4 NASHSOLVER

We implement an optimized version of the Govindan-Wilson algorithm [Blum et al., 2006] to solve for exact mixed NE of the learned graphical game. The graphical structure is exploited through the efficient computation of deviation payoffs.

Remark With many players, the main computational bottleneck of the Govindan-Wilson algorithm is calculating the adjugate matrix for the payoff Jacobian. Here we resort to the method

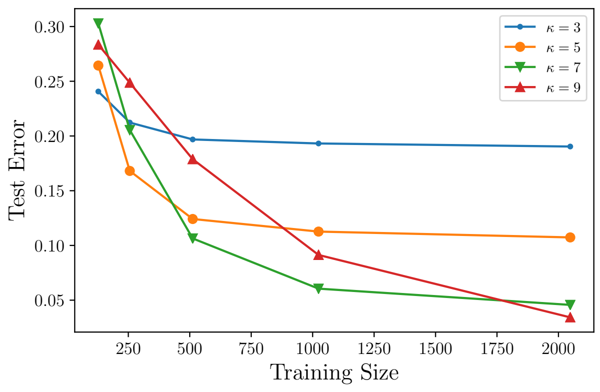
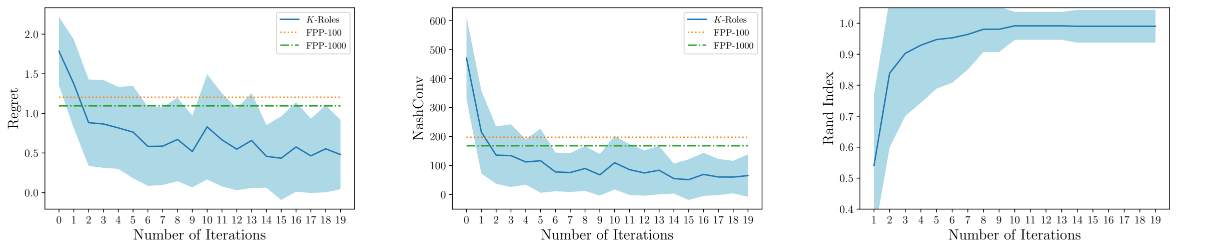


Figure 3.3: Performance of greedy forward learning for pure-strategy payoff estimation under different $\hat{\kappa}$. Training data are corrupted with Gaussian noise width 0.2^2.



(a) Regret (b) NashConv (c) Rand Statistics with Ground-truth

Figure 3.4: Performance of K-Roles over iterations. The results are averaged over 30 runs.

proposed by Stewart [1998], which utilizes perturbed decomposition to compute the adjugate as well as to handle the case when it is singular. We adopt the "wobble" trick and adaptive step size described by Blum et al. [2006] to speed up convergence. Other parameters are the same as the implementations in GameTracer [Blum et al., 2002].

3.6 Experiments

In this section, we evaluate our methods on both perfectly and approximately structured games. For games with perfect role structure, we generate the cluster assignment from a uniform distribution. For games with an underlying graphical model, we generate a directed random graph with expected number of neighbors 5. All payoff samples are added with Gaussian noise of width 0.2^2 when returned by the oracle.

For a game with a perfect role structure, we validate the model accuracy by the normalized Rand Statistics [Rand, 1971] within range [0,1], between the clusters output by the algorithm and the ground-truth. The higher the score the more similar two partitions are.

For a game with an underlying graph $\{\mathcal{N}(n)\}_n$, we define the *graph score* for the learned graph $\{\hat{\mathcal{N}}(n)\}_n$ as $GS = \frac{1}{N}\sum_n \frac{|\mathcal{N}(n)\cap\hat{\mathcal{N}}(n)|}{|\mathcal{N}(n)|} \in [0,1]$, measuring how well the learned graph resembles the original.

3.6.1 Random Role-Symmetric Games

We first test on a 300-agent 3-action, 3-role random role-symmetric game. We evaluate K-Roles choosing $\hat{K} = 3$ against the method of [Ficici et al., 2008] trained with 100 and 1000 data points, denoted as FPP-100 and FPP-1000 respectively. We maintain a data buffer of size 1000, and query 100 data points as $\mathcal{D}_{val}$ in each iteration. For regression of deviation function approximators, we use a neural network with two hidden layers of sizes 32 and 16.

As shown in Figures 3.4(a) and 3.4(b), K-Roles is able to find better solutions than FPP within a few iterations. Figure 3.4(c) shows that the procedure also succeeds in recovering the ground-truth role partition quickly, starting from a random initial assignment. The progress in finding better solutions with iteration can be attributed to both improved cluster representation and accumulation of data for payoff training.

3.6.2 Biased Voting Game

The biased voting game [Kearns et al., 2009] is a graphical game model designed to capture a tradeoff between maximizing one's own preferences and coordinating with neighbors. In an M-party biased voting game, agent n has a *preference score* $s_{n,m}$ for each party m. And if $f_{n,m}$ fraction of n's neighbor votes for party m, the payoff of n voting party m is $s_{n,m}f_{n,m}$.

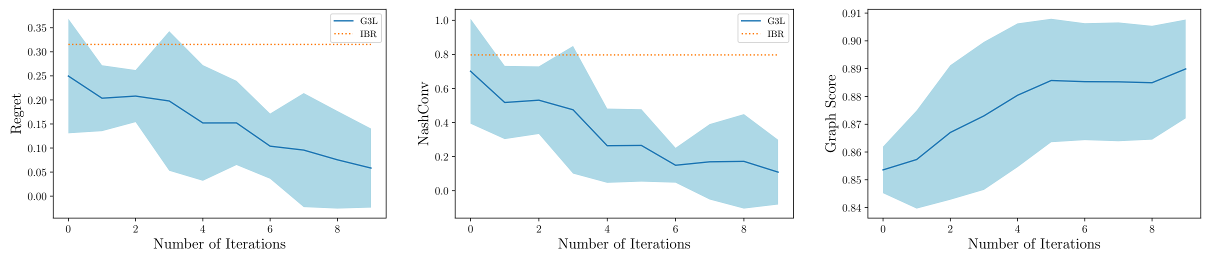
We first test on three games with different degrees of solving difficulty. In each experiment we query 1000 data points for one shot. We adopt Iterated Best Response (IBR) as our benchmark with the same number of data points queried. In each round of IBR a player is randomly selected to make a best response to the current state. IBR is guaranteed to reach pure-strategy Nash equilibrium (PSNE) for network games with strategic complements [Jackson, 2010, Section 9.3.3], however in general for network games and the biased voting game in particular, PSNE may not even exist.

We set $\hat{\kappa} = 6$ if $M = 2$ and $\hat{\kappa} = 4$ when $M = 3$. The results are shown in Table 3.1. We observe that choosing a large $\hat{\kappa}$ typically results in a fairly accurate graphical model, but since the size of payoff table for the learned game grows exponentially with $\hat{\kappa}$, we need to sacrifice model accuracy for efficient equilibrium computation when facing moderate M. Nevertheless the solutions returned by G3L exhibited better quality than IBR in all instances tested.

We then perform an iterated version of the experiment $N = 100, M = 2$, where buffer size is 1000 and 100 data points are queried in each iteration. As shown in Figures 3.5, G3L beats the

Table 3.1: Performance of G3L versus IBR on biased voting game instances. The entry format in the second and third columns is REGRET(NASHCONV). The last column gives the graph scores for the graphs learned by G3L. All results are averaged over 20 runs.

Game Instance	G3L	IBR	G3L GS
$N = 100, M = 2$	0.165 (0.337)	0.315 (0.797)	0.907
$N = 100, M = 3$	0.298 (1.447)	0.513 (1.601)	0.698
$N = 200, M = 2$	0.075 (0.207)	0.661 (5.178)	0.915



(a) Regret (b) NashConv (c) Graph Score with Ground-truth

Figure 3.5: Performance of G3L over iterations. The results are averaged over 10 runs

.

baseline even at the first iteration, and its capability of finding a good solution as well as recovering the graph structure evidently improves over iterations.

3.6.3 Criminal Network Game

In the peer-effect *criminal network game* studied by Bramoullé et al. [2014], each agent n is embedded in a graph G and has to choose a criminal level $a_n \in [0, 1]$. The payoff for agent n is defined as $u_n = y_n - \zeta \cdot x_n$. The symmetric game term y_n is a function of the total criminal levels of this network, capturing the competing effects and satisfying conditions $\frac{\partial y_n}{\partial a_n} \geqslant 0, \frac{\partial y_n}{\partial a_{n'}} \leqslant 0, \forall n' \neq n$. The graphical game term x_n measures peer-effects and satisfies $\frac{\partial x_n}{\partial a_n} \geqslant 0, \frac{\partial x_n}{\partial a_{n'}} \leqslant 0, \forall n' \in \mathcal{N}(n)$.

For our purposes, the most interesting feature of the criminal network game is that by varying the structure parameter $\zeta \geqslant 0$ for fixed y_n, x_n, we obtain a spectrum of games between perfect symmetry and perfect sparsity: With greater $\zeta \geqslant 0$ the game is closer to a graphical game as opposed to a symmetric game, and vice versa.

We let $M = 3$ by constraining the criminal level $a_n \in \{0, 0.5, 1\}$. We fix linear-quadratic functions for both y_n and x_n and vary ζ. The results of K-Roles and G3L are based on a single iteration of game learning with 1000 query samples. The solution qualities of the methods for game points of different structures are plotted in Figure 3.6. We find that K-Roles and G3L, respectively, out-

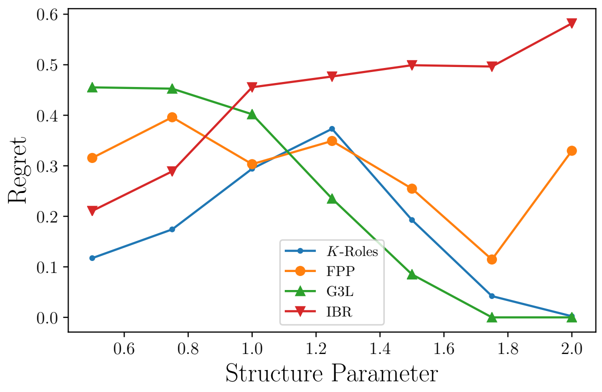


Figure 3.6: Performance of K-Roles, FPP, G3L, IBR on a 100-player, 3-action criminal network game instance. $\hat{K} = 2$ for K-Roles and FPP. Results are averaged over 10 runs.

perform the others when the game is closer to the symmetry or the sparsity extreme. Interestingly, when the game approaches a graphical game, K-Roles is able to discover an approximate role structure when the sparsity increases, with solutions nearly as good as those found by G3L. This suggests that symmetry can arise from sparsity in a game, and validates the efficacy of K-Roles in revealing such structure.

3.7 Conclusion

Scaling game modeling to large numbers of players perhaps inevitably requires some structural regularity in the situation and interactions of the agent population. We proposed an approach to reasoning about many-player games based on iterative structure learning, given black-box access to noisy payoff samples for a target game. Starting from a basic structural hypothesis, our method learns a candidate structure and associated game model, then gathers additional training data based on this candidate to learn a refined model. We instantiated this approach for two very different forms of structural hypothesis: role symmetry, and locality of interaction. Our K-Roles algorithm, inspired by k-means clustering, combines supervised and unsupervised techniques to learn a role-symmetric game model. Our G3L iteratively learns a graphical game model.

We performed computational experiments on three relevant albeit stylized games, on instances with at least 100 agents. One game exhibits strict role structure, the second strict graph structure, and a third has structure but neither strictly. We found that both methods achieved good performance for structure learning in the models with clear structure, and both also demonstrated advantages of the iterative structure-learning approach to equilibrium seeking.

It is important to note that the identification of symmetry by K-roles assumes that we already have a common set of actions for the agents. In other situations we may have different action sets, which would just impose a constraint on the clustering process. In a more general version of the problem, identification of correspondences between actions of different agents would itself be something that would have to be learned; as yet we have not considered how to extend K-roles in that direction.

For graphical game learning, action correspondence is not a concern. A possible focus for improvement of G3L would be more explicit management of the tradeoffs in maximum neighborhood size $\hat{\kappa}$, considering simultaneously its affect on learning (*i.e.*, use as a regularizer) and on game-theoretic computation with the resulting model.

Future work could encompass these issues as well as many other extensions to cover broader classes of games and additional forms of game structure.

3.8 Appendix: Implementation Details

In addition to the vanilla version of clustering method we described in the chapter which only uses $\nabla_{\boldsymbol{\sigma}^*}\hat{u}$ resulted from $\mathcal{D}_{val}$, it may be more robust to utilize the information contained in the data buffer $\mathcal{D}$. Thus in our implementation, at each iteration we first construct another embedding for agent n as a vector $\nabla\bar{u}_n \in \mathbb{R}^M$ based on $\mathcal{D}$ in the same way we construct $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_n$, and finally apply the average of $\nabla\bar{u}_n$ and $\nabla_{\boldsymbol{\sigma}^*}\hat{u}_n$ as the final input embeddings of n for k-means. For hierarchical clustering we also utilize the information in $\mathcal{D}$ by using the average of per-iteration distances calculated so far as the pre-computed distance matrix input. We adopt default configuration for k-means and adopt average linkage method for hierarchical clustering in Scikit-learn.

We implement a generalized version of the one adopted by Ficici et al. [2008] that support $\hat{K} > 2$. We follow the setting that in the view of cluster $\mathbb{C}$, $Pr(s_i|\mathbb{D}) = 0.5$, when only one of $\mathbb{D}$ and its twin $\mathbb{D}'$ chooses s_i, while other aspects of extension is straightforward. We use the global Newton method in GAMBIT here as the solver for the reduced game. In the original paper the authors propose a twin-reduction technique to consider the deviation effect. However we empirically found that GAMBIT may not always return a twin-symmetric equilibrium for a twin game. So in this case, we would switch back to a non-twin game and use the equilibrium of that game as the answer.

We implement function minimization by using the minimize method in SciPy software package. This method seek for $\boldsymbol{\sigma}$ that minimizes $\sum_n \sum_m (\max\{u_n(m, \boldsymbol{\sigma}) - u_n(\boldsymbol{\sigma}), 0\})^2$. The subroutine typically will find a local minimum of the objective function. So in each iteration we solve the learned role-symmetric games 100 times each with different initial solution and select the lowest scored one as the solution of current iteration.

For Govindan-Wilson algorithm in [Blum et al., 2006], the main computation bottleneck lies on how to compute the adjugate matrix for the payoff Jacobian. If the Jacobian matrix is non-singular, it will be easy since we could just computes its determinant and inverse using existing efficient algorithm, and then multiple them together. If the Jacobian is singular, however, we adopt method proposed in [Stewart, 1998] to calculate the adjugate. That is, first do a SVD decomposition on the Jacobian matrix, then do a small perturbation on the diagonal matrix, and finally multiply these matrices back. This practically works more efficient than the one directly calculating the minors. We enable "wobble" trick and adopt adaptive step size to speed up convergence. Other parameters are the same as the implementations in GameTracer.

For all experiments we calculate REGRET and NASHCONV for a given profile by sampling 1000 data points for every deviation payoffs within the oracle. For all iterative experiments we start the random guess $\boldsymbol{\sigma}^*$ as the uniform strategy.

The detailed configurations for specific game class is as follows.

Random Role-Symmetric Game: For each action m, we generate a vector $A_m \in \mathbb{R}^{KM}$ and a scalar b_m from the normal distribution. Then for each role k and action m, we generate a vector $A_{m,k}$(and a scaler $b_{m,k}$) from a Gaussian distribution with mean $A_m(b_m)$, and covariance matrix(variance) $0.5^2\boldsymbol{I}(0.5^2)$ respectively. Then for an action counts vector $\boldsymbol{f}$, we define the payoff for role k choosing action m as $\cos(5 \cdot (A_{m,k}\boldsymbol{f} + b_{m,k}))$.

Biased Voting Game The *preference scores* $s_{n,m}$ are all generated from the uniform distribution on $[0, 1]$.

Criminal Network Game: We let $y_n = a_n(1-\alpha\frac{\sum\limits_{i\in\mathcal{N}} a_i}{N})$ with $\alpha = 1.5$ and $x_n = a_n(1-\phi\frac{\sum\limits_{i\in\mathcal{N}(n)} a_i}{|\mathcal{N}(n)|})$ with $\phi = 1.2$

CHAPTER 4

Deep Evolutionary Search for Solving Large Bayesian Games

We address the problem of solving complex Bayesian games, characterized by high-dimensional type and action spaces, many (> 2) players, and general-sum payoffs. Our approach applies to symmetric one-shot Bayesian games, with no given analytic structure. We represent agent strategies in parametric form as neural networks, and apply natural evolution strategies (NES) [Wierstra et al., 2014] for deep model optimization. For pure equilibrium computation, we formulate the problem as bi-level optimization, and employ NES in an iterative algorithm to implement both inner-loop best response optimization and outer-loop regret minimization. In simple games including first- and second-price auctions, it is capable of recovering known analytic solutions. For mixed equilibrium computation, we adopt an incremental strategy generation framework, with NES as strategy generator producing a finite sequence of approximate best-response strategies. We then calculate equilibria over this finite strategy set via a model-based optimization process. Both our pure and mixed equilibrium computation methods employ NES to efficiently search for strategies over the function space, given only black-box simulation access to noisy payoff samples. We experimentally demonstrate the efficacy of all methods on two simultaneous sealed-bid auction games with distinct type distributions, and observe that the solutions exhibit qualitatively different behavior in these two environments.

4.1 Introduction

Bayesian games [Harsanyi, 1967] model incomplete information by encoding uncertainty over opponents' hidden information in terms of beliefs over player *types*. Types are drawn from a common knowledge prior distribution. Each player is informed of its own type, and decides its action strategically as a function of this private information. This framework provides a standard model for many economic games, such as auctions, and has informed the design of many real-world market-based systems, including mechanisms for online advertising. In this chapter, we focus on one-shot, *symmetric Bayesian games* (SBG), where both type space and action space are subsets of multidimensional Euclidean spaces [McAdams, 2003]. Multi-object auctions [Christodoulou et al., 2008] provide a canonical example of this game class, with types as parameter vectors defining valuations for sets of goods, and strategies mapping such types to bids for these goods.

A *Bayes-Nash equilibrium* (BNE) is a configuration of strategies that is stable in the sense that no player can increase the expected value of its outcome by deviating to another strategy, given its belief over other players' types. As our games are symmetric, it is natural to seek symmetric BNE, and to exploit symmetric representations for computational purposes. A classical approach [Milgrom and Weber, 1982, McAfee and McMillan, 1987] for deriving pure symmetric BNE is to solve a differential equation for a fixed point of an analytical best response mapping. However for many SBGs of interest, analytic solution is hindered by irregular type distributions and nonlinear payoffs, and dimensionality of type and action spaces.

Instead of seeking analytical solutions, we formulate the problem of computing BNE as a high-dimensional optimization, and present computational results that bridge classical economic theories and modern AI techniques. We parameterize agent strategies as neural networks, therefore approximate the original functional strategy space as a high-dimensional vector space of network weights. Our algorithmic approach is based on *natural evolution strategies* (NES) [Wierstra et al., 2014], a black-box optimization algorithm based on stochastic search in the parameter space. NES has been shown a competitive optimization technique for non-smooth environments like those often tackled by RL [Salimans et al., 2017]. We consider it well suited for our purpose of equilibrium computation, as SBGs typically exhibit large payoff discontinuities [Reny, 1999].

We present methods for computing both pure and mixed BNE (PBNE and MBNE). With player symmetry, computing PBNE can be cast as a minimax optimization. Our proposed algorithm employs one NES process to implement an approximate best response operator, and another NES to search for an approximate fixed-point of this operator via regret minimization. This approach to some extent mirrors the classical analytic approach mentioned above.

Finding stable profiles in pure strategies may be difficult for games endowed with complex strategic landscape. Indeed the existence of PBNE can be assured only with certain assumptions such as regularity of type distributions [Milgrom and Weber, 1985]. To search for approximate MBNE, we construct finite approximations of the original infinite game by discretizing the strategy space [Dasgupta and Maskin, 1986], and consider mixed equilibria of these restricted games. We resort to an incremental strategy generation framework [McMahan et al., 2003] to implement this approach, where NES again is employed as a strategy generator producing a finite sequence of approximate best-response strategies. We then extract strategic information of a restricted strategy set by regressing a finite game model via supervised learning, and calculate a mixed equilibrium of this learned game as the solution.

4.2 Related Work

Although there is a rich literature in economics on characterizing properties of equilibria in Bayesian games [Milgrom and Weber, 1985, Athey, 2001, McAdams, 2003, Reny, 2011], the problem of solving for BNE computationally is less explored. It is known that determining the existence of a PBNE for discrete strategy space (where both type and action are discrete) is NP-complete [Conitzer and Sandholm, 2008], and computing PBNE for simultaneous Vickrey auctions is even PP-hard [Cai and Papadimitriou, 2014]. These seem to discourage attempts to develop general solvers for Bayesian games.

However efforts directed to specific classes of Bayesian games has not been inhibited, and over the years substantial results have been obtained both theoretically and empirically. Some work focuses on the two-player case: Reeves and Wellman [2004] derived analytical best response for Bayesian game with piece-wise linear payoff functions, and Ceppi et al. [2009] gave experimental results on the support-enumeration approach. Other research addresses symmetric Bayesian games with more players. Vorobeychik and Wellman [2008] used self-play implemented by simulated annealing to find PBNE in SBGs. We likewise employ a form of stochastic search, but avoiding the limitations of self-play [Balduzzi et al., 2019] and with methods that extend to high-dimensional settings. Wellman et al. [2017] tackled simultaneous Vickrey auctions by applying empirical game-theoretical analysis on a set of hand-crafted strategies solving for MBNE. They identified effective heuristic bidding strategies, however these strategies generally require computation exponential in the number of goods.

Rabinovich et al. [2013] considered Bayesian games with continuous type space and discrete action space, and provided both theoretical convergence results as well as experimental validations for the fictitious play algorithm. Recent work [Wang et al., 2020] gave computational results for first-price auctions with discrete type space and continuous action space. For a more general class of multi-item auction, works [Christodoulou et al., 2008, Dütting and Kesselheim, 2017] studied algorithms of theoretical interest for combinatorial Vickrey auctions, and Bosshard et al. [2017] presented experimental results on applying grid search to solve the Local-Local-Global auctions. For succinct game models, works [Singh et al., 2004] and [Jiang and Leyton-Brown, 2010] formulated the notion of incomplete information in graphical games and action-graph games, together with corresponding computational methods, respectively. Armantier et al. [2008] approximated the Bayesian game to be solved by a sequence of constrained games, and designed numerical methods by solving approximate partial differential equations for equilibrium computation. Nevertheless, none of the above works developed a computational method that scales with player numbers as well as the dimensions of type space and action space.

Solving Bayesian games is also relevant to real-world applications such as advertising auctions.

Chawla and Hartline [2013] proved the uniqueness of symmetric BNE for a class of rank-based auction formats including generalized first-price auctions. Gomes and Sweeney [2014] derived a closed-form expression of the symmetric PBNE for generalized second-price auctions.

In work independent and concurrent with ours, Heidekrüger et al. [2021] also employ NES to compute BNE. Whereas we solve the game from a central perspective, they model the result of decentralized learning agents. Other key differences are that we assume and exploit symmetry, and develop methods to compute MBNE as well as PBNE. They in contrast consider a general asymmetric setting and focus on finding PBNE.

4.3 Preliminaries

4.3.1 Bayesian Games

One-shot, simultaneous-move symmetric Bayesian games provide a standard model for common strategic scenarios such as auctions [Krishna, 2009]. Formally, an SBG consists of $(\mathscr{N}, \mathscr{T}, \mathscr{A}, \mathcal{P}, u)$, with $\mathscr{N} = \{1, \ldots, N\}$ being the set of agents, $\mathscr{T}$ the set of types and $\mathscr{A}$ the set of actions. We focus on the case where $\mathscr{T}$ and $\mathscr{A}$ are compact subsets of $\mathbb{R}^T$ and $\mathbb{R}^A$, with T and A the dimensions of type and action space, respectively.

In a play of the SBG, agents simultaneously and privately observe their types, drawn i.i.d. from a common knowledge distribution $t \sim \mathcal{P}$, then independently choose their respective actions conditional on this private information. More formally, an agent's *pure strategy* s is a deterministic mapping from the type space to the action space $s : \mathscr{T} \to \mathscr{A}$. We denote the set of all such mappings as $\mathscr{S}$. For computational purposes in this chapter we represent a pure strategy as a neural network. Though the space of multilayer perceptrons of a fixed architecture may not perfectly coincide with $\mathscr{S}$, in effect we can represent any strategy to a close approximation due to the expressive power of deep models.

For playing action $a_n = s_n(t_n)$, agent n receives a real-valued payoff $u(a_n, \boldsymbol{a}_{-n} \mid t_n) : \mathscr{A} \times \mathscr{A}^{N-1} \times \mathscr{T} \to \mathbb{R}$, which depends on its own type and action along with the profile of other agents' actions $\boldsymbol{a}_{-n} \in \mathscr{A}^{N-1}$. Player symmetry entails that one's payoff value is permutation-invariant to the opponents' actions: $u(a_n, a_{\pi(1)}, \ldots, a_{\pi(N)} \mid t_n) = u(a_n, a_1, \ldots, a_N \mid t_n)$ for any permutation π of order $N - 1$. For a given strategy profile $(s_1, \ldots, s_N)$, the *ex interim* (EI) payoff is the expected payoff marginalized over opponents' types: $u(s_n, \boldsymbol{s}_{-n} \mid t_n) \triangleq \mathbb{E}_{\boldsymbol{t}_{-n} \sim \mathcal{P}^{N-1}}[u(s_n(t_n), s_1(t_1), \ldots, s_N(t_N)) \mid t_n)]$, and the *ex ante* (EA) payoff averages over one's own type randomness $u(s_n, \boldsymbol{s}_{-n}) = \mathbb{E}_{t_n \sim \mathcal{P}}[u(s_n, \boldsymbol{s}_{-n} \mid t_n)]$.

A *mixed strategy* $\sigma \in \Delta(\mathscr{S})$ defines a probability measure over the pure strategy space and allows one to make stochastic decisions.[1] For a given mixed strategy profile $(\sigma_1, \dots, \sigma_N)$, the EI and EA payoffs for agent n of type t_n are $u(\sigma_n, \boldsymbol{\sigma}_{-n} \mid t_n) \triangleq \mathbb{E}_{s_n \sim \sigma_n, \boldsymbol{s}_{-n} \sim \boldsymbol{\sigma}_{-n}}[u(s_n, \boldsymbol{s}_{-n} \mid t_n)]$ and $u(\sigma_n, \boldsymbol{\sigma}_{-n}) \triangleq \mathbb{E}_{t_n \sim \mathcal{P}}[u(\sigma_n, \boldsymbol{\sigma}_{-n} \mid t_n)]$.

For SBGs we are often interested in one's payoff given all $N-1$ others adopt the same strategy. We write $u(\sigma', \boldsymbol{\sigma})$ for the EA payoff of an agent choosing strategy σ' while the rest choose σ. For strategy σ and pure strategy s, we call $u(s, \boldsymbol{\sigma})$ the *deviation payoff* to pure strategy s against opponent mixture σ, or the *fitness* of s under σ.

4.3.2 Bayes-Nash Equilibrium

In a symmetric profile, every agent adopts the same strategy. For symmetric games, we generally prefer to find solutions in symmetric profiles, and these are guaranteed to exist in a variety of settings [Nash, 1951, Cheng et al., 2004, Chawla and Hartline, 2013, Hefti, 2017]. The *symmetric regret* of strategy σ is given by $\textsc{Regret}(\sigma) \triangleq \max_{s \in \mathscr{S}} u(s, \boldsymbol{\sigma}) - u(\sigma, \boldsymbol{\sigma})$. Our goal is to search for σ that minimizes loss $\textsc{Regret}(\sigma)$, in other words, a symmetric profile where each agent approximately best responds to the others. If $\textsc{Regret}(\sigma) \leqslant \epsilon$, we call σ an (EA) ϵ-BNE. For the pure case (*i.e.*, $\sigma \in \mathscr{S}$) we more specifically label σ an (EA) ϵ-PBNE, and for the mixed case an (EA) ϵ-MBNE.

4.3.3 Black-Box Games

While many classic Bayesian games possess closed-form solutions [Krishna, 2009], many others are intractable via analytical reasoning. To develop computational methods for general SBGs of interest, therefore, we adopt a more universal formulation and represent the game by a black-box payoff oracle $\mathcal{O} : \mathscr{S}^N \times \Omega \to \mathbb{R}^N$. In this black-box (also termed *simulation-based*) setting, the basic operation is a query by the game analyst, who submits a joint pure strategy $\boldsymbol{s} \in \mathscr{S}^N$ to $\mathcal{O}$, and receives in return a payoff vector $\boldsymbol{u} \in \mathbb{R}^N$ recording payoffs for each agent realized through agent-based simulation (with a random seed $\omega \in \Omega$). We assume the analyst is aware of $\mathscr{N}, \mathscr{T}, \mathscr{A}$ and player symmetry, but neither the type distribution $\mathcal{P}$ nor a direct representation of the payoff function u. Therefore the goal is to find approximate equilibria given only stochastic black-box simulation access to the game.

[1]A *behavioral strategy* $b : \mathscr{T} \to \Delta(\mathscr{A})$ is a mapping from the type space to the space of probability measures on the action space. It turns out that mixed and behavioral strategies can each be shown equivalent to the other under certain conditions [Milgrom and Weber, 1985].

Algorithm 4 Natural Evolution Strategies

```
1: Input: Black-box function $F$, hyperparameters $J, \alpha, \nu$
2: Output: Approximate maximum $\boldsymbol{\theta}$ of $F$
3: Initialize $\boldsymbol{\theta}$;
4: for $i = 1, 2, \ldots$ do
5:     Sample $\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_J \sim \mathcal{N}(0, \boldsymbol{I})$;
6:     $\forall j, r_{j+} \leftarrow F(\boldsymbol{\theta} + \nu\boldsymbol{\varepsilon}_j), r_{j-} \leftarrow F(\boldsymbol{\theta} - \nu\boldsymbol{\varepsilon}_j); r_j \leftarrow r_{j+} - r_{j-}$;
7:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\frac{1}{J\nu}\sum_j r_j\boldsymbol{\varepsilon}_j$;
8: end for
9: return $\boldsymbol{\theta}, F(\boldsymbol{\theta})$;
```

4.3.4 Natural Evolution Strategies

NES [Wierstra et al., 2014] belongs to a family of black-box optimization algorithms called *evolution strategies* (ES) that can be viewed as abstract versions of natural selection processes. The goal of NES is to maximize a fitness function $F(\boldsymbol{\theta})$ with respect to neural network weights $\boldsymbol{\theta}$, given only stochastic black-box access to its point values. Instead of directly optimizing F, the idea is to construct gradient estimators of a *Gaussian smoothing objective* $\mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(0,\boldsymbol{I})}[F(\boldsymbol{\theta}+\nu\boldsymbol{\varepsilon})]$ with bandwidth hyperparameter ν, and apply stochastic gradient ascent accordingly. As a technique for optimizing neural networks, NES has been shown competitive with other back-propagation based methods of deep RL [Salimans et al., 2017]. For the purpose of equilibrium computation, we employ NES as a subroutine to optimize neural strategies for different choices of fitness functions.

We elaborate our version of NES in Algorithm 4. Initialized with network weights $\boldsymbol{\theta}$, on each iteration NES constructs a finite difference approximation of the gradient through random search over the space of $\boldsymbol{\theta}$, and updates the deep model towards the direction of higher expected fitness values. To construct such gradients, NES first samples a population of noisy vectors $\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_J$ in the parameter space from a normal distribution. For each of these sampled $\boldsymbol{\varepsilon}$, it perturbs $\boldsymbol{\theta}$ into a pair of antithetic variables ($\boldsymbol{\theta} \pm \nu\boldsymbol{\epsilon}$), and evaluates the corresponding fitness values as $r_\pm$. This ensures that the quantity $v = \frac{1}{2\nu}(r_+ - r_-)\boldsymbol{\varepsilon}$ is an unbiased estimator of $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\varepsilon}}[F(\boldsymbol{\theta} + \nu\boldsymbol{\varepsilon}))]$, and the stochastic gradient is constructed as the empirical average of v enabling gradient ascent updates.

4.4 Computing Pure Equilibrium via Minimax Optimization

In this section, we demonstrate how to compute PBNE in SBGs by exploiting game structure and the power of NES as a black-box optimizer. We show that due to player symmetry the problem of calculating a symmetric PBNE is equivalent to computing a minimax equilibrium in a reduced two-player zero-sum game, and therefore design a co-evolutionary algorithm (Algorithm 5) implemented by two NES processes to solve this bi-level optimization problem. In this approach,

Algorithm 5 Minimax-NES for PBNE

```
Input: Payoff Oracle $\mathcal{O}$, hyperparameters $J_1, J_2, \alpha_1, \alpha_2, \nu_1, \nu_2$
Output: Approximate PBNE $s_{\boldsymbol{\theta}}$
1 Function MinusRegret($\boldsymbol{\theta}$)
2     $V \leftarrow \mathcal{O}(s_{\boldsymbol{\theta}}, \boldsymbol{s}_{\boldsymbol{\theta}})$; $\boldsymbol{\theta}', DEV \leftarrow$ NES($\mathcal{O}(\cdot, \boldsymbol{s}_{\boldsymbol{\theta}}), J_1, \alpha_1, \nu_1$); return $V - DEV$;
3 Algorithm MiniMax()
4     return NES(MinusRegret, $J_2, \alpha_2, \nu_2$);
```

we employ one NES to compute an approximate best response and a regret value for a given pure strategy in the inner loop, and another NES in the outer loop to minimize this approximate regret function over the pure strategy space. We next develop the details of the algorithm.

4.4.1 A Minimax Formulation

Recall that computing PBNE is equivalent to minimizing REGRET over the pure strategy space, which can be formulated as

$$\min_{s \in \mathscr{S}} \text{REGRET}(s) = \min_{s \in \mathscr{S}} \max_{s' \in \mathscr{S}} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s}). \tag{4.1}$$

That is, we can view this optimization as a two-player zero-sum game: a primary agent who aims to choose s that minimize REGRET, and an adversary who selects the best response s' against s to implement the REGRET function. This concise formulation exploits player symmetry such that the $N-1$ opponents can be abstracted as one agent when computing a symmetric equilibrium [Hefti, 2017, Vadori et al., 2020].

4.4.2 Inner Loop: NES as the Best Response Optimizer

To optimize objective (4.1), we need to acquire a best-response strategy as well as a maximum deviation value for a given pure strategy $s_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$. To achieve this we utilize NES and the payoff oracle $\mathcal{O}$ to optimize a neural strategy $s_{\boldsymbol{\theta}'}$ as an approximate best response. Since to compute a best response against strategy $s_{\boldsymbol{\theta}}$ is to maximize the deviation payoff function $u(\cdot, \boldsymbol{s}_{\boldsymbol{\theta}})$, we need to provide such stochastic value queries to this deviation function, by refactoring the payoff oracle as $\mathcal{O}(\cdot, \boldsymbol{s}_{\boldsymbol{\theta}})$. $\mathcal{O}(\cdot, \boldsymbol{s}_{\boldsymbol{\theta}})$ takes a deep strategy $s_{\boldsymbol{\theta}'}$ as input and outputs its stochastic fitness values under $s_{\boldsymbol{\theta}}$. This can be implemented by controlling one agent n adopting $s_{\boldsymbol{\theta}'}$ with the rest $s_{\boldsymbol{\theta}}$, sampling this joint profile many times, and taking the average payoff of n across these samples of different type realizations as the final output. Then we pass this wrapped refactored payoff oracle to NES for automatic best-response optimization.

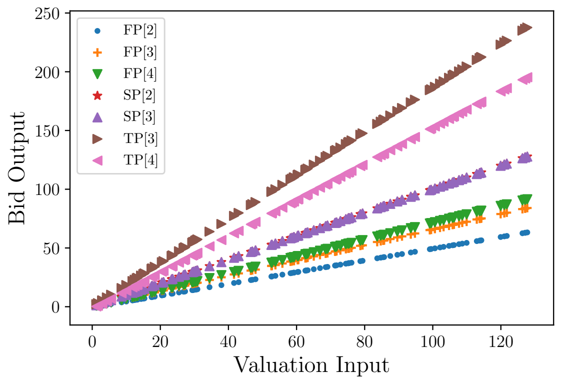


Figure 4.1: Point plots for strategy functions learned by minimax-NES in games with known analytical solutions

4.4.3 Outer Loop: NES as the Regret Minimizer

The inner loop just described provides with an approximate maximum deviation value for any pure strategy, DEV in line 2 of Algorithm 5. We can hence estimate $\text{REGRET}(s_{\theta})$ by first evaluating $u(s_{\theta}, \boldsymbol{s_{\theta}})$ as V via payoff oracle $\mathcal{O}$, shown in line 2 of Algorithm 5, and set the regret estimator as $DEV - V$. To compute an approximate PBNE, we employ another NES in the outer loop that minimizes such surrogate regret function over the pure strategy space. To accomplish that we wrap a black-box function `MinusRegret` that returns $V - DEV$ for a given deep strategy, which is delivered to the outer-loop NES for regret minimization. Hence the interaction between the agent in the outer loop and the adversary in the inner loop each implemented by an NES process enables an efficient algorithm to compute symmetric PBNE in SBGs.

4.4.4 Results for Games with Analytical Solutions

As a preliminary experiment, we investigate the strategies learned by minimax-NES in those SBGs with known analytical solutions. The algorithmic specifications are detailed in Section 4.6.1 and the appendix 4.8. Specifically, we consider[2] N-player unit-item first-price auctions, second-price auctions and third-price auctions, denoted as *FP*$[N]$ and *SP*$[N]$ and *TP*$[N]$. It is known that for uniform valuation distribution $U[0, \bar{t}]$ the canonical PBNE is $s(t) = (\frac{N-1}{N})t$ for *FP*$[N]$, $s(t) = t$ for *SP*$[N]$ and $s(t) = (\frac{N-1}{N-2})t$ for *TP*$[N]$. We plot the relation between the input valuation and output bid of the strategies produced by minimax-NES in these environments in Figure 4.1, where we let $\bar{t} = 128$. The results show that minimax-NES learns these canonical equilibria. Notice that classical derivations for these PBNE imposed a variety of constraints on the solutions, such

[2]In Section 4.8.4 we also include experiments for games with more complex solutions such as all-pay auctions, but we did not successfully reproduce their canonical analytic results.

as monotonicity [Riley and Samuelson, 1981], while our method only implicitly relies on the expressiveness and differentiability of deep models yet still learned these canonical PBNE. We hypothesize these equilibria are the unique solutions representable by deep neural networks.

4.5 Computing Mixed Equilibrium via Incremental Strategy Generation

4.5.1 Overview

In this section, we consider computing mixed equilibria for SBGs by interleaving strategy exploration and equilibrium calculation. This approach fits a generic paradigm for reasoning games with complex strategy space called incremental strategy generation (ISG). ISG maintains a finite restricted set of pure strategies S that discretizes the intractable strategic landscape, and iteratively enlarges this set via best responses to explore rational regions of the game. A mixture over such representative set of strategies is computed in each iteration, serving as a quality mixed equilibrium of the full game. ISG had been shown great success in domains of two-player zero-sum stochastic games [McMahan et al., 2003], security games [Jain et al., 2011, Bosansky et al., 2015, Wang et al., 2019, Wright et al., 2019], extensive-form games [Bosansky et al., 2014], multiagent RL [Lanctot et al., 2017] and multiplayer team games [Zhang and An, 2020].

We next elaborate the usage of ISG in our SBG context, as diagrammed in Algorithm 6. The ISG framework consists of two components: a meta-solver (MS) and a best response oracle (BR). Initialized as a singleton strategy set, on each iteration of ISG, MS takes the restricted strategy set S resulted from the previous iteration as input and outputs a probability mixture σ over S. The mixture is expected to constitute a quality equilibrium when none of the pure strategies is dominant over the others. Common meta-solvers include self-play (which puts all mass on the last strategy), fictitious play (uniformly mixing over S), replicator dynamics (that computes an NE in the restricted game) and other more developed ones [Balduzzi et al., 2019, Muller et al., 2020].

After MS had output σ, BR generates a new strategy into S that is a(n) (approximate) best response against opponent mixture σ. The functionality of BR is to explore and include strategies in regions where agents are more likely to exhibit rational behavior, producing a more robust strategy population and reinforcing the mixture quality calculated by MS in the next iteration. In our version of ISG we use NES to implement BR (line 6 of Algorithm 6) where in Algorithm 4 we let the fitness function F be $\mathcal{O}(\cdot, \boldsymbol{\sigma})$, being consistent with Section 4.4.

We next discuss our choices of meta-solvers.

Algorithm 6 Incremental Strategy Generation

```
Input: Payoff Oracle $\mathcal{O}$. Meta-solver $MS$. Hyperparameters $J, \alpha, \nu$.
Output: A finite strategy set $S$, a mixed strategy $\sigma$ over $S$.
Initial strategy set $S = \{s_0\}$, a singleton distribution $\sigma$ with $\sigma(s_0) = 1$.
for $i = 1, 2, \dots$ do
    $\sigma \leftarrow MS(\mathcal{O}, S, \sigma)$.
    $s', DEV \leftarrow \texttt{NES}(\mathcal{O}(\cdot, \boldsymbol{\sigma}), J, \alpha, \nu)$.
    $S \leftarrow S \cup \{s'\}$.
end for
```

4.5.2 Fictitious Play

The first choice of meta-solver is fictitious play (FP), which puts a uniform mixture on current strategy set S. FP-type algorithms had been shown great success in a variety of games beyond the version for tabular normal-form games [Rabinovich et al., 2013, Heinrich et al., 2015]. We believe FP is a competitive baseline in our problem, since a uniform mixture may capture the diversity of the strategic landscape to some extent.

4.5.3 Nash Equilibrium

Our second choice of meta-solver is to output a Nash equilibrium of the restricted game with support S, which also reflects the double-oracle algorithm [McMahan et al., 2003]. We adopt projected replicator dynamics (RD) [Lanctot et al., 2017] to reach an equilibrium. Suppose we are given the exact payoff function u, in each iteration with opponent mixture σ, RD will update the probability mass of each pure strategy s by an amount proportional to the its present *aggregated deviation value* $\sigma(s)(u(s, \boldsymbol{\sigma}) - u(\sigma, \boldsymbol{\sigma}))$, after which an L2 projection [Wang and Carreira-Perpinán, 2013] onto $\Delta(S)$ is operated to ensure it remain on the probability simplex. Therefore by evolving the mixture towards strategies with higher fitness values, the converged mixed strategy is expected to be stable at the end of the dynamics, and the new strategies are anticipated to be generated in rational regions.

4.5.3.1 Model Learning

However in our problem, the issue for replicator dynamics or other Nash-solvers is that we do not have the exact evaluation of deviation payoff $u(s, \boldsymbol{\sigma})$ but only its stochastic query values, which may require a significant number of samples to control the variance for each update, and inhibit Nash-solvers from converging to stable solutions. To reduce sample complexity and computational intractability, we adopt a supervised model-learning approach [Vorobeychik et al., 2007, Wiedenbeck et al., 2018, Sokota et al., 2019, Li and Wellman, 2020] that regresses the pure-strategy payoff function of this finite restricted game model, and further provides RD with deviation estimation for mixture computation. The key here is to exploit a succinct game representation. Notice that for finite symmetric games with M strategies, it suffices to record $u(s, f_1, \ldots, f_M)$, where s is the pure strategy chosen by the principal agent and f_m counts the fraction of its opponents choosing strategy m. Then by assuming the ground-truth payoff function varies smoothly with strategy counters, we can use a function approximator to learn such succinct representation by extracting correlations among different pure-strategy payoff entries. This is also reflected in a recent work about multi-agent training [Liu et al., 2022].

More concretely, we regress the restricted game with $|S| = M$ by sampling a dataset from the payoff oracle and learning a value network $\hat{u} : \Delta(S) \to \mathbb{R}^M$ as the empirical game model. The input training feature of $\hat{u}$ is a vector of strategy counters, each dimension counting the fraction of other players choosing the corresponding strategy, and the output targets are the payoffs for each of the M pure strategies when adopted by the principal agent. Then to estimate the deviation payoffs under opponent mixture σ, we directly set σ as the input to the value network, and obtain M values as deviation payoff estimations. We consider this approach applies to finite symmetric games with many players, since the strategy counters are expected to conform the probability mixture due to law of large number.

4.6 Experiments

4.6.1 Setups

4.6.1.1 Algorithmic Configurations

We represent each pure strategy as a two-layer perceptron with hidden node size 32 for each layer separated by ReLU units. The hyperparameters for NES are tuned as follows. We fixed population size $J = 4 + 3\lfloor \log d \rfloor$ as the default setting adopted by Wierstra et al. [2014], where d is the number of parameters of the deep model. For every fitness function we apply a grid search to select the best bandwidth ν and learning rate α within certain ranges, to maximize the performance of the resulted NES. We adopt Adam optimizer [Kingma and Ba, 2014] to automatically adjust the learning rate initialized with α during the stochastic gradient ascent process. In addition to the vanilla NES described in Section 4.3.4, our implementation employs a fitness-shaping trick [Wierstra et al., 2014] to enhance the robustness of the optimization process.

For black-box functions $\mathcal{O}(\cdot, \boldsymbol{s})$ and $\mathcal{O}(\cdot, \boldsymbol{\sigma})$, each query we run an agent-based simulation for 5000 times and take the corresponding average payoff values as the outputs. In the model-learning part of RD, for each empirical game model with $|S| = M$, we draw 2000 vectors of strategy counters from a Dirichlet distribution as training features, and for each feature we calculate M

pure strategy payoffs via 500 Monte Carlo samples as training targets. We adopt a two-layer network with hidden node size 32 each for game model learning.

Please refer to supplementary material [Li and Wellman, 2021a] for more details on implementation and hyperparameter selection.

4.6.1.2 Environments

We test our methods on two simultaneous sealed-bid auctions: market-based scheduling (MBS) [Reeves et al., 2005] and homogeneous-good auctions (HG) [Wellman et al., 2017], both of which are SBGs with multidimensional type space and action space possessing no analytical solution. We elaborate the game mechanisms as follows.

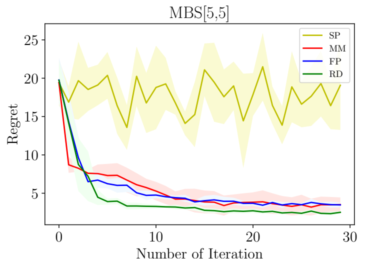
Market-Based Scheduling In an MBS environment, an agent's objective is to acquire enough number of slots to fulfil its task through strategic bidding. More specifically, for an MBS with K slots, each agent is rendered a type vector with dimension $K + 1$: an integer $\Lambda \in [1, K]$ specifying the total number of slots needed, and a valuation vector $v \in \mathbb{R}^K$, where v_k is the valuation realized if it acquired its Λ-th slot at the k-th auction. If it had not obtained Λ slots in the end then its valuation is 0. Λ is drawn from an exponential distribution, while v_k are constructed by first independently drawing K numbers uniformly from $[0, 50]$, and reordering the values to satisfy a non-increasing constraint. Each slot is allocated to the highest bidder and priced via a second-price payment rule, so an action is a bidding vector $b \in \mathbb{R}^K$ specifying the bids for each of the K auctions. The payoff of one agent is the difference between its realized valuation and payment. This auction format exhibits both complementarity and substitutability.

Homogeneous-Good Auctions In an HG of K goods, K Vickery auctions are simultaneously operated for selling homogeneous items. Each agent is rendered a type vector $t \in \mathbb{R}^K$, where t_k is the marginal valuation of acquiring the k-th good. t_1 is drawn uniformly from $[0, 128]$, with t_k drawn uniformly from $[0, t_{k-1}]$ for $k > 1$. An action is a vector of K bids for each of the K auctions. This auction format exhibits perfect substitutability.

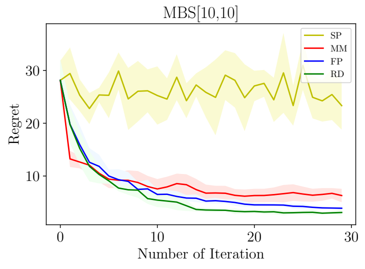
In our experimental presentation we use $MBS[N, K]$ and $HG[N, K]$ to denote MBS and HG with N agents and K goods, respectively.

4.6.1.3 Evaluation Metric and Methods

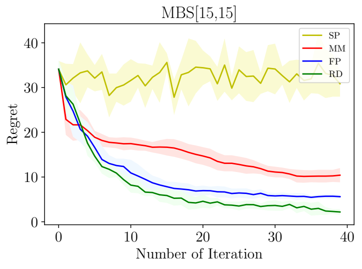
We compare four methods in all of our experiments: self-play and minimax that compute PBNE, with fictitious play and replicator dynamics solving for MBNE. Each method runs for $MAXT$ trials of different random seeds, with each trial continues for $MAXI$ iterations. Each of such iteration generates a new pure strategy, resulting in a restricted pure strategy set $\overline{S}$ with $|\overline{S}| = 4 \times MAXT \times MAXI$ totally. Denote $\sigma_{AL,i,t}$ the strategy output (which could



(a)



(b)



(c)

Figure 4.2: Results for market-based scheduling environments

be either pure or mixed) algorithm AL produces at iteration i of trial number t. Since we do not have an exact best response oracle, we estimate the regret of $\sigma_{AL,i,t}$ in the full game as $\max_{s \in \overline{S}} u(s, \boldsymbol{\sigma_{AL,i,t}}) - u(\sigma_{AL,i,t}, \boldsymbol{\sigma_{AL,i,t}})$, instead of using NES to compute an approximate best response. Then we measure the performance of an algorithm by plotting its regret curve averaged across different trials. We consider this evaluation approach utilizes the results the most and largely reduces the bias introduced by NES as an approximate best response operator. Furthermore to reduce statistical bias of taking maximum during regret calculations, we first estimate each of the $|\overline{S}|$ deviation values via 5000 Monte Carlo samples, select the 10 strategies with highest scores, calculate the deviation values for these 10 again via 50000 samples and take the maximum of these as the final maximum deviation estimation. We next specify the strategy output for each of the four algorithms.

Self-Play (SP): SP is also called iterated best response. At each iteration SP outputs a best response to the pure strategy of the previous iteration.

Minimax (MM): For Algorithm 5, we define one iteration as one step of gradient ascent (line 4 of Algorithm 4) in the outer loop, which outputs a pure strategy.

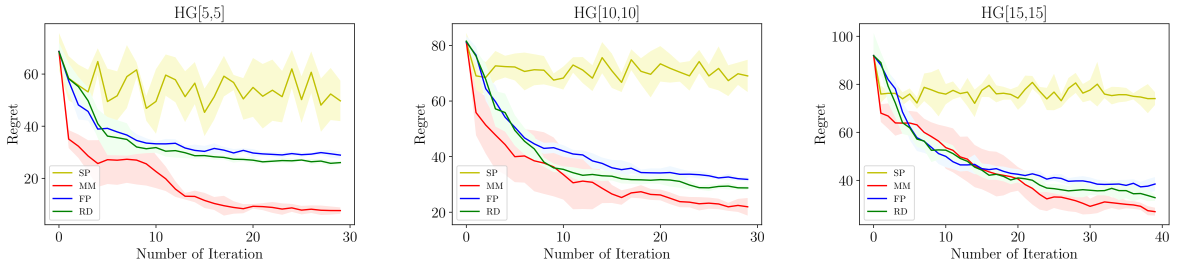
Fictitious Play (FP): On each iteration FP outputs a uniform mixture over S.

Replicator Dynamics (RD): RD outputs a Nash mixture for a restricted game, which is a mixed strategy.

4.6.2 Results

We test our methods on MBS and HG environments of 5, 10, and 15 agents with the same number of goods. Each experiment runs for 5 trials. The results are shown in Figures 4.2 and 4.3.

Our first observation is that self-play performs poorly in nearly all of our experiments. It tends to cycle in the strategic landscape and shows no explicit improvement from the initial strategy. This contrasts with the results of Vorobeychik and Wellman [2008], which were obtained for a different game with much lower-dimensional strategy space. In our context, we find that best responses



(a) (b) (c)

Figure 4.3: Results for homogeneous good environments

cycle around the intransitive regions of the game.

Our second observation is that the RD meta-solver generally outperforms FP. FP outputs a uniform mixture of all strategies explored, which may include strategies generated early on that are not effective against those learned later. Equilibrium based methods, in contrast, will ignore strategies once they are no longer part of a solution. However the observed performance gaps between FP and RD are relatively small. The equilibrium-based method entails much greater computational cost than FP (detailed in supplementary material) due to model learning and training data sampling, thus FP could be an advantageous computational method in certain settings.

We are particularly interested in the performance of minimax-NES, which exhibited qualitatively different behavior in our two experimental environments. In MBS, minimax-NES produces strategies that are generally less robust than the mixed equilibria, whereas in HG it is able to reach pure strategies that surpass both FP and RD in terms of stability. This advantage is especially pronounced in environment $HG[5, 5]$. We attribute the difference to the distinct game characteristics induced by MBS and HG valuations. MBS environments produce more complex strategic landscapes, for (1) its type representation involves Γ as an integer which prevents an atomless type distribution that is usually required for the existence of pure equilibria [Milgrom and Weber, 1985], and (2) complementarity in valuations makes strategy outcomes more sensitive to other-agent behavior. Therefore it may be difficult for minimax-NES to reach a pure strategy region that is robust globally. While for HG the types are vectors of marginal valuations which we hypothesize a representation more amenable for the deep models to extract strategic information.

4.6.2.1 Comparison to Hand-Crafted Strategies

We compare the performance of bidding strategies derived by our methods against state-of-the-art strategies for simultaneous sealed-bid auctions that optimize the bid vector based on probabilistic price predictions. Specifically, our reference is to hand-crafted bidding strategies based on *self-confirming* predictions, defined as probability distributions over prices that result when all bidders

Instance	SP	MM	FP	RD	SC
$MBS[5,5]$	19.1	3.51	3.47	2.51	5.30
$HG[5,5]$	49.7	7.62	28.9	26.0	11.0

Table 4.1: Regret of SC compared with other methods within $\overline{S}$

Instance	SP	MM	FP	RD
$MBS[5,5]$	0.0	3.46	0.74	1.71
$HG[5,5]$	0.45	3.05	0.0	0.0

Table 4.2: Regret of our methods with respect to SC

optimize with respect to these distributions. Such strategies, which we denote SC, were found in an study employing empirical game-theoretical analysis by Wellman et al. [2017] to be effective against a broad range of bidding strategies from prior literature.

The idea of a self-confirming price prediction (SCPP) is that it summarizes opponents' behavior near an equilibrium. Assuming all the $N - 1$ opponents employ some strong heuristic bidding strategies, SC computes an approximate best response to the resulted SCPP, using a hand-crafted bid-generation methods. Since the calculation involves searching for an optimal bundle with respect to predicted price, effectively enumerating all possible bundles, SC generally takes exponential time in the number of goods. This also makes it more expensive to evaluate an action, compared with one forward pass of the neural strategies.

The version of SC we adopt in the following experiments is LocalBid initialized with ExpectedMU64 [Wellman et al., 2017]. Due to the exponential computational cost we are only able to test on $MBS[5,5]$ and $HG[5,5]$ environments.

First we compare the robustness of SC equilibrium to our methods, by measuring their regret values with our restricted set $\overline{S}$, as shown in Table 4.1. The performances of our methods are measured by taking the outputs of the last iteration, averaged across different runs. The results showed that in $MBS[5,5]$ the SC equilibrium surpasses only self-play in terms of stability, while in $HG[5,5]$ it is more robust than methods SP, FP, RD, but is still inferior to MM. This demonstrates that our methods produce comparably or even more quality equilibria than SC globally.

Next we investigate more on the strategic dependencies among these strategies, by testing the robustness of our methods against SC. We here measure the regret values of our methods to SC, shown in Table 4.2. The results shows that SC especially exploits MM in both environments. This validates that MM produces near-equilibrium strategies and SC exploits such class of strategies according to its designs. We further consider the results suggest a strategic dependency among different strategies and illustrate that there exist no single strategy that can outperform the others in all cases.

4.7 Conclusion

In contrast to classic works in auction theory that seek analytical solutions, we formulate the problem of computing BNE as an empirical optimization problem. Scaling computational methods to the dimensions of types and actions as well as player number immediately calls for tractable solution representation and efficient optimization techniques. We found that combining the expressive power of deep models with NES as a black-box optimization technique effectively supports solution of a general class of complex symmetric Bayesian games.

Our pure equilibrium computation method, minimax-NES, parallels the classical analytical approach and could be regarded as a high-dimensional generalization of the global convergence method of Vorobeychik and Wellman [2008]. By exploiting player symmetry, the method employs NES for best-response optimization and regret minimization. Here two NES processes are integrated in one algorithm to reach minimax solutions. Our mixed equilibrium computation method, ISG, employs NES as both best-response optimizer and strategy generator. We tested our methods on two simultaneous-auction games with qualitatively different properties, and found that the mixed equilibria showed lower regret on environments with more complex strategic landscape, while the solutions output by minimax-NES appeared to be more robust in games with smoother topology.

Our methods rely on the power of NES as a function searching tool in the strategy space. But since it is difficult to reach the true optimum in such function space we hypothesize certain biases could be introduced by the algorithm NES itself. Our evaluation approach was designed to mitigate this by measuring regret with respect to all the pure strategies we generated across experiment runs. In future work, we are interested in testing other optimization alternatives including genetic algorithms [Such et al., 2017] and comparing them with NES.

4.8 Appendix

4.8.1 More Implementation Details

All of our deep models and training processes are implemented by PyTorch, with ReLU as the nonlinear activation function and Xavier-uniform as the initialization method. Furthermore to prevent the action values explode to infinity we truncate the output of the last linear layer to $[0, \bar{t}]$, where $\bar{t}$ is the upper bound on the action values. $\bar{t} = 50$ for MBS and $\bar{t} = 128$ for HG. The value networks are trained with SmoothL1Loss with batch normalization.

For implementation of RD, we combine a support enumeration approach with the vanilla RD to search for a low-support Nash equilibrium. More concretely, for current support $\underline{S} \in S$, we test whether there is a strategy in the remaining restricted strategy set $S \backslash \underline{S}$ with beneficial deviation. If the beneficial deviation of a strategy is larger than some threshold, we will add this strategy to support $\underline{S}$, and recalculate the Nash mixture focusing on $\underline{S}$.

Algorithm 7 RD with Support Exploration

```
Input: Finite strategy set $S$, current support $\underline{S} \in S$, payoff oracle $\mathcal{O}$, Nash-Solver RD, initial
       mixture $\sigma$, threshold $E$
Output: Support $\underline{S} \in S$, a mixture $\sigma$ over $\underline{S}$
5  repeat
6      for $s \in S \backslash \underline{S}$ do
7          if $\mathcal{O}(s, \boldsymbol{\sigma}) > \mathcal{O}(\sigma, \boldsymbol{\sigma}) + E$ then
8              $\underline{S} \leftarrow \underline{S} \cup s$
9          end
10     end
11     $\sigma \leftarrow RD(\mathcal{O}, \underline{S})$
12 until Convergence;
```

4.8.2 Time Scales

We test the computational cost for all of our methods, as shown in Table 4.3. The computational time is measured on 2x 3.0 GHz Intel Xeon Gold 6154 with 4 cores and RAM 32GB.

Instance	SP[s]	MM[s]	FP[s]	RD[s]
$MBS[5, 5]$	8.49	272	11.0	240
$MBS[10, 10]$	25.6	780	28.8	1015
$MBS[15, 15]$	52.7	2331	62.0	2767
$HG[5, 5]$	7.22	212	10.7	486
$HG[10, 10]$	21.9	829	27.3	2166
$HG[15, 15]$	53.5	1052	61.1	4015

Table 4.3: Average computational time per iteration

4.8.3 Hyperparameter Selection

The detailed process for tuning hyperparameters is as follows. For minimax-NES we fix $\alpha_1 = 0.01, \nu_1 = 0.1$ as we found the selections differs a little. We tune α_2 by applying a grid search in range $\{0.01, \ldots, 0.1\}$, and ν_2 in $\{0.01, \ldots, 0.1\}$. The configuration we employed in our experiments is provided in Table 4.4.

Instance	α_2	ν_2
$MBS[5,5]$	0.05	0.05
$MBS[10,10]$	0.04	0.04
$MBS[15,15]$	0.02	0.08
$HG[5,5]$	0.05	0.1
$HG[10,10]$	0.06	0.04
$HG[15,15]$	0.05	0.07

Table 4.4: Hyperparameter selection for minimax-NES

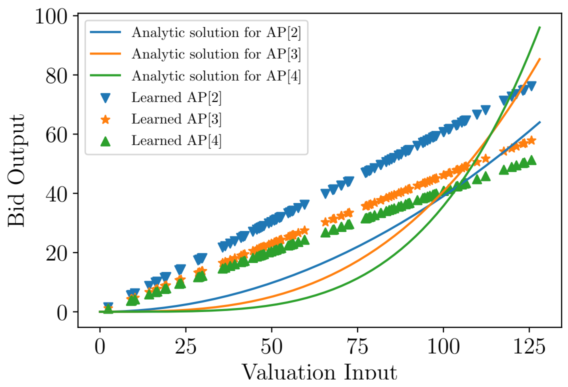


Figure 4.4: Point plot for strategy functions learned by minimax-NES in games with known analytical solutions

4.8.4 More Experiments

We test minimax-NES on unit-item N-player all-pay auction, denoted as $AP[N]$. For uniform distribution $U[0,\bar{t}]$, the canonical solution for all-pay auction is [Krishna, 2009]: $s(t) = (\frac{N-1}{N})\frac{t^N}{\bar{t}^{N-1}}$. As shown in figure 4.4, we find minimax-NES tends to converge to linear solutions under current architecture, where we let $\bar{t} = 128$.

CHAPTER 5

Combining Game Tree-Search and Population-Based Reinforcement Learning for Solving Large Extensive-Form Games

Algorithms that combine deep reinforcement learning and search to train agents, such as AlphaZero, have demonstrated remarkable success in producing human-level game-playing AIs for large adversarial domains. We propose a like combination that can be applied to general-sum, imperfect information games, by integrating a novel search procedure with a population-based deep RL training framework. The outer loop of our algorithm is implemented by Policy Space Response Oracles (PSRO), which generates a diverse population of rationalizable policies by interleaving game-theoretic analysis and deep RL. We train each policy using an Information-Set Monte Carlo Tree Search (IS-MCTS) procedure, with concurrent learning of a deep generative model for handling imperfect information during search. We furthermore propose two new meta-strategy solvers for PSRO based on the Nash bargaining solution. Our approach thus combines planning, inferring environmental state, and predicting opponents' strategies during online decision-making. To demonstrate the efficacy of this training framework, we evaluate PSRO's ability to compute approximate Nash equilibria in benchmark games. We further explore its performance on two negotiation games: Colored Trails, and Deal-or-No-Deal. Employing our integrated search method, we conduct behavioral studies where human participants negotiate with our agents. We find that search with generative modeling finds stronger policies during both training time and test time, enables online Bayesian co-player prediction, and can produce agents that achieve comparable social welfare negotiating with humans as humans trading among themselves.

5.1 Introduction

Computer game research has witnessed tremendous progress over the past decade, marked prominently by the development of human-level game-playing bots in the games of Go [Silver et al., 2018], Poker [Brown et al., 2020, Schmid et al., 2023], and Diplomacy [Bakhtin et al., 2023]. Two broad algorithmic techniques are primarily responsible for this success: (1) deep reinforcement learning (RL) and (2) game-tree search. Deep RL methods are capable of training quality value functions or policies represented by neural nets which generalize well across large state spaces. Search techniques such as Monte Carlo Tree Search (MCTS) [Browne et al., 2012] leverage computational resources at decision time to improve the strength of a strategy. AlphaZero [Silver et al., 2018] provides an elegant framework that coherently combines the power of both methods: a deep policy-and-value network (PVN) is trained using self-play trajectories generated by MCTS, and the updated PVN further guides the search procedure and improves the quality of the trajectory data. By iteratively training the PVN and simulating self-play matches, AlphaZero produces progressively stronger play, which eventually surpass professional human players without any human data. Outside recreational game domains, AlphaZero-style methods also achieved remarkable successes in discovering faster matrix completion methods [Fawzi et al., 2022] and sorting algorithms [Mankowitz et al., 2023].

AlphaZero was originally designed to master large, adversarial, perfect-information games. There are several barriers to generalization of this approach to general-sum, imperfect information domains. First, self-play training is specifically geared to two-player zero-sum domains and implicitly depends on transitivity of the game [Balduzzi et al., 2019]. For games that are not purely adversarial, issues like *equilibrium selection* appear: agents trained entirely through self-play optimize to their opponents at training time, thus may not perform well to opponents at test time, which may correspond to alternative equilibria. In cooperative settings like coordination or common-interest games, this issue can be alleviated by publishing the algorithms and random seeds as mutual knowledge among players [Foerster et al., 2019, Lerer et al., 2020]. However, this is generally an unreasonable assumption for games involving mixed cooperative and competitive elements. Population-based training methods provide one approach to dealing with this issue. By training against a diverse population of opponents, the agent optimizes against a variety of opponent strategies. Population-based training has shown success in pure coordination settings [Lupu et al., 2021] as well as completely adversarial games [Vinyals et al., 2019].

A second major barrier is reasoning with imperfect information. In partially observable environments (e.g., Poker), an agent needs to maintain its belief over world states (e.g., the hands of the opponents) during a planning procedure. Specific techniques such as counterfactual regret minimization (CFR) were developed [Zinkevich et al., 2008, Brown et al., 2020, Schmid et al., 2023] for computer poker, where belief states can be characterized exactly [Moravčík et al., 2017]. Exact reasoning about belief states can be intractable for domains with more complex forms of imperfect information, such as Stratego [Perolat et al., 2022]. Approaches such as particle filtering may be applicable [Silver and Veness, 2010], but are also subject to scaling challenges.

We propose a general-purpose multiagent RL training regime to address the above issues, and extend AlphaZero-style RL and MCTS methods to large general-sum, imperfect information domains. The outer loop adopts a population-based training framework instantiated by *Policy Space Response Oracles* (PSRO) [Lanctot et al., 2017]. PSRO incrementally generates a set of diverse opponents by repeating the following two steps: the *i) meta-strategy solver (MSS) step*, which computes a distribution over existing strategies via empirical game-theoretic analysis (EGTA) [Wellman, 2006], and *ii)* the *best response (BR) step*, which computes approximate best response policies using deep RL against the MSS distribution, adding them to the pool. This procedure effectively builds a belief hierarchy consisting of game-theoretic rationalizable strategies [Bernheim, 1984], bearing some resemblance to the K-level cognitive hierarchy [Camerer et al., 2004, Cui et al., 2021] of behavioral game theory and recursive reasoning in multiagent applications [Gmytrasiewicz and Durfee, 2000].

We employ an enhanced version of AlphaZero-style MCTS to train each best response strategy, thereby equipping our agent with the capability to both plan and infer the environmental state as well as opponents' strategic choices during online decision-making. This novel search method integrates deep RL with *Information Set MCTS* (IS-MCTS). To handle large imperfect information, we augment a deep generative model that samples world states at the root of the search tree, and iteratively refine its quality together with a PVN using RL trajectory data during the training loop. On each simulation step, a world state is sampled, and posterior mixed strategies of the opponents are updated, given the history implied by this world state. Then the opponent nodes are replaced with a sampled pure strategies from this distribution. Each pure strategy in the opponent pool serves as a "type" [Harsanyi, 1967] of play by viewing the environment as a Bayesian game. This type-based reasoning is also reflected by a recent work on Diplomacy [Bakhtin et al., 2023]. While their work generates different types by sampling different regularization parameters of human policies, our approach automates the generations of types during the best response step, and calibrates the type distribution during the MSS step. Therefore, our agent is capable of performing test-time search while automatically inferring opponents' types given an observation history.

Experimentally, we first assess the capacity of PSRO to compute a Nash equilibrium across various benchmark games. We then test on two negotiation game domains: colored-trails and deal-or-no-deal. Our negotiation-based PSRO agents, selected using fairness criteria, reach Pareto frontier and achieve and a social welfare when negotiating with humans that is comparable to humans trading among themselves. Importantly, as was recently demonstrated in the cooperative game Overcooked [Strouse et al., 2021], this is achieved without using any human data in the training procedure.

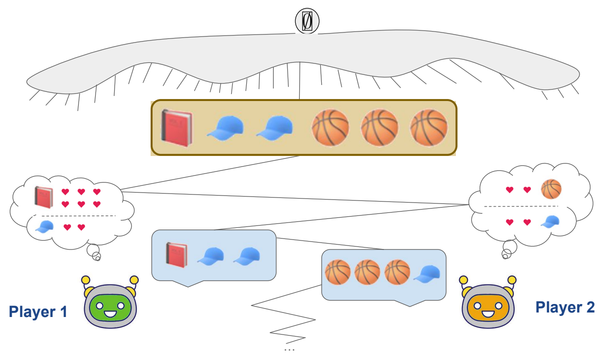


Figure 5.1: Example negotiation game in extensive-form. In "Deal or No Deal", the game starts at the empty history ($\varnothing$), chance samples a public pool of resources and private preferences for each player, then players alternate proposals for how to split the resources.

5.2 Background and Related Work

An N-player ***normal-form game*** consists of a set of players $\mathcal{N} = \{1, 2, \dots, N\}$, N finite pure strategy sets Π_i (one per player) with joint strategy set $\Pi = \Pi_1 \times \Pi_2 \times \cdots \Pi_N$, and a utility tensor (one per player), $u_i : \Pi \to \mathbb{R}$, and we denote player i's utility as $u_i(\pi)$. Two-player (2P) normal-form games are called ***matrix games***. A ***two-player zero-sum*** (purely adversarial) game is such that, $N = 2$ and for all joint strategies $\pi \in \Pi : \sum_{i \in \mathcal{N}} u_i(\pi) = 0$, whereas a ***common-payoff*** (purely cooperative) game: $\forall \pi \in \Pi, \forall i, j \in \mathcal{N} : u_i(\pi) = u_j(\pi)$. A ***general-sum*** game is one without any restrictions on the utilities. A ***mixed strategy*** for player i is a probability distribution over Π_i denoted $\sigma_i \in \Delta(\Pi_i)$, and a strategy profile $\sigma = \sigma_1 \times \cdots \times \sigma_N$, and for convenience we denote $u_i(\sigma) = \mathbb{E}_{\pi \sim \sigma}[u_i(\pi)]$. By convention, $-i$ refers to player i's opponents. A ***best response*** is a strategy $b_i(\sigma_{-i}) \in \mathrm{BR}(\sigma_{-i}) \subseteq \Delta(\Pi_i)$, that maximizes the utility against a specific opponent strategy: for example, $\sigma_1 = b_1(\sigma_{-1})$ is a best response to σ_{-1} if $u_1(\sigma_1, \sigma_{-1}) = \max_{\sigma_1'} u_1(\sigma_1', \sigma_{-1})$. An approximate ***$\epsilon$-Nash equilibrium*** is a profile σ such that for all $i \in \mathcal{N}, u_i(b_i(\sigma_{-i}), \sigma_{-i}) - u_i(\sigma) \leqslant \epsilon$, with $\epsilon = 0$ corresponding to an exact Nash equilibrium.

A "correlation device", $\mu \in \Delta(\Pi)$, is a distribution over the *joint* strategy space, which secretly recommends strategies to each player. Define $u_i(\pi_i', \mu)$ to be the expected utility of i when it deviates to π_i' given that other players follow their recommendations from μ. Then, μ is ***coarse-correlated equilibrium (CCE)*** when no player i has an incentive to unilaterally deviate *before* receiving their recommendation: $u_i(\pi_i', \mu) - u_i(\mu) \leqslant 0$ for all $i \in \mathcal{N}, \pi_i' \in \Pi_i$. Similarly, define $u_i(\pi_i', \mu | \pi_i'')$ to be the expected utility of deviating to π_i' given that other players follow μ and player i has received recommendation π_i'' from the correlation device. A ***correlated equilibrium (CE)*** is a correlation device μ where no player has an incentive to unilaterally deviate *after* receiving their recommendation: $u_i(\pi_i', \mu | \pi_i'') - u_i(\mu | \pi_i'') \leqslant 0$ for all $i \in \mathcal{N}, \pi_i' \in \Pi_i, \pi_i'' \in \Pi_i$.

In an ***extensive-form game***, play takes place over a sequence of actions $a \in \mathcal{A}$. Examples of such games include chess, Go, and poker. An illustrative example of interaction in an extensive-form game is shown in Figure 6.1. A ***history*** $h \in \mathcal{H}$ is a sequence of actions from the start of the game taken by all players. Legal actions are at h are denoted $\mathcal{A}(h)$ and the player to act at h as $\tau(h)$. Players only partially observe the state and hence have imperfect information. There is a special player called ***chance*** that plays with a fixed stochastic policy (selecting outcomes that represent dice rolls or private preferences). Policies π_i (also called behavioral strategies) is a collection of distributions over legal actions, one for each player's ***information state***, $s \in \mathcal{S}_i$, which is a set of histories consistent with what the player knows at decision point s (*e.g.*all the possible private preferences of other players), and $\pi_i(s) \in \Delta(\mathcal{A}(s))$.

There is a subset of the histories $\mathcal{Z} \subset \mathcal{H}$ called ***terminal histories***, and utilities are defined over terminal histories, e.g. $u_i(z)$ for $z \in \mathcal{Z}$ could be –1 or 1 in Go (representing a loss and a win for player i, respectively). As before, expected utilities of a joint profile $\pi = \pi_1 \times \cdots \times \pi_N$ is defined as an expectation over the terminal histories, $u_i(\pi) = \mathbb{E}_{z\sim\pi}[u_i(z)]$, and best response and Nash equilibria are defined with respect to a player's full policy space.

5.2.1 EGTA and Policy-Space Response Oracles

Empirical game-theoretic analysis (EGTA) [Wellman, 2006] is an approach to reasoning about large sequential games through normal-form ***empirical game*** models, induced by simulating enumerated subsets of the players' full policies in the sequential game. Policy-Space Response Oracles (PSRO) [Lanctot et al., 2017] uses EGTA to incrementally build up each player's set of policies ("oracles") through repeated applications of approximate best response using RL. Each player's initial set contains a single policy (*e.g.*uniform random) resulting in a trivial empirical game U^0 containing one cell. On epoch t, given N sets of policies Π_i^t for $i \in \mathcal{N}$, utility tensors for the empirical game U^t are estimated via simulation. A ***meta-strategy solver*** (MSS) derives a profile σ^t, generally mixed, over the empirical game strategy space. A new best response oracle, say $b_i^t(\sigma_{-i}^t)$, is then computed for each player i by training against opponent policies sampled from σ_{-i}^t. These are added to strategy sets for the next epoch: $\Pi_i^{t+1} = \Pi_i^t \cup \{b_i^t(\sigma_{-i}^t)\}$. Since the opponent policies are fixed, the oracle response step is a single-agent problem [Oliehoek and Amato, 2014], and (deep) RL can feasibly handle large state and policy spaces.

5.2.2 Algorithms for Meta-Strategy Solvers

A key motivation for introducing the MSS abstraction in PSRO [Lanctot et al., 2017] was the observation that best-responding to exact Nash equilibrium tended to produce new policies overfit to the current solution. Abstracting the solver allows for consideration of alternative response

Algorithm 8 Policy-Space Response Oracles (PSRO)

```
Input: Game $\mathcal{G}$, Meta Strat. Solver MSS, oracle BR.
function PSRO($\mathcal{G}$, MSS, BR)
    Initialize strategy sets $\forall i, \Pi_i = \{\pi_i^0\}$. Initialize mixed strategies $\sigma_i(\pi_i^0) = 1, \forall i$, payoff tensor $U^0$.
    for $t \in \{0, 1, 2 \cdots, T\}$ do
        for $i \in \mathcal{N}$ do
            $\Pi_i \leftarrow \Pi_i \bigcup \{\mathtt{BR}(i, \sigma, num\_eps)\}$
        end for
        Update missing entries in $U^t$ via simulations
        $\sigma \leftarrow \mathrm{MSS}(U^t)$
    end for
    return $\Pi = (\Pi_1, \Pi_2, \cdots, \Pi_N), \sigma$
end function
```

targets, for example those that ensure continual training against a broader range of past opponents, and those that keep some lower bound probability $\gamma/|\Pi_i|$ of being selected.

The current work considers a variety of previously proposed MSSs: ***uniform*** (corresponding to fictitious play [Brown, 1951]), ***projected replicator dynamics*** (PRD), a variant of replicator dynamics with directed exploration [Lanctot et al., 2017], α***-rank*** [Omidshafiei et al., 2019, Muller et al., 2020], ***maximum Gini (coarse) correlated equilibrium*** (MGCE and MGCCE) solvers [Marris et al., 2021], and exploratory ***regret-matching*** (RM) [Hart and Mas-Colell, 2000], a parameter-free regret minimization algorithm commonly used in extensive-form imperfect information games [Zinkevich et al., 2008, Moravčík et al., 2017, Brown et al., 2020, Schmid et al., 2023]. We also use and evaluate ADIDAS [Gemp et al., 2021] as an MSS for the first time. ADIDAS is a recently proposed general approximate Nash equilibrium (limiting logit equilibrium / QRE) solver.

5.2.3 Combining MCTS and RL for Best Response

The performance of EGTA and PSRO depend critically on the quality of policies found in the best-response steps; to produce stronger policies and enable test-time search, AlphaZero-style combined RL+MCTS [Silver et al., 2018] can be used in place of the RL alone. This has been applied recently to find exploits of opponent policies in Approximate Best Response (ABR) [Timbers et al., 2022, Wang et al., 2023] and also combined with auxiliary tasks for opponent prediction in BRExIt [Hernandez et al., 2023]. This combination can be particularly powerful; for instance, ABR found an exploit in a human-level Go playing agent trained with significant computational resources using AlphaZero.

When computing an approximate best response in imperfect information games, ABR uses a variant of Information Set Monte Carlo tree search [Cowling et al., 2012] called IS-MCTS-BR. At the root of the IS-MCTS-BR search (starting at information set s), the posterior distribution over world states, $\Pr(h \mid s, \pi_{-i})$ is computed explicitly, which requires both (i) enumerating every history in s, and (ii) computing the opponents' reach probabilities for each history in s. Then, during each search round, a world state is sampled from this belief distribution, then the game-tree regions are explored in a similar way as in the vanilla MCTS, and finally the statistics are aggregated on the information-set level. Steps (i) and (ii) are prohibitively expensive in games with large belief spaces. Hence, we propose learning a generative model online during the BR step; world states are sampled directly from the model given only their information state descriptions, leading to a succinct representation of the posterior capable of generalizing to large state spaces.

5.3 Search-Improved Generative PSRO

Our main algorithm has three components: the main driver (PSRO) [Lanctot et al., 2017], a search-enhanced BR step that concurrently learns a generative model, and the search with generative world state sampling itself. The main driver (Algorithm 8) operates as described in Subsection 5.2.1. In classical PSRO, the best response oracle is trained entirely via standard RL. For the first time, we introduce Approximate Best Response (ABR) as a search-based oracle in PSRO with a generative model for sampling world states.

The approximate best response step (Algorithm 9) proceeds analogously to AlphaZero's self-play based training, which trains a value net $\boldsymbol{v}$, a policy net $\boldsymbol{p}$, along with a generative network $\boldsymbol{g}$ using trajectories generated by search. There are some important differences from AlphaZero. Only one player is learning (*e.g.*player i). The (set of) opponents are fixed, sampled at the start of each episode from the opponent's meta-distribution σ_{-i}. Whenever it is player i's turn to play, since we are considering imperfect information games, it runs a POMDP search procedure based on IS-MCTS (Algorithm 12) from its current information state s_i. The search procedure produces a policy target π^*, and an action choice a^* that will be taken at s_i at that episode. Data about the final outcome and policy targets for player i are stored in data sets $D_{\boldsymbol{v}}$ and $D_{\boldsymbol{p}}$, which are used to improve the value net and policy net that guide the search. Data about the history, h, in each information set, $s(h)$, reached is stored in a data set $D_{\boldsymbol{g}}$, which is used to train the generative network $\boldsymbol{g}$ by supervised learning.

The MCTS search we use (Algorithm 12) is based on IS-MCTS-BR in [Timbers et al., 2022] (described in Section 5.2.3) and POMCP [Silver and Veness, 2010]. Here it utilizes value net $\boldsymbol{v}$ to truncate the search at an unexpanded node and policy net $\boldsymbol{p}$ for action selection at an expanded node s using the PUCT [Silver et al., 2018] formula: $\texttt{MaxPUCT}(s, \boldsymbol{p}) = \arg\max_{a \in \mathcal{A}(s)} \frac{s.child(a).value}{s.child(a).visits} + c_{uct} \cdot \boldsymbol{p}(s, a) \cdot \frac{\sqrt{s.total\_visits}}{s.child(a).visits+1}$, for some constant c_{uct}. Then at the end of the search call, it returns

Algorithm 9 ABR with generative model learning

```
function ABR($i, \sigma, num\_eps$)
  Initialize value nets $\boldsymbol{v}, \boldsymbol{v}'$, policy nets $\boldsymbol{p}, \boldsymbol{p}'$, generative nets $\boldsymbol{g}, \boldsymbol{g}'$, data buffers $D_{\boldsymbol{v}}, D_{\boldsymbol{p}}, D_{\boldsymbol{g}}$
  for $eps = 1, \ldots, num\_eps$ do
    $h \leftarrow$ initial state. $\mathcal{T} = \{s_i(h)\}$
    Sample opponents $\pi_{-i} \sim \sigma_{-i}$.
    while $h$ not terminal do
      if $\tau(h) = chance$ then
        Sample chance event $a \sim \pi_c$
      else if $\tau(h) \neq i$ then
        Sample $a \sim \pi_{\tau(h)}$
      else
        $a, \pi \leftarrow$ Search($s_i(h), \sigma, \boldsymbol{v}', \boldsymbol{p}', \boldsymbol{g}'$)
        $D_{\boldsymbol{p}} \leftarrow D_{\boldsymbol{p}} \bigcup \{(s_i(h), \pi)\}$
        $D_{\boldsymbol{g}} \leftarrow D_{\boldsymbol{g}} \bigcup \{(s_i(h), h)\}$
      end if
      $h \leftarrow h.apply(a), \mathcal{T} \leftarrow \mathcal{T} \bigcup \{s_i(h)\}$
    end while
    $D_{\boldsymbol{v}} \leftarrow D_{\boldsymbol{v}} \bigcup \{(s, r) \mid s \in \mathcal{T}\}$, where $r$ is the payoff of $i$ in this trajectory
    $\boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g} \leftarrow$ Update($\boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g}, D_{\boldsymbol{v}}, D_{\boldsymbol{p}}, D_{\boldsymbol{g}}$)
    Replace parameters of $\boldsymbol{v}', \boldsymbol{p}', \boldsymbol{g}'$ by the latest parameters of $\boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g}$ periodically.
  end for
  return Search($\cdot, \sigma, \boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g}$), or policy network $\boldsymbol{p}$, or greedy policy towards $\boldsymbol{v}$
end function
```

an action a^* which receives the most visits at the root node, and a policy π^* representing the action distribution of the search at the root node.

Algorithm 12 has two important differences from previous methods. Firstly, rather than computing exact posteriors, we use the deep generative model $\boldsymbol{g}$ learned in Algorithm 9 to sample world states. As such, this approach may be capable of scaling to large domains where previous approaches such as particle filtering [Silver and Veness, 2010, Somani et al., 2013] fail. Secondly, in the context of PSRO the imperfect information of the underlying POMDP consists of both (i) the actual world state h and (ii) opponents' pure-strategy commitment π_{-i}. We make use of the fact $\Pr(h, \pi_{-i} \mid s, \sigma_{-i}) = \Pr(h \mid s, \sigma_{-i}) \Pr(\pi_{-i} \mid h, \sigma_{-i})$ such that we approximate $\Pr(h \mid s, \sigma_{-i})$ by $\boldsymbol{g}$ and compute $\Pr(\pi_{-i} \mid h, \sigma_{-i})$ exactly via Bayes' rule. Computing $\Pr(\pi_{-i} \mid h, \sigma_{-i})$ can be interpreted as doing inference over opponents' types [Albrecht et al., 2016, Kreps and Wilson, 1982a, Hernandez-Leal and Kaisers, 2017, Kalai and Lehrer, 1993] or styles during play [Synnaeve and Bessiere, 2011, Ponsen et al., 2010].

Algorithm 10 IS-MCTS-BR with generative sampling

```
function Search($s$, $\sigma$, $\boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g}$)
  for $iter = 1, \ldots, num\_sim$ do
    $\mathcal{T} = \{\}$
    Sample a world state (gen. model): $h \sim \boldsymbol{g}(h \mid s)$
    Sample an opponent profile using Bayes' rule: $\pi'_{-i} \sim \Pr(\pi_{-i} \mid h, \sigma_{-i})$. Replace opponent
    nodes with chance events according to $\pi'_{-i}$
    while  do
      if $h$ is terminal then
        $r \leftarrow$ payoff of $i$. Break
      else if $\tau(h) = chance$ then
        $a \leftarrow$ sample according to chance
      else if $s_i(h)$ not in search tree then
        Add $s_i(h)$ to search tree.
        $r \leftarrow \boldsymbol{v}(s_i(h))$
      else
        $a \leftarrow$ MaxPUCT($s_i(h), \boldsymbol{p}$)
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s_i(h), a)\}$
      end if
      $h.apply(a)$
    end while
    for $(s, a) \in \mathcal{T}$ do
      $s.child(a).visits \leftarrow s.child(a).visits + 1$
      $s.child(a).value \leftarrow s.child(a).value + r$
      $s.total\_visits \leftarrow s.total\_visits + 1$
    end for
  end for
  return action $a^*$ that receives max visits among children of $s$, and a policy $\pi^*$ that represents
  the visit frequency of children of $s$
end function
```

5.3.1 Extracting a Final Agent at Test Time

How can a single decision-making agent be extracted from Π? The naive method samples $\pi_i \sim \sigma_i$ at the start of each episode, then follows σ_i for the episode. The **self-posterior** method *resamples* a new oracle $\pi_i \in \Pi_i^T$ at information states each time an action or decision is requested at s. At information state s, the agent samples an oracle π_i from the posterior over its own oracles: $\pi_i \sim \Pr(\pi_i \mid s, \sigma_i)$, using reach probabilities of its own actions along the information states leading to s where i acted, and then follows π_i. The self-posterior method is based on the equivalent behavior strategy distribution that Kuhn's theorem [Kuhn, 1953] derives from the mixed strategy distribution over policies. The **aggregate policy** method takes this a step further and computes the average (expected self-posterior) policy played at each information state, $\bar{\pi}_i^T(s)$, exactly (rather than via

Algorithm	Abbreviation	Independent/Joint	Solution Concepts	Description
α-Rank	—	Joint	MCC	[Omidshafiei et al., 2019, Muller et al., 2020]
ADIDAS	—	Independent	LLE/QRE	[Gemp et al., 2021]
Max Entropy (C)CE	ME(C)CE	Joint	(C)CE	[Ortiz et al., 2007]
Max Gini (C)CE	MG(C)CE	Joint	(C)CE	[Marris et al., 2021]
Max NBS (C)CE	MN(C)CE	Joint	(C)CE	Sec 5.4.3
Max Welfare (C)CE	MW(C)CE	Joint	(C)CE	[Marris et al., 2021]
Nash Bargaining Solution (NBS)	NBS	Independent	P-E	Sec 5.4.2
NBS Joint	NBS_joint	Joint	P-E	Sec 5.4.2
Projected Replicator Dynamics	PRD	Independent	?	[Lanctot et al., 2017, Muller et al., 2020]
Regret Matching	RM	Independent	CCE	[Lanctot et al., 2017]
Social Welfare	SW	Joint	MW	Sec 5.4
Uniform	—	Independent	?	[Brown, 1951, Shoham and Leyton-Brown, 2009]

Table 5.1: Meta-strategy solvers. For each MSS, we indicate whether its output is over joint or individual strategy spaces, and the solution concept it captures. P-E stands for Pareto efficiency.

samples); it is described in detail in [Lanctot et al., 2017, Section E.3]. The **rational planning** method enables decision-time search instantiated with the final oracles: it assumes the opponents at test time exactly match the σ_{-i} of training time, and keeps updating the posterior $\Pr(h, \pi_{-i} \mid s, \sigma_{-i})$ during an online play. Whenever it needs to take an action at state s at test time, it employs Algorithm 12 to search against this posterior. This method combines online Bayesian opponent modeling and search-based best response, which resembles the rational learning process [Kalai and Lehrer, 1993].

5.4 New Meta-Strategy Solvers

Recall from Section 5.2 that a meta-strategy solver (MSS) selects a strategy profile from the current empirical game for use as best-response target. This target can take the form of either: (i) μ, a joint distribution over Π, or (ii) $(\sigma_1, \sigma_2, \ldots, \sigma_N)$, a set of (independent) distributions over Π_i, respectively. These distributions (μ_{-i} or σ_{-i}) are used to sample opponents when player i is computing an approximate best response. We use many MSSs: several new and from previous work, summarized in Appendix 5.7.1 and Table 5.1. We present several new MSSs for general-sum games inspired by bargaining theory, which we now introduce.

5.4.1 Bargaining Theory and Solution Concepts

The ***Nash Bargaining solution*** (NBS) selects a Pareto-optimal payoff profile that uniquely satisfies axioms specifying desirable properties of invariance, symmetry, and independence of irrelevant alternatives [Nash, 1950a, Ponsati and Watson, 1997]. The axiomatic characterization of NBS abstracts away the process by which said outcomes are obtained through strategic interaction. However, Nash showed that it corresponds to a strategic equilibrium if threats are credible [Nash, 1953], and in fact, in bargaining games where agents take turns, under certain conditions the perfect equilibrium corresponds to the NBS [Binmore et al., 1986].

Define the set of achievable payoffs as all expected utilities $u_i(\mu)$ under a joint-policy profile μ [Harsanyi and Selten, 1972, Morris, 2012]. Denote the disagreement outcome of player i, which is the payoff it gets if no agreement is achieved, as d_i. The NBS is the set of policies that maximizes the ***Nash bargaining score*** (A.K.A. *Nash product*):

$$\max_{\mu \in \Delta(\mathit{\Pi})} \Pi_{i \in \mathcal{N}} \left(u_i(\mu) - d_i\right), \tag{5.1}$$

which, when $N = 2$, leads to a quadratic program (QP) with the constraints derived from the policy space structure [Griffin, 2010]. However, even in this simplest case of two-player matrix games, the objective is non-concave posing a problem for most QP solvers. Furthermore, scaling to N players requires higher-order polynomial solvers.

5.4.2 Empirical Game Nash Bargaining Solution

Instead of using higher-order polynomial solvers, we propose an algorithm based on (projected) gradient ascent [Singh et al., 2000, Boyd and Vandenberghe, 2004]. Let $\mathbf{x} \in \Delta(\mathit{\Pi})$ represent a distribution over joint strategies in an empirical game. Let $u_i(\mathbf{x}) = \mathbb{E}_{\pi \sim \mathbf{x}}[u_i(\pi)]$ be the expected utility for player i under the joint distribution $\mathbf{x}$. Let $\Pi_{i \in \mathcal{N}}(u_i(\mathbf{x}) - d_i)$ be the Nash product defined in Equation 5.1. In practice, d_i is either clearly defined from the context, or is set as a value that is lower than the minimum achievable payoff of i in $\Delta(\mathit{\Pi})$. We restrict $u_i(\mathbf{x}) - d_i > 0$ for all $i, \mathbf{x}$. Note that the Nash product is non-concave, so instead of maximizing it, we maximize the ***log Nash product*** $g(\mathbf{x}) =$

$$\log\left(\Pi_{i \in \mathcal{N}}(u_i(\mathbf{x}) - d_i)\right) = \sum_{i \in \mathcal{N}} \log(u_i(\mathbf{x}) - d_i), \tag{5.2}$$

which has the same maximizers as (5.1), and is a sum of concave functions, hence concave. The process is depicted in Algorithm 11; `Proj` is the ℓ_2 projection onto the simplex.

Theorem 5.4.1. *Assume any deal is better than no deal by $\kappa > 0$, i.e., $u_i(\mathbf{x}) - d_i \geqslant \kappa > 0$ for all $i, \mathbf{x}$. Let $\{\mathbf{x}^t\}$ be the sequence generated by Algorithm 11 with starting point $\mathbf{x}^0 = |\mathit{\Pi}|^{-1}\mathbf{1}$ and step size sequence $\alpha^t = \frac{\kappa\sqrt{(|\mathit{\Pi}|-1)/|\mathit{\Pi}|}}{u^{\max} N}(t+1)^{-1/2}$. Then, for all $t > 0$ one has*

$$\max_{\mathbf{x} \in \Delta^{|\mathit{\Pi}|-1}} g(\mathbf{x}) - \max_{0 \leqslant s \leqslant t} g(\mathbf{x}^s) \leqslant \frac{u^{\max} N \sqrt{|\mathit{\Pi}|}}{\kappa\sqrt{t+1}} \tag{5.3}$$

where $u^{\max} = \max_{i,\mathbf{x}} u_i(\mathbf{x})$, $|\mathit{\Pi}|$ is the number of possible pure joint strategies, and $\mathbf{x}$ is assumed to be a joint correlation device (μ).

Algorithm 11 NBS by projected gradient ascent

```
Input: Initial iterate $\mathbf{x}$, payoff tensor $U$.
function NBS($\mathbf{x}^0$, $U$)
    Let $g(\mathbf{x})$ be the log Nash product defined in eqn (5.2)
    for $t = 0, 1, 2 \cdots, T$ do
        $\mathbf{y}^{t+1} \leftarrow \mathbf{x}^t + \alpha^t \nabla g(\mathbf{x}^t)$
        $\mathbf{x}^{t+1} \leftarrow \texttt{Proj}(\mathbf{y}^{t+1})$
    end for
    Return $\arg\max_{\mathbf{x}^{t=0:T}} g(\mathbf{x}^t)$
end function
```

For a proof, see Appendix 5.7.2.

5.4.3 Max-NBS (Coarse) Correlated Equilibria

The second new MSS we propose uses NBS to select a (C)CE. For all normal-form games, valid (C)CEs are a convex polytope of joint distributions in the simplex defined by the linear constraints. Therefore, maximizing any strictly concave function can uniquely select an equilibrium from this space (for example Maximum Entropy [Ortiz et al., 2007], or Maximum Gini [Marris et al., 2021]). The log Nash product (Equation (5.2)) is concave but not, in general, strictly concave. Therefore to perform unique equilibrium selection, a small additional strictly concave regularizer (such as maximum entropy) may be needed to select a uniform mixture over distributions with equal log Nash product. We use existing off-the-shelf exponential cone constraints solvers (e.g., ECOS [Domahidi et al., 2013], SCS [O'Donoghue et al., 2021]) which are available in simple frameworks (CVXPY [Diamond and Boyd, 2016, Agrawal et al., 2018]) to solve this optimization problem.

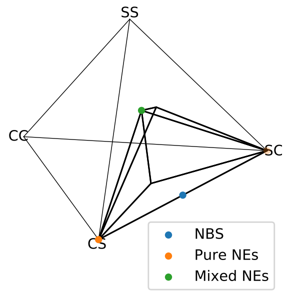
NBS is a particularly interesting selection criterion. Consider the game of chicken in where players are driving head-on; each may continue (C), which may lead to a crash, or swerve (S), which may make them look cowardly. Many joint distributions in this game are valid (C)CE equilibria (Figure 5.2). The optimal outcome in terms of both welfare and fairness is to play SC and CS each 50% of the time. NBS selects this equilibrium. Similarly in Bach-or-Stravinsky where players coordinate but have different preferences over events: the fairest maximal social welfare outcome is a compromise, mixing equally between BB and SS.

5.4.4 Social Welfare

This MSS selects the pure joint strategy of the empirical game that maximizes the estimated social welfare.

	C	S
C	-5,-5	+1,-1
S	-1,+1	-1,-1

	B	S
B	3,2	0,0
S	0,0	2,3



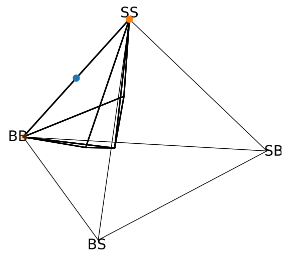


Figure 5.2: (C)CE polytopes in Chicken (left) and Bach-or-Stravinsky (right) showing NBS equilibrium selection.

5.5 Experiments

We initially assessed PSRO as a general approximate Nash equilibrium solver and collection of MSSs over 12 benchmark games commonly used throughout the literature. The full results are presented in Appendix 5.5.1.

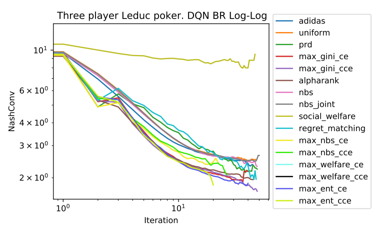
Then we focus our evaluation on negotiation, a common human interaction and important class of general-sum games with a tension between incentives (competing versus cooperating).

5.5.1 Approximate Nash Equilibrium Solving on Benchmark Games

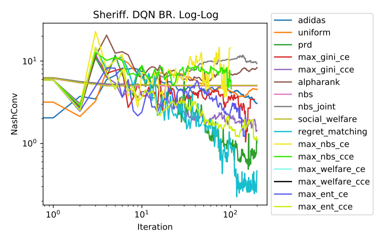
Here we evaluate the capacity of PSRO with different meta-strategy solvers [1] to act as a general Nash equilibrium solver for sequential N-player games. For these initial experiments, we run PSRO on its own without search. Since these are benchmark games, they are small enough to compute exact exploitability, or the Nash gap (called NashConv by Lanctot et al. [2017, 2019]), and search is not necessary. We run PSRO using 16 different meta-strategy solvers across 12 different benchmark games (three 2P zero-sum games, three N-player zero-sum games, two common-payoff games, and four general-sum games): instances of Kuhn and Leduc poker, Liar's dice, Trade Comm, Tiny Bridge, Battleship, Goofspiel, and Sheriff. These games have recurred throughout previous work, so we describe them in Appendix 5.7.3; they are all partially-observable and span a range of different types of games.

A representative sample of the results is shown in Figure 5.3, whereas all the results are shown below. $\text{NashConv}(\bar{\pi}) = \sum_{i \in \mathcal{N}} u_i(b_i(\bar{\pi}_{-i}), \bar{\pi}_{-i}) - u_i(\bar{\pi})$, where b_i is an exact best response, and

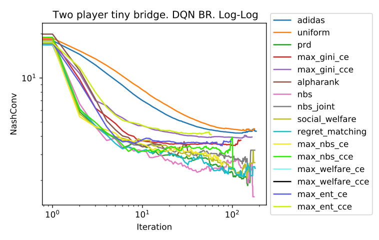
[1]For simplicity we use the same MSS for both guiding the BR step (strategy exploration) and evaluating NashConv at each iteration. Firmer conclusions about relative effectiveness for strategy exploration would require defining a fixed solver for evaluation purposes [Wang et al., 2021b].



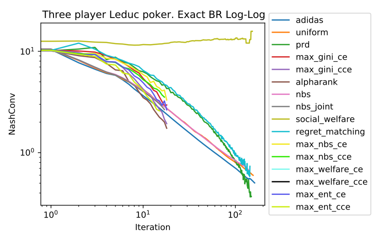
(a)



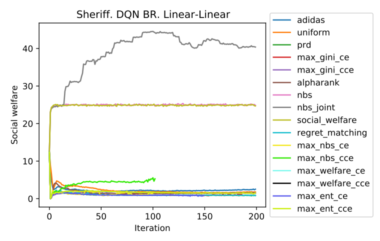
(b)



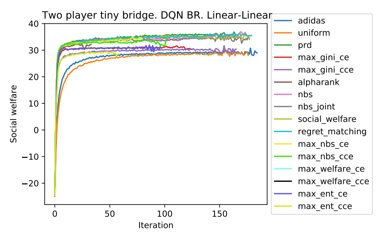
(c)



(d)



(e)



(f)

Figure 5.3: NASHCONV and social welfare along PSRO iterations across game types. NASHCONV in 3P Leduc poker using (a) DQN oracles vs. (d) exact oracles. NASHCONV (b) and (e) social welfare in Sheriff. NASHCONV (c) and social welfare (f) in 2P Tiny Bridge.

$\bar{\pi}$ is the exact average policy using the aggregator method described in Section 5.3.1. A value of zero means $\bar{\pi}$ is Nash equilibrium, and values greater than zero correspond to an gap from Nash equilibrium. Social welfare is defined as $\text{SW}(\bar{\pi}) = \sum_i u_i(\bar{\pi})$.

Most of the meta-strategy solvers seem to reduce NASHCONV faster than a completely uninformed meta-strategy solver (uniform) corresponding to fictitious play, validating the EGTA approach taken in PSRO. In three-player (3P) Leduc, the NashConv achieved for the exact best response is an order of magnitude smaller than when using DQN. ADIDAS, regret-matching, and PRD are a good default choice of MSS in competitive games. The correlated equilibrium meta-strategy solvers are surprisingly good at reducing NASHCONV in the competitive setting, but can become unstable and even fail when the empirical game becomes large in the exact case. In the general-sum game Sheriff, the reduction of NASHCONV is noisy, with several of the meta-strategy solvers having erratic graphs, with RM and PRD performing best. Also in Sheriff, the Nash bargaining (and social welfare) meta-strategy solvers achieve significantly higher social welfare than most meta-strategy solvers. Similarly in the cooperative game of Tiny Bridge, the MSSs that reach closest to optimal are NBS (independent and joint), social welfare, RM, PRD, and Max-NBS-(C)CE. Many of these meta-strategy solvers are not *guaranteed* to compute an approximate Nash equilibrium (even in the empirical game), but any limiting logit equilibrium (QRE) solver can get arbitrarily close. ADIDAS does not require the storage of the meta-tensor U^t, only samples from it. So, as the number of iterations grow ADIDAS might be one of the safest and most memory-efficient choice for reducing NASHCONV long-term.

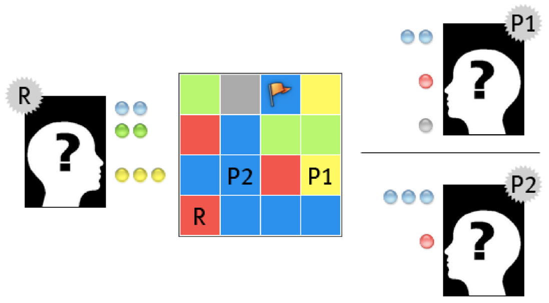


Figure 5.4: Three-Player Colored Trails.

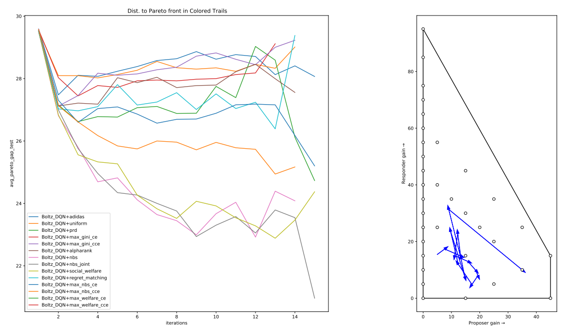


Figure 5.5: Empirical reduction in Pareto Gap on test game configurations, and example evolution toward Pareto front (right).

5.5.2 Negotiation Game: Colored Trails

We start with a highly configurable negotiation game played on a grid [Gal et al., 2010a] of colored tiles, which has been actively studied by the AI community [Grosz et al., 2004, Ficici and Pfeffer, 2008b, Gal et al., 2010b]. Colored Trails does not require search since the number of moves is small, so we use classical RL based oracles (DQN and Boltzmann DQN) to isolate the effects of the new meta-strategy solvers. Furthermore, it has a property that most benchmark games do not: it is parameterized by a board (tile layout and resource) configuration, which allows for a training/testing set split to evaluate the capacity to generalize across different instances of similar games.

We use a three-player variant [Ficici and Pfeffer, 2008a, de Jong et al., 2011] depicted in Figure 5.4. At the start of each episode, an instance (a board and colored chip allocation per player) is randomly sampled from a database of strategically interesting and balanced configurations [de Jong et al., 2011, Section 5.1]. There are two proposers (P1 and P2) and a responder (R). R can see all players' chips, both P1 and P2 can see R's chips; however, proposers cannot see each other's chips. Each proposer, makes an offer to the receiver. The receiver than decides to accept one offer and trades chips with that player, or passes. Then, players spend chips to get as close to the flag as possible (each chip allows a player to move to an adjacent tile if it is the same color as the chip). For any configuration (player i at position p), define $\textsc{Score}(p, i) = (-25)d + 10t$, where d is the Manhattan distance between p and the flag, and t is the number of player i's chips. The utility for player i is their *gain*: score at the end of the game minus the score at the start.

This game has been decomposed into specific hand-crafted meta-strategies for both proposers and receiver [de Jong et al., 2011]. These meta-strategies cover the Pareto-frontier of the payoff space by construction. Rather than relying on domain knowledge, we evaluate the extent to which

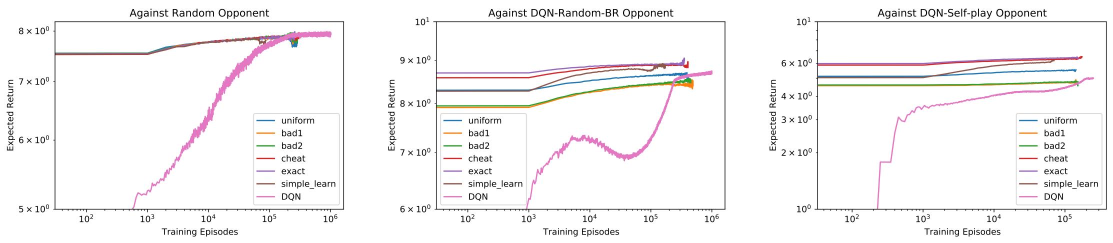


Figure 5.6: Best response performance using different generative models, against (left) uniform random opponent, (middle) DQN response to uniform random, (right) self-play DQN opponent. **Uniform** samples a legal preference vector uniformly at random, **bad1** always samples the first legal instance in the database, **bad2** always samples the last legal instance in the database, **cheat** always samples the actual underlying world state, **exact** samples from the exact posterior, **simple learn** is the method described in Algorithm 9 (detailed in Appendix 5.7.4.1), and **DQN** is a simple DQN responder that does not use a generative model nor search.

PSRO can learn such a subset of representative meta-strategies. To quantify this, we compute the Pareto frontier for a subset of configurations, and define the Pareto Gap (P-Gap) as the minimal ℓ_2 distance from the outcomes to the outer surfaces of the convex hull of the Pareto front, which is then averaged over the set of configurations in the database.

Figure 5.5 shows representative results of PSRO agents on Colored Trails (for full graphs, and evolution of score diagrams, see Appendix 5.7.5.1). The best-performing MSS is NBS-joint, beating the next best by a full 3 points. The NBS meta-strategy solvers comprise five of the six best MSSs under this evaluation. An example of the evolution of the expected score over PSRO iterations is also shown, moving toward the Pareto front, though not via a direct path.

5.5.3 Negotiation Game: Deal or No Deal

"Deal or No Deal" (DoND) is a simple alternating-offer bargaining game with incomplete information, which has been used in many AI studies [DeVault et al., 2015, Lewis et al., 2017, Cao et al., 2018, Kwon et al., 2021]. Our focus is to train RL agents to play against humans *without human data*, similar to previous work [Strouse et al., 2021]. An example game of DoND is shown in Figure 6.1. Two players are assigned *private* preferences $\boldsymbol{w}_1 \geqslant \boldsymbol{0}, \boldsymbol{w}_2 \geqslant \boldsymbol{0}$ for three different items (books, hats, and basketballs). At the start of the game, there is a pool $\boldsymbol{c}$ of 5 to 7 items drawn randomly such that: (i) the total value for a player of all items is 10: $\boldsymbol{w}_1 \cdot \boldsymbol{c} = \boldsymbol{w}_2 \cdot \boldsymbol{c} = 10$, (ii) each item has non-zero value for at least one player: $\boldsymbol{w}_1 + \boldsymbol{w}_2 > \boldsymbol{0}$, (iii) some items have non-zero value for both players, $\boldsymbol{w}_1 \odot \boldsymbol{w}_2 \neq \boldsymbol{0}$, where $\odot$ represents element-wise multiplication.

The players take turns proposing how to split the pool of items, for up to 10 turns (5 turns each). If an agreement is not reached, the negotiation ends and players both receive 0. Otherwise, the agreement represents a split of the items to each player, $\boldsymbol{o}_1 + \boldsymbol{o}_2 = \boldsymbol{c}$, and player i receives a utility of $\boldsymbol{w}_i \cdot \boldsymbol{o}_i$. DoND is an imperfect information game because the other player's preferences are private. We use a database of 6796 bargaining instances made publicly available in [Lewis et al., 2017]. Deal or No Deal is a significantly large game, with an estimated $1.32 \cdot 10^{13}$ information states for player 1 and $5.69 \cdot 10^{11}$ information states for player 2 (see Appendix 5.7.3.2 for details).

5.5.3.1 Generative World State Sampling

We now show that both the search and the generative model contribute to achieving higher reward (in the BR step of PSRO) than RL alone. The input of our deep generative model is one's private values $\mathbf{v}_i$ and public observations, and the output is a distribution over $\mathbf{v}_{-i}$ (detailed in the Appendix). We compute approximate best responses to three opponents: uniform random, a DQN agent trained against uniform random, and a DQN agent trained in self-play. We compare different world state sampling models as well to DQN in Figure 5.6, where the deep generative model approach is denoted as simple_learn.

The benefit of search is clear: the search methods achieve a high value in a few episodes, a level that takes DQN many more episodes to reach (against random and DQN response to random) and a value that is not reached by DQN against the self-play opponent. The best generative models are the true posterior (exact) and the actual underlying world state (cheat). However, the exact posterior is generally intractable and the underlying world state is not accessible to the agent at test-time, so these serve as idealistic upper-bounds. Uniform seems to be a compromise between the bad and ideal models. The deep generative model approach is roughly comparable to uniform at first, but learns to approximate the posterior as well as the ideal models as data is collected. In contrast, DQN eventually reaches the performance of the uniform model against the weaker opponent but not against the stronger opponent even after 20000 episodes.

5.5.3.2 Studies with Human Participants

We recruited participants from Prolific [Pe'er et al., 2021, Peer et al., 2017] to evaluate the performance of our agents in DoND (overall 346; 41.4% female, 56.9% male, 0.9% trans or nonbinary; median age range: 30–40). Crucially, participants played DoND for real monetary stakes, with an additional payout for each point earned in the game.

Participants first read game instructions and completed a short comprehension test to ensure they understood key aspects of DoND's rules. Participants then played five episodes of DoND with a randomized sequence of opponents. Episodes terminated after players reached a deal, after 10 moves without reaching a deal, or after 120 seconds elapsed. After playing all five episodes, participants completed a debrief questionnaire collecting standard demographic information and

Name	Values
Individual return (IR)	r_i
Inequity aversion [Fehr and Schmidt, 1999] (IE)	$r_i - 0.5 \cdot \max\{r_{-i} - r_i, 0\}$ [Gal et al., 2010a]
Social welfare (SW)	$r_i + r_{-i}$
Nash bargaining score (NBS)	$r_i \cdot r_{-i}$

Table 5.2: Different tree back-propagation value types. r_i is the return for player i.

open-ended feedback on the study.

Training Details Our infrastructure restricts that each human participant can only play five matches with our bots. Therefore we decided to select five different agents so every participant can play each of these once. For comparison, we decided to include one independent RL agent and four search-improved PSRO agents of different playing styles.

For the independent RL agent, we trained two classes of independent RL agents in selfplay: (1) DQN [Mnih et al., 2015] and Boltzmann DQN [Cui and Koeppl, 2021], and (2) policy gradient algorithms such as A2C, QPG, RPG and RMPG [Srinivasan et al., 2018]. For DQN and Boltzmann DQN, we used replay buffer sizes of 10^5, ϵ decayed from $0.9 \to 0.1$ over 10^6 steps, a batch size of 128, and swept over learning rates of $\{0.01, 0.02, 0.005\}$. For Boltzmann DQN, we varied the temperature $\eta \in 0.25, 0.5, 1$. For all self-play policy gradient methods we used a batch size of 128, and swept over critic learning rate in $\{0.01, 0.001\}$, policy learning rate in $\{.001, 0.0005, 0.0001\}$, number of updates to the critic before updating the policy in $\{1, 4, 8\}$, and entropy cost in $\{0.01, 0.001\}$. DQN trained with the settings above and a learning rate of 0.005 was the agent we found to achieve highest individual returns (and social welfare, and Nash bargaining score), so we select it as the representative agent for the independent RL category.

For PSRO agents, we consider 16 different meta-strategy solvers, and 4 different back-propagating value types during the tree search procedure, making it 64 different combination in total. Notice that the original MCTS algorithm (Algorithm 12) back-propagates individual rewards during each simulation phase for the search agent. We also explore other choices such as social welfare and inequity aversion in our DoND human behavioral studies, as listed in Table 5.2.

We consider self-posterior (SP) and rational planning (RP) methods described in Section 5.3.1 to extract a final decision agent, as the other approaches are either infeasible computationally or is subsumed by the current methods. That makes it 128 agents totally in principle. We train the neural networks for 3 days, and screen out those combination that cannot make it to the 3rd PSRO iteration (due to break of the MSS optimizer). We eventually have 112 different PSRO agents at hand. As detailed in App. 5.7.5.2, we apply empirical game-theoretic analysis on the resulting 112×112 meta-game and using different categories to select the final four PSRO agents. We

Agent	$\bar{u}_{\text{Humans}}$		$\bar{u}_{\text{Agent}}$		$\bar{u}_{\text{Comb}}$		NBS
IndRL	5.86	$[5.37, 6.40]$	$\mathbf{6.50}$	$[\mathbf{5.93}, \mathbf{7.06}]$	6.18	$[5.82, 6.56]$	38.12
Comp1	5.14	$[4.56, 5.63]$	5.49	$[4.87, 6.11]$	5.30	$[4.93, 5.76]$	28.10
Comp2	6.00	$[5.49, 6.55]$	5.54	$[4.96, 6.10]$	5.76	$[5.33, 6.12]$	33.13
Coop	6.71	$[6.23, 7.20]$	6.17	$[5.66, 6.64]$	6.44	$[6.11, 6.75]$	41.35
Fair	$\mathbf{7.39}$	$[\mathbf{6.89}, \mathbf{7.87}]$	5.98	$[5.44, 6.49]$	$\mathbf{6.69}$	$[\mathbf{6.34}, \mathbf{7.01}]$	$\mathbf{44.23}$

Table 5.3: Humans vs. agents performance with $\mathbf{129}$ human participants, $\mathbf{547}$ games total. $\bar{\boldsymbol{u}}_{\boldsymbol{X}}$ refers to the average utility to group $\boldsymbol{X}$ (for the humans when playing the agent, or for the agent when playing the humans), Comb refers to Combined (human and agent). Square brackets indicate 95% confidence intervals. IndRL refers to Independent RL (DQN), Comp1 and Comp2 are the two top-performing competitive agents, Coop is the most cooperative agent, and Fair is fairest agent. NBS is the Nash bargaining score (Eq 5.1).

eventually selected: (i) two most competitive agents (Comp1, Comp2) (maximizing utility), (ii) the most cooperative agents (Coop) (maximizing social welfare), the (iii) the fairest agent (Fair) (minimizing social inequity [Fehr and Schmidt, 1999]); (iv) a separate top-performing independent RL agent (IndRL) trained in self-play (DQN). Here Comp1, Comp2 are extracted using SP while Coop and Fair are using RP. Both Coop and Fair are using Nash product as the back-propagating values during tree search, while Comp1 uses inequity aversion and Comp2 uses individual rewards. Comp1, Comp2 and Fair are trained using Max-Gini CE or CCE MSS, while Coop uses uniform distribution as the MSS.

Results We collect data under two conditions: human vs. human (HvH), and human vs. agent (HvA). In the HvH condition, we collect 483 games: 482 end in deals made (99.8%), and achieve a return of 6.93 (95% c.i. [6.72, 7.14]), on expectation. We collect 547 games in the HvA condition: 526 end in deals made (96.2%; see Table 5.3). DQN achieves the highest individual return. By looking at the combined reward, it achieves this by aggressively reducing the human reward (down to 5.86)–possibly by playing a policy that is less human-compatible. The competitive PSRO agents seem to do the same, but without overly exploiting the humans, resulting in the lowest social welfare overall. The cooperative agent achieves significantly higher combined utility playing with humans. Better yet is Human vs. Fair, the only Human vs. Agent combination to achieve social welfare comparable to the Human vs. Human social welfare.

Another metric is the objective value of the empirical NBS from Eq. 5.1, over the symmetric game (randomizing the starting player) played between the different agent types. This metric favors Pareto-efficient outcomes, balancing between the improvement obtained by both sides. From App 5.7.5.2, the NBS of Coop decreases when playing humans, from $44.51 \rightarrow 41.35$– perhaps due to overfitting to agent-learned conventions. Fair increases slightly ($42.56 \rightarrow 44.23$). The NBS of DQN rises from $23.48 \rightarrow 38.12$. The NBS of the competitive agents also rises playing against humans ($24.70 \rightarrow 28.10$, and $25.44 \rightarrow 28.10$), and also when playing with Fair ($24.70 \rightarrow 29.63$, $25.44 \rightarrow 28.73$).

The fair agent is both adaptive to many different types of agents, and cooperative, increasing the social welfare in all groups it negotiated with. This could be due to its MSS (MGCE) putting significant weight on many policies leading to Bayesian prior with high support, or its backpropagation of the product of utilities rather than individual return.

5.6 Conclusion and Future Work

We proposed a general-purpose multiagent training regime that combines the power of MCTS search and a population-based training framework, for general-sum imperfect information domains. We developed a novel search technique that combines IS-MCTS with a deep belief learning module coupled with the RL training loop, which scale to large belief and state spaces. The outer loop of our algorithm is implemented by PSRO, which iteratively trains and adds search strategies guided by game-theoretic analysis. On one hand, search serves as a strong best response method within the PSRO loop, which provides an instance of the framework of its own interests. On the other hand, PSRO automatically produces a belief hierarchy over the opponents' strategies, which endows the search with the capability of inferring opponent types during online decision makings. This dual view of the whole training architecture illustrates its effectiveness in producing agents that are capable of opponent modeling through game-theoretic analysis and planning forward at test-time.

Our experimental results found that ADIDAS, regret-matching, and PRD MSSs work well generally and even better in competitive games. In cooperative, general-sum games and negotiation games, NBS-based meta-strategy solvers can increase social welfare and find solutions closer to the Pareto frontier. In our human-agent study of a negotiation game, self-play DQN exploits humans most, and agents trained with PSRO (selected using fairness criteria) adapt and cooperate well with humans.

Future work could further enhance the best response by predictive losses [Hernandez et al., 2023], scale to even larger domains, and convergence to other solution concepts such as self-confirming or correlated equilibria.

5.7 Appendix

5.7.1 Meta-Strategy Solvers

In this section, we describe the algorithms used for the MSS step of PSRO, which computes a set of meta-strategies σ_i or a correlation device μ for the normal-form empirical game.

5.7.1.1 Classic PSRO Meta-Strategy Solvers

Projected Replicator Dynamics (PRD) In the replicator dynamics, each player i used by mixed strategy σ_i^t, often interpreted as a distribution over population members using pure strategies. The continuous-time dynamic then describes a change in weight on strategy $\pi_k \in \Pi_i$ as a time derivative:

$$\frac{d\sigma_i^t(\pi_k)}{dt} = \sigma_i^t(\pi_k)[u_i(\pi_k, \sigma_{-i}) - u_i(\sigma)].$$

The projected variant ensures that the strategies stay within the exploratory simplex such σ_i^t remains a probability distribution, and that every elements is subject to a lower-bound $\frac{\gamma}{|\Pi_i|}$. In practice, this is simulated by small discrete steps in the direction of the time derivatives, and then projecting σ_i^t back to the nearest point in the exploratory simplex.

Exploratory Regret-Matching Regret-Matching is based on the algorithm described in [Hart and Mas-Colell, 2000] and used in extensive-form games for regret minimization [Zinkevich et al., 2008]. Regret-matching is an iterative solver that tabulates cumulative regrets $R_i^T(\pi)$, which initially start at zero. At each trial t, player i would receive $u_i(\sigma_i^t, \sigma_{-i}^t)$ by playing their strategy σ_i^t. The instantaneous regret of *not* playing pure strategy $\pi_k \in \Pi_i$ is

$$r^t(\pi_k) = u_i(\pi_k, \sigma_{-i}^t) - u_i(\sigma_i^t, \sigma_{-i}^t).$$

The cumulative regret over T trials for the pure strategy is then defined to be:

$$R_i^T(\pi_k) = \sum_{t=1}^{T} r^t(\pi_k).$$

Define $(x)^+ = \max(0, x)$. The policy at time $t+1$ is derived entirely by these regrets:

$$\sigma_{i,RM}^{t+1}(\pi_k) = \frac{R_i^{T,+}(\pi_k)}{\sum_{\pi_k' \in \Pi_i} R_i^{T,+}(\pi_k')},$$

if the denominator is positive, or $\frac{1}{|\Pi_i|}$ otherwise. As in original PSRO, in this work we also add exploration:

$$\sigma_i^{t+1} = \gamma \textsc{Uniform}(\Pi_i) + (1-\gamma)\sigma_{i,RM}^{t+1}.$$

Finally, the meta-strategy returned for all players at time t is their average strategy $\bar{\sigma}^T$.

5.7.1.2 Joint and Correlated Meta-Strategy Solvers

The jointly correlated meta-strategy solvers were introduced in [Marris et al., 2021], which was the first to propose computing equilibria in the joint space of the empirical game. In general, a correlated equilibrium (and coarse-correlated equilibrium) can be found by satisfying a number of constraints, on the correlation device, so the question is what to use as the optimization criterion, which effectively selects the equilibrium.

One that maximizes Shannon entropy (ME(C)CE) seems like a good choice as it places maximal weight on all strategies, which could benefit PSRO due to added exploration among alternatives. However it was found to be slow on large games. Hence, Marris et al. [Marris et al., 2021] propose to use a different but related measure, the Gini impurity, for correlation device μ,

$$\textsc{GiniImpurity}(\mu) = 1 - \mu^T\mu,$$

which is a form of Tsallis entropy. The resulting equilibria Maximum Gini (Coarse) Correlated Equilibria (MG(C)CE) have linear constraints and can be computed by solving a quadratic program. Also, Gini impurity has similar properties to Shannon entropy: it is maximized at the uniform distribution and minimized when all the weight is placed on a single element.

In the Deal-or-no-Deal experiments, 4 of 5 selected winners of tournaments used MG(C)CE (see Section 5.7.5.2, including the one that cooperated best with human players, and the other used uniform. Hence, it is possible that the exploration motivated my high-support meta-strategies indeed does help when playing against a population of agents, possibly benefiting the Bayesian inference implied by the generative model and resulting search.

5.7.1.3 ADIDAS

Average Deviation Incentive with Adaptive Sampling (ADIDAS) [Gemp et al., 2021] is an algorithm designed to approximate a particular Nash equilibrium concept called the *limiting logit equilibrium* (LLE). The LLE is unique in almost all N-player, general-sum, normal-form games and is defined via a homotopy [McKelvey and Palfrey, 1995]. Beginning with a quantal response (logit) equilibrium (QRE) under infinite temperature (known to be the uniform distribution), a continuum of QREs is then traced out by annealing the temperature. The LLE, aptly named, is the

QRE defined in the limit of zero temperature. ADIDAS approximates this path in a way that avoids observing or storing the entire payoff tensor. Instead, it intelligently queries specific entries from the payoff tensor by leveraging techniques from stochastic optimization.

5.7.2 Nash Bargaining Solution of Normal-form games via Projected Gradient Ascent

In this section, we elaborate on the method proposed in Section 5.4.2. As an abuse of notation, let u_i denote the payoff tensor for player i flattened into a vector; similarly, let $\mathbf{x}$ be a vector as well. Let d be the number of possible joint strategies, e.g., M^N for a game with N agents, each with M pure strategies. Let Δ^{d-1} denote the $(d-1)$-simplex, i.e., $\sum_{k=1}^{d} \mathbf{x}_k = 1$ and $\mathbf{x}_k \geqslant 0$ for all $k \in \{1, \ldots, d\}$. We first make an assumption.

Assumption 5.7.1. Any agreement is better than no agreement by a positive constant κ, i.e.,

$$u_i^\top \mathbf{x} - d_i \geqslant \kappa > 0 \;\forall\; i \in \{1, \ldots, N\}, \mathbf{x} \in \Delta^{d-1}. \tag{5.4}$$

Lemma 5.7.2. *Given Assumption 5.7.1, the negative log-Nash product, $f(\mathbf{x}) = -\sum_i \log(u_i^\top \mathbf{x} - d_i)$, is convex with respect to the joint distribution $\mathbf{x}$.*

Proof. We prove $f(\mathbf{x})$ is convex by showing its Hessian is positive semi-definite. First, we derive the gradient:

$$\nabla f(\mathbf{x}) = -\sum_i \frac{u_i}{u_i^\top \mathbf{x} - d_i}. \tag{5.5}$$

We then derive the Jacobian of the gradient to compute the Hessian. The kl-th entry of the Hessian is

$$H_{kl} = \sum_i \frac{u_{ik} u_{il}}{(u_i^\top \mathbf{x} - d_i)^2}. \tag{5.6}$$

We can write the full Hessian succinctly as

$$H = \sum_i \frac{u_i u_i^\top}{(u_i^\top \mathbf{x} - d_i)^2} = \sum_i \frac{u_i u_i^\top}{\gamma_i^2}. \tag{5.7}$$

Each outer product $u_i u_i^\top$ is positive semi-definite with eigenvalues $||u_i||$ (with multiplicity 1) and 0 (with multiplicity $M^N - 1$).

Each γ_i is positive by Assumption 5.7.1, therefore, H, which is the weighted sum of $u_i u_i^\top$ is positive semi-definite as well. $\square$

We also know the following.

Lemma 5.7.3. *Given Assumption 5.7.1, the infinity norm of the gradients of the negative log-Nash product are bounded by* $\frac{u^{\max} N}{\kappa}$ *where* $u^{\max} = \max_{i,k}[u_{ik}]$.

Proof. As derived in Lemma 5.7.2, the gradient of $f(\mathbf{x})$ is

$$\nabla f(\mathbf{x}) = -\sum_i \frac{u_i}{u_i^\top \mathbf{x} - d_i}. \tag{5.8}$$

Using triangle inequality, we can upper bound the infinity (max) norm of the gradient as

$$\begin{aligned} ||\nabla f(\mathbf{x})||_\infty &= ||\sum_i \frac{u_i}{u_i^\top \mathbf{x} - d_i}||_\infty & (5.9)\\ &\leqslant \sum_i ||\frac{u_i}{u_i^\top \mathbf{x} - d_i}||_\infty & (5.10)\\ &= \sum_i \frac{||u_i||_\infty}{\gamma_i} & (5.11)\\ &\leqslant \frac{u^{\max} N}{\kappa} & (5.12)\end{aligned}$$

where $u^{\max} = \max_{i,k}[u_{ik}]$. □

Theorem 5.7.4. *Let* $\{\mathbf{x}^t\}$ *be the sequence generated by Algorithm 11 with starting point* $\mathbf{x}^0 = d^{-1}\mathbf{1}$ *and step size sequence* $\alpha^t = \frac{\sqrt{(d-1)/d}}{||g'(\mathbf{x}^s)||_2}(t+1)^{-1/2}$. *Then, for all* $t > 0$ *one has*

$$\begin{aligned} \max_{\mathbf{x}\in\Delta^{d-1}} g(\mathbf{x}) - \max_{0\leqslant s\leqslant t} g(\mathbf{x}^s) &\leqslant \frac{\sqrt{2B_\psi(\mathbf{x}^*, \mathbf{x}^0)}||g'(\mathbf{x}^s)||_2}{\sqrt{t+1}} & (5.13)\\ &\leqslant \frac{\sqrt{\frac{d-1}{d}}||g'(\mathbf{x}^s)||_2}{\sqrt{t+1}} & (5.14)\\ &\leqslant \frac{\sqrt{d}||g'(\mathbf{x}^s)||_\infty}{\sqrt{t+1}}. & (5.15)\end{aligned}$$

where $g(x) = -f(x)$ *is the log-Nash product defined in Sec 5.4.2.*

Proof. Given Assumption 5.7.1, $f(x)$ is convex (Lemma 5.7.2) and its gradients are bounded in norm (Lemma 5.7.3). We then apply Theorem 4.2 of [Beck and Teboulle, 2003] with $f(x) = -g(x)$, $\psi(\mathbf{x}) = \frac{1}{2}||\mathbf{x}||_2^2$ and note that $B_\psi(\mathbf{x}^*, \mathbf{x}^0) \leqslant \frac{1}{2}(d-1)/d$ to achieve the desired result. □

Theorem 5.7.5. *Let* $\{\mathbf{x}^t\}$ *be the sequence generated by EMDA [Beck and Teboulle, 2003] with starting point* $\mathbf{x}^0 = d^{-1}\mathbf{1}$ *and step size sequence* $\alpha^t = \frac{\sqrt{2\log(d)}}{||g'(\mathbf{x}^s)||_\infty}(t+1)^{-1/2}$. *Then, for all* $t > 0$ *one*

Game	N	Type	Description From
Kuhn poker (2P)	2	2P Zero-sum	[Lockhart et al., 2019]
Leduc poker (2P)	2	2P Zero-sum	[Lockhart et al., 2019]
Liar's dice	2	2P Zero-sum	[Lockhart et al., 2019]
Kuhn poker (3P)	3	NP Zero-sum	[Lockhart et al., 2019]
Kuhn poker (4P)	4	NP Zero-sum	[Lockhart et al., 2019]
Leduc poker (3P)	3	NP Zero-sum	[Lockhart et al., 2019]
Trade Comm	2	Common-payoff	[Sokota et al., 2021, Lanctot et al., 2019]
Tiny Bridge (2P)	2	Common-payoff	[Sokota et al., 2021, Lanctot et al., 2019]
Battleship	2	General-sum	[Farina et al., 2019]
Goofspiel (2P)	2	General-sum	[Lockhart et al., 2019]
Goofspiel (3P)	3	General-sum	[Lockhart et al., 2019]
Sheriff	2	General-Sum	[Farina et al., 2019]

Table 5.4: Benchmark games. N is the number of players.

has

$$\max_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x}) - \max_{0\leqslant s\leqslant t} g(\mathbf{x}^s) \leqslant \frac{\sqrt{2\log d}||g'(\mathbf{x}^s)||_\infty}{\sqrt{t+1}} \tag{5.16}$$

where $g(x) = -f(x)$ is the log-Nash product defined in Sec 5.4.2.

Proof. Given Assumption 5.7.1, $f(x)$ is convex (Lemma 5.7.2) and its gradients are bounded in norm (Lemma 5.7.3). We then apply Theorem 5.1 of [Beck and Teboulle, 2003] with $f(x) = -g(x)$. □

This argument concerns the regret of the joint version of the NBS where $\mathbf{x}$ is a correlation device. However, it also makes sense to try compute a Nash bargaining solution where players use independent strategy profiles (without any possibility of correlation across players), $\sigma = (\sigma_1 \times \cdots \sigma_N)$. In this case, $\mathbf{x}$ represents a concatenation of the individual strategies σ_i and the projection back to the simplex is applied to each player separately. Ensuring convergence is trickier in this case, because the expected utility may not be a linear function of the parameters which may mean the function is nonconcave.

5.7.3 Game Domain Descriptions and Details

This section contains descriptions of the benchmark games and a size estimate for Deal-or-no-Deal.

5.7.3.1 Benchmark Games

In this section, we describe the benchmark games used in this chapter. The game list is show in Table 5.4. As they have been used in several previous works, and are openly available, we simply copy the descriptions here citing the sources in the table.

Kuhn Poker Kuhn poker is a simplified poker game first proposed by Harold Kuhn. Each player antes a single chip, and gets a single private card from a totally-ordered $(n + 1)$-card deck, e.g. J, Q, K for the $(n = 2)$ two-player case. There is a single betting round limited to one raise of 1 chip, and two actions: pass (check/fold) or bet (raise/call). If a player folds, they lose their commitment (2 if the player made a bet, otherwise 1). If no player folds, the player with the higher card wins the pot. The utility for each player is defined as the number of chips after playing minus the number of chips before playing.

Leduc Poker Leduc poker is significantly larger game with two rounds and a two suits with $(N + 1)$ cards each, e.g. JS,QS,KS, JH,QH,KH in the $(N = 2)$ two-player case. Like Kuhn, each player initially antes a single chip to play and obtains a single private card and there are three actions: fold, call, raise. There is a fixed bet amount of 2 chips in the first round and 4 chips in the second round, and a limit of two raises per round. After the first round, a single public card is revealed. A pair is the best hand, otherwise hands are ordered by their high card (suit is irrelevant). Utilities are defined similarly to Kuhn poker.

Liar's Dice Liar's Dice(1,1) is dice game where each player gets a single private die in $\{1, \ldots, 6\}$, rolled at the beginning of the game. The players then take turns bidding on the outcomes of both dice, i.e. with bids of the form q-f referring to quantity and face, or calling "Liar". The bids represent a claim that there are at least q dice with face value f among both players. The highest die value, 6, counts as a wild card matching any value. Calling "Liar" ends the game, then both players reveal their dice. If the last bid is not satisfied, then the player who called "Liar" wins. Otherwise, the other player wins. The winner receives +1 and loser -1.

Trade Comm Trade Comm is a common-payoff game about communication and trading of a hidden item. It proceeds as follows.

1. Each player is independently dealt one of num items with uniform chance.
2. Player 1 makes one of num utterances utterances, which is observed by player 2.
3. Player 2 makes one of num utterances utterances, which is observed by player 1.

4. Both players privately request one of the num items $\times$ num items possible trades.

The trade is successful if and only if both player 1 asks to trade its item for player 2's item and player 2 asks to trade its item for player 1's item. Both players receive a reward of 1 if the trade is successful and 0 otherwise. We use num items = num utterances = 10.

Tiny Bridge A very small version of bridge, with 8 cards in total, created by Edward Lockhart, inspired by a research project at University of Alberta by Michael Bowling, Kate Davison, and Nathan Sturtevant.

This smaller game has two suits (hearts and spades), each with four cards (Jack, Queen, King, Ace). Each of the four players gets two cards each.

The game comprises a bidding phase, in which the players bid for the right to choose the trump suit (or for there not to be a trump suit), and perhaps also to bid a 'slam' contract which scores bonus points.

The play phase is not very interesting with only two tricks being played, so it is replaced with a perfect-information result, which is computed using minimax on a two-player perfect-information game representing the play phase.

The game comes in two varieties - the full four-player version, and a simplified two-player version in which one partnership does not make any bids in the auction phase.

Scoring is as follows, for the declaring partnership:

- +10 for making 1H/S/NT (+10 extra if overtrick)
- +30 for making 2H/S
- +35 for making 2NT
- -20 per undertrick

Doubling (only in the 4p game) multiplies all scores by 2. Redoubling by a further factor of 2.

An abstracted version of the game is supported, where the 28 possible hands are grouped into 12 buckets, using the following abstractions:

- When holding only one card in a suit, we consider J/Q/K equivalent
- We consider KQ and KJ in a single suit equivalent
- We consider AK and AQ in a single suit equivalent (but not AJ)

Battleship Sheriff is a general-sum variant of the classic game Battleship. Each player takes turns to secretly place a set of ships S (of varying sizes and value) on separate grids of size $H \times W$. After placement, players take turns firing at their opponent—ships which have been hit at all the tiles they lie on are considered destroyed. The game continues until either one player has lost all of their ships, or each player has completed r shots. At the end of the game, the payoff of each player is computed as the sum of the values of the opponent's ships that were destroyed, minus γ times the value of ships which they lost, where $\gamma \geqslant 1$ is called the loss multiplier of the game.

In this work we use $\gamma = 2, H = 2, W = 2$, and $r = 3$. The OpenSpiel game string is:

```
battleship(board_width=2,board_height=2,
ship_sizes=[1;2], ship_values=[1.0;1.0],
num_shots=3,loss_multiplier=2.0)
```

Goofspiel Goofspiel or the Game of Pure Strategy, is a bidding card game where players are trying to obtain the most points. shuffled and set face-down. Each turn, the top point card is revealed, and players simultaneously play a bid card; the point card is given to the highest bidder or discarded if the bids are equal. In this implementation, we use a fixed deck of decreasing points. In this work, we an imperfect information variant where players are only told whether they have won or lost the bid, but not what the other player played. The utility is defined as the total value of the point cards achieved.

In this work we use $K = 4$ card decks. So, e.g. the OpenSpiel game string for the three-player game is:

```
goofspiel(imp_info=True,returns_type=total_points,
players=3,num_cards=4)
```

Sheriff Sheriff is a simplified version of the Sheriff of Nottingham board game. The game models the interaction of two players: the Smuggler—who is trying to smuggle illegal items in their cargo– and the Sheriff– who is trying to stop the Smuggler. At the beginning of the game, the Smuggler secretly loads his cargo with $n \in \{0, ..., n_{max}\}$ illegal items. At the end of the game, the Sheriff decides whether to inspect the cargo. If the Sheriff chooses to inspect the cargo and finds illegal goods, the Smuggler must pay a fine worth $p \cdot n$ to the Sheriff. On the other hand, the Sheriff has to compensate the Smuggler with a utility if no illegal goods are found. Finally, if the Sheriff decides not to inspect the cargo, the Smuggler's utility is $v \cdot n$ whereas the Sheriff's utility is 0. The game is made interesting by two additional elements (which are also present in the board game): bribery and bargaining. After the Smuggler has loaded the cargo and before the Sheriff chooses whether or not to inspect, they engage in r rounds of bargaining. At each round

Hyperparameter Name	Values in DoND	Values in CT (PSRO w/ DQN BR)
AlphaZero training episodes num_eps	10^4	10^4
Network input representation	an infoset vector $\in \mathbb{R}^{309}$ implemented in [Lanctot et al., 2019]	an infoset vector $\in \mathbb{R}^{463}$
Network size for policy & value nets	[256, 256] for the shared torso	[1024, 512, 256]
Network optimizer	SGD	SGD
UCT constant c_{uct}	20 for IR and IE, 40 for SW, 100 for NP	
Returned policy type	greedy towards $\boldsymbol{v}$	argmax over learned q-net
# simulations in search	300	
Learning rate for policy & value net	2e-3 for IR and IE, 1e-3 for SW, 5e-4 for NP	1e-3
Delayed # episodes for replacing $\boldsymbol{v}, \boldsymbol{p}, \boldsymbol{g}$	200	200
Replay buffer size $\lvert D \rvert$	2^{16}	
Batch size	64	128
PSRO empirical game entries # simulation	200	2000
L2 coefficient in evaluator c_1	1e-3	
Network size for generator net	[300, 100] MLP	
Learning rate for generator net	1e-3	
L2 coefficient in generator net c_2	1e-3	
PSRO minimum pure-strategy mass lower bound	0.005	0.005

Table 5.5: Hyper-parameters.

$i = 1, ..., r$, the Smuggler tries to tempt the Sheriff into not inspecting the cargo by proposing a bribe $b_i \in \{0, ...b_{max}\}$, and the Sheriff responds whether or not they would accept the proposed bribe. Only the proposal and response from round r will be executed and have an impact on the final payoffs—that is, all but the r^{th} round of bargaining are non-consequential and their purpose is for the two players to settle on a suitable bribe amount. If the Sheriff accepts bribe b_r, then the Smuggler gets $p \cdot n - b_r$, while the Sheriff gets b_r.

In this work we use values of $n_{max} = 10, b_{max} = 2, p = 1, v = 5$, and $r = 2$. The OpenSpiel game string is:

```
sheriff(item_penalty=1.0,item_value=5.0,
max_bribe=2,max_items=2,num_rounds=2,
sheriff_penalty=1.0)
```

5.7.3.2 Approximate Size of Deal or No Deal

To estimate the size of Deal or No Deal described in Section 5.5.3, we first verified that there are 142 unique preference vectors per player. Then, we generated 10,000 simulations of trajectories using uniform random policy computing the average branching factor (number of legal actions per player at each state) as $b \approx 23.5$.

Since there are 142 different information states for player 1's first decision, about $142bb$ player 1's second decision, etc. leading to $142(1+b^2+b^4+b^8) \approx 13.2\times10^{12}$ information states. Similarly, player 2 has roughly $142(1 + b^1 + b^3 + b^5 + b^7) = 5.63 \times 10^{11}$ information states.

5.7.4 Hyper-parameters and Algorithm Settings

We provide detailed description of our algorithm in this section. Our implementation differs from the original ABR [Timbers et al., 2022] and AlphaZero [Silver et al., 2018] in a few places. First, instead of using the whole trajectory as a unit of data for training, we use per-state pair data. We maintain separate data buffers for policy net, value net, and generate net, and collect corresponding (s_i, r), (s_i, p), (s_i, h) to them respectively. On each gradient step we will sample a batch from each of these data buffer respectively. The policy net and value net shared a torso part, while we keep a separate generative net (detailed in Section 5.7.4.1). The policy net and value net are trained by minimizing loss $(r - v)^2 - \pi^{*T} \log p + c_1 \|\theta\|_2^2$, where (1) π^* are the policy targets output by the MCTS search, (2) p are the output by the policy net, (3) v are the output by the value net, (4) r are the outcome value for the trajectories (5) θ are the neural parameters. During selection phase the algorithm select the child that maximize the following PUCT [Silver et al., 2018] term: $\texttt{MaxPUCT}(s, \boldsymbol{p}) = \arg\max_{a \in \mathcal{A}(s)} \frac{s.child(a).value}{s.child(a).visits} + c_{uct_c} \cdot \boldsymbol{p}(s)_a \cdot \frac{\sqrt{s.total\_visits}}{s.child(a).visits+1}$. Other hyperparameters are listed in Table 5.5.

5.7.4.1 Generative Models

Deal or No Deal In DoND given an information state, the thing we only need to learn is the opponent's hidden utilities. And since the utilities are integers from 0 to 10, we design our generative model as a supervised classification task. Specifically, we use the 309-dimensional state s_i as input and output three heads. Each head corresponds to one item of the game. each head consists of 11 logits corresponding to each of 11 different utility values. Then each gradient step we sample a batch of (s_i, h) from the data buffer. And then do one step of gradient descent to minimize the cross-entropy loss $-h^T \log \boldsymbol{g}(s) + c_2 \|\theta'\|_2^2$, where here we overload h to represent a one-hot encoding of the actual opponent utility vectors. And each time we need to generate a new state at information state s_i, we just feed forward s_i and sample the utilities according to the output logits. But since we have a constraints $\mathbf{v}_1 \cdot \mathbf{p} = \mathbf{v}_2 \cdot \mathbf{p} = 10$, the sampled utilities may not always be feasible under this constraint. Then we will do an additional L2 projection on to all feasible utility vector space and then get the final results.

5.7.5 Additional Results

The full analysis of empirical convergence to approximate Nash equilibrium and social welfare are shown for various settings:

- Two-player Zero-Sum Games: Figure 5.7.
- N-player Zero-Sum Games: Figure 5.8.

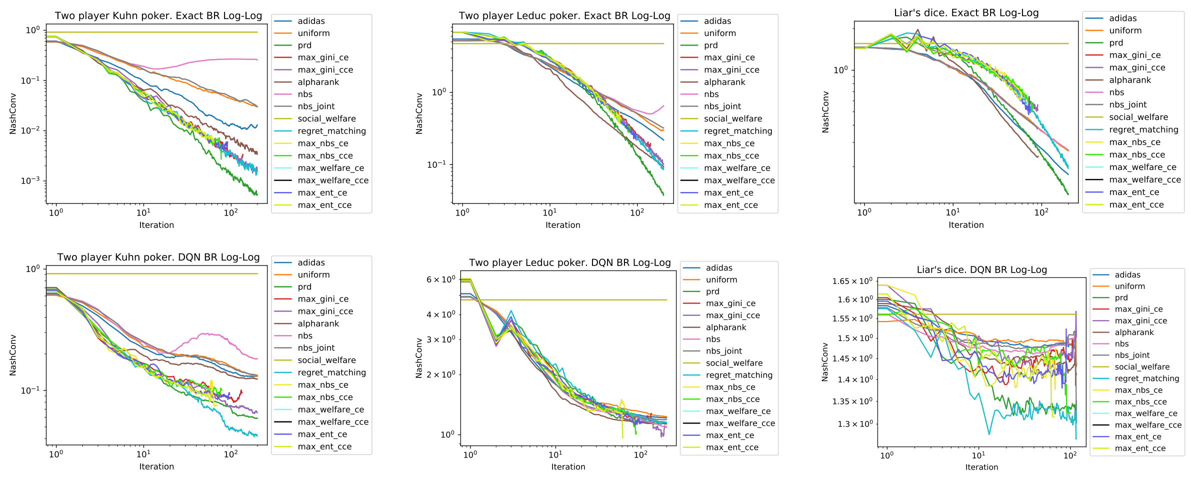


Figure 5.7: Empirical Convergence to Nash Equilibria using Exact vs. DQN Best Response versus in Two-Player Zero-Sum Benchmark Games.

- Common Payoff Games: Figure 5.9.
- General-Sum Games: Figure 5.10.

5.7.5.1 Colored Trails

The full Pareto gap graphs as a function of training time is shown in Figure 5.11. Some examples of the evolution of expected outcomes over the course of PSRO training are shown in Figure 5.12.

5.7.5.2 Deal or No Deal

In this section, we described how we train and select the PSRO agents in DoND human behavioral studies. Due the experimental limitation, we can only select 5 of our agents to human experiments. For convenience from now on we label a PSRO agent as $(\mathit{MSS}, \mathit{BACKPROP\_TYPE}, \mathit{FINAL\_TYPE})$ if it is trained under meta-strategy solver *MSS*, its back-propagation type is *BACKPROP_TYPE* and we use *FINAL_TYPE* as its decision architecture. We apply standard empirical game theoretic analysis [Wellman, 2006, Jordan et al., 2007] on our agent pool: we create a 113x113 symmetric empirical game by simulating head to head results between every pair of our agents. During each simulation we toss a coin to assign the roles (first-mover or second-mover in DoND) to our agents. Then we decide the selected agents based on this empirical game matrix.

Specifically, we want to select: (1) the most competitive agents (2) the most collaborative agents and (3) the fairest agent in our pool in a principled way. Now we explain how we approach these criterions one by one.

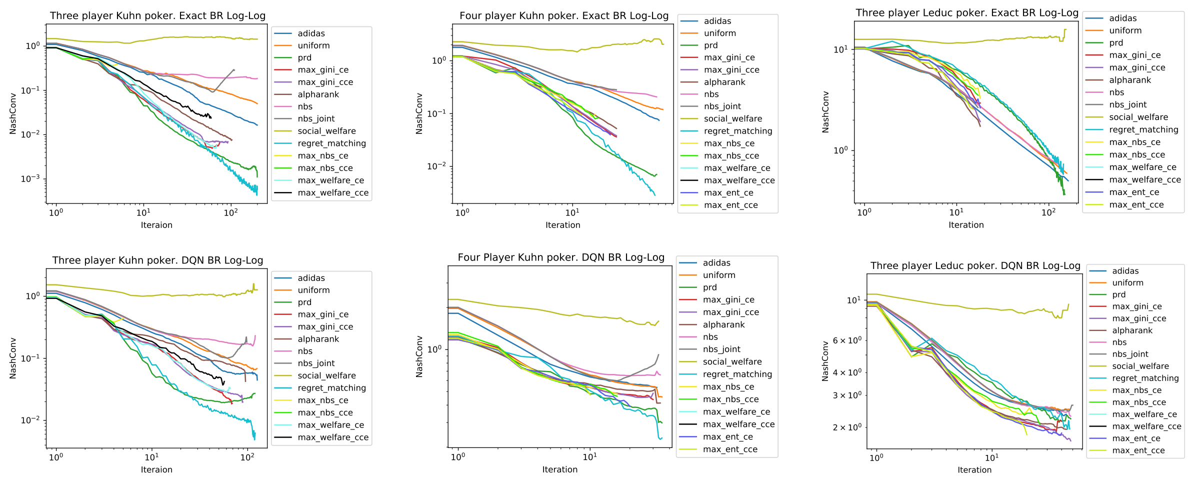


Figure 5.8: Empirical Convergence to Nash Equilibria using Exact vs. DQN Best Response in N-Player Zero-Sum Benchmark Games.

For (1), we apply our competitive MSSs (e.g., ADIDAS, CE/CCE solvers) on this empirical game matrix and solve for a symmetric equilibrium. Then we rank the agents according to their expected payoff when the opponents are playing according to this equilibrium. This approach is inspired by Nash-response ranking in [Jordan et al., 2007] and Nash averaging in [Balduzzi et al., 2018b] which is recently generalized to any N-player general-sum games [Marris et al., 2022]. We found that Independent DQN, (MGCE, IA, SP) (denoted as Comp1), (MGCCE, IR, SP) (denoted as Comp2) rank at the top under almost all competitive MSS.

For (2), we create two collaborative games where the payoffs of agent1 v.s. agent2 are their social welfare/Nash product. We conduct the same analysis as in (1), and found the agent (uniform, NP, RP) normally ranks the first (thus we label it as Coop agent).

For (3), we create an empirical game where the payoffs of agent1 v.s. agent2 are the negative of their absolute payoff difference, and apply the same analysis. We also conduct a Borda voting scheme: we rank the agent pairs in increasing order of their absolute payoff difference. Each of these agent pairs will got a Borda voting score. Then the score of an agent is the summation of the scores of agents pairs that this agent is involved in. In both approaches, we find the agent (MGCE, NP, RP) ranks the top. Therefore we label it as Fair agent.

To summary, the five agents we finally decided to conduct human experiments are (1) DQN trained through self-play (IndRL) (2) (MGCE, IA, SP) (Comp1) (3) (MGCCE, IR, SP) (Comp2) (4) (uniform, NP, RP) (Coop) (5) (MGCE, NP, RP) (Fair).

We show the head-to-head empirical game between these five agents in Table 5.6, social-welfare in Table 5.7, empirical Nash product in Table 5.8.

Agent \ Opponent	IndRL	Com1	Com2	Coop	Fair
IndRL	4.85	4.98	5.02	7.05	6.96
Com1	3.75	4.97	4.66	7.19	6.70
Com2	3.20	5.30	5.04	6.84	6.86
Coop	5.63	4.43	4.32	6.67	6.64
Fair	5.47	4.43	4.19	6.59	6.52

Table 5.6: Head-to-head empirical game matrix among our selected agents . The (i, j)-th entry is the payoff of the i-th agent when it is playing with the j-th agent.

Agent \ Opponent	IndRL	Com1	Com2	Coop	Fair
IndRL	9.70	8.73	8.22	12.68	12.42
Com1	8.73	9.94	9.96	11.62	11.12
Com2	8.22	9.96	10.09	11.16	11.05
Coop	12.68	11.62	11.16	13.34	13.22
Fair	12.42	11.12	11.05	13.22	13.05

Table 5.7: Head-to-head empirical social welfare matrix among our selected agents . The (i, j)-th entry is the social welfare of when i-th agent is playing with the j-th agent.

Agent \ Opponent	IndRL	Com1	Com2	Coop	Fair
IndRL	23.48	18.69	16.05	39.66	38.02
Com1	18.69	24.70	24.68	31.84	29.63
Com2	16.05	24.68	25.44	29.57	28.73
Coop	39.66	31.84	29.57	44.51	43.70
Fair	38.02	29.63	28.74	43.70	42.56

Table 5.8: Head-to-head empirical Nash product matrix among our selected agents . The (i, j)-th entry is the Nash product when i-th agent is playing with the j-th agent.

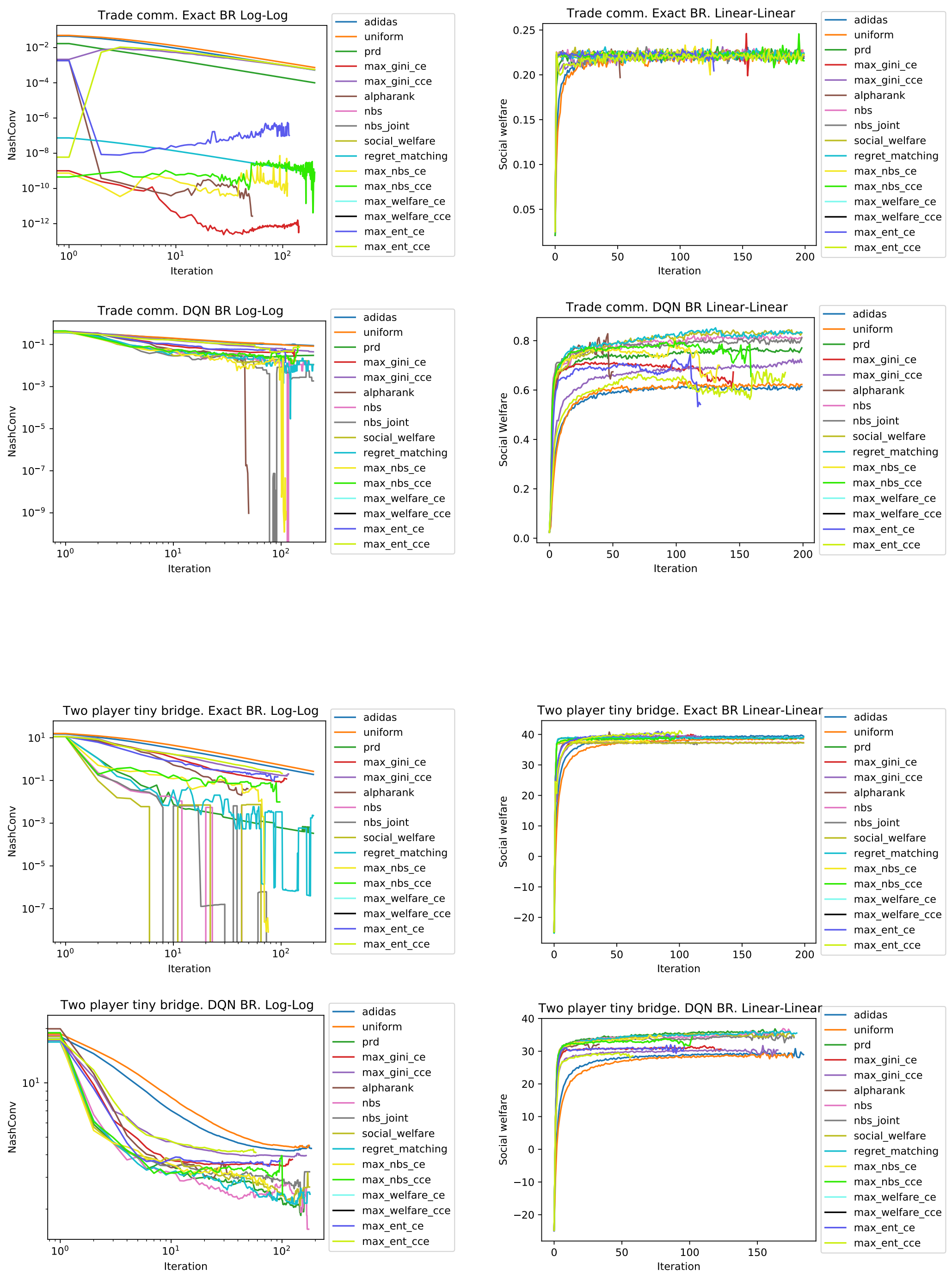


Figure 5.9: Empirical Convergence to Nash Equilibria and Social Welfare in Common Payoff Benchmark Games.

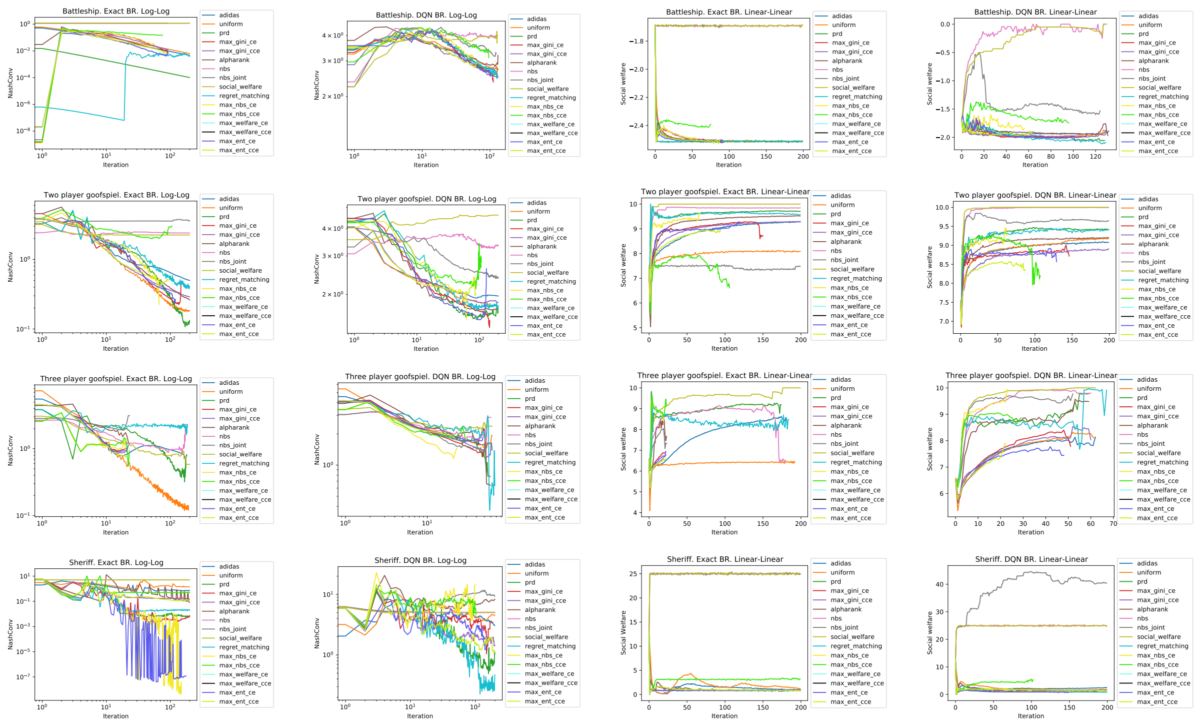
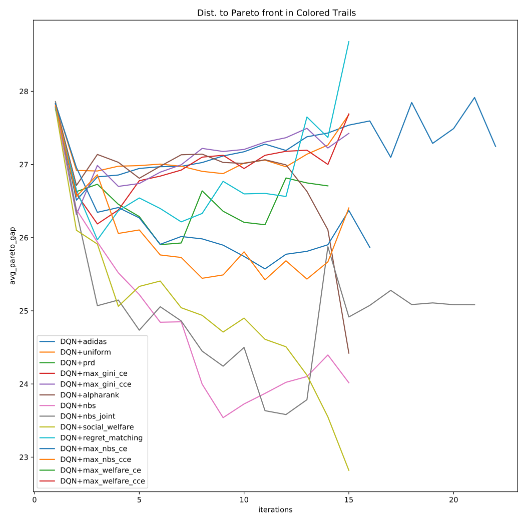
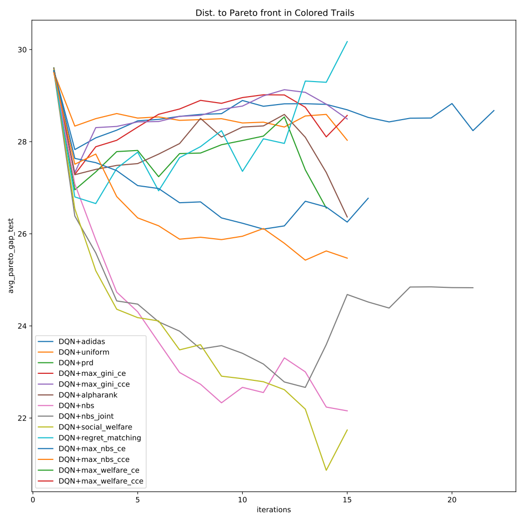


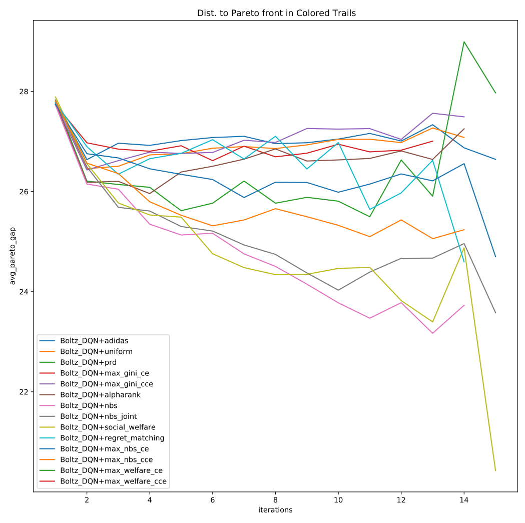
Figure 5.10: Empirical Convergence to Nash Equilibria and Social Welfare in General-Sum Benchmark Games.



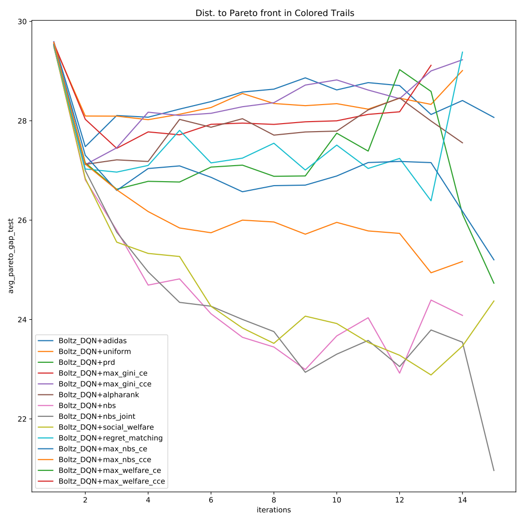
(a)



(b)



(c)



(d)

Figure 5.11: Average Pareto gap using DQN best response (top: (a) and (b)) and Boltzmann DQN (bottom: (c) and (d)), training gap (left: (a) and (c)) and gap on held-out test boards (right: (b) and (d)).

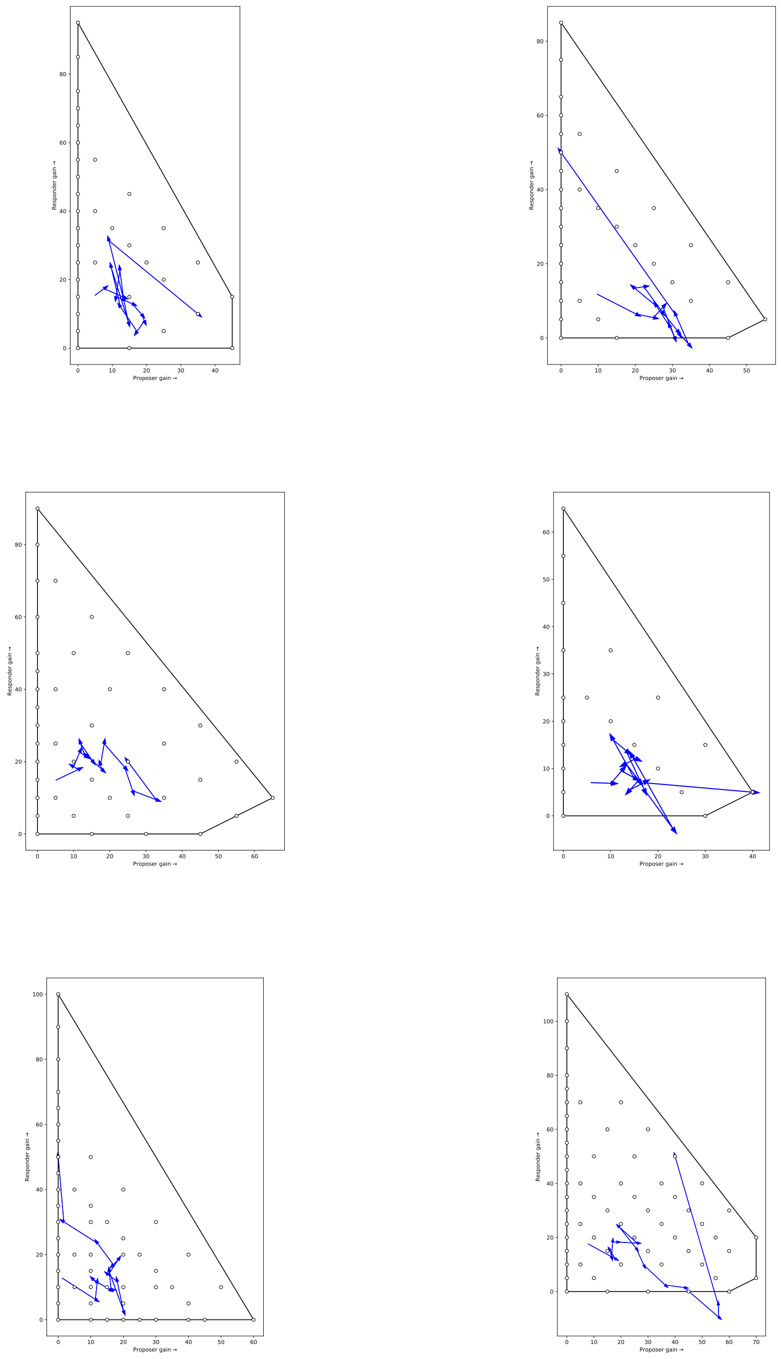


Figure 5.12: Evolution of the expected outcomes of the PSRO agents using the DQN best response type and social welfare MSS. Each diagram depicts the outcome of the agent for a single configuration of Colored Trails: circles represent the rational outcomes (pure joint strategies where players have non-negative gain). The outer surface of the convex hull represents the Pareto front/envelope. To make a 2D image, the proposers' gains are aggregated and only the winning proposer's value is included in the outcome computation. The blue directed path represents the PSRO agents' expected outcomes at iterations $t \in \{0, 1, \cdots, 15\}$, where each point estimated from 100 samples. Note that values can be negative due to sampling approximation but also due to choosing legal actions that result in negative gain.

CHAPTER 6

From Solutions to Evaluation: A Meta-Game Analysis Framework for Evaluating Interactive AI Algorithms

Evaluating interactive AI algorithms, such as deep multiagent reinforcement learning (MARL) algorithms, is complicated by stochasticity in training and sensitivity of agent performance to the behavior of other agents. We propose a meta-game evaluation framework for deep MARL, by framing each MARL algorithm as a *meta-strategy*, and repeatedly sampling normal-form empirical games over combinations of meta-strategies resulting from different random seeds. Each empirical game captures both *self-play* and *cross-play* factors across seeds. These empirical games provide the basis for constructing a sampling distribution, using bootstrapping, over a variety of game analysis statistics. We use this approach to evaluate state-of-the-art deep MARL algorithms on a class of negotiation games. From statistics on individual payoffs, social welfare, and empirical best-response graphs, we uncover strategic relationships among self-play, population-based, model-free, and model-based MARL methods. We also investigate the effect of run-time search as a *meta-strategy operator*, and find via meta-game analysis that the search version of a meta-strategy generally leads to improved performance.

Evaluating complex AI algorithms requires careful attention to stochastic factors. Deep reinforcement learning (RL) algorithms in particular are subject to randomness within the algorithm and the operational environment, and variations with choices of hyperparameters and initial conditions. It is conventional to address these uncertainties in part by aggregating results across multiple runs [Bellemare et al., 2013, Machado et al., 2018]. Deep multiagent RL (MARL) algorithms present these issues, and additional challenges due to agent interactions. Evaluating performance against humans has been one source of compelling demonstrations [Silver et al., 2016, Vinyals et al., 2019, Wurman et al., 2022, Perolat et al., 2022], but this approach is limited by the range of tasks for which human expertise exists, and the cost of engaging it when it is available. Generally speaking, we lack evaluation protocols for comparing different MARL methods in a statistically principled way.

In purely adversarial (*i.e.*, two-player zero-sum) environments, distance to Nash equilibrium may be a sufficient evaluation metric [Brown et al., 2020, Schmid et al., 2023], as all equilibria are interchangeably optimal. More generally, where there are multiple equilibria or where we do not necessarily expect equilibrium behavior, the metrics for MARL performance may be less clear. In collaborative domains, global team return is the common objective [Foerster et al., 2018, Rashid et al., 2020], however complex learning dynamics may lead agents using the same MARL algorithm to equilibria of distinct machine conventions in different runs [Hu et al., 2020, Bakhtin et al., 2021].

We seek an approach to evaluating deep MARL in general-sum domains. We propose a *meta-game* evaluation framework (§6.4), which frames MARL algorithms as *meta-strategies*: mapping games and random seeds to joint policies. We sample seed combinations to generate meta-game instances, from which we compute evaluation metrics of interest based on game-theoretic solution concepts. Through resampling and bootstrap techniques, we generate a statistical characterization of algorithm performance in these games. Our contributions:

- The meta-game evaluation framework.
- A new search algorithm for imperfect information games, *Gumbel Information-Set Monte Carlo Tree Search* (§6.5), based on the recent development of Gumbel AlphaZero [Danihelka et al., 2022].
- Extensive experiments on state-of-the-art MARL algorithms (§6.6.2) on a class of negotiation games (§6.6.1), illustrating the framework and providing new evidence regarding the algorithms studied.

6.1 Related Work

While evaluating single-agent deep RL algorithms is well-studied [Henderson et al., 2018, Jordan et al., 2020, Agarwal et al., 2021], there are relatively few works that consider evaluation principles for deep MARL [Gronauer and Diepold, 2022]. Typically, agents are evaluated against a selected set of background opponents or emergent behaviors in certain contexts [Lowe et al., 2017, Li et al., 2019, Song et al., 2020, Leibo et al., 2021]. Other work has defined evaluation protocols for cooperative settings [Papoudakis et al., 2020, Gorsane et al., 2022], where a global team reward is well-defined. Our scenario falls into the *agent-vs-agent* category of Balduzzi et al. [2018b], who argue for the necessity of comprehensively considering the possible joint interactions.

Kiekintveld and Wellman [2008] employed a concept of meta-games for evaluating general game-playing methods. In their setting, meta-strategies map normal-form game specifications to strategy choices. Treutlein et al. [2021] construct *label-free coordination* (LFC) games among instances of a cooperative MARL algorithm, in order to study zero-shot coordination [Hu et al., 2020]. LFC games are a kind of meta-game in our sense, as they take dynamic game descriptions (Dec-POMDPs) as input.

Bootstrapping is a non-parametric statistical approach that constructs sampling distributions for any specified statistic by resampling from the original dataset [Davison and Hinkley, 1997]. Bootstrapping techniques have been applied in machine learning methods such as aggregating decision-tree models [Breiman, 1996]. Wiedenbeck et al. [2014] applied the bootstrap for statistical analysis of game-theoretic models estimated from simulations.

We test our methods on a class of sequential bargaining games, specifically a bilateral alternating-offer negotiation version developed for recent studies in multiagent emergent communication [Lewis et al., 2017, Cao et al., 2018]. These games also resemble the multi-issue negotiation settings explored in multiagent systems research, exemplified in the Automated Negotiation Agent Competition [Baarslag et al., 2012]. Much research on agent-based negotiation focuses on heuristic, knowledge-based strategies [Jennings et al., 2001, Fatima et al., 2014]. A few recent works apply RL methods for bargaining games. Bakker et al. [2019] proposed Q-learning-based negotiation strategies with domain knowledge incorporated. Higa et al. [2023] employed policy gradient approaches and Chen et al. [2023] trained deep RL strategies using off-line interaction data with humans.

6.2 Game Theory Preliminaries

A ***normal-form representation*** of a ***game*** $\mathcal{G}$ consists of a ***player set*** $\mathcal{N} = \{1, \dots, N\}$, and for each player $i \in \mathcal{N}$ a ***pure strategy space*** Π_i and a ***utility function*** $u_i : \Pi \to \mathbb{R}$. $\Pi = \Pi_1 \times \cdots \times \Pi_N$ is the ***joint pure strategy space***. A ***mixed strategy*** $\sigma_i \in \Delta(\Pi_i)$ for player i defines a probability distribution over that player's pure strategy space. Player i's ***payoff*** for choosing σ_i while others play $\sigma_{-i} = (\sigma_j)_{j \neq i}$ is given by expectation over the respective mixtures: $u_i(\sigma) = \mathbb{E}_{\pi_i \sim \sigma_i, \pi_{-i} \sim \sigma_{-i}}[u(\pi_i, \pi_{-i})]$.

The ***regret*** of player i who plays σ_i when others are playing σ_{-i} is $\textsc{Regret}_i(\sigma_i, \sigma_{-i}) = \max_{\pi'_i \in \Pi_i} u_i(\pi'_i, \sigma_{-i}) - u_i(\sigma_i, \sigma_{-i})$. A ***Nash equilibrium*** (NE) is a mixed strategy profile σ such that nobody has positive regret: $\forall i.\ \textsc{Regret}_i(\sigma_i, \sigma_{-i}) = 0$. As a measure of approximation to NE, we define $\textsc{NashConv}(\sigma) = \sum_i \textsc{Regret}_i(\sigma_i, \sigma_{-i})$.

If a game is known to be symmetric, we can reduce its normal-form description complexity. Formally, in a ***symmetric game***, players share the same strategy space $\Pi_1 = \cdots = \Pi_N = \Pi$, and utility function $u_1 = \cdots = u_N = u$. Furthermore, each player's utility is permutation-invariant to other players' strategies. For symmetric games we overload Π to refer to the common individual strategy space, rather than the joint space. In a ***symmetric profile***, every player adopts the same (generally mixed) strategy. Solutions in symmetric profiles are guaranteed to exist in relevant settings [Nash, 1951, Cheng et al., 2004, Hefti, 2017], and are generally preferred absent any basis for breaking symmetry [Kreps, 1990].

Also given symmetry, let $u(\sigma', \boldsymbol{\sigma})$ be the expected payoff of a player when it chooses a strategy $\sigma' \in \Delta(\Pi)$ while the other $N - 1$ play the same mixed strategy $\sigma \in \Delta(\Pi)$. We thus have $\textsc{Regret}(\sigma', \sigma) = \max_{\pi' \in \Pi} u(\pi', \boldsymbol{\sigma}) - u(\sigma', \boldsymbol{\sigma})$. A symmetric NE strategy σ satisfies $\textsc{Regret}(\sigma, \sigma) = 0$.

An ***extensive-form game*** (EFG) representation goes beyond normal form to include temporal and information structure. Effectively, an EFG defines a dynamical system (usually visualized by a tree) with a world state or history h that is not necessarily fully observable by every player. At each state h, starting with the initial state h_0, there is a single player $\tau(h) \in \mathcal{N} \cup \{c\}$, where c is the ***chance player***, designated to select an action $a \in \mathcal{A}(h)$. The chance player chooses according to a fixed random policy. Player $i \in \mathcal{N}$ chooses according to its ***information state/set***, $s_i(h)$, which comprises the information that i has observed at h. Often $s_i(h)$ is represented by a concatenation of public and private action-observation histories. Following action a, the world transits to $h' = ha$ until a terminal state $z \in \mathcal{Z}$ is reached. Then each player $i \in \mathcal{N}$ receives a utility $u_i(z)$ as a function of z. A ***behavioral strategy*** or ***policy***, π_i, of player i is a mapping from i's infostates to distributions over actions. For EFGs we overload Π_i to refer to the set of behavioral strategies of player i. A joint behavioral strategy profile π thus induces a probability distribution over the terminal states and we define $u_i(\pi)$ as the expected payoff for player i under this distribution. In this paper, we consider EFGs with ***perfect recall***, that is, information sets $s_i(h)$ distinguish all actions i had taken to reach h. A consequence is that any mixed strategy $\sigma_i \in \Delta(\Pi_i)$ is payoff-equivalent to some behavioral strategy $\pi_i \in \Pi_i$ [Aumann, 1964].

6.3 Multiagent Training Algorithms

We define a ***multiagent training algorithm*** (MATA) $\mathcal{M}$ as a stochastic procedure that produces a policy profile $\pi = \mathcal{M}(\mathcal{G}, \Theta, \omega)$ for an EFG. In general, the input EFG $\mathcal{G}$ cannot be tractably represented as an explicit game tree. Instead, we assume the game is given in the form of a black-box simulator that the algorithm can exercise by submitting actions and receiving observations and rewards. Θ is the set of hyperparameters of the algorithm, and ω is a random seed. If $\mathcal{G}$ is symmetric, then it is often natural to constrain the output π to be likewise symmetric (*i.e.*, single policy to be played by all). More generally, π is a policy profile. For the AlphaZero MATA, for example, $\mathcal{G}$ could be represented by a Go simulator, and Θ would include the learning rate schedule and neural architecture. The output profile π specifies Go-playing policies for white and black, respectively.

A MATA is effectively a form of ***meta-strategy***: a procedure that given a game $\mathcal{G}$, generates a strategy profile for $\mathcal{G}$. We can employ the MATA to play from the perspective of any particular player i, simply by selecting the i^{th} element $\pi(i)$ of the output profile.

A key issue for analysis of MATAs is uncertainty in strategy generation. It has been well observed (*e.g.*, by Hu et al. [2020] and Bakhtin et al. [2021]) that a MATA with the same $\mathcal{G}$ and Θ but different ω may generate policies with vastly different strategic behaviors. For example, in a negotiation game, different runs of a MATA may lead to strategies that adopt distinct offering conventions. In the present work, we assume the hyperparameters Θ^m for each MATA $\mathcal{M}^m$ have been fixed, so the uncertainty in behavior of a training algorithm is fully captured by the random seeds. Note that there is always discretion about what one considers a distinct MATA $\mathcal{M}$ versus a parametric variation, so it is possible to bring choice among hyperparameter settings within the scope of our analysis framework.

6.4 Meta-Game Evaluation Framework

Given M different MATAs $\{(\mathcal{M}^1, \Theta^1), \ldots, (\mathcal{M}^M, \Theta^M)\}$ with associated hyperperameters, how can we evaluate their relative performance with respect to a given game $\mathcal{G}$? Viewing the MATAs simply as game-solvers, we could focus on measures of their effectiveness in deriving a solution—for example, time and accuracy of convergence to Nash equilibrium. Viewing the MATAs' role as generating strategies to play a game, however, requires a different focus that considers the interaction with other strategy generators. This is particularly salient for games that have a multiplicity of solutions (the general case), or for which the operable solution concept may be open to question. Consequently, we propose to analyze competing MATAs by framing their interaction as itself a game. As MATAs are *meta-strategies*, we refer to this approach as a ***meta-game*** evaluation framework.

As noted above, we are particularly concerned with uncertainty in the results of multiagent training. In analysis of the MATA meta-game, therefore, we aim to characterize the implications of this uncertainty in probabilistic form.

6.4.1 Empirical Game-Theoretic Analysis

Our approach employs ***empirical game-theoretic analysis*** (EGTA), a methodology for reasoning about games through agent-based simulation [Tuyls et al., 2020, Wellman, 2016]. EGTA aligns with our assumption that the game of interest $\mathcal{G}$ is defined by a black-box simulator. In the typical framing, EGTA operates by estimating an ***empirical game model*** in normal form over an enumerated strategy set. The enumerated strategies are a small selection from the full strategy space of the extensive game represented by the simulator.

We employ EGTA with respect to an N-player game of interest $\mathcal{G}$, and strategies defined by the output of MATAs. The meta-game is likewise over N players, and is symmetric regardless of whether $\mathcal{G}$ is symmetric or not. Let $\hat{\pi}^m$ denote the output from MATA m, for instance $\hat{\pi}^m = \mathcal{M}^m(\mathcal{G}, \Theta^m, \omega)$: the result from running the MATA on $\mathcal{G}$ for a particular random seed. We also allow that $\hat{\pi}^m$ be an aggregate of policy profiles from multiple random seeds. From these MATA-generated policy profiles $\hat{\Pi} = \{\hat{\pi}^1, \ldots, \hat{\pi}^M\}$, we construct an empirical meta-game $\mathcal{MG}(\hat{\Pi})$ over policy space $\hat{\Pi}$ as follows.

If the base game $\mathcal{G}$ is symmetric, then the $\hat{\pi}^m$ are single-player policies, and we estimate the meta-game utility function by the standard EGTA approach of simulating profiles over $\hat{\Pi}$. If $\mathcal{G}$ is not symmetric, then each $\hat{\pi} \in \hat{\Pi}$ is an N-player profile for $\mathcal{G}$. Simulating a meta-game profile $(\hat{\pi}_1, \ldots, \hat{\pi}_N)$ of these base-game profiles entails first assigning the $\hat{\pi}$ to players, according to a random permutation $perm$ drawn uniformly from the $N!$ possibilities. Then it simulates a play where player i of $\mathcal{MG}$ plays $\hat{\pi}_i(perm(i))$ as if it is player $perm(i)$ in $\mathcal{G}$. This construction also corresponds to an EFG beginning with a root chance node that uniformly chooses among $N!$ different outcomes, each followed by a copy of the original EFG with player indices permuted.

Symmetry of the meta-game reflects a view that, for multiagent training, developing effective strategies for each of the player positions is equally important. If this were not the case, or if one wished to perform an analysis of the differential effectiveness of various MATAs from the perspectives of different players or roles, a non-symmetric (or role-symmetric) meta-game model could be constructed instead.

6.4.2 Meta-Game Evaluation Procedure

Just as single-agent RL algorithms are statistically evaluated by return performance across different random seeds, we can analyze strategic properties among MATAs across different *combinations* of seeds. Let $\boldsymbol{X}$ denote statistics characterizing the strategic properties of interest (discussed below). Given an N-player game $\mathcal{G}$ and parametrized MATAs $\{(\mathcal{M}^1, \Theta^1), \ldots, (\mathcal{M}^M, \Theta^M)\}$, our evaluation procedure comprises of the following steps:

1. Select a finite set of seeds Ω^m for each MATA m. Generate $\hat{\pi}^m(\omega) = \mathcal{M}^m(\mathcal{G}, \Theta^m, \omega)$ for each $\omega \in \Omega^m$.

2. For each m, uniformly sample $|\Omega^m|$ seeds from Ω^m with replacement, yielding the sequence $(\omega_1, \ldots, \omega_{|\Omega^m|})$. Let $\hat{\pi}^m$ be a profile that that is payoff-equivalent to a uniform mixture over the multiset $\{\hat{\pi}^m(\omega_1), \ldots, \hat{\pi}^m(\omega_{|\Omega^m|})\}$.

3. Given $\hat{\Pi} = \{\hat{\pi}^1, \ldots, \hat{\pi}^M\}$, estimate the symmetric empirical meta-game $\mathcal{MG}(\hat{\Pi})$ as described in §6.4.1.

4. Compute the statistics-of-interest $\boldsymbol{X}$ from $\mathcal{MG}(\hat{\Pi})$

5. Repeat Steps 2 through 4. Estimated profile payoffs should be memoized for reuse across iterations. Obtain an empirical distribution of $\boldsymbol{X}$ and report statistical properties of $\boldsymbol{X}$.

One way to understand this evaluation procedure is to view each MATA as a *mixed strategy*, selecting policies uniformly over the possible random seeds. The "ground-truth" meta-game represents an expectation over the results of this randomization. The empirical meta-game estimates this from finite samples Ω^m. By resampling from these seeds at hand, Step 2 constructs multiple empirical games among MATAs, from which we construct sampling distributions over $\boldsymbol{X}$ using bootstrapping.

There are a variety of choices for statistics $\boldsymbol{X}$ to gather. The only requirement is that $\boldsymbol{X}$ can be computed from the information in a normal-form game model. For example, one possible metric on MATA performance is ***uniform-score***: average payoff against a uniform distribution over other MATAs. Such scores have been employed in a variety of contexts, however putting equal weight on the possible counterparts is questionable, as they are not equally relevant [Balduzzi et al., 2018b].

An alternative proposed by Jordan et al. [2007] is ***NE-regret***: $\text{REGRET}(\pi, \sigma^*)$, where σ^* is a symmetric mixed equilibrium of $\mathcal{MG}(\hat{\Pi})$. The motivation for this measure is that it focuses on behavior against rational opponents. Performance against obviously flawed opponents should carry much less weight, as they are less likely to be encountered in practice, all else equal.[1] For games with multiple equilibria, however, NE-regret is sensitive to the choice of solutions σ^*. This sensitivity is inherent to situations with multiple equilibria, but it can still be helpful to adopt a focal equilibrium to reduce ambiguity in analysis. Balduzzi et al. [2018b] proposed ***Nash averaging***, which is essentially NE-regret with respect to the max-entropy equilibrium. The intuition for preferring to evaluate with respect to higher-entropy solutions is that they reflect diversity, and thus reward robustness to a wide range of rational opponents.

6.4.3 Max-Entropy Nash Equilibrium

Computing max-entropy NE is hard in general, but practically feasible to approximate for bi-matrix games of modest size. We adapt the mixed-integer programming formulation of Sandholm et al.

[1]If however there is some basis to expect opponents who are flawed or boundedly rational in some particular way, then by all means it would make sense to measure regret with respect to a solution concept capturing that basis.

[2005]:

$$\begin{aligned} \min_{\sigma^*} \quad & \sum_{\pi \in \hat{\Pi}} \sigma^*(\pi) \log \sigma^*(\pi) \\ \text{s.t.} \quad & \forall \pi \in \hat{\Pi}. \quad u_\pi = \sum_{\pi' \in \hat{\Pi}} \sigma^*(\pi') u(\pi, \boldsymbol{\pi}'), \\ & u^* \geqslant u_\pi, \quad u^* - u_\pi \leqslant U b_\pi, \quad \sigma^*(\pi) \leqslant 1 - b_\pi, \\ & \sigma^*(\pi) \geqslant 0, \quad \sum_\pi \sigma^*(\pi) = 1, \quad b_\pi \in \{0, 1\}, \end{aligned} \tag{6.1}$$

with real variables $\sigma^*(\pi)$, u_π, binary variables b_π for each strategy π, and an equilibrium payoff real variable u^*. U is the maximum difference across payoffs. The variables b_π indicate whether strategy π is outside the equilibrium support. That is, $b_\pi = 1$ iff $\sigma^*(\pi) = 0$. Otherwise $u_\pi = u^*$: strategies in the support have the same payoff.

In practice, it might be expensive to directly optimize the objective, due to the nonlinearity of the entropy function. We instead optimize a piecewise linear approximation of the objective. We have the following result (details in App. 6.8.1).

Theorem 6.4.1. *Given $\epsilon > 0$, an ϵ-maximum-entropy symmetric Nash can be solved by a mixed-integer linear program based on* (6.1) *with an additional $O(|\hat{\Pi}|^2/\epsilon)$ linear constraints.*

Although approximate or even exact max-entropy NE is not generally unique, it narrows the possibilities considerably compared to unconstrained (approximate or exact) NE.

6.5 Search as a Meta-Strategy Operator

Many MATAs produce a policy network $\boldsymbol{p}$ that maps directly from an infostate to a distribution over actions in a forward pass for every player. Recent work has found that leveraging computation at run-time and adding search to $\boldsymbol{p}$ can improve performance in large EFG domains [Silver et al., 2018, Brown et al., 2020, Schmid et al., 2023]. As a case study for our meta-game evaluation framework, we apply it to investigate the effect of search as a general policy improver.

Toward that end, we propose a heuristic search method for large EFGs based on information-set MCTS (IS-MCTS) [Cowling et al., 2012] and Gumbel AlphaZero [Danihelka et al., 2022]. Details of this procedure, ***Gumbel IS-MCTS***, are specified in Alg. 12. Parameterized by a policy net $\boldsymbol{p}$ and a value net $\boldsymbol{v}$, the procedure conducts multiple passes over the game-tree guided by $\boldsymbol{v}$ and $\boldsymbol{p}$ at an input infostate s, and outputs an action a for decision-making. We can apply this procedure to a variety of underlying MATAs, as a ***meta-strategy operator***: transforming $\mathcal{M}$ to $\mathcal{M}'$ (with additional hyperparameters like simulation budget). The meta-strategy $\mathcal{M}'$ in effect adds run-time search to the output policy $\boldsymbol{p}$ of $\mathcal{M}$. Notice that unlike AlphaZero—which uses the same

Algorithm 12 Gumbel IS-MCTS

```
function Gumbel-Search($s$, $\boldsymbol{v}, \boldsymbol{p}$)
    $\forall(s,a).\ R(s,a) \leftarrow 0, C(s,a) \leftarrow 0$
    $\forall a \in \mathcal{A}(s)$. sample $g(a) \sim \mathit{Gumbel}(0), \hat{\mathcal{A}} \leftarrow \mathcal{A}(s)$
    repeat
        Sample a world state: $h \sim \Pr(h \mid s, \boldsymbol{p})$
        while do
            if $h$ is terminal then
                $\boldsymbol{r} \leftarrow$ payoffs of players Break
            else if $i \triangleq \tau(h)$ is chance then
                $a \leftarrow$ sample according to chance
            else if $s_i(h)$ not in search tree then
                Add $s_i(h)$ to search tree
                $\boldsymbol{r} \leftarrow \boldsymbol{v}(s_i(h))$. Break
            else if $s_i(h)$ is root node $s$ then
                $a, \hat{\mathcal{A}} \leftarrow$ one step of sequential halving (Alg. 14) based on $GS(s,a)$ and remaining
                actions in $\hat{\mathcal{A}}$
            else
                Select $a$ according to Eq.(6.3) in App. 6.8.2.2.
            end if
            $h \leftarrow ha$
        end while
        for $(s_i, a)$ in this trajectory do
            Increment $R(s_i, a)$ by $\boldsymbol{r}_i$, $C(s_i, a)$ by 1.
        end for
    until $\mathit{num\_sim}$ simulations done
    return Action $a$ that remains in $\hat{\mathcal{A}}$
end function
```

MCTS method for training and run-time, with meta-strategy operators we can explore a variety of MATAs as training-time methods which produce $\boldsymbol{v}$ and $\boldsymbol{p}$ for search at test-time [Sokota et al., 2023] (details in §6.6.2.4).

We next explain the key design of Gumbel IS-MCTS and how it differs from previous search methods. We defer further technical details in App. 6.8.2. Just like MCTS, IS-MCTS incrementally builds and traverses a search tree and aggregates statistics such as visit counts $C(s,a)$ and aggregated values $R(s,a)$ for visited (s,a) pairs. During each simulation of the search (line 5), a world state is sampled from a posterior belief $\Pr(h \mid s, \boldsymbol{p})$, which is computed via Bayes's rule, assuming the opponent were acting according to the policy net $\boldsymbol{p}$ prior to s.

The key feature of Gumbel IS-MCTS is how it selects actions at the search nodes. At the beginning of the search (line 3), a Gumbel random variable, $g(a)$, is sampled i.i.d. for each legal action a of the root, for later use in action selection. At the root (line 15), the algorithm treats

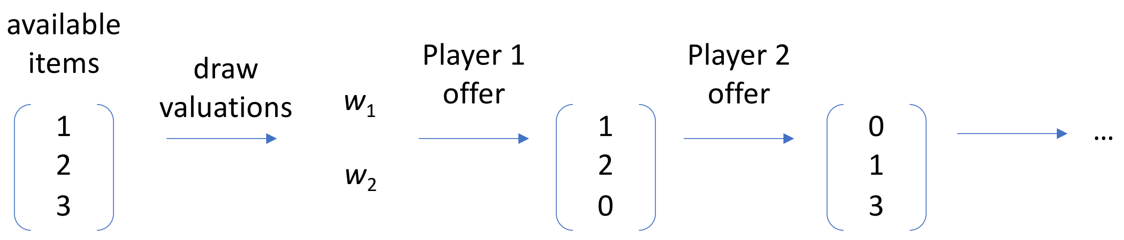


Figure 6.1: Example start of sequential bargaining game instance.

each legal action as an arm of a stochastic bandit, and uses a sequential-halving algorithm [Pepels et al., 2014] (Alg. 14) to distribute the simulation budget. Sequential-halving algorithms usually are designed for minimizing the *simple regret* [Bubeck et al., 2009], which is the regret at the last-iteration action recommendation. By contrast, UCB-styled algorithms are usually designed for minimizing the *accumulated regret* during an online learning process. For a game-playing search algorithm, minimizing simple regret makes more sense in terms of producing a single optimal action at a decision point.

We assign to each arm a a ***Gumbel score*** $GS(s,a) = g(a) + \text{logit}\, \boldsymbol{p}(s,a) + G(\hat{q}(s,a))$. The second term is the logit of a produced by $\boldsymbol{p}$, and the third term is a monotone transformation of the action value $\hat{q}(s,a)$, which is estimated by $R(s,a)$, $C(s,a)$, and $\boldsymbol{v}$. The intuition is that a high $\hat{q}(s,a)$ value indicates a direction for policy improvement. Indeed, the improved policy $Imp(\boldsymbol{p})(s,a) \triangleq \text{SoftMax}(\text{logit}\, \boldsymbol{p}(s,a) + G(\hat{q}(s,a)))$ provably achieves higher expected values Danihelka et al. [2022, App. B]. The forms of G and $\hat{q}$ is detailed in App. 6.8.2.1.

Adding Gumbel noise $g(a)$ implements the "Gumbel top-K-trick": deterministically selecting the top K actions according to $GS(s,a)$ is equivalent to sampling K actions from $Imp(\boldsymbol{p})(s,a)$ without replacement [Huijben et al., 2022]. The Gumbel score induces a low-variance non-deterministic action selection of the root node during the sequential halving process, which encourages exploration while distributing the simulation budget toward actions likely to yield higher expected values.

At a non-root node (line 17), an action is selected to minimize the discrepancy between $Imp(\boldsymbol{p})$ and the produced visited frequency (details in App. 6.8.2.2). At the end of the search, Gumbel IS-MCTS outputs the action a that survives the sequential halving procedure.

6.6 Evaluation Study

6.6.1 Domain: Alternating Negotiation

We test our MATAs and evaluation framework on the same class of negotiation games "Deal-or-No-Deal" (DoND) [Lewis et al., 2017] we tested in Section 5.5.3. Here we consider a general game parametrization: for the game instance $\texttt{Barg}(T, \varepsilon, \gamma)$, the game ends if either (1) a deal is made, (2) a maximum number of rounds T is reached, or (3) chance decides to terminate the game, which happens at every round with probability ε. If an agreement $(\boldsymbol{o}_1, \boldsymbol{o}_2)$ was reached at the t^{th} round, player i receives payoff $\gamma^t \boldsymbol{w}_i \cdot \boldsymbol{o}_i$. Otherwise both players receive zero payoff.

DoND is a family of challenging general-sum environments with imperfect information. Players have a common interest in reaching a deal (and quickly, for $\gamma < 1$), and the different ways of dividing the pool have different total value. In our studies, we sample configurations uniformly from a database of 6796 published by Lewis et al. [2017]. This leads to prohibitive complexity: a game with $T = 10$ has $O(10^{11})$ infosets for every player, which is intractable for enumeration (detailed in App. 5.7.3.2), and grows exponentially with T.

6.6.2 Benchmark Algorithms

Our meta-game evaluation considers the following $M = 17$ MATAs. These algorithms represent a comprehensive set of state-of-the-art MARL algorithms, including methods using self-play based training, population based training, and model-free and model-based approaches. Details are below and in App. 6.8.4.

6.6.2.1 Independent/Multiagent PPO (IDPPO/MAPPO)

Both IDPPO and MAPPO train policy and value nets using self-play trajectories by minimizing the trust-region clipped loss [Schulman et al., 2017] using the generalized advantage estimator (GAE) [Mnih et al., 2016]. Value nets are trained by minimizing L2 loss from the targets produced by GAEs. In IDPPO each player maintains its own policy and value nets whereas in MAPPO [Yu et al., 2022] players share the same neural nets.

6.6.2.2 Regularized Nash Dynamics (R-NaD) and NFSP

Both R-NaD [Perolat et al., 2021] and Neural Fictitious Self-Play (NFSP) [Heinrich and Silver, 2016] are self-play model-free MARL algorithms originally designed for purely adversarial settings. R-NaD has recently shown success in producing human-level agents in Stratego [Perolat et al., 2022], where it iteratively trains policy nets by minimizing NeuRD loss [Hennes et al., 2020] and value nets using the V-Trace estimator [Espeholt et al., 2018] on a sequence of regularized games. NFSP mimics the classic fictitious play algorithm [Heinrich and Silver, 2016] by alternating between training a supervised learning net that summarizes historical plays and training a DQN policy that serves as a best response.

6.6.2.3 Policy Space Response Oracles (PSRO) and FCP

PSRO [Lanctot et al., 2017] and Fictitious Co-Play (FCP) [Strouse et al., 2021] are two population-based MARL algorithms. PSRO is an EGTA method that iteratively adds policies that are best responses to the current solution. We use max-entropy Nash as the solution concept (solving an asymmetric version of (6.1)), and PPO as the best-response method. We consider two ways of extracting the final agents: (i) PSRO: using the final-iteration solution and (ii) PSRO-LAST: using the final-iteration best-response policies. FCP has notably demonstrated success in collaborating with humans from scratch [Strouse et al., 2021]. FCP builds a population by picking policies of different skill levels on multiple self-play runs, and trains final best responses against such populations. Our implementation of FCP uses IDPPO to generate self-play runs, picks the agents based on social welfare, and uses PPO as the best-response method.

6.6.2.4 Gumbel Search and Vanilla AlphaZero-style Search

We include the following variants of Gumbel IS-MCTS: (i) G-Search: Using Gumbel IS-MCTS for both training the networks (Alg. 15) at training time and conducting search at test time. (ii) G-Search-PN: Using search at training time, but only policy net without search at test time. (iii) G-Search-IDPPO: Using IDPPO to train the policy and value nets, based on which Gumbel IS-MCTS executes test-time search. (iv) G-Search-MAPPO and (v) G-Search-R-NaD are similarly defined. In addition, we implement an extension of the original AlphaZero-style MCTS to IS-MCTS. The differences between this search method (Alg. 13 in App. 6.8.3) and Gumbel IS-MCTS are in the mechanisms for exploration (*e.g.*, Dirichlet noise) and action selection (*e.g.*, PUCT) within the search tree. We include (vi) VA-Search: Using this search for both training time and test time, and (vii) VA-Search-PN: similarly defined as (ii).

6.6.2.5 Heuristic Strategies

We further include the following baseline bargaining strategies: (i) **Uniform**, which uniformly samples among all legal actions at each decision point. (ii) **Tough**, which never agrees, and always proposes uniformly among offers that maximize its own payoff. (iii) **Soft**, which always agrees, or uniformly proposes among all offers if it is the first-mover.

MATA Name	NE-Regret-Score	Uniform-Score	NE-NBS	MATA Name	NE-Regret-Score	Uniform-Score	NE-NBS
G-Search-R-NaD	**0.002±0.020**	6.158±0.040	44.010±0.794	MAPPO	0.721±0.059	5.602±0.035	32.832±0.818
G-Search	0.010±0.046	**6.215±0.015**	**44.345±0.496**	IDPPO	0.766±0.133	5.645±0.050	30.240±2.051
R-NaD	0.045±0.043	5.977±0.027	42.008±0.738	NFSP	0.806±0.050	5.766±0.014	36.534±0.721
VA-Search	0.092±0.023	6.074±0.029	40.836±0.551	VA-Search-PN	0.843±0.071	5.570±0.026	36.540±0.717
G-Search-MAPPO	0.354±0.047	6.038±0.027	37.808±0.962	FCP	1.116±0.096	5.804±0.013	38.665±1.092
PSRO-LAST	0.414±0.051	5.890±0.026	39.187±0.762	Tough	1.801±0.093	4.017±0.031	6.581±0.137
PSRO	0.417±0.055	5.904±0.022	38.288±0.873	Soft	3.189±0.098	3.773±0.013	27.023±0.855
G-Search-IDPPO	0.480±0.062	6.088±0.031	36.385±1.317	Uniform	4.274±0.062	2.391±0.004	11.980±0.494
G-Search-PN	0.575±0.048	5.811±0.013	42.174±0.493				

Table 6.1: Results for $\texttt{Barg}(10, 0, 1)$, with 95% confidence intervals. MATAs are listed in increasing order of NE-Regret.

MATA Name	NE-Regret-Score	Uniform-Score	NE-NBS	MATA Name	NE-Regret-Score	Uniform-Score	NE-NBS
G-Search-R-NaD	**0.000±0.001**	**6.147±0.013**	43.681±0.497	PSRO	0.488±0.056	5.688±0.052	39.851±0.728
G-Search	0.005±0.033	6.078±0.019	43.198±0.545	G-Search-MAPPO	0.729±0.095	5.656±0.056	32.983±0.994
R-NaD	0.045±0.010	6.121±0.012	**44.072±0.417**	IDPPO	0.749±0.059	5.504±0.042	36.350±0.685
VA-Search	0.075±0.023	6.058±0.014	42.605±0.610	PSRO-LAST	0.906±0.213	5.391±0.161	35.723±2.119
G-Search-PN	0.294±0.019	5.833±0.016	42.285±0.448	MAPPO	1.135±0.144	5.306±0.086	31.878±1.378
FCP	0.360±0.026	5.864±0.013	43.321±0.265	Soft	1.918±0.018	4.352±0.011	35.990±0.267
VA-Search-PN	0.412±0.033	5.668±0.014	40.314±0.325	Uniform	4.161±0.031	2.370±0.007	12.562±0.211
NFSP	0.421±0.011	5.786±0.014	39.151±0.481	Tough	4.542±0.082	2.685±0.025	2.270±0.097
G-Search-IDPPO	0.450±0.038	5.820±0.019	36.997±0.432				

Table 6.2: Results for $\texttt{Barg}(30, 0.125, 0.935)$, with 95% confidence intervals. MATAs are listed in increasing order of NE-Regret.

6.6.3 Experimental Setup

All methods are implemented within the programming model of OpenSpiel [Lanctot et al., 2019]. Each MATA that involves neural training uses models of approximately equal size (*e.g.*, number of layers, hidden nodes). The neural net inputs are likewise the same across all MATAs, and include complete history information (*i.e.*, losslessly represent infosets) for the DoND game. For procedures that involve self-play or RL training we employ roughly the same number of training trajectories. For all search methods we use 200 simulations per call. We fine-tuned the learning rates based on NashConv performance. Most of the MATAs reliably produce approximate equilibria. Details are in App. 6.8.5.

We set $|\Omega^m| = 10$, $m \in \{1, \ldots, M\}$. To simplify the evaluation procedure, we pre-compute an empirical payoff matrix covering every policy pair: $(\hat{\pi}_1^m(\omega), \hat{\pi}_2^{m'}(\omega'))$, for $m, m' \in \{1, \ldots, M\}$, ω and ω' among the seeds sampled for $\mathcal{M}^m$ and $\mathcal{M}^{m'}$, respectively. For each payoff entry we run 2×10^4 simulations to compute the expected payoff. We then sample and analyze 10^6 $M \times M$ empirical meta-games from this matrix, per Steps 2–4 of §6.4.2, to obtain distributions over the statistics of interest.

6.6.4 Results

We test all 17 MATAs on two DoND instances: `Barg`(10, 0, 1) and `Barg`(30, 0.125, 0.935). The two settings are qualitatively different. The latter includes discounting and per-round ending probability, incentivizing the players to find and agree on a good deal in early rounds. Without these factors, the first game reduces to an ultimatum-game-like environment. Meta-game statistics are reported in Tables 6.1 and 6.2.

In addition to NE-Regret and uniform scores, we also report a statistic we term the ***NE-Nash-Bargaining-Score*** (NE-NBS). The NE-NBS for MATA m in an empirical game is defined as the utility-product between π^m and an opponent strategy σ^*: $u(\pi^m, \boldsymbol{\sigma}^*) \cdot u(\sigma^*, \boldsymbol{\pi^m})$. Here σ^* is a max-entropy Nash computed by (6.1). This statistic is intended to measure the effectiveness of an agent in achieving high social welfare and fairness with a rational opponent.

From the tables we can see that while there is positive correlation between NE-Regret Scores and Uniform Scores, NE-Regret Scores are better at identifying the most robust MATAs. The rankings of the top four MATAs in NE-regret scores are the same for both environments, and from the empirical distribution plots in App. 6.8.6 we can see G-Search-R-NaD and G-Search consistently produce strategies that are selected in a max-entropy equilibrium support with high probability. By contrast uniform-score comparisons can be distorted by non-salient success against weak strategies. It is also interesting to note positive correlation between NE-Regret and NE-NBS. This reflects the non-zero-sum nature of these environments.

The performance of search-based agents is especially noteworthy. Three of the top four MATAs employ search at test time. All search methods are stronger than their policy net counterparts, confirming that IS-MCTS is a policy-improver in imperfect information games. We hypothesize that this is because a search-based algorithm is usually a better responder to its policy network counterpart, while both behave strategically similarly against other methods. The relative strength among search-based algorithms mirrors that of their policy-net counterparts. In both environments, G-Search and VA-Search are among the top four. This suggests that using search at training time is compatible with the same search at test time, and produces better policy and value nets compared with other methods.

We notice some algorithms behave qualitatively differently in these two environments. We found FCP consistently produces collaborative strategies with rather high agreement probability. This may explain its better performance in the second environment, which encourages settling a deal in early rounds. Likewise, Soft achieves higher individual scores and NE-NBS in the second environment. By contrast, while Tough achieves better individual performance than Soft in the ultimatum game, it is still worse than all learning methods, as shown by its terrible NE-NBS values for both environments.

Another interesting view is provided by ***best-response graphs***. In a best-response graph for a game instance, strategies are vertices, and there is a directed edge $(m_1 \rightarrow m_2)$ iff $m_2 = \arg\max_{m'} u(\pi^{m'}, \boldsymbol{\pi}^{\boldsymbol{m_1}})$. We generate an aggregate graph for a MATA meta-game by recording the frequency of such edges in the sampled empirical games. These are shown for the two game settings in Fig. 6.2.

The empirical best-response graphs support several observations. First, many MATA vertices such as IDPPO and MAPPO have self-edges with non-negligible frequencies. This is consistent with previous observations that self-play MARL algorithms are likely to produce agents that overfit to themselves [Hu et al., 2020]. However, self-edges do not necessarily suggest poor generalization performances. For example, G-Search-NaD has a self-edge with a high probability in both environments, but still performs the best. Self-edges persist even after applying search as policy-improver at test time. For example, there are self-edges for G-Search-IDPPO and G-Search-MAPPO. This may suggest that there are certain strategic correlations between a search agent and its policy net counterpart. Interestingly, we also observe certain self-edge probabilities for PSRO and PSRO-LAST, which are normally regarded as population-based training methods. It could be that the equilibria selected at each iteration introduce correlations between different players. By contrast, NFSP and FCP are two MATAs without strong self-edges; both use a uniform distribution for opponent modeling, which may reduce the correlation between player positions.

Edges across MATAs illuminate strategic interaction structure. Tough is unsurprisingly the best response to Soft. G-Search-NaD is a strong attractor in both graphs, as G-Search is in the first. Sometimes a search-based MATA is a best response to its policy-net counterpart, as illustrated by R-NaD to G-Search-R-NaD, and VA-Search-PN to VA-Search. This could also explain why MATAs such as G-Search-PN and VA-Search-PN do not have apparent self-edges.

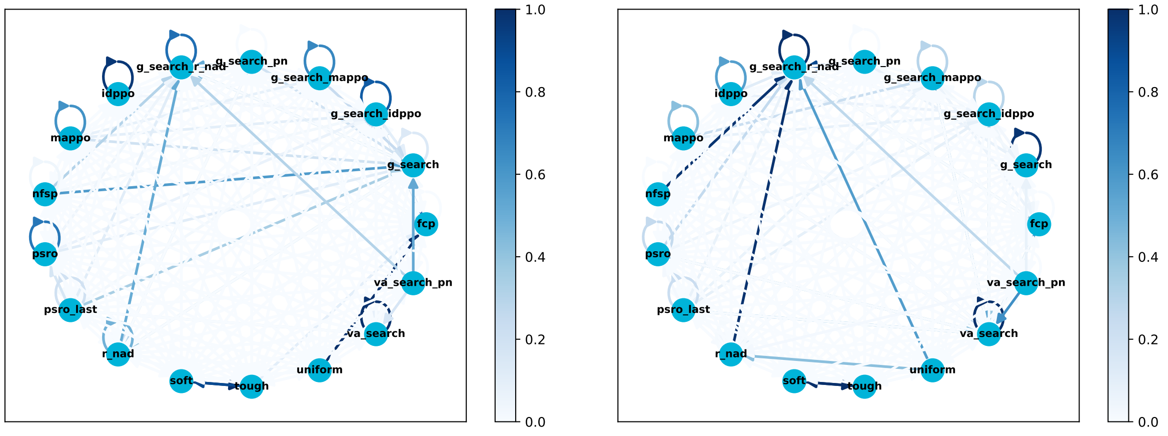


Figure 6.2: Empirical best-response graphs for Barg(10, 0, 1) and Barg(30, 0.125, 0.935).

6.7 Conclusion

We propose a meta-game evaluation framework for MARL in general-sum environments. Our approach is analogous to the evaluation process for single-agent RL, effectively aggregating the strategic analysis procedures across possible worlds defined by different seed combinations. We illustrated the method by constructing a meta-game model over a comprehensive set of multiagent training algorithms using deep RL on a class of negotiation games. The meta-game analysis evaluated the algorithms in multiple ways, most prominently through max entropy NE-regret ranking and the structure of best-response graphs. Bootstrap statistics provide a basis for assessing uncertainty in evaluation results.

Experimental results support several interesting observations about Gumbel IS-MCTS as a meta-strategy operator, especially regarding the value of search at both training time and test time, and on patterns of strategic interactions among the algorithms.

A key feature of our evaluation framework is the flexibility to investigate a variety of statistics of interest through a carefully structured process. Future work will focus on understanding the robustness of alternative measures that can be employed for algorithm assessment through meta-games, and developing effective tools for statistical analysis.

6.8 Appendix

6.8.1 Max-Entropy Nash

6.8.1.1 Proof of Theorem 6.4.1

Proof. The idea is to transform (6.1) into the following mixed-integer linear program:

$$\begin{aligned}
\min_{\sigma^*} \quad & \sum_{\pi \in \hat{\Pi}} \gamma_\pi \\
\text{s.t.} \quad & \forall \pi \in \hat{\Pi}, \\
& \forall k \in [K], \gamma_\pi \geqslant l_k(\sigma^*(\pi)) \\
& u_\pi = \sum_{\pi' \in \hat{\Pi}} \sigma^*(\pi) u(\pi, \boldsymbol{\pi'}) \\
& u^* \geqslant u_\pi \\
& u^* - u_\pi \leqslant U b_\pi \\
& \sigma(\pi) \leqslant 1 - b_\pi \\
& \sigma^*(\pi) \geqslant 0 \\
& \sum_\pi \sigma^*(\pi) = 1 \\
& b_\pi \in \{0, 1\}
\end{aligned} \tag{6.2}$$

Where $l_k(x) = f(\frac{k}{K}) + \frac{f(\frac{k+1}{K}) - f(\frac{k}{K})}{\frac{1}{K}}(x - f(\frac{k}{K}))$ is the k-th piecewise linear segment of the function $f(x) = x \log x$, for a total of K segments. Denote the convex envelope of all these K segments as $l(x)$. The additional constraints are $\gamma_\pi \geqslant l_k(\sigma^*(\pi))$. When we solve (6.2) exactly, one of these K inequalities will be satisfied at equality.

Let $f(x^*)$ be the minimum of f and $l(x')$ be the minimum of the piecewise approximation. Then if $|f - l|_\infty < \epsilon$, we can see that $f(x^*) \geqslant l(x^*) - \epsilon \geqslant l(x') - \epsilon$. Then x' is an ϵ-optimal solution of f.

Then to achieve an ϵ-optimal solution of (6.1) we need to choose some K, such that $|f - l|_\infty < \frac{\epsilon}{|\hat{\Pi}|}$.

For the segment $I_k = [\frac{k}{m}, \frac{k+1}{m}]$, by elementary calculus we can find $g(k) = \max_{x \in I_k} |x \log x - l_k(x)| = \frac{k+1}{eK}(1 + \frac{1}{k})^k - \frac{k(k+1)}{K} \log(1 + \frac{1}{k})$. This happens when $x = \frac{k+1}{eK}\left(1 + \frac{1}{k}\right)^k$.

We will now prove

$$g'(k) = \frac{e + \left(\left(1+\frac{1}{k}\right)^k (1+k) - e(1+2k)\right) \log\left(1+\frac{1}{k}\right)}{eK} < 0$$

To show this, by arranging the numerator, we need to show that

$$\left(k + (1+k)\left(1 - \frac{1}{e}\left(1+\frac{1}{k}\right)^k\right)\right) \log\left(1+\frac{1}{k}\right) > 1$$

First notice that

$$\begin{aligned}
1 - \frac{1}{e}\left(1+\frac{1}{k}\right)^k &= 1 - e^{k\log(1+\frac{1}{k})-1} \\
&= 1 - e^{-\frac{1}{2k}+\frac{1}{3k^2}-\frac{1}{4k^3}+\frac{1}{5k^4}+\ldots} \\
&\geqslant 1 - e^{-\frac{1}{2k}+\frac{1}{3k^2}} \\
&\geqslant (-\frac{1}{2k}+\frac{1}{3k^2}) - \frac{1}{2}\left(-\frac{1}{2k}+\frac{1}{3k^2}\right)^2 \\
&= \frac{1}{2k} - \frac{11}{24k^2} + \frac{1}{6k^3} - \frac{1}{18k^4}
\end{aligned}$$

Then when $k \geqslant 7$

$$\begin{aligned}
&(1+k)\left(1 - \frac{1}{e}\left(1+\frac{1}{k}\right)^k\right) \\
\geqslant &(1+k)\left(\frac{1}{2k} - \frac{11}{24k^2} + \frac{1}{6k^3} - \frac{1}{18k^4}\right) \\
= &\frac{1}{2} + \frac{1}{24k} - \frac{7}{24k^2} + \frac{1}{9k^3} - \frac{1}{18k^4} > \frac{1}{2}
\end{aligned}$$

Therefore

$$\left(k + (1+k)\left(1 - \frac{1}{e}\left(1+\frac{1}{k}\right)^k\right)\right) \log\left(1+\frac{1}{k}\right) > \left(k+\frac{1}{2}\right)\log\left(1+\frac{1}{k}\right)$$

Let $h(k) = \left(k+\frac{1}{2}\right)\log\left(1+\frac{1}{k}\right)$. Then $h'(k) = \log\left(1+\frac{1}{k}\right) - \frac{1}{2k} - \frac{1}{2(k+1)}$ $h''(k) = -\frac{1}{k(k+1)} + \frac{1}{2k^2} + \frac{1}{2(k+1)^2} > 0$. Therefore $h'(k)$ increases. And since $\lim_{k\to\infty} h'(k) = 0$ we can deduce $h'(k) < 0$. Therefore $h(k)$ is decreasing. And since $\lim_{k\to\infty} h(k) = 1$ we can see that $h(k) > 1$. Therefore the we have proved $g'(k) < 0$ for $k \geqslant 7$ is decreasing. It is easy to verify it also holds for $k < 7$. So $g(k)$ achieves maximum $\frac{1}{eK}$ when $k = 0$.

Then we have when $K = \left\lfloor \frac{|\hat{\Pi}|}{e\epsilon} \right\rfloor + 1$, we can obtain an ϵ-optimal maximum entropy Nash by solving (6.2) with additional $K \cdot (|\Pi|) = O\left(\frac{|\hat{\Pi}|^2}{\epsilon}\right)$ linear constraints.

□

6.8.1.2 Setup

We use GUROBI [Gurobi Optimization, LLC, 2023] as the solver. In our experiments we always solve for a 0.05-optimal max-entropy Nash.

6.8.2 Details of Gumbel IS-MCTS

6.8.2.1 Value Estimation

When $C(s,a) > 0$, we simply let $\hat{q}(s,a) = \frac{R(s,a)}{C(s,a)}$. When $C(s,a) = 0$, following [Danihelka et al., 2022], we let

$$\hat{q}(s,a) = \frac{1}{1 + \sum_b C(s,b)} \left(\boldsymbol{v}(s,a) + \frac{\sum_b C(s,b)}{\sum_{b:C(s,b)>0} \boldsymbol{p}(s,b)} \sum_{b:C(s,b)>0} \frac{R(s,b)}{C(s,b)} \boldsymbol{p}(s,b) \right)$$

as an estimator. And we use $G(\hat{q}(s,a)) = c_2(c_1 + \max_b C(s,b))\hat{q}(s,a)$, for some $c_1, c_2 > 0$.

6.8.2.2 Action Selection at Non-Root Nodes

At a non-root node, an action is selected to minimize the discrepancy between $Imp(\boldsymbol{p})$ and the produced visited frequency:

$$\arg\min_a \sum_b \left(Imp(\boldsymbol{p})(s_i, b) - \frac{C(s_i,b) + 1\{a = b\}}{1 + \sum_c C(s_i,c)} \right)^2 \tag{6.3}$$

6.8.3 Algorithms Pseudocode

6.8.3.1 Vanilla AlphaZero Search

Pseudocode presented as Alg. 13.

6.8.3.2 Sequential Halving

Pseudocode presented as Alg. 14.

Algorithm 13 Vanilla AlphaZero-styled IS-MCTS

```
1:  function VA-Search($s, \boldsymbol{v}, \boldsymbol{p}$)
2:      $\forall (s,a), R(s,a) = 0, C(s,a) = 0.$
3:      $\forall a \in \mathcal{A}(s)$, sample $d(a) \sim Dirichlet(\alpha)$
4:      for $iter = 1, \ldots, num\_sim$ do
5:          Sample a world state: $h \sim \Pr(h \mid s, \boldsymbol{p})$
6:          while do
7:              if $h$ is terminal then
8:                  $\boldsymbol{r} \leftarrow$ payoffs of players. Break
9:              else if $i \triangleq \tau(h)$ is chance then
10:                 $a \leftarrow$ sample according to chance
11:             else if $s_i(h)$ not in search tree then
12:                 Add $s_i(h)$ to search tree.
13:                 $\boldsymbol{r} \leftarrow \boldsymbol{v}(s_i(h))$. Break
14:             else if $s_i(h)$ is root node $s$ then
15:                 $a \leftarrow \arg\max_a \frac{R(s_i,a)}{C(s_i,a)} + c_{puct}((1-\epsilon)\boldsymbol{p}(s,a) + \epsilon d(a))\frac{\sqrt{C(s_i,a)}}{1+\sum_b C(s_i,b)}$ .
16:             else
17:                 $a \leftarrow \arg\max_a \frac{R(s_i,a)}{C(s_i,a)} + c_{puct}\boldsymbol{p}(s,a)\frac{\sqrt{C(s_i,a)}}{1+\sum_b C(s_i,b)}$
18:             end if
19:             $h \leftarrow ha$
20:         end while
21:         for $(s_i, a)$ in this trajectory do
22:             $R(s_i,a) \mathrel{+}= \boldsymbol{r}_i, C(s_i,a) \mathrel{+}= 1$
23:         end for
24:     end for
25:     return Action $a = \arg\max C(s,a)$, (During training) a policy target that is the normalized
        visited count of $s$.
26: end function
```

6.8.3.3 Self-play based Training

Pseudocode presented as Alg. 15.

6.8.4 Hyperparameters

6.8.4.1 Input Representation

We use the infostate_tensor in OpenSpiel [Lanctot et al., 2019] as the representation of infostate. In Bargaining games, this includes information (1) which player the current agent is playing (2) pool configuration (3) the player's own private valuation (4) the current round number.

Algorithm 14 Sequential Halving

```
1: function Seq-Hal($\hat{\mathcal{A}}, K$)
2:     Maintain a static variable $epoch$ across different calls of this function, initialized as 0
3:     if $epoch == 0$ then
4:         $\hat{\mathcal{A}} \leftarrow K$ actions with largest $g(a) + \text{logit}\, \boldsymbol{p}(s, a)$
5:         $epoch \mathrel{+}= 1$
6:     end if
7:     $a \leftarrow$ an action that has not been visited $\lfloor \frac{sim\_num}{\frac{K}{2^{epoch-1}} \log_2 K} \rfloor$ times at current epoch in $\hat{\mathcal{A}}$
8:     if All actions in $\hat{\mathcal{A}}$ have been visited $\lfloor \frac{sim\_num}{\frac{K}{2^{epoch-1}} \log_2 K} \rfloor$ times in current epoch then
9:         $\hat{\mathcal{A}} \leftarrow$ Top $\lfloor \frac{K}{2^{epoch}} \rfloor$ actions in $\hat{\mathcal{A}}$ based on $g(a) + \text{logit}\, \boldsymbol{p}(s, a) + G(\hat{q}(s, a))$
10:        $epoch \leftarrow epoch + 1$
11:    end if
12:    return $a, \hat{\mathcal{A}}$.
13: end function
```

6.8.4.2 PPO Algorithms

We use the same PPO hyperparameters for every place when it is used. See Table 6.3. For

Hyperparameter Name	Values
Learning rate	$2e-4$
Optimizer	Adam
Batch size	16
Number of minibatch	4
Number of updates per epoch	10
Number of steps per PPO trajectory	64
Number of game trajectories in total	1e6
Entropy weight	0.01
Clipped parameter ϵ	0.2
GAE lambda	0.95
RL discount factor	1
Torso for policy nets	[256, 256]
Torso for value nets	[256, 256]

Table 6.3: Hyper-parameters for PPO.

single-agent PPO and IDPPO, the PPO algorithm trains policy nets by minimizing the clipped loss $\mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}, 1+\epsilon, 1-\epsilon \right) \hat{A}_t \right) \right]$, where A_t is estimated by generalized advantage estimation (GAE) [Schulman et al., 2016]. Value nets are updated by minimizing L2 loss from the value targets produced by GAE. For MAPPO, the loss are simply the summation of individual PPO losses of every player.

Algorithm 15 Self-Play Training for Search Methods

```
1:  function Self-Play-Train($\mathcal{G}$)
2:      Initialize $\boldsymbol{v}, \boldsymbol{v}', \boldsymbol{p}, \boldsymbol{p}'$
3:      $D_{\boldsymbol{v}} = \{\}, D_{\boldsymbol{p}} = \{\}$
4:      for $iter = 1, \ldots, num\_epoch$ do
5:          $h \leftarrow$ Initial state $h_0$ of $\mathcal{G}$
6:          while  do
7:              if $h$ is terminal then
8:                  $\boldsymbol{r} \leftarrow$ payoffs of players Break
9:              else if $i \triangleq \tau(h)$ is chance then
10:                 $a \leftarrow$ sample according to chance
11:             else
12:                 $a, \pi \leftarrow$ SEARCH$(s_i(h), \boldsymbol{v}', \boldsymbol{p}')$
13:                 $D_{\boldsymbol{p}} \leftarrow D_{\boldsymbol{p}} \cup \{(s_i(h), \pi)\}$, $\pi$ is the visiting frequency during search
14:             end if
15:             $h \leftarrow ha$
16:         end while
17:         for $s_i$ in this trajectory do
18:             $D_{\boldsymbol{v}} \leftarrow D_{\boldsymbol{v}} \cup \{s_i, \boldsymbol{r}_i\}$
19:         end for
20:         $\boldsymbol{v}, \boldsymbol{p} \leftarrow$ UPDATE$(D_{\boldsymbol{v}}, D_{\boldsymbol{p}})$
21:         Replace parameters of $\boldsymbol{v}', \boldsymbol{p}'$ with the latest parameters of $\boldsymbol{v}, \boldsymbol{p}$ periodically.
22:     end for
23:     return $\boldsymbol{v}, \boldsymbol{p}$
24: end function
```

6.8.4.3 R-NaD

We fine-tuned the learning rate to be 1e-3, the total number of game trajectories to be around 1e6. Others are the same as the default hyperparameters in OpenSpiel [Lanctot et al., 2019].

6.8.4.4 NFSP

See Table 6.4.

6.8.4.5 PSRO

See Table 6.5.

6.8.4.6 FCP

See Table 6.6

Hyperparameter Name	**Values**
RL network Learning rate	0.01
DQN buffer size	2^{17}
SL network learning rate	0.01
Optimizer	Adam
Batch size	256
Number of game trajectories in total	1e6
Torso for RL net	[256, 256]
Torso for SL net	[256, 256]

Table 6.4: Hyper-parameters for NFSP.

Hyperparameter Name	**Values**
Number of PSRO iterations	32
best-response method	PPO
Number of game trajectories to train each BR	1e6
Meta-strategy solver	Max-entropy Nash

Table 6.5: Hyper-parameters for PSRO.

6.8.4.7 Gumbel Search

See Table 6.7.

6.8.4.8 VA Search

See Table 6.8.

6.8.5 NASHCONV Results

All NASHCONVs are estimated using PPO as a best response method. The results are reported across three random seeds.

6.8.5.1 `Barg`$(10, 0, 1)$

See Figures 6.3.

6.8.5.2 `Barg`$(30, 0.125, 0.935)$

See Figures 6.4.

Hyperparameter Name	Values
Number of self-play runs	32
Number of checkpoint strategies considered per run	100
Strategies picked each run	The first one, the last one, and the one that just achieves half of the social welfare of the last one
Self-play method	IDPPO
best-response method	PPO

Table 6.6: Hyper-parameters for FCP.

Hyperparameter Name	Values
Learning rate	$2e-4$
Optimizer	SGD
Batch size	256
Max buffer size	2^{17}
c_1	50
c_2	0.1
sim_num	200
K	16
Network delayed period	1000
Number of game trajectories in total	1e6
Torso for PVN	[256, 256]

Table 6.7: Hyper-parameters for Gumbel Search.

6.8.6 Empirical Distribution of REGRET

6.8.6.1 `Barg`$(10, 0, 1)$

See Figures 6.5.

6.8.6.2 `Barg`$(30, 0.125, 0.935)$

See Figures 6.6

Hyperparameter Name	**Values**
Learning rate	$1e-3$
Optimizer	SGD
Batch size	256
Max buffer size	2^{17}
c_{puct}	20
sim_num	200
Dirichlet α	$\frac{1}{\lvert\mathcal{A}\rvert} \cdot \mathbf{1}$
ϵ	0.25
Network delayed period	1000
Number of game trajectories in total	1e6
Torso for PVN	[256, 256]

Table 6.8: Hyper-parameters for VA Search.

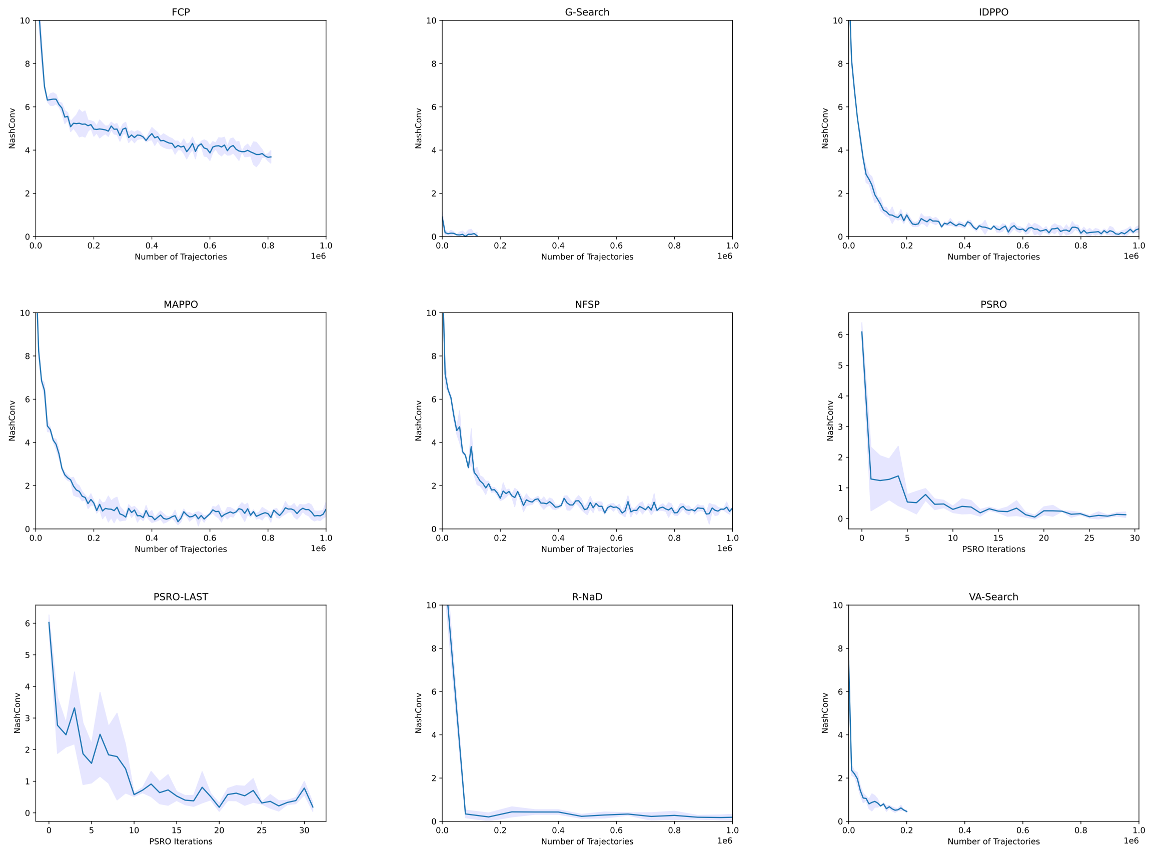


Figure 6.3: NASHCONV of $\texttt{Barg}(10, 0, 1)$

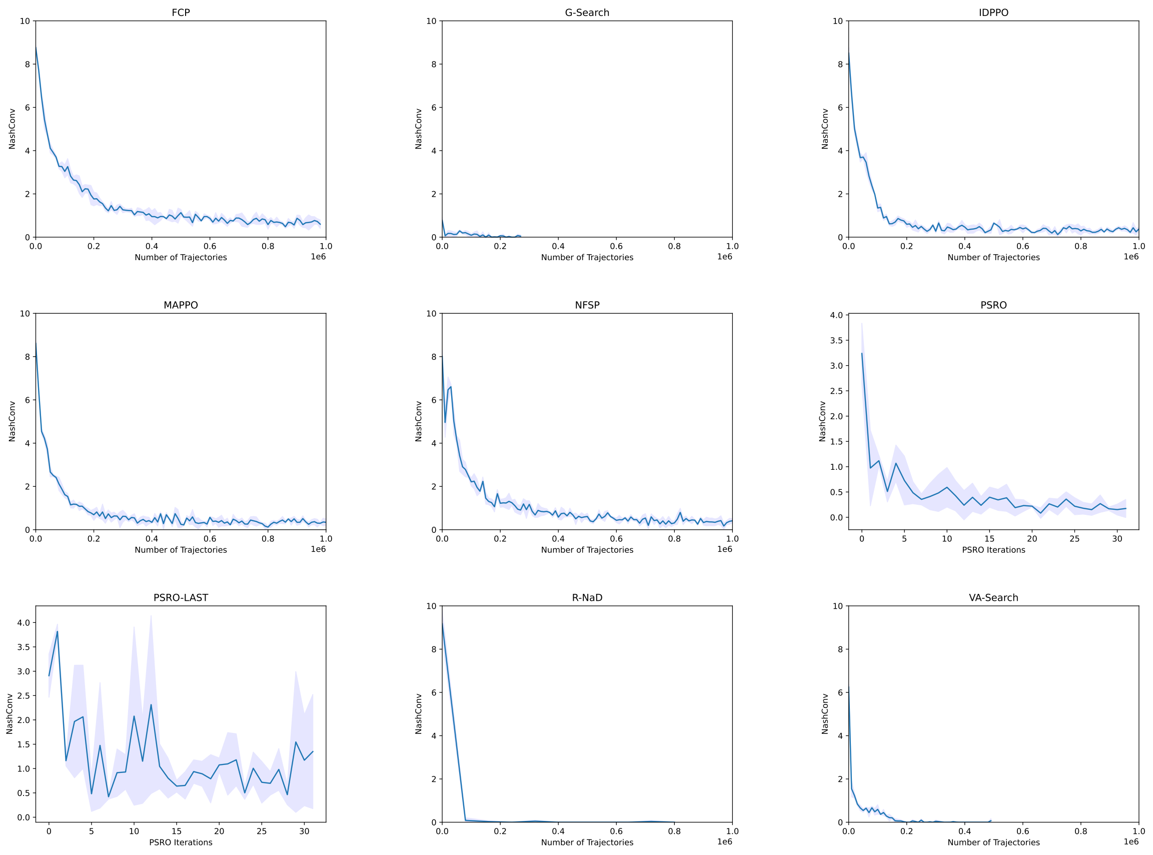


Figure 6.4: NASHCONV of $\texttt{Barg}(30, 0.125, 0.935)$

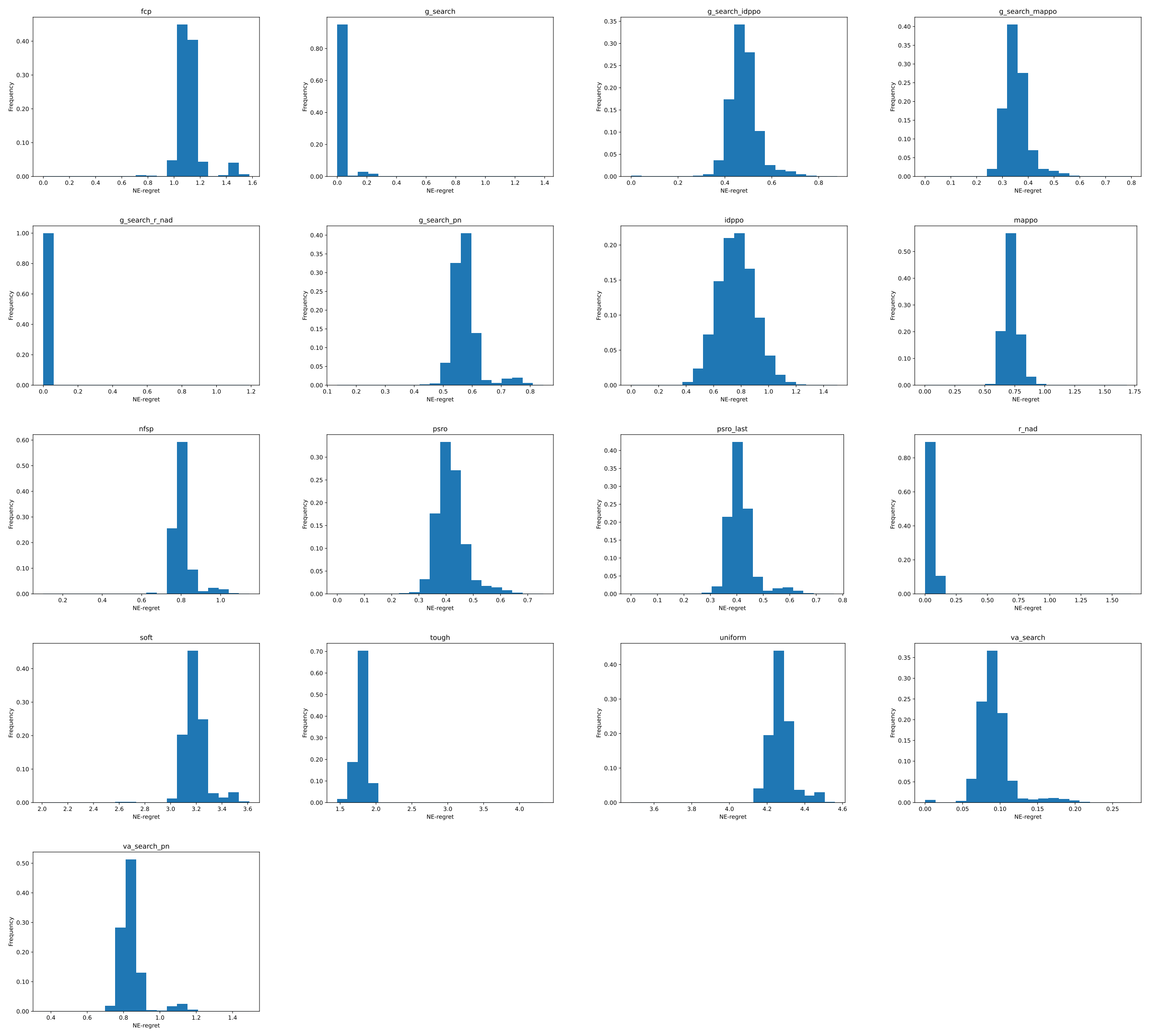


Figure 6.5: Empirical Distribution of NE-Regret of $\texttt{Barg}(10, 0, 1)$

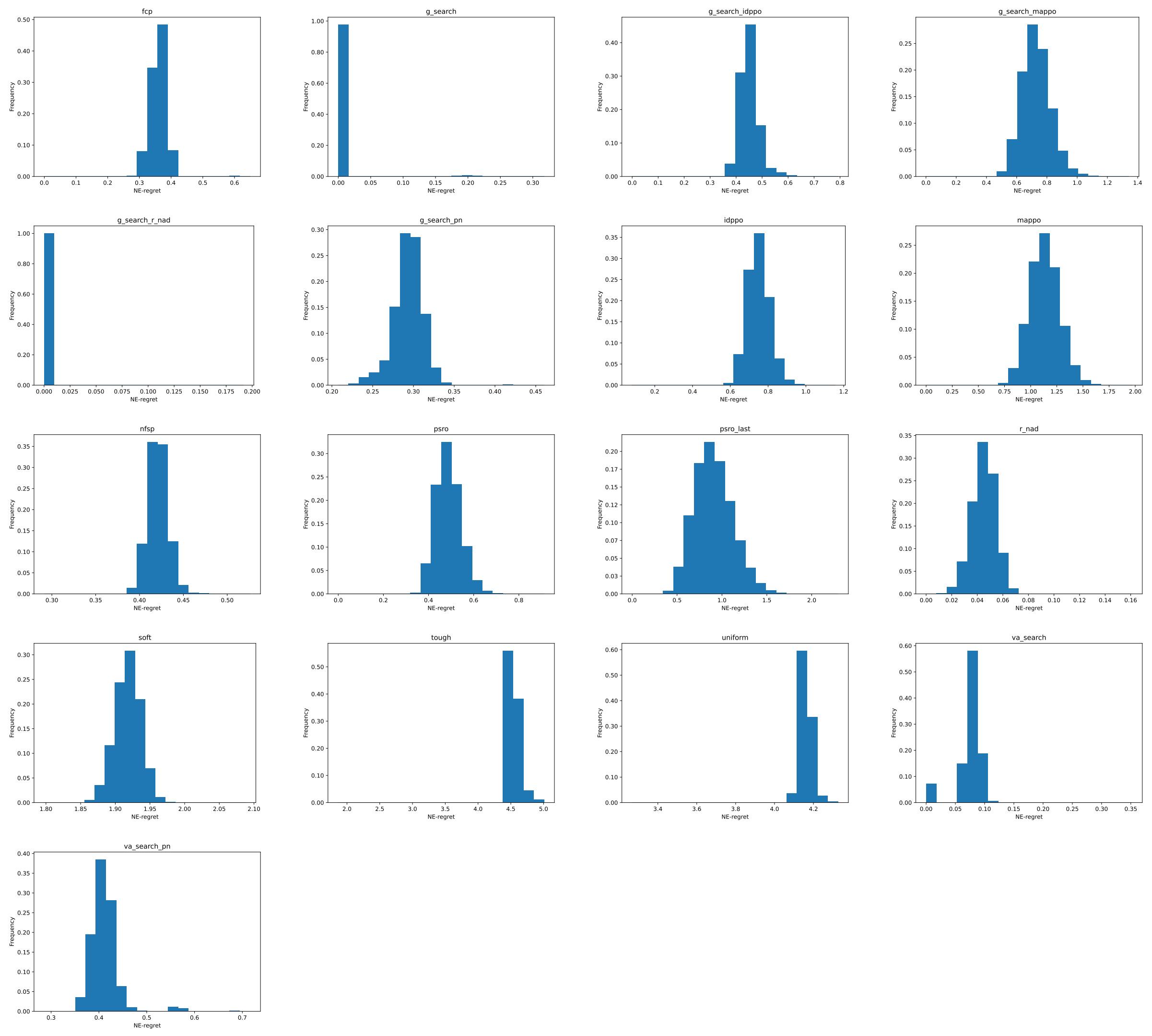


Figure 6.6: Empirical Distribution of NE-Regret of $\texttt{Barg}(30, 0.125, 0.935)$

CHAPTER 7

Conclusions and Future Works

Understanding intelligence has been one of the most fundamental questions and has been progressing along with the development of computation, psychology, statistics, and many other interdisciplinary domains. The emergence of human intelligence is already the most amazing phenomenon that has taken place in the universe where we are currently situated: everyone is born as an infant who can barely understand basic arithmetic, yet nowadays humans are able to manage nuclear energy and explore outer space. The power of artificial intelligence is even more unimaginable: leveraging computation and a huge corpus of empirical data, we may create artifacts that surpass the capability of human brains. For example, the AlphaGo agent made moves that could beat classical human strategic patterns accumulated through thousands of years of history. On one hand, the capability of human intelligence is largely bounded by biological limitations which are hard to break through only millions of years of evolution. On the other hand, the limitations of computers are far from known under the classical computational model, not to mention the development of hardware keeps improving as well as several possible new computational models such as quantum computers. During the time when I was writing this thesis, the world was revolutionized by the development of large language models (LLM) — those large deep neural nets (usually variants of the transformer architecture [Vaswani et al., 2017]) trained by large amounts of text data on webs using self-supervised learning. LLMs are able to condense and transfer the latent structure of existing human knowledge to achieve a variety of tasks such as question-answering and code generation in the format of dialogues. This marks an influential achievement towards the path of artificial general intelligence — a computation entity that can achieve a variety of tasks on or beyond human level. We are in the most exciting age of AI in history.

These achievements could not happen without two factors: the transfer of knowledge and skills across generations, and interactions, particularly collaboration, of individuals and organizations. In other words, it is not only individual intelligence but perhaps the multi-agent social learning aspect that really prosper our humans as a species. This thesis studies the applications of AI algorithms on games and the implications of intelligence through multi-agent lens at the same time. For multi-agent research, economists and social scientists have developed mathematical theories for markets and games for decades, while computer scientists and AI practitioners devised concrete algorithms for these multi-agent systems. And since computational agents are bounded rational, careful designs need to be made to confront the uncertainty and dynamic of the environments as well as to reason strategically about other decision-makers, who are also probably computational. Therefore artificial intelligence algorithms suit perfectly well for these cases, especially when the problems are so complex that no analytical solutions can be derived.

In this thesis, I particularly studied three of the most fundamental multi-agent representations ranging from the most elementary one to the most expressive one: normal-formal representation which has a minimal structure, Bayesian games which incorporates beliefs over other-players' uncertainty, and extensive-form game which models temporal and information structure of a game. And in the last part of this thesis, I use game theory to evaluate the interaction among AI algorithms themselves. I employ a variety of artificial intelligence algorithms to reason these game representations: supervised learning, unsupervised learning, graphical models, game-tree search, deep reinforcement learning and so on. I obtain results and observations over a variety of many-player, general-sum games, with applications on auctions, negotiations, and security domains.

7.1 Summary of Contributions

7.1.1 Solving Normal-Form Games

Normal-form is the most elementary representation of a game: it specifies the joint utilities for an arbitrary joint strategy input as the only description of a game. Every game has a normal-form representation, but it does not exploit the structure of the strategy space if one is known or can be approximately inferred. In Chapter 3, I considered one direction of normal-form complexity — the player set. A direct tabular encoding of a N-player and M-strategy normal-form game needs $O(NM^N)$ real values which is prohibitive to reason when N is large.

Regularity of structure can afford compact representation and efficient reasoning for many-player games. Therefore, I employ machine learning techniques to recover these structures using black-box simulation data for efficient equilibrium computation. The kinds of structure I considered are symmetry and sparsity, and I utilize unsupervised learning and unsupervised learning methods as the main approaches to learning such structures. I found my methods recover ground-truth structures quickly during the training processes, and provide quality equilibrium solutions more efficiently. My methods scale to games with hundreds of players, without storing the whole game tensor. We even found a class of games that can fit either category of structure as a game parameter varies. This suggests a broader application of my framework – we can solve a many-player game by first learning an approximate concise structure of it and then solving this learned approximated game.

7.1.2 Solving Bayesian Games

The second game representation I considered is Bayesian game representation, which models incomplete information by a concept called types. Types usually refer to the hidden information of opponents, either the parameters of their utility functions or their strategy choices. Using the Harsanyi formulation [Harsanyi, 1967], I only consider the former concept of type. And the uncertainty of types is usually encoded in a probability distribution.

I considered a standard version of Bayesian games in auction theory: types are i.i.d. for every player, and the game is symmetric. By exploiting this particular structure, I transformed the equilibrium computation into a bi-level optimization problem. I then employed an approach from deep reinforcement learning to develop the concrete optimization process: viewing the strategy as a function mapping types to actions, I can parameterize the strategy as a neural net, and use a zero-order method called natural evolution strategy to optimize the neural net. This particular optimization method is well-suited for Bayesian games since many of the standard games like auctions involve discontinuity and lack access for first-order gradient.

I developed one algorithm for computing pure equilibria, and another for mixed equilibria based on double-oracle where NES is used for best response step. My methods scale to high-dimensional domains where there is no analytical solution. For some who do have, my method can even recover these mathematical results. We also compare my methods with heuristic-based strategies, where we find certain strategic dependencies among these strategies.

7.1.3 Solving Extensive-Form Games

My last development for solving games is on the extensive-form game representation. EFG is probably the most expressive of game forms — it allows fine-grained encoding of temporal structure and information perfectness.

In Chapter 5, I studied solving general-sum, imperfect EFG, and a natural idea is to exploit the structure of the game tree. I particularly consider the adaptation of AlphaZero-styled MCTS method for game solving and agent construction. However to handle large imperfect information the vanilla MCTS needs to be modified — I incorporate a deep generative model to represent the root belief state, and use this belief distribution to guide the search procedure. For the outer training loop, I use PSRO, a population-based training regime based on EGTA to train the neural nets. I discovered a negotiation agent that can achieve human-level performance in a class of negotiation games, where I found one of my agents could achieve comparable social welfare with humans when humans are trading with themselves.

7.1.4 Evaluating Interactive AI Algorithms

In the final piece of this thesis, I consider the problem of *evaluation*, instead of *game-solving*, of different interactive AI algorithms. The motivation of this work is largely due to the applicability of solution concepts in multi-agent environments in general: Nash equilibria are not interchangeable in general-sum environments. So if the opponent at test-time is trained by a different AI algorithm, or even by the same algorithm but with a different random seed, then there is nothing to guarantee by merely minimizing the distance to the set of Nash equilibria.

We define the concept of *meta-game* as a means to evaluate interactive AI algorithms. A meta-game is a symmetric normal-form game where the strategies are the AI algorithms, and the payoffs are defined as the expected values marginalized across random seeds. My meta-game concept is constructed to capture both the *self-play* and *cross-play* information among different AI algorithms. By resampling different combinations of random seeds, my meta-game evaluation framework uses multiple approximated meta-game samples to construct confidence intervals of certain statistics. We discover a variety of properties among different multi-agent RL algorithms.

7.2 Future Works

The applications of modern AI methods in computational game theory are far broader than game-solving algorithms. In this section, I highlight a few future directions that naturally extend the development of this thesis.

7.2.1 Advanced Machine Learning Methods and Game Structures for Game Solving

The idea of structure learning generalizes beyond the sparsity and symmetry I considered in Chapter 3. There are more interesting structures that are worth studying. For example, the action-graph game representation I mentioned in Section 2.1.7.3 considers locality and symmetry of the strategy space [Jiang et al., 2011]. A possible combination of the sparsity and symmetry structure learning can be devised for learning action-graph games. Another idea is to automatically cluster similar strategies based on their payoff performances [Bard et al., 2015] and solve a fewer-strategy game thereafter.

The current state-of-the-art machine learning methods for solving many-player games use stochastic gradient descent (SGD) methods by designing a proper loss function for equilibrium computation [Gemp et al., 2021, 2023]. It may be possible to interleave this SGD method with structure learning or incorporate structural inductive biases into the loss functions.

Furthermore, the idea of structure learning extends beyond non-cooperative games. A recent work by Xu et al. [2023] studies how to learn coalition structures for cooperative games and apply game-solving methods thereafter. It may be possible to adopt a similar EGTA setting for cooperative game theory for scenarios like voting and coalition formation.

7.2.2 Learning Analytical Solutions in Bayesian Games

My pure equilibrium computation method for Bayesian games in Chapter 4 can recover analytical solutions in simple auction games. However for more complex analytical games such as all-pay auctions, the classical solution seems to be more difficult to be recovered. This also leaves an open question — investigating the effects of neural parameterization in these delicate games with probably a more sophisticated optimization procedure. During the completion of this thesis, I noticed recent works [Thoma et al., 2023, Pieroth et al., 2023] developed learning algorithms that can recover analytical equilibria even in sequential auctions, where the work by Pieroth et al. [2023] used a similar algorithm based on evolution strategies. This further shows that employing learning methods in incomplete information games is promising. Furthermore, one may ask can these learning methods be applied beyond independent type settings and recover some classical economic anomalies like the winner's curse [Thaler, 1988] or the declining price anomaly [McAfee and Vincent, 1993] in some format? Can these learning methods further guide the analytical understanding of complex auction games?

7.2.3 Advanced Search Methods and Solution Concepts for Extensive-Form Games

My methods developed in Chapter 5 combine tree-search with normal-form level analysis. However, the IS-MCTS is mainly used for single-agent best response search, and the equilibrium analysis on the normal-form level abstracts the EFG structure away. A follow-up question is whether we can exploit more of the EFG representations. For example, is it possible to utilize search on a multi-agent level for general-sum imperfect information games such as the ones proposed by Lerer et al. [2020], Sokota et al. [2021] and more recently in Kubicek et al. [2023]? Can we improve the game-theoretic analysis by using more refined notions of Nash, such as sequential equilibrium [Kreps and Wilson, 1982b], trembling-hand perfect equilibrium [Farina et al., 2018], self-confirming equilibrium [Fudenberg and Levine, 1993], or extensive-form correlated equilibrium [Von Stengel and Forges, 2008]?

7.2.4 Re-Evaluating Meta-Game Evaluation

My meta-game evaluation framework was developed in part to address the impracticability of pursuing equilibrium as a solution concept in general-sum games. Yet, I am still performing game-theoretic analysis on the meta-game instances based on which I conclude the performances of different algorithms. This brings a profound question in general: while it is clear that interactive AI algorithms have a multi-agent perspective of interpretations, is non-cooperative game theory the ultimate tool we can resort to? Or, can we utilize other frameworks, e.g., social choice theory, to provide a more theoretically sound evaluation protocol? Another concurrent work [Lanctot et al., 2023] may provide useful insights for this question. For example, instead of bootstrapping normal-form empirical games, a notion of empirical voting matrices can be developed for evaluation.

Another direction is to consider a variety of environments for meta-game evaluation. In my current work, I only use a single environment to evaluate the performance of an algorithm as a means to produce effective strategies. But such performance should be assessed across a variety of games if the meta-strategy is designed to be general-purpose. Many of the current deep MARL algorithms are indeed agnostic to the environments. Then a follow-up question is how do we select the priorities over different environments? This is related to the agent-vs-task setting by [Balduzzi et al., 2018b], and what I am thinking of is a “agent-vs-agent-vs-task” setting.

Furthermore, it is even reasonable to consider meta reinforcement learning in this context —— developing algorithms that perform well across a variety of environments. A typical setup for meta-RL is to search for the best hyperparameters (e.g., initial hyperparameters) for a class of RL algorithm (say PPO) given a distribution of tasks. My meta-game construction open a new revenue for studying meta multi-agent reinforcement learning methods.

7.2.5 Scaling Dynamic Empirical Mechanism Design via Stackelberg Deep Multi-Agent Reinforcement Learning

In Section 2.4, I mentioned mechanism design as another application of computational game theory. The idea is that a designer may publish the rule of a mechanism, and then the players play and reach some equilibrium under this rule. The designer’s goal is to select the rule that maximizes its own payoff considering the players’ responses.

Standard mechanism design problems in auction theory usually consider one-shot auctions, where the players bid once and receive outcomes accordingly. However in real-world auction systems, such as advertising auctions, players are involved in auctions repeatedly, and now players’ strategies become policies that map historical outcomes to bids at the current stage. This is called dynamical mechanism design [Bergemann and Välimäki, 2019].

As a concrete example, consider a seller that is selling an item multiple times for H rounds

Algorithm 16 Stackelberg Double Oracle

```
1: function Stackelberg-DO()
2:     Initialize $\hat{S} = \{\pi_l^0\}, \hat{T} = \{\pi_f^0\}$
3:     for $i = 1, \dots$ do
4:         Compute $\pi_l^i = \arg\min_{\pi_l} u_f(\pi_l, BR^i(\pi_l))$
5:         $\hat{S} \leftarrow \hat{S} \cup \{\pi_l^i\}$
6:         Simulate the empirical game matrix $\hat{G}^i$ defined on $\hat{S}$ and $\hat{T}$
7:         Solve for an empirical SSE $\delta_l^i, \delta_f^i$ on $\hat{G}^i$ by using the multiple LP [Conitzer and Sandholm, 2006]
8:         $\pi_f^i \leftarrow \arg\max_{\pi_f} u_f(\delta_l^i, \pi_f)$
9:         $\hat{T} \leftarrow \hat{T} \cup \{\pi_f^i\}$
10:    end for
11: end function
```

to a single buyer. At each round t, the buyer draws its valuation $v_t \sim F$. The seller decided a price p_t, as a function of the outcome of previous rounds. Then the buyer decides whether or not to buy this item as a function of p_t and the outcome of previous rounds. The buyer could also potentially communicate with the seller before deciding to buy or not, e.g., reporting its valuation, or just selecting one of many symbolic tokens for communication. In this dynamic game, the goal of the seller is to learn the valuation distribution F as well as to become aware of the buyer's potential capability to misreport its valuation or manipulate the seller's future pricing strategy by deliberately not buying in early rounds.

In contrast to standard mechanism design theory, the above complex scenario may not have a tractable analytical solution. I aim to solve this class problem computationally, or even approximately using modern AI methods. This is related to *automated mechanism design* [Sandholm, 2003] or *empirical mechanism design* [Vorobeychik et al., 2006]. Since now I am considering a policy strategy space, I may also term it *dynamic empirical mechanism design*.

Another way to think this problem is to view mechanism design as a Stackelberg game: the mechanism designer first commits to a policy, and then the followers decide their policies as a function of the mechanism. Vorobeychik and Singh [2012], Letchford et al. [2012] provided results for tabular cases. For high-dimensional cases, I may need to resort to deep reinforcement learning methods. One possible approach is to modify the update the rule of a joint-action Q-learner or multi-agent policy gradient update such that it incorporates a bi-level structure in the optimization [Sengupta and Kambhampati, 2020, Zhang et al., 2020]. Another approach I think of is to adopt a double-oracle-like approach. While there are already several works used DO in Stackelberg game settings [Karwowski and Mańdziuk, 2020, Černỳ et al., 2018, Durkota et al., 2015], they all require some kind of structure of strategy space (e.g., tree policy). I consider a Stackelberg double oracle plainly from the normal-form level in Algorithm 16.

Here $\hat{S}$, $\hat{T}$ are the constrained strategy space for the leader and the follower. u_l, u_f are their utility functions. A crucial definition is $BR^i(\pi_l) = \arg\max_{\pi_f \in \hat{T}} u_f(\pi_l, \pi_f)$, where assume the follower is choosing best responses in the current constrained set. The idea of this algorithm is that the leader is trying to induce a novel follower strategy by deliberatively minimizing its payoff. This is trying to prevent the algorithm from being stuck at a local Stackelberg leader strategy where the follower already has exact best response in the constrained set, while the global Stackelberg leader strategy is still not recovered.

BIBLIOGRAPHY

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Thirty-Fifth International Conference on Neural Information Processing Systems*, pages 29304–29320, 2021.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Stefano V. Albrecht, Jacob W. Crandall, and Subramanian Ramamoorthy. Belief and truth in hypothesised behaviours. *Artificial Intelligence*, 235:63–94, 2016.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL https://www.marl-book.com.

Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, Satinder Singh, Thore Graepel, and Yoram Bachrach. Learning to play no-press diplomacy with best response policy iteration. In *Thirty-Fourth International Conference on Neural Information Processing Systems*, pages 17987–18003, 2020.

Olivier Armantier, Jean-Pierre Florens, and Jean-Francois Richard. Approximation of Nash equilibria in Bayesian games. *Journal of Applied Econometrics*, 23(7):965–981, 2008.

Kenneth J. Arrow. An extension of the basic theorems of classical welfare economics. In *Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 507–532, 1951.

Susan Athey. Single crossing properties and the existence of pure strategy equilibria in games of incomplete information. *Econometrica*, 69(4):861–889, 2001.

Robert J. Aumann. Mixed and behavior strategies in infinite extensive game. In M. Dresher, L. S. Shapley, and A. W. Tucker, editors, *Advances in Game Theory*, Annals of Mathematics Studies, pages 627–650. Princeton University Press, 1964.

Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211(4489): 1390–1396, 1981.

Tim Baarslag, Koen Hindriks, Catholijn Jonker, Sarit Kraus, and Raz Lin. The first automated negotiating agents competition (anac 2010). *New Trends in agent-based complex automated negotiations*, pages 113–135, 2012.

Tim Baarslag, Mark JC Hendrikx, Koen V. Hindriks, and Catholijn M. Jonker. Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques. *Autonomous Agents and Multi-Agent Systems*, 30(5):849–898, 2016.

Anton Bakhtin, David Wu, Adam Lerer, and Noam Brown. No-press diplomacy from scratch. In *Thirty-Fifth International Conference on Neural Information Processing Systems*, 2021.

Anton Bakhtin, David J. Wu, Adam Lerer, Jonathan Gray, Athul Paul Jacob, Gabriele Farina, Alexander H. Miller, and Noam Brown. Mastering the game of no-press diplomacy via human-regularized reinforcement learning and planning. In *Eleventh International Conference on Learning Representation*, 2023.

Jasper Bakker, Aron Hammond, Daan Bloembergen, and Tim Baarslag. Rlboa: A modular reinforcement learning framework for autonomous negotiating agents. In *Eighteenth International Conference on Autonomous Agents and Multi-Agent systems*, pages 260–268, 2019.

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *Thirty-Fifth International Conference on Machine Learning*, 2018a.

David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. Re-evaluating evaluation. In *Thirty-Second International Conference on Neural Information Processing Systems*, pages 3272–3283, 2018b.

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech M. Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *Thirty-sixth International Conference on Machine Learning*, 2019.

Nolan Bard, Deon Nicholas, Csaba Szepesvaári, and Michael Bowling. Decision-theoretic clustering of strategies. In *Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 17–25, 2015.

Ana L. C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18(3):342–375, 2009.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Kimmo Berg and Tuomas Sandholm. Exclusion method for finding Nash equilibrium in multiplayer games. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Dirk Bergemann and Juuso Välimäki. Dynamic mechanism design: An introduction. *Journal of Economic Literature*, 57(2):235–274, 2019.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

B. Douglas Bernheim. Rationalizable strategic behavior. *Econometrica*, 52:1007–1028, 1984.

Ken Binmore, Ariel Rubinstein, and Asher Wolinksy. The Nash bargaining solution in economic modelling. *Rand Journal of Economics*, 17(2):176–188, 1986.

David Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. Evolutionary dynamics of multi-agent learning: A survey. *Journal of Artificial Intelligence Research*, 53:659–697, 2015.

Ben Blum, Daphne Koller, and Christian R. Shelton. Game theory: Gametracer. http://dags.stanford.edu/Games/gametracer.html, 2002.

Ben Blum, Christian R. Shelton, and Daphne Koller. A continuation method for Nash equilibria in structured games. *Journal of Artificial Intelligence Research*, 25:457–502, 2006.

Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014.

Branislav Bosansky, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining compact representation and incremental generation in large games with sequential strategies. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 812–818, 2015.

Vitor Bosshard, Benedikt Bünz, Benjamin Lubin, and Sven Seuken. Computing Bayes-Nash equilibria in combinatorial auctions with continuous value and action spaces. In *Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 119–127, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Yann Bramoullé, Rachel Kranton, and Martin D'Amours. Strategic interaction and networks. *American Economic Review*, 104(3):898–930, 2014.

Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

M. Broom, C Cannings, and GT Vickers. Multi-player matrix games. *Bulletin of mathematical biology*, 59(5):931–952, 1997.

George W. Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation.*, 13(1):374, 1951.

Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *Thirty-First International Conference on Neural Information Processing Systems*, 2017.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Thirty-Second International Conference on Neural Information Processing Systems*, 2018.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. In *Thirty-Fourth International Conference on Neural Information Processing Systems*, pages 17057–17069, 2020.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Twentieth International Conference Algorithmic Learning Theory*, pages 23–37, 2009.

Maciej Bukowski and Jacek Miekisz. Evolutionary and asymptotic stability in symmetric multi-player games. *International Journal of Game Theory*, 33:41–54, 2004.

Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Yang Cai and Christos Papadimitriou. Simultaneous Bayesian auctions and computational complexity. In *Fifteenth ACM Conference on Economics and Computation*, pages 895–910, 2014.

Colin F. Camerer, Teck-Hua Ho, and Juin-Kuan Chong. A cognitive hierarchy model of games. *Quarterly Journal of Economics*, 119(3):861–898, 2004.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z. Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. In *Sixth International Conference on Learning Representations*, 2018.

Sofia Ceppi, Nicola Gatti, and Nicola Basilico. Computing Bayes-Nash equilibria through support enumeration methods in Bayesian two-player strategic-form games. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 541–548, 2009.

Jakub Černỳ, Branislav Boỳanskỳ, and Christopher Kiekintveld. Incremental strategy generation for Stackelberg equilibria in extensive-form games. In *Nineteenth ACM Conference on Economics and Computation*, pages 151–168, 2018.

Shuchi Chawla and Jason D. Hartline. Auctions with unique equilibria. In *Fourteenth ACM Conference on Electronic Commerce*, pages 181–196, 2013.

Siqi Chen, Jianing Zhao, Gerhard Weiss, Ran Su, and Kaiyou Lei. An effective negotiating agent framework based on deep offline reinforcement learning. In *Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 324–335, 2023.

Shih-Fen Cheng, Daniel M. Reeves, Yevgeniy Vorobeychik, and Michael P. Wellman. Notes on equilibria in symmetric games. In *Sixth International Workshop on Game Theoretic and Decision Theoretic Agents*, 2004.

Shih-Fen Cheng, Evan Leung, Kevin M. Lochner, Kevin O'Malley, Daniel M. Reeves, Julian L. Schvartzman, and Michael P. Wellman. Walverine: A walrasian trading agent. *Decision Support Systems*, 39(2):169–184, 2005.

Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. In *Eighteenth International Conference on Neural Information Processing Systems*, 2004.

George Christodoulou, Annamária Kovács, and Michael Schapira. Bayesian combinatorial auctions. In *International Colloquium on Automata, Languages, and Programming*, pages 820–832, 2008.

Vincent Conitzer. On Stackelberg mixed strategies. *Synthese*, 193(3):689–703, 2016.

Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Seventh ACM conference on Electronic commerce*, pages 82–90, 2006.

Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:120–143, 2012.

Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions*. The MIT Press, 2006.

Vincent P. Crawford and Joel Sobel. Strategic information transmission. *Econometrica*, pages 1431–1451, 1982.

Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. k-level reasoning for zero-shot coordination in hanabi. In *Thirty-Fifth International Conference on Neural Information Processing Systems*, volume 34, pages 8215–8228, 2021.

Kai Cui and Heinz Koeppl. Approximately solving mean field games via entropy-regularized deep reinforcement learning. In *Twenty-Fourth International Conference on Artificial Intelligence and Statistics*, 2021.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *Tenth International Conference on Learning Representations*, 2022.

Partha Dasgupta and Eric Maskin. The existence of equilibrium in discontinuous economic games, i: Theory. *Review of Economic Studies*, 53(1):1–26, 1986.

Constantinos Daskalakis and Christos H. Papadimitriou. Approximate Nash equilibria in anonymous games. *Journal of Economic Theory*, 156:207–245, 2015.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

Anthony Christopher Davison and David Victor Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, 1997.

Steven de Jong, Daniel Hennes, Karl Tuyls, and Ya'akov Gal. Metastrategies in the colored trails game. In *Tenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 551–558, 2011.

David DeVault, Johnathan Mell, and Jonathan Gratch. Toward natural turn-taking in a virtual human negotiation agent. In *AAAI Spring Symposium on Turn-taking and Coordination in Human-Machine Interaction*, 2015.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference*, pages 3071–3076, 2013.

Quang Duong, Yevgeniy Vorobeychik, Satinder Singh, and Michael P. Wellman. Learning graphical game models. In *Eighteenth International Joint Conference on Artificial Intelligence*, pages 116–121, 2009.

Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. Approximate solutions for attack graph games with imperfect information. In *Sixth International Conference Decision and Game Theory*, pages 228–249, 2015.

Paul Dütting and Thomas Kesselheim. Best-response dynamics in combinatorial auctions with item bidding. In *Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 521–533, 2017.

Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David Parkes, and Sai Srivatsa Ravindranath. Optimal auctions through deep learning. In *Thirty-Sixth International Conference on Machine Learning*, pages 1706–1715, 2019.

Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review*, 97(1):242–259, 2007.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Thirty-Fifth International Conference on Machine Learning*, pages 1407–1416, 2018.

FAIR, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob1, Mojtaba Komeili1, Karthik Konath1, Minae Kwon1, Adam Lerer, Mike Lewis, Alexander H. Miller1, Sasha Mitts, Adithya Renduchintala1, Stephen Roller, Dirk Rowe1, Weiyan Shi, Joe Spisak, Alexander Wei, David Wu, Hugh Zhang, and Markus Zijlstra. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Peyman Faratin, Carles Sierra, and Nick R Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.

Gabriele Farina, Nicola Gatti, and Tuomas Sandholm. Practical exact algorithm for trembling-hand equilibrium refinements in games. In *Thirty-Second International Conference on Neural Information Processing Systems*, volume 31, 2018.

Gabriele Farina, Chun Kai Ling, Fei Fang, and Tuomas Sandholm. Correlation in extensive-form games: Saddle-point formulation and benchmarks. In *Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

Shaheen Fatima, Sarit Kraus, and Michael Wooldridge. *Principles of automated negotiation*. Cambridge University Press, 2014.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

John Fearnley, Martin Gairing, Paul W. Goldberg, and Rahul Savani. Learning equilibria of games via payoff queries. *Journal of Machine Learning Research*, 16(1):1305–1344, 2015.

E. Fehr and K. Schmidt. A theory of fairness, competition and cooperation. *Quarterly Journal of Economics*, 114:817–868, 1999.

S. G. Ficici and A. Pfeffer. Modeling how humans reason about others with partial information. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, 2008a.

Sevan G. Ficici and Avi Pfeffer. Modeling how humans reason about others with partial information. In *Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 315–322, 2008b.

Sevan G. Ficici, David C. Parkes, and Avi Pfeffer. Learning and solving many-player games through a cluster-based representation. In *Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 188–195, 2008.

Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *Thirty-Sixth International Conference on Machine Learning*, pages 1942–1951, 2019.

Daniel Friedman. Evolutionary games in economics. *Econometrica: Journal of the Econometric Society*, pages 637–666, 1991.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer, 2001.

Drew Fudenberg and David K. Levine. Self-confirming equilibrium. *Econometrica: Journal of the Econometric Society*, pages 523–545, 1993.

Y. Gal, B. Grosz, S. Kraus, A. Pfeffer, and S. Shieber. Agent decision-making in open-mixed networks. *Artificial Intelligence*, 174:1460–1480, 2010a.

Ya'akov Gal, Barbara Grosz, Sarit Kraus, Avi Pfeffer, and Stuart Shieber. Agent decision-making in open mixed networks. *Artificial Intelligence*, 174(18):1460–1480, 2010b.

Xi Alice Gao and Avi Pfeffer. Learning game representations from data using rationality constraints. In *Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 185–192, 2010.

Vikas Garg and Tommi Jaakkola. Learning tree structured potential games. In *Thirtieth International Conference on Neural Information Processing Systems*, 2016.

Vikas Garg and Tommi Jaakkola. Local aggregative games. In *Thirty-First International Conference on Neural Information Processing Systems*, 2017.

Ian Gemp, Rahul Savani, Marc Lanctot, Yoram Bachrach, Thomas W. Anthony, Richard Everett, Andrea Tacchetti, Tom Eccles, and János Kramár. Sample-based approximation of Nash in large many-player games via gradient descent. In *Twenty-First International Conference on Autonomous Agents and Multi-Agent Systems*, 2021.

Ian Gemp, Luke Marris, and Georgios Piliouras. Approximating Nash equilibria in normal-form games via stochastic optimization. *arXiv preprint arXiv:2310.06689*, 2023.

Herbert Gintis. *Game theory evolving*. Princeton University Press, 2009.

Piotr J. Gmytrasiewicz and Edmund H. Durfee. Rational coordination in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 3:319–350, 2000.

Ben Goertzel. Artificial general intelligence: Concept, state of the art, and future prospects. *Journal of Artificial General Intelligence*, 5(1):1, 2014.

Paul W. Goldberg and Stefano Turchetta. Query complexity of approximate equilibria in anonymous games. *Journal of Computer and System Sciences*, 90:80–98, 2017.

Renato Gomes and Kane Sweeney. Bayes-Nash equilibria of the generalized second-price auction. *Games and Economic Behavior*, 86:421–437, 2014.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Twenty-Eighth International Conference on Neural Information Processing Systems*, volume 27, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Rihab Gorsane, Omayma Mahjoub, Ruan John de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. In *Thirty-Sixth International Conference on Neural Information Processing Systems*, volume 35, pages 5510–5521, 2022.

Srihari Govindan and Robert Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65–86, 2003.

Srihari Govindan and Robert Wilson. Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control*, 28(7):1229–1241, 2004.

Jonathan Gray, Adam Lerer, Anton Bakhtin, and Noam Brown. Human-level performance in no-press diplomacy via equilibrium search. In *Eighth International Conference on Learning Representations*, 2020.

Christopher Griffin. Quadratic programs and general-sum games. In *Game Theory: Penn State Math 486 Lecture Notes*, pages 138–144. Online note., 2010. https://docs.ufpr.br/~volmir/Math486.pdf.

Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review*, 55:895—-943, 2022.

Barbara J. Grosz, Sarit Kraus, Shavit Talman, Boaz Stossel, and Moti Havlin. The influence of social dependencies on decision-making: Initial investigations with a new game. In *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 782–789, 2004.

Xin Guo, Anran Hu, Renyuan Xu, and Junzi Zhang. Learning mean-field games. In *Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

John C. Harsanyi. Games with incomplete information played by Bayesian players, Part I. the basic model. *Management Science*, 14(3):159–182, 1967.

John C. Harsanyi and Reinhard Selten. A generalized Nash solution for two-person bargaining games with incomplete information. *Management Science*, 18:80–106, 1972.

S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

Andreas Hefti. Equilibria in symmetric games: Theory and applications. *Theoretical Economics*, 12(3):979–1002, 2017.

Stefan Heidekrüger, Paul Sutterer, Nils Kohring, Maximilian Fichtl, and Martin Bichler. Equilibrium learning in combinatorial auctions: Computing approximate Bayesian Nash equilibria via pseudogradient dynamics. In *AAAI-21 Workshop on Reinforcement Learning in Games*, 2021.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games, 2016. preprint arXiv:1603.01121.

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *Thirty-Second International Conference on Machine Learning*, pages 805–813, 2015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Rémi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duéñez-Guzmán, and Karl Tuyls. Neural replicator dynamics: Multiagent learning via hedging policy gradients. In *Ninth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 492–501, 2020.

P. Jean-Jacques Herings and Ronald Peeters. Homotopy methods to compute equilibria in game theory. *Economic Theory*, 42(1):119–156, 2010.

Daniel Hernandez, Hendrik Baier, and Michael Kaisers. Brexit: On opponent modelling in expert iteration. In *Thirty-Second International Joint Conference on Artificial Intelligence*, 2023.

Pablo Hernandez-Leal and Michael Kaisers. Learning against sequential opponents in repeated stochastic games. In *Third Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, volume 25, 2017.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

Ryota Higa, Katsuhide Fujita, Toki Takahashi, Takumu Shimizu, and Shinji Nakadai. Reward-based negotiating agent strategies. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, volume 37, pages 11569–11577, 2023.

Josef Hofbauer, Sylvain Sorin, and Yannick Viossat. Time average replicator and best-reply dynamics. *Mathematics of Operations Research*, 34(2):263–269, 2009.

Jean Honorio and Luis E. Ortiz. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*, 16(1):1157–1210, 2015.

Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. “Other-play” for zero-shot coordination. In *Thrity-Seventh International Conference on Machine Learning*, pages 4399–4410, 2020.

Junling Hu and Michael P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.

Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

Iris AM Huijben, Wouter Kool, Max B. Paulus, and Ruud JG. Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1353–1371, 2022.

Tommi Jaakkola, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Eighteenth International Conference on Neural Information Processing Systems*, 1994.

Matthew O. Jackson. *Social and Economic Networks*. Princeton University Press, 2010.

Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *Tenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 327–334, 2011.

Nicholas R. Jennings, Peyman Faratin, Alessio R. Lomuscio, Simon Parsons, Carles Sierra, and Michael Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

Albert Xin Jiang and Kevin Leyton-Brown. Bayesian action-graph games. In *Twenty-Fourth International Conference on Neural Information Processing Systems*, pages 991–999, 2010.

Albert Xin Jiang, Kevin Leyton-Brown, and Navin AR Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, 2011.

Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1181–1189, 2015.

Catholijn Jonker, Reyhan Aydogan, Tim Baarslag, Katsuhide Fujita, Takayuki Ito, and Koen Hindriks. Automated negotiating agents competition (anac). In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Patrick R. Jordan and Michael P. Wellman. Designing an ad auctions game for the trading agent competition. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 147–162. Springer, 2009.

Patrick R. Jordan, Christopher Kiekintveld, and Michael P. Wellman. Empirical game-theoretic analysis of the TAC supply chain game. In *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1–8, 2007.

Patrick R. Jordan, Yevgeniy Vorobeychik, and Michael P. Wellman. Searching for approximate equilibria in empirical games. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1063–1070, 2008.

Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. Evaluating the performance of reinforcement learning algorithms. In *Thirty-Seventh International Conference on Machine Learning*, pages 4962–4973, 2020.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Ehud Kalai and Ehud Lehrer. Rational learning leads to Nash equilibrium. *Econometrica: Journal of the Econometric Society*, pages 1019–1045, 1993.

Jan Karwowski and Jacek Mańdziuk. Double-oracle sampling method for Stackelberg equilibrium approximation in general-sum extensive-form games. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 2054–2061, 2020.

Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical models for game theory. In *Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

Michael Kearns, Stephen Judd, Jinsong Tan, and Jennifer Wortman. Behavioral experiments on biased voting in networks. *Proceedings of the National Academy of Sciences*, 106(5):1347–1352, 2009.

Christopher Kiekintveld and Michael P. Wellman. Selecting strategies using empirical game models: An experimental analysis of meta-strategies. In *Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1095–1101, 2008.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. preprint arXiv:1412.6980.

Donald E Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Seventeenth European Conference on Machine Learning*, pages 282–293, 2006.

Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisỳ. Rethinking formal models of partially observable multiagent decision making. *Artificial Intelligence*, 303: 103645, 2022.

David M. Kreps. *Game Theory and Economic Modelling*. Oxford University Press, 1990.

David M. Kreps and Robert Wilson. Reputation and imperfect information. *Journal of Economic Theory*, 27(2):253–279, 1982a.

David M. Kreps and Robert Wilson. Sequential equilibria. *Econometrica*, pages 863–894, 1982b.

Vijay Krishna. *Auction theory*. Academic Press, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Twenty-Sixth International Conference on Neural Information Processing Systems*, 2012.

Ondrej Kubicek, Neil Burch, and Viliam Lisy. Look-ahead search on top of policy networks in imperfect information games. *arXiv preprint arXiv:2312.15220*, 2023.

H. W. Kuhn. Extensive games and the problem of information. *Annals of Mathematics Studies*, 28:193–216, 1953.

Volodymyr Kuleshov and Okke Schrijvers. Inverse game theory: Learning utilities in succinct games. In *Eleventh Conference on Web and Internet Economics*, 2015.

Minae Kwon, Siddharth Karamcheti, Mariano-Florentino Cuellar, and Dorsa Sadigh. Targeted data acquisition for evolving negotiation agents. In *Thirty-Eighth International Conference on Machine Learning*, volume 139, pages 5894–5904, 2021.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Thirty-First International Conference on Neural Information Processing Systems*, pages 4190–4203, 2017.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis.

Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Marc Lanctot, Kate Larson, Yoram Bachrach, Luke Marris, Zun Li, Avishkar Bhoopchand, Thomas Anthony, Brian Tanner, and Anna Koop. Evaluating agents using social choice theory. *arXiv preprint arXiv:2312.03121*, 2023.

Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Sixteenth International Conference on Autonomous Agents and Multi-Agent Systems*, 2017.

Joel Z. Leibo, Edgar A. Dueñez-Guzman, Alexander Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *Thirty-Eighth International Conference on Machine Learning*, pages 6187–6199, 2021.

Carlton E. Lemke and Joseph T Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 7187–7194, 2020.

Joshua Letchford and Vincent Conitzer. Computing optimal strategies to commit to in extensive-form games. In *Eleventh ACM conference on Electronic commerce*, pages 83–92, 2010.

Joshua Letchford, Liam MacDermed, Vincent Conitzer, Ronald Parr, and Charles L. Isbell. Computing optimal strategies to commit to in stochastic games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning for negotiation dialogues. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Thirty-Third AAAI Conference on Artificial Intelligence*, pages 4213–4220, 2019.

Zun Li and Michael P. Wellman. Structure learning for approximate solution of many-player games. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 2119–2127, 2020.

Zun Li and Michael P. Wellman. Evolution strategies for approximate solution of Bayesian games: Supplementary material. Avaliable at `https://rezunli96.github.io/`, 2021a.

Zun Li and Michael P. Wellman. Evolution strategies for approximate solution of Bayesian games. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021b.

Chun Kai Ling, Fei Fang, and J. Zico Kolter. What game are we playing? End-to-end learning in normal and extensive form games. In *Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

Chun Kai Ling, Fei Fang, and J. Zico Kolter. Large scale learning of agent rationality in two-player zero-sum games. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

Michael Littman. Friend-or-foe q-learning in general-sum games. In *Eighteenth International Conference on Machine Learning*, pages 322–328, 2001.

Siqi Liu, Luke Marris, Daniel Hennes, Josh Merel, Nicolas Heess, and Thore Graepel. Neupl: Neural population learning. In *Tenth International Conference on Learning Representations*, 2022.

Xiangyu Liu, Hangtian Jia, Ying Wen, Yaodong Yang, Yujing Hu, Yingfeng Chen, Changjie Fan, and Zhipeng Hu. Towards unifying behavioral and response diversity for open-ended learning in zero-sum games. In *Thirty-Fifth International Conference on Neural Information Processing Systems*, 2021.

Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. In *Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Thirty-First International Conference on Neural Information Processing Systems*, 2017.

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *Thirty-Eighth International Conference on Machine Learning*, pages 7204–7213, 2021.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618:257–263, 2023.

Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *Twenty-Eighth International Conference on Machine Learning*, 2021.

Luke Marris, Marc Lanctot, Ian Gemp, Shayegan Omidshafiei, Stephen McAleer, Jerome Connor, Karl Tuyls, and Thore Graepel. Game theoretic rating in n-player general-sum games with equilibria. *arXiv preprint arXiv:2210.02205*, 2022.

Andreu Mas-Colell, Michael Dennis Whinston, and Jerry R. Green. *Microeconomic theory*, volume 1. Oxford University Press New York, 1995.

Eric Maskin and Jean Tirole. Markov perfect equilibrium. *Journal of Economic Theory*, 100(2): 191–219, 2001.

Eric Mazumdar, Lillian J. Ratliff, and S. Shankar Sastry. On gradient-based learning in continuous games. *SIAM Journal on Mathematics of Data Science*, 2(1):103–131, 2020.

David McAdams. Isotone equilibrium in games of incomplete information. *Econometrica*, 71(4): 1191–1214, 2003.

R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, 1987.

R Preston McAfee and Daniel Vincent. The declining price anomaly. *Journal of Economic Theory*, 60(1):191–212, 1993.

John McCarthy. Ai as sport, 1997.

Richard D. McKelvey. A liapunov function for Nash equilibria. *Unpublished Work*, 1998.

Richard D. McKelvey and Andrew McLennan. Computation of equilibria in finite games. *Handbook of Computational Economics*, 1:87–142, 1996.

Richard D. McKelvey and Thomas R. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behavior*, 10(1):6–38, 1995.

Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. *Gambit: Software Tools for Game Theory*. Gambit Project, 2006. Version 0.2006.01.20.

H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Twentieth International Conference on Machine Learning*, pages 536–543, 2003.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Twenty-Seventh International Conference on Neural Information Processing Systems*, 2013.

Paul Milgrom. *Putting auction theory to work*. Cambridge University Press, 2004.

Paul R. Milgrom and Robert J. Weber. A theory of auctions and competitive bidding. *Econometrica*, pages 1089–1122, 1982.

Paul R. Milgrom and Robert J. Weber. Distributional strategies for games with incomplete information. *Mathematics of Operations Research*, 10(4):619–632, 1985.

Koichi Miyasawa. On the convergence of the learning process in a 2×2 non-zero-sum two-person game. Technical report, Princeton University, 1961.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Thirty-Third International Conference on Machine Learning*, pages 1928–1937, 2016.

Dov Monderer and Lloyd S. Shapley. Fictitious play property for games with identical interests. *Journal of economic theory*, 68(1):258–265, 1996a.

Dov Monderer and Lloyd S Shapley. Potential games. *Games and Economic Behavior*, pages 124–143, 1996b.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton University Press, 1953.

Peter Morris. *Introduction to game theory*. Springer Science & Business Media, 2012.

Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. A generalized training approach for multiagent learning. In *Eighth International Conference on Learning Representations*, 2020.

Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

John Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950a.

John Nash. Non-cooperative games. *Annals of Mathematics*, pages 286–295, 1951.

John Nash. Two-person cooperative games. *Econometrica*, 21(1):128–140, 1953.

John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950b.

Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. Decentralized stochastic control with partial history sharing: A common information approach. *IEEE Transactions on Automatic Control*, 58(7):1644–1658, 2013.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Twenty-Third International Conference on Machine Learning*, pages 673–680, 2006.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Seventeenth International Conference on Machine Learning*, 2000.

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. SCS: Splitting conic solver, version 3.2.1. https://github.com/cvxgrp/scs, November 2021.

Frans A. Oliehoek and Christopher Amato. Best response Bayesian reinforcement learning for multiagent systems with state uncertainty. In *Ninth AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2014.

Frans A. Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific reports*, 9(1):1–29, 2019.

Shayegan Omidshafiei, Karl Tuyls, Wojciech M. Czarnecki, Francisco C. Santos, Mark Rowland, Jerome Connor, Daniel Hennes, Paul Muller, Julien Pérolat, Bart De Vylder, Audrunas Gruslys, and Rémi Munos. Navigating the landscape of multiplayer games. *Nature Communications*, 11 (1):1–17, 2020.

Luis E. Ortiz, Robert E. Schapire, and Sham M. Kakade. Maximum entropy correlated equilibria. In *Eleventh International Conference on Artificial Intelligence and Statistics*, pages 347–354, 2007.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative evaluation of cooperative multi-agent deep reinforcement learning algorithms. *arXiv preprint arXiv:2006.07869*, 2020.

Philip Paquette, Yuchen Lu, Seton Steven Bocco, Max Smith, Satya O-G, Jonathan K. Kummerfeld, Joelle Pineau, Satinder Singh, and Aaron C. Courville. No-press diplomacy: Modeling multi-agent gameplay. In *Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

David C Parkes and Michael P. Wellman. Economic reasoning and artificial intelligence. *Science*, 349(6245):267–272, 2015.

Eyal Peer, Laura Brandimarte, Sonam Samat, and Alessandro Acquisti. Beyond the Turk: Alternative platforms for crowdsourcing behavioral research. *Journal of Experimental Social Psychology*, 70:153–163, 2017.

Eyal Pe'er, David Rothschild, Andrew Gordon, Zak Evernden, and Ekaterina Damer. Data quality of platforms and panels for online behavioral research. *Behavior Research Methods*, pages 1–20, 2021.

Tom Pepels, Tristan Cazenave, Mark H.M. Winands, and Marc Lanctot. Minimizing simple and cumulative regret in monte-carlo tree search. In *Third Workshop on Computer Games, CGW 2014 on Twenty-First European Conference on Artificial Intelligence*, pages 1–15, 2014.

Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *Thirty-Eighth International Conference on Machine Learning*, pages 8525–8535, 2021.

Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378(6623): 990–996, 2022.

Fabian R. Pieroth, Nils Kohring, and Martin Bichler. Equilibrium computation in multi-stage auctions and contests. *arXiv preprint arXiv:2312.11751*, 2023.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *arXiv preprint arXiv:2312.01072*, 2023.

Asaf Plan. Symmetry in n-player games. *Journal of Economic Theory*, 207:105549, 2023.

Clara Ponsati and Joel Watson. Multiple-issue bargaining and axiomatic solutions. *International Journal of Game Theory*, 62:501–524, 1997.

Marc Ponsen, Geert Gerritsen, and Guillaume Chaslot. Integrating opponent models with monte-carlo tree search in poker. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, pages 642–662, 2008.

Zinovi Rabinovich, Victor Naroditskiy, Enrico H. Gerding, and Nicholas R. Jennings. Computing pure Bayesian-Nash equilibria in games with finite actions and continuous types. *Artificial Intelligence*, 195:106–139, 2013.

TES Raghavan. Zero-sum two-person games. *Handbook of game theory with economic applications*, 2:735–768, 1994.

William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):7234–7284, 2020.

Daniel M. Reeves and Michael P. Wellman. Computing best-response strategies in infinite games of incomplete information. In *Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 470–478, 2004.

Daniel M. Reeves, Michael P. Wellman, Jeffrey K. MacKie-Mason, and Anna Osepayshvili. Exploring bidding strategies for market-based scheduling. *Decision Support Systems*, 39(1):67–85, 2005.

Philip J. Reny. On the existence of pure and mixed strategy Nash equilibria in discontinuous games. *Econometrica*, 67(5):1029–1056, 1999.

Philip J. Reny. On the existence of monotone pure-strategy equilibria in Bayesian games. *Econometrica*, 79(2):499–553, 2011.

Philip J. Reny and Arthur J. Robson. Reinterpreting mixed strategy equilibria: a unification of the classical and Bayesian views. *Games and Economic Behavior*, 48(2):355–384, 2004.

John G. Riley and William F Samuelson. Optimal auctions. *The American Economic Review*, 71 (3):381–392, 1981.

Julia Robinson. An iterative method of solving a game. *Annals of mathematics*, pages 296–301, 1951.

J Rosenmüller. On a generalization of the lemke–howson algorithm to noncooperative n-person games. *SIAM Journal on Applied Mathematics*, 21(1):73–79, 1971.

Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of encounter: Designing conventions for automated negotiation among computers*. MIT press, 1994.

Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica: Journal of the Econometric Society*, pages 97–109, 1982.

Stuart Russell. Rationality and intelligence: A brief update. In *Fundamental Issues of Artificial Intelligence*, pages 7–28. Springer, 2016.

Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Pearson, 4 edition, 2020.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. preprint arXiv:1703.03864.

Tuomas Sandholm. Automated mechanism design: A new application area for search algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 19–36. Springer, 2003.

Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Nineteenth AAAI Conference on Artificial Intelligence*, pages 495–501, 2005.

William H. Sandholm. *Population games and evolutionary dynamics*. MIT Press, 2010.

Robert E. Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 2013.

Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Joshua Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Student of games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances*, 2023.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Fourth International Conference on Learning Representations*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Aner Sela. Fictitious play in 'one-against-all' multi-player games. *Economic Theory*, 14(3):635–651, 1999.

Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. In *Models of Strategic Rationality*, pages 1–31. Springer, 1988.

Sailik Sengupta and Subbarao Kambhampati. Multi-agent reinforcement learning in Bayesian Stackelberg markov games for adaptive moving target defense. *arXiv preprint arXiv:2007.10457*, 2020.

Jack Serrino, Max Kleiman-Weiner, David C. Parkes, and Josh Tenenbaum. Finding friend and foe in multi-agent games. In *Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10): 1095–1100, 1953.

Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Twenty-Fourth International Conference on Neural Information Processing Systems*, volume 23, 2010.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Satinder Singh, Michael Kearns, and Yishay Mansour. Nash convergence of gradient dynamics in general-sum games. In *Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.

Satinder Singh, Vishal Soni, and Michael Wellman. Computing approximate Bayes-Nash equilibria in tree-games of incomplete information. In *Fifth ACM Conference on Electronic Commerce*, pages 81–90, 2004.

Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. Stackelberg security games: Looking beyond a decade of success. In *Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

Maurice Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1):171–176, 1958.

Brian Skyrms. *The stag hunt and the evolution of social structure*. Cambridge University Press, 2004.

John Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

Samuel Sokota, Caleb Ho, and Bryce Wiedenbeck. Learning deviation payoffs in simulation-based games. In *Thirty-Third AAAI Conference on Artificial Intelligence*, pages 2173–2180, 2019.

Samuel Sokota, Edward Lockhart, Finbarr Timbers, Elnaz Davoodi, Ryan D'Orazio, Neil Burch, Martin Schmid, Michael Bowling, and Marc Lanctot. Solving common-payoff games with approximate policy iteration. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 9695–9703, 2021.

Samuel Sokota, Gabriele Farina, David J. Wu, Hengyuan Hu, Kevin A. Wang, J. Zico Kolter, and Noam Brown. The update equivalence framework for decision-time planning. *arXiv preprint arXiv:2304.13138*, 2023.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Twenty-Seventh International Conference on Neural Information Processing Systems*, volume 26, 2013.

Yuhang Song, Andrzej Wojcicki, Thomas Lukasiewicz, Jianyi Wang, Abi Aryan, Zhenghua Xu, Mai Xu, Zihan Ding, and Lianlong Wu. Arena: A general evaluation platform and building toolkit for multi-agent intelligence. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 7253–7260, 2020.

Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *Thirty-First International Conference on Neural Information Processing Systems*, 2018.

GW Stewart. On the adjugate matrix. *Linear Algebra and its Applications*, 283(1-3):151–164, 1998.

DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. In *Thirty-Fifth International Conference on Neural Information Processing Systems*, pages 14502–14515, 2021.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017. preprint arXiv:1712.06567.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

Gabriel Synnaeve and Pierre Bessiere. A Bayesian model for opening prediction in rts games with application to starcraft. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 281–288. IEEE, 2011.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Thirtieth International Conference on Neural Information Processing Systems*, 2016.

Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge University Press, 2011.

Richard H Thaler. Anomalies: The winner's curse. *Journal of economic perspectives*, 2(1):191–202, 1988.

Vinzenz Thoma, Vitor Bosshard, and Sven Seuken. Computing perfect Bayesian equilibria in sequential auctions, 2023. preprint arXiv:2312.04516.

Finbarr Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, and Michael Bowling. Approximate exploitability: Learning a best response in large games. In *Thirty-First International Joint Conference on Artificial Intelligence*, pages 3487–3493, 2022.

Johannes Treutlein, Michael Dennis, Caspar Oesterheld, and Jakob Foerster. A new formalism, method and open issues for zero-shot coordination. In *Thirty-Eighth International Conference on Machine Learning*, pages 10413–10423, 2021.

Karl Tuyls, Julien Perolat, Marc Lanctot, Edward Hughes, Richard Everett, Joel Z. Leibo, Csaba Szepesvári, and Thore Graepel. Bounds and dynamics for empirical game-theoretic analysis. *Autonomous Agents and Multi-Agent Systems*, 34(7), 2020.

Nelson Vadori, Sumitra Ganesh, Prashant Reddy, and Manuela Veloso. Calibration of shared equilibria in general sum partially observable markov games. In *Thirty-Fourth International Conference on Neural Information Processing Systems*, pages 14118–14128, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Thrity-First International Conference on Neural Information Processing Systems*, volume 30, 2017.

David Vickrey and Daphne Koller. Multi-agent algorithms for solving graphical games. In *Sixteenth AAAI Conference on Artificial Intelligence*, 2002.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

Enrique Areyan Viqueira, Cyrus Cousins, Eli Upfal, and Amy Greenwald. Learning simulation-based games from data. In *Eighteenth International Conference on Autonomous Agents and Multi-Agent Systems*, 2019.

Heinrich Von Stackelberg. *The theory of the market economy*. Oxford University Press, 1952.

Bernhard Von Stengel and Françoise Forges. Extensive-form correlated equilibrium: Definition and computational complexity. *Mathematics of Operations Research*, 33(4):1002–1022, 2008.

Bernhard Von Stengel and Shmuel Zamir. Leadership games with convex strategy sets. *Games and Economic Behavior*, 69(2):446–457, 2010.

Yevgeniy Vorobeychik and Murat Kantarcioglu. Adversarial machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–169, 2018.

Yevgeniy Vorobeychik and Satinder Singh. Computing Stackelberg equilibria in discounted stochastic games. In *Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1478–1484, 2012.

Yevgeniy Vorobeychik and Michael P. Wellman. Stochastic search methods for Nash equilibrium approximation in simulation-based games. In *Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1055–1062, 2008.

Yevgeniy Vorobeychik, Christopher Kiekintveld, and Michael P. Wellman. Empirical mechanism design: Methods, with application to a supply-chain scenario. In *Seventh ACM conference on Electronic commerce*, pages 306–315, 2006.

Yevgeniy Vorobeychik, Michael P. Wellman, and Satinder Singh. Learning payoff functions in infinite games. *Machine Learning*, 67(1-2):145–168, 2007.

William E. Walsh, Rajarshi Das, Gerald Tesauro, and Jeffrey O. Kephart. Analyzing complex strategic interactions in multi-agent systems. In *AAAI-02 Workshop on Game-Theoretic and Decision-Theoretic Agents*, pages 109–118, 2002.

William E. Walsh, David Parkes, and Rajarshi Das. Choosing samples to compute heuristic-strategy Nash equilibrium. In *International Workshop on Agent-Mediated Electronic Commerce*, 2003.

Tony Tong Wang, Adam Gleave, Nora Belrose, Tom Tseng, Joseph Miller, Kellin Pelrine, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, and Stuart Russell. Adversarial policies beat superhuman Go AIs. In *Fortieth International Conference on Machine Learning*, 2023.

Weiran Wang and Miguel A Carreira-Perpinán. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application, 2013. preprint arXiv:1309.1541.

Xintong Wang, Christopher Hoang, Yevgeniy Vorobeychik, and Michael P. Wellman. Spoofing the limit order book: A strategic agent-based analysis. *Games*, 12(2):46, 2021a.

Yongzhao Wang, Qiurui Ma, and Michael P Wellman. Evaluating strategy exploration in empirical game-theoretic analysis. In *Twentieth International Conference on Autonomous Agents and Multi-Agent Systems*, 2021b.

Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *Thirty-Third AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019.

Zihe Wang, Weiran Shen, and Song Zuo. Bayesian Nash equilibrium in first-price auction with discrete value distributions. In *Nineteenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1458–1466, 2020.

Kevin Waugh, Brian D. Ziebart, and J. Andrew Bagnell. Computational rationalization: The inverse equilibrium problem. In *Twenty-Eighth International Conference on Machine Learning*, pages 1169–1176, 2011.

Michael P. Wellman. Methods for empirical game-theoretic analysis. In *Twenty-First National Conference on Artificial intelligence*, pages 1552–1556, 2006.

Michael P. Wellman. Trading agents. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–107, 2011.

Michael P. Wellman. Putting the agent in agent-based modeling. *Autonomous Agents and Multi-Agent Systems*, 30:1175–1189, 2016.

Michael P. Wellman, Daniel M. Reeves, Kevin M. Lochner, and Rahul Suri. Searching for walverine 2005. In *Agent-Mediated Electronic Commerce. Designing Trading Agents and Mechanisms*, pages 157–170, 2005.

Michael P. Wellman, Amy Greenwald, and Peter Stone. *Autonomous bidding agents: Strategies and lessons from the trading agent competition*. Mit Press, 2007.

Michael P. Wellman, Eric Sodomka, and Amy Greenwald. Self-confirming price-prediction strategies for simultaneous one-shot auctions. *Games and Economic Behavior*, 102:339–372, 2017.

Bryce Wiedenbeck, Ben-Alexander Cassell, and Michael P. Wellman. Bootstrap statistics for empirical games. In *Thirteenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 597–604, 2014.

Bryce Wiedenbeck, Fengjun Yang, and Michael P. Wellman. A regression approach for modeling games with many symmetric players. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 1266–1273, 2018.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

Robert Wilson. Computing equilibria of n-person games. *SIAM Journal on Applied Mathematics*, 21(1):80–87, 1971.

Mason Wright and Michael P. Wellman. Evaluating the stability of non-adaptive trading in continuous double auctions. In *Seventeenth International Conference on Autonomous Agents and Multi-Agent Systems*, 2018.

Mason Wright, Yongzhao Wang, and Michael P. Wellman. Iterated deep reinforcement learning in games: History-aware training for improved stability. In *Twentieth ACM Conference on Economics and Computation*, pages 617–636, 2019.

Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896): 223–228, 2022.

Yixuan Even Xu, Chun Kai Ling, and Fei Fang. Learning coalition structures with games. *arXiv preprint arXiv:2312.09058*, 2023.

Jiachen Yang, Xiaojing Ye, Rakshit Trivedi, Huan Xu, and Hongyuan Zha. Deep mean field games for learning optimal behavior policy of large populations. In *Sixth International Conference on Learning Representations*, 2018.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Thirty-Sixth International Conference on Neural Information Processing Systems*, pages 24611–24624, 2022.

Haifeng Zhang, Weizhe Chen, Zeren Huang, Minne Li, Yaodong Yang, Weinan Zhang, and Jun Wang. Bi-level actor-critic for multi-agent coordination. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 7325–7332, 2020.

Youzhi Zhang and Bo An. Converging to team-maxmin equilibria in zero-sum multiplayer games. In *Thirty-Seventh International Conference on Machine Learning*, 2020.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Twentieth Conference on Neural Information Processing Systems*, pages 905–912, 2008.