Scalable Learning and Solving of Extensive-Form Games

Chun Kai Ling

CMU-CS-23-121

June 2023

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

Thesis Committee:
J. Zico Kolter (co-chair)
Fei Fang (co-chair)
Vincent Conitzer
Tuomas Sandholm
Michael Bowling (University of Alberta)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by: Space and Naval Warfare Systems Center under award number N66001-17-1-4036; Lockheed Martin Corporation under award numbers 1012628, PO4104860685 and PO4104551024; U.S. Army under award number 063000; National Science Foundation under award number IIS2046640; Defense Advanced Research Projects Agency under award number HR00112020006; and University Of Illinois At Urbana-Champaign under award number OSP00004741l. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Game Theory, Machine Learning, Multiagent Decision Making

To my family and friends, for everything.

Abstract

Strategic interactions between agents can be formalized by game theory. In recent years modern learning techniques, coupled with the ubiquity of data and proliferation of compute, have paved the way towards computationally solving large games, with broad applications ranging from poker to security. However, several challenges arise when considering the current state of the art in large-scale game solving: (i) game parameters in many realistic settings are often *unspecified*; and (ii) there is a notable lack of scalable solvers for large, *general-sum* extensive-form games (most existing approaches being limited to zero-sum games).

In this thesis, we tackle in these two challenges in two ways. First, we study the inverse problem in game theory, where game parameters are to be learnt given only samples drawn from its quantal response equilibrium. An end-to-end, differentiable game-solving module fully compatible with modern deep learning architectures is presented. This approach allows for the learning of game parameters in two-player normal-form and extensive-form zero-sum games in a scalable manner via gradient descent. Second, we extend state of the art *online* Nash solvers currently used in zero-sum games to Stackelberg and correlated equilibria in general-sum extensive-form games. In both cases, we present efficient online search algorithms which avoid computation in branches of the game tree not encountered in actual play. These algorithms allow approximate solving of large general-sum games which offline solvers struggle with, while still providing guarantees on performance. Third, we show how to solve Stackelberg equilibrium in large perfect information general-sum games by learning Enforceable Payoff Functions—functions which capture tradeoffs in utility between players — rather than the usual scalar values. We demonstrate how to learn these functions using a variant of fitted value iteration, and quantify the suboptimality incurred by errors in function approximation.

Acknowledgments

I am forever grateful to my family: my parents for their unconditional love and support, my two brothers for much of richness, color and excitement in my life, and my niece and nephew for being absolutely adorable and uplifting. Living abroad for an extended period has illuminated the importance of family, and it is my hope that I would be able to spend many more years with all of you.

This thesis would not be possible without my advisors, Fei and Zico, both of which have been a continual source of guidance for the past six years. I am grateful for your infinite patience during my periods of low productivity, as well as the confidence and support when looking for a job in my senior years. I could not have asked for a better pair of advisors, and will definitely miss working with both of you. I am also grateful for my thesis committee members for their time—your comments have been indispensable to the writing of this thesis.

Research is rarely performed alone, and this thesis is no exception. In particular, I would like to thank Gabriele and Noam, who have mentored me during my PhD. Both of you have been a great source of inspiration which I'd continue to look up to long after graduation. Much credit goes to my undergraduate advisor Bryan for giving me unofficial advice long past graduation, alongside others who have taught and assisted me in other ways—Ashique, Arunesh, Amulya, Bo An, Nihar, Anupam, Zhihao, Christian, Pang Wei, Tay Yi, and Alvina.

Being coadvised, I am fortunate to have met amazing students from not one, but two labs. Thank you, Brandon for explaining differentiable optimization, Eric and Priya for easing me into CMU (and persuading me to accept the PhD offer!), Ryan for sharing tips for giving talks, Gaurav and Rex for helping with any server issues, and Sam for being the only one I can relate to regarding game solving.

Doing research during the pandemic was a surreal experience. Many thanks to Stephanie, Steven, Keegan, Daniel, Janel, Tze Hui, Yong Kiam and Yangfan for the memorable board game sessions. Till today, it remains a mystery if I were gifted so many wins on purpose, but that's perhaps a topic worthy of a separate thesis.

I am blessed with the support and encouragement from my close friends. Paul for the candid discussions, Hanwei for watching my back, and Davin for keeping me sane. The same goes to Tom, Mihir, Shawn, Chris, Yee Jia, Michelle, Hui Han, Samuel, Zhi Chen, Chee Hoe, Jun Ren, Ling Zhang, Wentao, Aline, Bojian, Lynette, Tian, Minji, as well as countless others who have been great company over the years.

This journey would not have been as smooth without the timely and reliable support from the CMU administrative staff, including Deb, Linda, Sara and Ann. Lastly, I am grateful to the SCS administration for keeping the graduate student population well fed with the numerous research seminars, as well as SCS building management for subsequently keeping the rodent population in check.

Contents

Chapter 1

Introduction

Recent work in artificial intelligence has led to huge advances in methods for solving large-scale, zero-sum, extensive-form games, both from methodological and applied standpoints. Methods based on counterfactual regret minimization (CFR) [193], first-order methods for game solving [103], online search [28, 136, 156], reinforcement and meta-game learning [32, 33, 85, 110, 136, 164] have enabled the computation of solutions for larger and larger games. This has resulted in an impressive number of landmark breakthroughs, including exceeding human performance in 2-player and multiplayer no-limit poker [29, 31, 136], professional-level play in the real-time strategy game *Starcraft* [174], essentially weakly solving limit poker [23], as well as applications to popular games like *Dark Chess* [189], *Mahjong* [188] and hidden identity games such as *Avalon* [158] and *Among Us* [101].

Much of research effort today is devoted towards solving board and digital games such as those aforementioned. Beyond game-playing, a parallel line of work in "real-world" settings involves modeling real-world problems as games and developing algorithms to solve them. These have applications ranging from infrastructure security [150], wildlife poaching [59, 60] to cyber-security [10, 145, 187] mostly under the formalism of security games [167]. There are, however two key fundamental challenges in computational game theory which hinder the modeling and solving of games. This first lies with difficulties in defining the game and the second has to do with the scalability of game solvers.

1.1 Unspecified Game Parameters

A cornerstone of game theory is the *von Neumann–Morgenstern utility theorem* [176], which states that under certain rationality axioms, (i) all preference orderings can be summarized by an underlying utility function and (ii) rational agents behave such to maximize their expected utility. Often, it is conveniently assumed that these utilities are known to solvers a-priori, or at least that the solver has access to a black box simulator from which one can sample trajectories. The assumption that utilities are readily available greatly simplifies downstream tasks—for example, computer scientists need only focus on equilibrium selection and computation.

However, in many real-world situations certain elements of the game (most commonly, player utilities) are unknown. If a game is grossly misspecified, then whatever solution that emerges must be treated with skepticism. Without some specification of (or at least, the ability to sample from) the game, it is virtually impossible to reason about, much less solve. Here, we will describe three existing (interrelated) approaches when faced with unknown payoffs.

- **Estimate utilities using domain knowledge.** In these situations, one way forward is to simply assign payoffs based on one's domain knowledge. For example, in an airport security setting, Pita et al. [149] base payoffs on the possible damage caused in the event of an attack. Such an approach can unsatisfying and extremely subjective. Consider the defender's perspective when faced with a successful attack in a security application. Naturally, one would assign a "very large" negative number to that outcome, but this precise magnitude will directly affect the mixed equilibrium predicted. Attempting to obtain conclusive results when payoffs are somewhat arbitrarily chosen appears to be at best a thought experiment and at worst a vexing attempt at forced storytelling which we hope to avoid.
- **Specify ordinal utilities instead.** Domain knowledge may not give us cardinal (numerical) utilities, but may be enough for us to rank each possible outcome. This may be sufficient to solve some games—for example, subgame perfect equilibrium can be found in perfect information games by backward induction. Such an approach is favored in applications such as modeling interstate conflict for its simplicity. For example, Brams [25] advocates this approach in what he terms as the "Theory of Moves". However, avoiding cardinal utilities restricts both the equilibrium and games we can handle [1]. Indeed, a follow-up paper by Stone [165] criticizes Brams's models as "backsliding" and "holding no promise of generating theoretical progress". Fundamentally, ordinal utilities cannot help us compute expected utilities and hence lack the ability to predict or explain mixed strategies, including mixed equilibrium in simultaneous move games such as Chicken.
- **Basing parameters on field data.** A more scientific approach would be to based payoffs on data gathered from the field. For example, Fang et al. [60] model payoffs for anti-poaching patrols by a zero-sum game with utilities based on the animal density in each region. These animal densities were obtained by extensive prior studies such as tiger surveys in the area of interest. However, field measurements are often difficult to directly be mapped to utilities. In the anti-poaching application, player utilities can reasonably assumed to be monotonic in animal density, yet it is unclear what their precise relationship is—it could well be linear, convex, concave or otherwise. Furthermore, utilities are often based on a combination of factors. Here, the authors adjusted utilities to penalized rough terrain based on their own first-hand experience in walking routes, leading to a situation where utility is now dependent on multiple factors whose importance has to be weighed appropriately. This once again poses the same question: where do these weighting functions come from?

The above examples are included not to downplay their importance, but rather to highlight the harsh reality in applying game theory to the real world. Clearly, the problem is not computational, but one which stems from the inherent difficulty in specifying models[2]. One approach to

[1]In some 2×2 matrix games, it is possible to characterize some equilibrium depending based on the relative payoffs, but for larger games, analytically describing equilibrium as a function of payoffs is not possible.

[2]Sometimes, this problem can be alleviated by tweaking the solution concept. For example robust equilibrium such as those by Kroer et al. [105] allow for the specification of the range of utilities allowed for each outcome,

mitigating this problem lies exploiting data collected from past player actions.

Inverse Game Theory. If we indeed believe that *some* underlying game(s) underpins players' decisions, then with some samples of players' actions under equilibrium, we should be able to learn the underlying game (and not just the equilibrium). This is in line with the current trend in Machine Learning—that the data should describe the model. Just as game solving can be seen as a "forward" problem, this reverse learning process is known as *Inverse Game Theory*[3], and can be treated in a similar fashion to a supervised learning problem. Crucially, none of the existing work in computational game theory is compatible with the rich representational models afforded by modern deep learning, which requires one to be able to take *gradients* with respect to game parameters. This leads to our first research problem.

> *Q1: How could one approach Inverse Game Theory in a differentiable manner?*

1.2 Scalable Solving of General-Sum Extensive Form Games

Many games are by nature general-sum—players are neither fully competitive nor cooperative. In fact, these are often games which exhibit some of the most interesting behavior such as signaling and threats, and include most classical games such as the Prisoners Dilemma, Chicken and Ultimatum games, but also many of the modern applications such as those mentioned in Chapter 1.1 and multiplayer board games.

Types of equilibria in general-sum games. Unlike with zero-sum games where these solution concepts coincide, in general-sum games, solution concepts other than Nash equilibrium are extensively studied and sometimes preferred, typically because they yield higher social welfare (or utility to a single player), or because they are easier to compute. The appropriate choice of equilibrium concept is highly domain specific. Here, we show two such alternatives.

- *Stackelberg equilibrium* (SSE) is often the equilibrium of choice in security games. Here, the leader (sometimes termed the defender) is assumed to have commitment privileges, allowing it to commit to some mixed strategy prior to the game beginning, while the follower (the attacker) is assumed to be best-responding. This asymmetry is justified by the assumption that in the repeated setting, the leader can build a "reputation" for playing the strategy committed to.
- *Correlated equilibrium* (CE) is preferred when players are able to coordinate their actions. Traditionally, one assumes the existence of a trusted mediator or signaling device which serves to coordinate player actions. For CE, the goal is often to maximize some notion of social welfare (SW). It is well-known that by taking advantage of correlations, the CE

guaranteeing performance even if models are slightly misspecified. However, these approaches requires the provision of adequate confidence bounds and do not remedy the root problem—that these bounds themselves are not easily specified.

[3]Not to be confused with the problem of mechanism design.

can enjoy higher expected social welfare than Nash in games such as Chicken and, in the extensive-form case, a non-zero sum variant of the classic game *Battleship* [63].

Computational complexity in solving general-sum games. Solving for Nash, SSE, or CE in general-sum games is particularly difficult in extensive-form games (EFG). For example, while finding an SSE in a matrix game can be done in polynomial time [47], the problem is known to be NP-hard (in the size of the game tree) in virtually all extensive-form games, including games with imperfect information and games with chance [115]. Similarly, finding a social-welfare maximizing CE is NP-hard in extensive-form games [179]. Even in special cases where efficient algorithms theoretically exist, the correlation plan is often too large to be stored or be of practical use. For example, the benchmark game of *Battleship* played on a 3x2 grid with a single ship of size 2 and 4 rounds of firing for each player may appear to be small, and in fact has an extensive-form correlated equilibrium (EFCE) which can be found in polynomial time [64][4]. However in reality, it has a joint strategy profile requiring more than 100M entries just to represent.

Recent methods for large-scale game solving. There is a vast literature on game solving, ranging from traditional methods involving mathematical programming to more recent algorithms involving regret minimization. In very large games, computational constraints imply that we often cannot even represent such game trees explicitly, calling for the need for algorithms able to scale up. This has led to a series of approximate solvers using techniques such as (manual) state abstraction to modern methods involving meta-game learning and reinforcement learning. We pay special attention to the following two methods which have gained much popularity and acceptance for solving *zero-sum extensive form* games.

- **Online, depth-limited search.** For large enough games, even traversing the game tree is computationally infeasible. Instead of attempting to search to the end of the game tree, one performs search to a limited depth and evaluates leaves based on heuristics or other evaluation functions. After every move, search is then restarted from the new state, hence avoiding the need to initiate search from states that are not reached in actual play. Such methods have been used in part of classic minimax search in perfect information games, with major successes in chess. More recently, such approaches have been extended to imperfect information games with aliases such as online search and continual resolving, and have been proven to be crucial in leading poker bots.
- **Function approximation.** Traditionally, one would estimate the value of a state by means of heuristic evaluation functions. In chess, this can involve a significant amount of human engineering. Today, the preferred approach would be to *learn* the value of states as a function of their features—typically one that is represented by a neural network. For imperfect information games, the value of individual game states is undefined. Instead, a value vector for *public belief states* is learnt. The use of function approximators has not only saved on human engineering efforts, but has been shown to be able to generalize well even to states not seen before.

[4]The EFCE is a relaxation of the CE. Its SW-maximizing solution can be found in polynomial time in games without chance[179].

First used in zero-sum games of perfect information, online search and function approximation have non-trivial extensions to imperfect information games. Both approaches have not only altered the landscape for game-solving, but are said to produce strategies which are not only better performing, but also more "human". However, neither method has seen widespread use in general-sum games. This is somewhat surprising, since general-sum games are often more computationally challenging than their zero-sum counterparts.

> *Q2: How do we extend existing techniques in solving zero-sum games to equilibrium in general-sum games?*

1.3 Organization of this Thesis

In Chapter 3, we present our solution to *Q1*—a method to learn games in a differentiable manner. Our work leverages both the effectiveness of modern deep learning while still retaining some rigor afforded by game theory. We present the first end-to-end differentiable game solving module, fully capable of being embedded in any deep learning pipeline requiring gradient based updates. This chapter is based on our papers [121, 122].

In Chapters 4 and 5, we present part of our answer to *Q2*—methods extending subgame resolving for both Stackelberg and extensive-form correlated equilibrium. Our algorithms are, to the best of our knowledge, the first instances where an online method has been applied to either of these equilibrium notions with theoretical guarantees on performance[5]. We also touch on the gadgets used in our algorithms for upper bounding player payoffs, which may be of independent interest to other researchers. Both these chapters are based off [119, 120]. Part of the foundation for Chapter 5 were part of collaborations and presented in [63, 64]; we do not discuss them here. While independent, we recommend reading these chapters in order as the key techniques are closely related.

In Chapter 6 we present the second part of our answer to *Q2*—the use of Function Approximation in answering in the simplest of settings: finding Stackelberg equilibrium with signaling (also known as SEFCE) in perfect-information games. We propose learning *enforceable payoff frontiers* for each state, an analog of state values used in 2-player zero-sum games. This gives rise to a variant of Fitted Value Iteration (FVI) allowing us to scale to large game trees. This is one of the few methods applying function approximation to solve large EFGs while giving performance guarantees. This line of work is based on [123].

[5]Subgame resolving was used in the multiplayer poker bot Pluribus [31]. However, their guarantees are significantly different from ours.

Chapter 2

Preliminaries

Many problems in game theory begin with the specification of 2 components. First, we have the game definition. Who are the players and what actions can each take? Given that players have acted in a certain way, what are the payoffs to each of them? When do players take actions, and what information does each player have when making decisions? Essentially, the game specification captures the 'rules of play'.

Once the game is defined, one has to decide on what is meant by 'solving' the game, known as selecting an *equilibrium concept*. Even with the commonly held assumption that players are self interested, there are often multiple reasonable equilibrium concepts, each modeling a different human/bot behavior. Can players randomize? If so, are they allowed to correlate their strategies? What if players are not perfectly rational and can make mistakes? These are design decisions that depend on the domain and application one is interested in. For this thesis, we are concerned with the behavior of players *at* the equilibrium, and not the dynamics taken to arrive at the equilibrium (e.g., evolutionary game theory [185]). We are, however interested in the the algorithmic aspects involved in the computation of the equilibrium, which can involve similar simulations.

This chapter covers the ideas and notation associated with games, various approaches for representing player strategies, as well as some of the equilibrium concepts relevant to future chapters. The bulk of this section is synthesized based on references from [115, 130, 143, 167, 179].

2.1 Normal Form Games

The most common game is the matrix, or *normal form* game. These represent situations where players make their move simultaneously, with each player receiving a payoff depending on the action of all players. In this thesis, we focus on 2-player games, with players $\mathsf{P}_1, \mathsf{P}_2$ equipped with *finite* action set $\mathcal{A}_1, \mathcal{A}_2$ respectively. A normal form game G is represented using 2 payoff matrices $P_i = \mathbb{R}^{|\mathcal{A}_1| \times |\mathcal{A}_n|}$, one for each player. Given an action profile $a = (a_1, a_2), a_i \in \mathcal{A}_i$, P_i receives a payoff of $P_i(a_1, a_2)$. The space of strategies for P_1 and P_2 respectively are the probability simplices $\Delta_{\mathcal{A}_1}$ and $\Delta_{\mathcal{A}_2}$. Given randomized strategies $u \in \Delta_{\mathcal{A}_1}, v \in \Delta_{\mathcal{A}_2}$, the expected utility that P_i obtains is given by the bilinear expression $u^T P_i v$.

Zero-sum Games. A game is *zero-sum* if $P_1 = -P_2$, otherwise it is *general-sum*. 2-player zero-sum games can be concisely represented by a single payoff matrix P. Zero-sum games are extremely well studied and enjoy nice properties, such as having a unique game value and being easier to solve computationally.

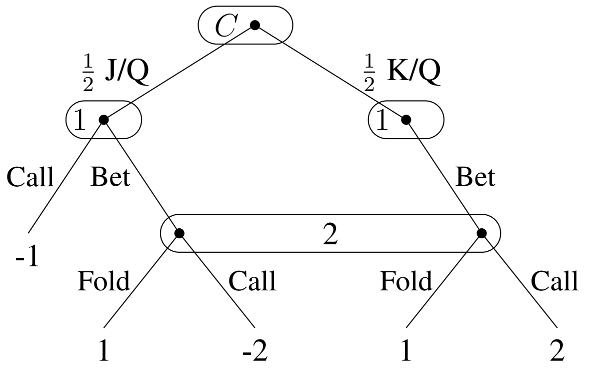
2.2 Extensive Form Games

In many real-world situations, decisions are made sequentially, with each decision potentially having consequences later. Furthermore, players are often required to make decisions without perfect knowledge of the game state—for example, poker players do not know their opponent's hidden cards. This structure is neatly formalized by Extensive Form Games (EFGs) and form the basis for the bulk of this thesis.

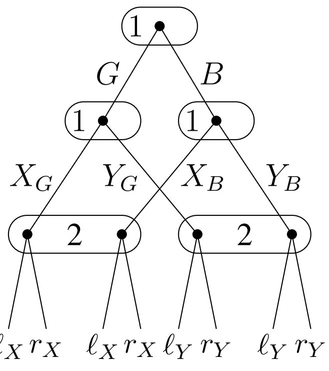
Game Structure. An EFG G is played on a rooted, finite game tree with nodes representing game states, each belonging to P_1, P_2 or a special *chance player* C representing randomness in nature (e.g., the dealer randomly distributing cards in a game of poker). The set of states is given by $\mathcal{S}$, which can be further partitioned into 4 sets $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_\mathsf{C}, \mathcal{L}$, representing states belonging to P_1, P_2, C, as well as the set of leaves (or terminal states). Each state $s \in \mathcal{S} \backslash \mathcal{L}$ is associated with a finite (possibly empty) action set $\mathcal{A}(s)$. An edge directed downwards from state s to s' has a one-to-one mapping with $\mathcal{A}(s)$, and represent actions by players or randomness from nature. Every state s and action $a \in \mathcal{A}(s)$, leads immediately to a single next state s', which we denote by shorthand as sa. We say that state s precedes s' if $s \neq s'$ and s' is a descendent of s in the game tree. This is denoted by $s \prec s'$. We write $s \preceq s'$ when allowing $s = s'$, and used $\succ$ and $\succeq$ when relationships are reversed. Note that $\prec$ is a partial order; states are comparable only when they lie on some path starting from the root.

Utilities. In EFGs it is customary to associate player utilities to each $s \in \mathcal{L}$. Each leaf is associated with a single path starting from the root (equivalently a list of actions taken by chance, or either of the players). Leaves are associated with utility functions $r_i : \mathcal{L} \to \mathbb{R}$ which define the utility received by P_i is $r_i(\ell)$ upon reaching leaf ℓ. As before, the *social welfare* is given by $r_{\text{sw}} = r_1(\ell) + r_2(\ell)$, and a game is zero-sum if $r_{\text{sw}}(\ell) = 0$ for all $\ell \in \mathcal{L}$.

Imperfect Information. To account for imperfect information, states in $\mathcal{S}_i$ are grouped into *information sets* (abbrev. infosets). States belonging the same infoset are indistinguishable. That is, $\mathcal{I}_i$ are partitions of $\mathcal{S}_i$. Each set (of states) $I_i \in \mathcal{I}_i$ constitute a single infoset. We insist that states belonging to the same infoset contain the same actions, i.e., $s, s' \in I_i \implies \mathcal{A}(s) = \mathcal{A}(s')$. If not, P_i could distinguish such states by comparing available actions. Hence, for simplicity we can write $\mathcal{A}(I_i)$ to refer to actions available at any $s \in I_i$. We further assume players exhibit perfect recall, which means that players never forget past observations and actions. Formally, perfect recall requires that (i) if s and s' belonging to P_i are in distinct information sets I_i and I_i', then their descendants $s \succ s$ and $s' \succ s'$ never belong to the same information set, (ii) if states s, s', possibly equal, belong to the same information set I_i, which contains distinct actions



(a) Simplified Poker



(b) Signaling Game

Figure 2.1: Examples of EFGs. Rounded boxes indicate information sets, with player (and chance) ownership denoted inside the box. Edges are labeled by action names. Note that we have removed the option to call after P_1 draws a king, since from a strategic perspective there is no reason to do so.

a, a', then the states $s \geq sa$ and $s' \geq s'a'$ belong to different information sets. Perfect recall is satisfied by virtually any game of practical interest. [1] The function $C : \mathcal{S} \rightarrow [0, 1]$ is the probability of reaching s assuming all players play to do so (or equivalently, the contribution of C towards reaching s).

Example 1 (EFGs): Consider a very simple (zero-sum) variant of poker shown in Figure 2.1 adapted from Burch [34]. The game starts with chance, which deals either a jack or king to us. The opponent is always dealt a queen. We (P_1) are given the choice to check, which causes the game to end with us receiving a payoff of -1. Alternatively, we could bet, which gives our opponent (P_2) the chance to call or fold. If it folds, we will earn a payoff of 1. If it calls, then we enter a showdown, where we earn 2 if we held a king, and -2 if we held a jack. Notice the information set containing both our opponent's state: these are bundled together since our opponent does not know exactly which card we were dealt. This has important strategic consequences: if the opponent could see our card, the optimal strategy (obtained by minimax search) involves us betting (and our opponent folding) when we are dealt a king, and checking when dealt a jack. However, exploiting imperfect information allows us to squeeze out additional reward. This is done by betting with probability $1/3$ when dealt a jack. This way, we earn $1/3$ regardless of whether the opponent calls or folds. The strategy just described is called the *Nash equilibrium*, which will be explained in Section 2.4. While the reasoning in this simple example was straightforward to

[1]Sometimes, whether a game has perfect recall can depend on how it is specified. For example, the card game *Bridge* can be seen as having 4 players with perfect recall, with each pair of teammates sharing the same utilities that are zero-sum *across* teams. However, one can also view this as a 2-player zero-sum game, with each team representing a single player. Now, each player has imperfect recall. While this second formulation is somewhat unnatural, it underpins why team games like Bridge cannot be solved via the same methods that were used in Poker. Indeed, designing solvers which exploit this special substructure of imperfect recall is an active area of research.

conduct by hand, the situation becomes significantly more complicated in larger games, often requiring the use of more sophisticated solvers. Indeed, the subject of efficiently solving zero-sum EFGs has been the topic of numerous influential theses [26, 34, 156] and important papers [27, 136], and is the foundation of many modern bots in games like Poker, Dark Chess, and Stratego.

2.3 Representing Strategies in Imperfect Information Extensive Form Games

An important decision when computationally solving games is how to represent strategies for each player P_i. Unlike single-player settings, we will have to represent *randomized* strategies. The choice of representation will greatly influence algorithm design, including aspects like memory and time complexities. This section's focus is on the case where players are playing independently. The case where players correlate strategies is deferred to later chapters.

2.3.1 Normal Form Strategies

The traditional, albeit inefficient representation is obtained by converting G to normal form $\mathsf{G}^{\text{normal}}$. In this new game, the set of deterministic strategies for each player is given by Cartesian product of the set of actions available at each infoset, i.e.,

$$\mathcal{A}_i^{\text{normal}} = \bigtimes_{I_i \in \mathcal{I}_i} \mathcal{A}(I_i).$$

Any action $a_i^{\text{normal}} \in \mathcal{A}_i^{normal}$ tells P_i exactly what to do at each infostate. Randomized strategies lie on the probability simplex $\Delta_{|\mathcal{A}_i^{\text{normal}}|}$. For every action $a_i^{\text{normal}} \in \mathcal{A}_i^{normal}$, we denote the action taken at infoset $I_i \in \mathcal{I}_i$ by $a_i^{\text{normal}}(I_i)$, and with a slight abuse of notation $a_i^{\text{normal}}(s_i)$ for $s \in \mathcal{S}_i$. We say that a_i^{normal} is consistent with a state s if it makes P_i play to s, expressed by the binary function

$$\mathsf{Consistent}(s, a_i^{\text{normal}}) = \begin{cases} 1, & \text{for all } s' \in \mathcal{S}_i \text{ where } s' \prec s, \quad s' a_i^{\text{normal}}(s') \preceq s \\ 0, & \text{otherwise} \end{cases}.$$

The payoff matrix P_i^{normal} has entries given by

$$P_i^{\text{normal}}(a_1^{\text{normal}}, a_2^{\text{normal}}) = \sum_{\ell \in \mathcal{L}} r_i \cdot C(\ell) \cdot \prod_{i \in \{1,2\}} \mathsf{Consistent}(\ell, a_i^{\text{normal}}).$$

The probability of reaching each leaf is bilinear in a_1^{normal} and a_2^{normal}, which lends itself well to optimization. Unfortunately, explicitly enumerating elements of $\mathcal{A}_i^{\text{normal}}$ incurs $\prod_{I_i \in \mathcal{I}_i} |\mathcal{A}(I_i)|$ space. This is computationally unappealing, since the number of infosets is typically exponential in the depth of the tree.

2.3.2 Reduced Normal Form Strategies

The *reduced normal form* seeks to prune the space of normal form actions. Consider the modified signaling game in Figure 2.1b. Here, P_1 chooses to play G or B, followed by X or Y. P_2 only observes the second action. The normal form representation contains 8 deterministic actions. However, some actions are strategically (and payoff) equivalent. For example, (G, X_G, X_B) and (G, X_G, Y_B) reach the same outcome regardless of whatever P_2 plays. Indeed, if action G was taken, it does not matter what action one would have taken if B was played. Thus, (G, X_G, X_B) and (G, X_G, Y_B) can be compressed into a single action $(G, X_G, *)$, where $*$ is a "dummy" action. This pruning step results in 4 actions $(G, X_G), (G, Y_G), (B, X_B), (B, Y_B)$.

Remark 1: In certain settings, particularly when players are boundedly rational, the reduced normal form is not equivalent to the normal form. This distinction will surface in Chapter 3.

2.3.3 Behavioral Form Strategies

Behavioral strategies overcome the problem of large representations. Instead of taking Cartesian products for each player P_i, we represent strategies as a mapping $\pi_i : \mathcal{I}_i \to \Delta_{|\mathcal{A}_i(I_i)|}$. From an implementation standpoint, we take the $\pi_i(I_i)$ at every infoset $I_i \in \mathcal{I}_i$ and concatenate these distributions into a vector of size $\sum_{I_i \in \mathcal{I}_i} |\mathcal{A}(I_i)|$.

Strategies in behavioral form are compact: it is of size $\sum_{I_i \in \mathcal{I}_i} |\mathcal{A}(I_i)|$, i.e., roughly linear in the size of the game tree, and thus significantly more memory efficient than the normal form representation. Behavioral strategies derive their power from Kuhn's Theorem [12], which states that in games with perfect recall, every normal form (potentially mixed) strategy is payoff equivalent to some behavioral strategy.[2] This implies that when solving games, we may restrict our search to the relatively small space of behavioral strategies.

Unfortunately, given behavioral strategies π_1 and π_2, the probability of reaching a specific leaf $\ell \in \mathcal{L}$ is the product of behavioral strategies (including chance) at each infoset leading up to ℓ, yielding an expression containing multinomials with degree potentially as high as the depth of the game. Thus the expected payoffs under π_1 and π_2 are linear combinations of multinomials. This can be undesirable when computing equilibrium. That said, behavioral form strategies are more "natural" representations compared to the normal and reduced form. In addition, they are more in line with methods originating from reinforcement learning and other techniques used in single-player settings.

2.3.4 Sequence Form Strategies

The sequence-form strategy, introduced by Von Stengel and Forges [179] achieves the spatial compactness of behavioral strategies while maintaining the convenience afforded by normal form strategies. Roughly speaking, for each player P_i, the sequence form is similar to the behavioral strategy except that we scale the distribution $\pi_i(I_i)$ by the product of all actions that have to be taken by P_i to reach I_i (due to perfect recall, there is only one such path).

[2]Loosely speaking, a strategy is payoff equivalent to another if they achieve the same payoffs regardless of whatever action the opponent takes.

More formally, we define the set of *sequences* for P_i as the set $\Sigma_i := \{(I, a) : I \in \mathcal{I}_i, a \in \mathcal{A}(I) \cup \{\varnothing\}$, where $\varnothing$ is known as the *empty sequence*. For any such sequence $\sigma_i \in \Sigma_i$, we write the unique last infoset of the sequence $\textbf{Inf}_i(\sigma_i') = I$. For any infoset $I_i \in \mathcal{I}_i$, we denote by denoted by $\textbf{Seq}_i(I_i)$ the *parent sequence* of I, which is the (unique) sequence which precedes I from the root to any node in I. (If no action is played by P_i prior to I, then $\textbf{Seq}_i(I) = \varnothing$.) For convenience, we overload $\textbf{Seq}_i(s)$ to represent parent sequence leading to s (this could be or $\varnothing$). Furthermore, we will drop player ids and write **Seq** and **Inf** when the context is clear. Sequences in Σ_i form a partial order.[3] For sequences $\tau = (I, a), \tau' = (I', a') \in \Sigma_i$, we write $\tau \sqsubset \tau'$ if there exists states sa, $s' \in I'$ belonging to P_i such that $sa \leq s'$, and write $\tau \sqsubseteq \tau'$ if allowing $\tau = \tau'$. If in addition, $\sigma(I') = \tau$ (or equivalently $\textbf{Seq}(\textbf{Inf}(\tau')) = \tau$), we say that τ' is an immediate successor of τ, denoted by $\tau \sqsubset_1 \tau'$ and $\textbf{Par}(\tau') = \tau$. Where necessary we will use τa to denote τ'. Lastly, infosets $I_i, I_i' \in \mathcal{I}_i$ which share the same parent sequence $\textbf{Seq}(I_i) = \textbf{Seq}(I_i')$ are called *parallel infosets*.

The sequence form replaces pure strategies by partial description of sequences specifying the player's moves over the game tree. Instead of probability vectors for each infoset, we are interested in *realization plans* $u \in \mathbb{R}^n_+, v \in \mathbb{R}^m_+$, each a vector of size equal to possible actions throughout the game , where n and m are $1 + \sum_{I_i \in \mathcal{I}_i} |\mathcal{A}(I_i)|$ respectively). Realization plans are indexed by sequences, with entries $u(\sigma)$ and $v(\sigma)$ represent probabilities of performing a sequence of actions σ, in isolation from chance and other player's moves. Note that u, v do not generally sum to 1.

Treeplexes The sequence form can visualized as *treeplexes* [88, 104], with one for each player. A treeplex is a tree with adjacent nodes alternating between infosets and sequences, with the empty sequence forming the root of the treeplex. Treeplexes encode tree-form decision problems (recall that infosets are essentially decisions points). An example of a treeplex for the classic game of Kuhn poker [106] is given in Figure 2.2. In this example, a valid (pure) realization plan would be to raise when one obtains a King, or Jack, and when dealt a Queen, call, and follow by folding if the opponent raises thereafter.

Mixed strategies in sequence form are enforced by *sequence form constraints* for P_1 (and analogously for P_2),

$$\begin{aligned} &u(\varnothing) = 1 \\ &u(\sigma_i) = \sum_{a \in \mathcal{A}(I_i)} u(\sigma_i a) \qquad \forall I_i \in \mathcal{I}_i, \sigma_i = \textbf{Seq}_i(I_i). \end{aligned} \tag{2.1}$$

The sequence form constraints may be visualized as 'flow' conservation constraints on the treeplex, with this flow being duplicated for parallel information sets. We will sometimes express the sequence form compactly by the linear constraints $Eu = e, Fv = f$. The E, F are matrices of size $|\mathcal{I}_1|$ and $|\mathcal{I}_2|$ with entries in $\{-1, 0, 1\}$, while e, f are vectors of size n and m with zeros everywhere except for $u(\varnothing) = v(\varnothing) = 1$. Thus, E and F encode parent-child relationships in the treeplex, and may be seen as a generalization of normal form requirement that u, v lie in probability simplexes. Deterministic strategies (i.e., those corressponding to a pure action in $\mathcal{A}_i^{\text{normal}}$) satisfy (2.1) and contain only zeros and ones.

[3]Warning: sequences and information sets *across* players do not form a partial order, even with perfect recall.

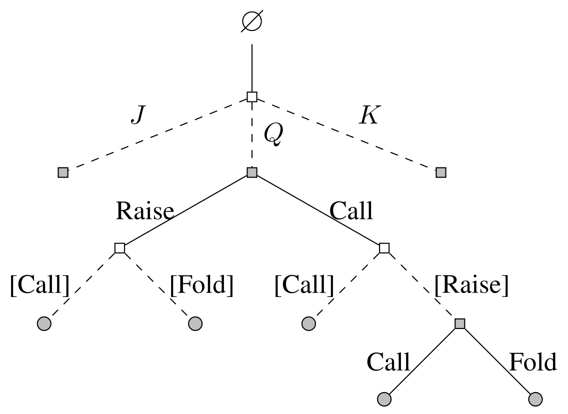


Figure 2.2: Treeplex of player 1 in Kuhn Poker. Filled squares represent information sets, circled nodes are terminal payoffs, hollow squares are points which lead to parallel information sets, which are preceded by dashed lines. Actions/sequences are given by full lines, and information from the second player is in square brackets. The treeplex is 'rooted' at the empty sequence. Subtreeplexes for the J and K outcomes are identical and thus omitted.

Example 2 (Treeplexes): The treeplex of P_1 for the game of Kuhn Poker [106] is shown in Figure 2.2 (note that for simplicity we have only shown the treeplex after a queen was drawn; the other branches are identical). There are 6 infosets, each with 2 actions. The initial 3 infosets corresponding to the actions taken immediately after being dealt a card are parallel. There are $2^6 = 64$ normal form actions, $3^3 = 27$ reduced normal for actions (each card drawn gives us 3 possible actions to continue with, i.e., Raise, Call-Call and Call-Fold), and behavioral and sequence form strategy of size $2 \cdot 6 = 12$. The sequences are of the form [card dealt]$\rightarrow$Raise, [card dealt]$\rightarrow$Call, [card dealt]$\rightarrow$Call$\rightarrow$Call and [card dealt]$\rightarrow$Call $\rightarrow$Fold.

Payoffs in sequence form can be written as a $n \times m$ *sequence form payoff matrix*, which like the normal form is denoted by P. Every leaf can be associated with a pair of unique sequences $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$. (Note that the converse is not true, since not any pair of sequences correspond to a leaf. In fact, there could be multiple leaves for each pair of sequences—e.g., when there are chance nodes after both players have taken their final actions.) The sequence form payoff matrix has entries indexed by (σ_1, σ_2), with entries

$$P_i(\sigma_1, \sigma_2) = \sum_{\ell \in \mathcal{L}: \sigma_k = \mathbf{Seq}_k(\ell)} r_i(\ell) \cdot C(\ell),$$

i.e., the expected utility of player i over all nodes reached when executing the sequence pair (σ_1, σ_2). For most practical games, P_i would be sparse, since the majority of sequence pairs do not correspond to any leaf. Just like the normal form, the P_i's payoff under strategies u, v is given by the bilinear term $u^T P_i v$.

2.4 Equilibrium Concepts

This thesis handles different equilibrium concepts. Here, we briefly touch on some of the key ideas — more precise details will be introduced when needed.

2.4.1 Best Response

Consider a G and a fixed, but possibly randomized strategy v of P_2 (in EFGs, v can be in normal, reduced, behavioral or sequence forms). The best-response of P_1 to v, denoted by $\mathsf{BR}_1(v)$, is the *set* of payoff maximizing strategies that P_1 may play assuming P_2 fixes their strategy to be v. The best response of P_2 is defined an analogous way. In a normal form game G with payoff matrices P_i, P_1's best response to P_2 strategy v is $\mathsf{BR}_1(v) = \operatorname{Arg\,max}_{u \in \Delta_{|\mathcal{A}_1|}} u^T P_1 v$, where $\operatorname{Arg\,max}$ is taken to be set valued. Similarly, when P and strategies u, v are written in sequence form, we have $\mathsf{BR}_1(v) = \operatorname{Arg\,max}_{u \geqslant 0, Eu=e} u^T P_1 v$. BR_2 is defined analogously.

The best response in EFGs is easily visualized using treeplexes. In particular, once Py is computed (in sequence form), *a* best response can be computed via a bottom-up traversal of the treeplex — when parallel information sets are encountered, their children values are summed, and when an information set is reached, we select the (deterministic) best action. After the treeplex is traversed, actions chosen in each information set describe the behavioral strategy of the computed best response.

Remark 2: In large games EFGs, we do not explicitly compute $P_1 v$ by matrix-vector multiplication. Instead, we can exploit sparsity by traversing the game tree bottom up while populating $P_1 v$ accordingly.

2.4.2 Nash Equilibrium

A 2-tuple of randomized profiles $x = (x_1, x_2)$ is a Nash equilibrium (NE) if neither player can strictly benefit by independently deviating to another (mixed) strategy, assuming the other player plays according to their distribution. In normal (or reduced normal), and sequence form representations, these may be expressed with linear constraints $\langle x_1', P_1 x_2 \rangle \leqslant \langle x_1, P_1 x_2 \rangle$ and $\langle x_1, P_2 x_2' \rangle \leqslant \langle x_1, P_2 x_2 \rangle$, where x_1', x_2' are taken to be over all strategies and P_i is taken to be over normal or sequence form representations. A NE is *pure* if x is deterministic, and *mixed* if either x_i is randomized. NE (possibly randomized) are guaranteed to exist [141], though they may not be unique.

Computing a NE even in 2-player normal form games is PPAD-hard and inapproximable [45, 52]. In fact, even simple variants such as finding the social-welfare maximizing NE or deciding if more than one NE exists is NP-hard [48, 74]. As a result, much effort has been spent in distilling out games with special structure which can be exploited yield efficient solvers (e.g., potential/congestion games [134], network [71], routing [153] and graphical [93, 98] games). There have also been attempts at deriving polynomial-time approximate solutions, particularly in the 2-player setting [21, 50, 51, 100, 124, 172]. As a generalization of normal form games, computing NE in general-sum EFGs is as expected, also computationally challenging. In practice, finding some NE can be found in small to medium-size normal and EFGs by various methods, e.g., reformulation into a Linear Complementary Problem (LCP), Lemke's algorithm [113, 179], mixed integer programming [151, 154], or homotopy type methods [86].

An important case arises when G is zero-sum, where finding a NE corresponds to finding a saddle point in a convex-concave function. In this case, equilibrium can be found efficiently (relative to the general case) by methods such as linear programming [2, 49, 70] and first order methods such as online learning/regret minimization [16, 43, 68, 69, 80, 83, 191]. These algorithms can often be extended to EFGs, for example by formulating the problem as a linear program using the sequence form [178] as well as newer methods based on smoothing [103]. This will surface later in Chapter 3. Amongst the most successful methods to solve EFGs is perhaps counterfactual regret minimization (CFR) [193] and its many variants [30, 109, 168]. These form one of the key components of the landmark accomplishments in poker AI [27, 136]. Other approaches include meta game solving [110] (and its variants, e.g., [128]), reinforcement learning [125, 147], and occasionally a combination of aforementioned techniques.

2.4.3 Stackelberg Equilibrium

The Stackelberg equilibrium models strategic interactions where P_1 plays the role of a *leader* and P_2 the *follower*. It was first used to model competitive firms and the first mover's advantage ([177]) and the ideas later extended by [180], which we use in this thesis. Informally, the leader commits to a strategy while the follower best responds to this commitment. The leader's goal is to maximize its own utility under the assumption that the follower best responds to it; in the event where there are multiple best-responses, the follower tiebreaks. That is, a pair of strategies (x_1, x_2) is a Stackelberg equilibrium if they jointly maximizes P_1's payoff under the constraint that $x_2 \in \mathsf{BR}(x_1)$. When using the sequence or (reduced) normal form, this is equivalent to finding a pair (x_1, x_2) such that (i) $x_2 \in \mathsf{BR}_2(x_1)$ and (ii) $\langle x_1, P_1 x_2 \rangle \geqslant \langle x_1, P_1 x_2' \rangle$ for all $x_2' \in \mathsf{BR}_2(x_1)$.

Remark 3: The case where tiebreaks are done in favor of P_1 is known as the *Strong* Stackelberg equilibrium (SSE). The alternative, where tiebreaking seeks to minimize P_1's utility is known as the *Weak* Stackelberg equilibrium — this amounts to swapping the sign of inequality (2) in the above definition. SSE are more commonly used in real-world applications—in almost all games, the leader can avoid tiebreaking issues by committing to a strategy arbitrarily close to x_1 such that the follower's best response is unambiguously x_2, thus it's expected payoff is arbitrarily close to $x_1^T P_1 x_2$.

SSE will always exist and yields P_1 no lower utility for P_1 than *any* NE. This follows from the fact that any (x_1^*, x_2^*) pair satisfying the NE incentive constraints also satisfies the SSE best-response constraints. In fact, Von Stengel and Zamir [180] show that SSE will yield a leader payoff no lower than any correlated equilibrium, a superset of NE which we describe in Section 2.4.4.[4] In zero-sum games, SSE coincide exactly with NE. Computing a SSE in general-sum, normal-form games can be done in polynomial time using the multiple linear program method. However, it is intractable in most EFGs, with the exception of the special game of games with perfect information and without chance[114]. SSEs have many applications, most notably in security applications such as wildlife poaching protection [60] and airport patrols [148]. The reader is directed to [92, 95, 161, 162, 167, 186] for some excellent surveys on the topic. Such games are often called *security games*, which model the interactions between attackers (followers) and defenders (leaders), often impose additional structure in payoffs to ensure compact models and

[4]Von Stengel and Zamir [180] also show that a coarse correlated equilibrium could yield an even higher payoff to the leader than SSE.

more efficient game solving [99, 102]. One such example would be the classic "water-filling" by Kiekintveld et al. [99], which has since been extended to more complicated settings (e.g., [72]).

2.4.4 Correlated Equilibrium

So far, we have assumed that players are behaving independently. For two-player zero-sum games, this is sensible, since there is no reason to coordinate player actions. This is not the case for general-sum games and cooperative games. The *Correlated Equilibrium* (CE) allows for players to coordinate actions with the aid of a trusted mediator or a randomized correlation device [11]. In a CE, the mediator recommend actions privately to the players according to a probability distribution over joint actions that is known to all players, where this distribution is such that rational players have no incentive to deviate from the recommended action. CE is known to allow for outcomes which lead to a significantly higher social welfare as compared to NE—the most notable example manifests in the classic "Game of Chicken". Furthermore, unlike NEs, CE in normal form games can be computed in polynomial time, making them computational viable alternatives to the NE.

There are numerous ways to induce correlation between players in EFGs. For example, one can convert the game to normal form and solve for a joint distribution there; this is known as the Normal Form Correlated Equilibrium (NFCE). Another example is the Extensive Form Correlated Equilibrium (EFCE), where the mediator gives recommendations for actions incrementally at each infoset [179]. Recently, Morrill et al. [137] show a strong connection between online learning and player deviations using the idea of ϕ-regret minimization [77, 90]. By carefully instantiating the set of available player deviations in an EFG, they recover a entire class of EFCE-like equilibrium. In this thesis, we avoid these nuances and focus specifically on EFCE, which will be introduced in more detail Chapter 5. It is also possible to combine correlation and commitment—this leads to equilibrium concepts like the Stackelberg EFCE (SEFCE), which we touch on in Chapter 6.

2.5 Notation Table

General

Notation	Meaning
$[n]$	$\{1, \ldots, n\}$
Δ_d	Probability simplex with dimension d
$\nabla_x f$	Gradient of f with respect to x
$\operatorname{Arg\,max}_S f$	Set of values in S which maximize f
$\arg\max_S f$	One of the values in S which maximize f (tiebreaks done arbitrarily)
$\mathbb{R}_+, \mathbb{R}^n_+$	Nonnegative reals/orthant

Extensive Form Games

Notation	Meaning
P_i, C	Player i, $i \in \{1, 2\}$, Chance/Nature
$\mathcal{S}, \mathcal{S}_i, \mathcal{S}_\mathsf{C}$	Set of all states, Set of states belonging to P_i/Chance
$s \prec s'$, $s \preceq s'$	State s precedes (is an ancestor of) s'
$\mathcal{L}$	Set of leaves/terminal states
$\mathcal{I}_i$	Set of information sets for P_i
$I_i \in \mathcal{I}_i, I_i \subseteq \mathcal{S}_i$	An information set for P_i
$\mathcal{A}(s)$, $\mathcal{A}(I_i)$	Set of actions at state s (resp. infoset I_i)
$r_i(\ell)$	P_i's utility at leaf $\ell \in \mathcal{L}$
$r_{\text{sw}}(\ell)$	$r_1(\ell) + r_2(\ell)$
$C(s)$	Probability of reaching s assuming both players play to do so
$sa \in \mathcal{S}$	Next state after player (or chance) takes a from s

Sequence Form Representation

Notation	Meaning
Σ_i	Set of sequences for P_i
$\varnothing$	Empty sequence
$\textbf{Inf}(\sigma)$	Infoset for last action taken in sequence σ
$\textbf{Seq}(I)$	Parent sequence (sequence taken by P_i to reach) infoset I_i
$\textbf{Par}(\sigma)$	Parent sequence of σ, i.e., $\textbf{Seq}(\textbf{Inf}(\sigma))$
$\sigma \sqsubset_1 \sigma'$	Sequence σ immediately precedes σ', i.e., $\textbf{Par}(\sigma') = \sigma$
$\sigma \sqsubset \sigma', \sigma \sqsubseteq \sigma'$	Sequence σ precedes (or is) σ'
P_i	Sequence form payoff matrix

Chapter 3

Differentiable Learning of Zero-Sum Games

In this chapter, we present an end-to-end framework for learning the parameters of uncertain *zero-sum* games (both for normal-form and extensive-form games), purely by observing the actions of the agents. Although there has been a great deal of work at the intersection of game theory and reinforcement learning [22, 36], most game-theoretic analysis either assumes that the payoffs underlying the game are known (this is the standard game theory setting), or forgoes trying to learn an explicit and complete representation of the game and instead looks for merely learning agent strategies that will perform well [66, 115, 181]. However, in many cases when the true underlying payoffs of the agents are *not* known, our primary goal is precisely to recover or understand the payoffs. The few exceptions that focus on learning the payoffs often rely on special structures of the game (e.g., symmetry in multiplayer setting [181]), or querying the best response of the agent with unknown payoffs by asking other agents to play carefully designed strategies [17, 115]. One relatively well studied area is the problem of identification (both in valuations and structure) for the special case of auctions. For example, Athey and Haile [9] study the problem of constructing tests which can differentiate between information structure in asymmetric auctions (i.e., what does each bidder know) using varying level of observations (e.g., winning bid, highest bid in second-price auctions etc). However, the general problem of learning game parameters by observing actions is still relatively under-explored. Some of the most closely-related works to our own is the computational rationalization framework of Waugh et al. [184], as well as Kuleshov and Schrijvers [107] though 1) our approach differs in how the utilities/payoffs are modeled; and 2) we crucially focus heavily on the extensive form settings, whereas these past work considered only normal form or succinct games; and 3) we allow for the game parameters to be a function of separately observed contextual features.

The crux of our approach is to consider the *quantal response equilibrium* (QRE), a generalization of Nash equilibrium (NE) that includes some possibility of agents acting suboptimally. We show that the solution of the QRE is a *differentiable* function of the game payoff matrix, and backpropagation can be computed analytically via implicit differentiation. We develop a solver that jointly solves the QRE for two-player zero-sum games using a primal-dual Newton Method, and allows us to compute the derivatives of agent actions with respect to the underlying payoff matrix. This enables us to develop end-to-end learning approaches that can infer the payoff ma-

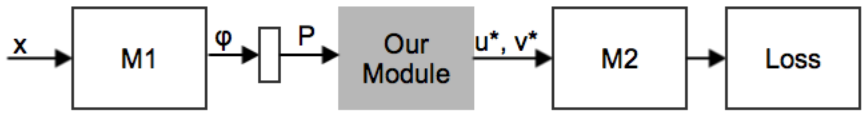


Figure 3.1: An example architecture utilizing our proposed module.

trix or other parameters underlying a game merely from samples of the agents acting according to their QREs. More generally, the method allows for (both normal form and extensive form) game-solving to be integrated as a module in deep learning systems, a strategy that can find use in multiple application areas.

In Section 3.1, we introduce the concepts of quantal response equilibrium and basics of our differentiable game solving module. This yields a simple method applicable to normal form games. In Section 3.2 we extend the method to imperfect information EFGs, and in Section 3.3 follow up by introducing an novel behavioral model for 2 player games. In Section 3.4 we describe how to scale our method up using efficient first order methods. Lastly, Section 3.5 we showcase some empirical results based on our method.

3.1 Learning and Quantal Response in Normal Form Games

Our game-solving module provides all the elements required to perform differentiable learning through the game solution. The resulting learning approach learns a mapping from *contextual features* x to payoff matrices P and computes equilibrium strategies (u^*, v^*) under a new set of contextual features. For simplicity, we will assume that x is d-dimensional, i.e., $x \in \mathcal{X} \subseteq \mathbb{R}^d$.

An example architecture is presented in Figure 3.1. Here, P is parameterized by a domain-dependent low-dimensional vector ϕ, which is dependent on a differentiable function $M_1(x)$. Similarly, the loss function is taken after applying any differentiable $M_2(u^*, v^*)$. For technical reasons, we focus on zero-sum games, which capture the simple but wide class of adversarial environments (general-sum games suffer from a variety of complications, e.g., equilibrium selection, having solutions that are nonsmooth/continuous, being difficult to compute in general).

This section focuses on normal form games. Although normal form games have limited real-world utility due to the fact that they can only handle relatively small-scale settings, the game solver and learning approach in this restricted setting captures much of the intuition and basic methodology of our approach which extends naturally to sequential settings.

3.1.1 Zero-Sum Normal Form Games

In two-player zero-sum game with payoff matrix P, a classic min-max formulation to compute the NE is as follows

$$\min_u \max_v \quad u^T P v \tag{3.1}$$

$$\text{subject to} \quad 1^T u = 1, u \geqslant 0 \tag{3.2}$$

$$1^T v = 1, v \geqslant 0, \tag{3.3}$$

u and v denote the (mixed) strategies employed by the min and max player respectively. The solution (u^*, v_0) to this optimization problem and the solution (u_0, v^*) of the corresponding problem with inversed player order (i.e., $\min_v \max_u u^T P v$) forms the Nash equilibrium (u^*, v^*).

Warning. For this chapter, we assume that P_1 is the minimizing player while P_2 is maximizing. This is in contrast to the convention in Chapter 2.

Here we present an introduction to our approach considering the case where the payoff matrix P is *not* known a priori. P could represent either a single fixed but unknown payoff matrix, or, in a more complex setting, depend on some external contextual features x. For example, in anti-poaching games [59], P depends on temperature, precipitation, or terrain. In general, however, we consider the case where we observe

1. Samples of actions $a^{(j)}$, $j = 1, \ldots, N$, each consisting of observed actions, from one or both players (i.e., in $\mathcal{A}_i$ or $\mathcal{A}_1 \times \mathcal{A}_2$), sampled from the equilibrium strategies (u^*, v^*).
2. Where existent, the contextual features $x^{(j)} \in \mathcal{X}$.

The goal is to recover the true underlying payoff matrix P, or a function form $P(x)$ depending on the current context.

3.1.2 Quantal Response Equilibria

While extremely powerful both theoretically and as a modeling tool, the NE is poorly-suited for our purposes because:

1. NEs are overly strict. In practice, many payoff matrices result in actions never being played. This tends to be overly restrictive and does not adequately describe real-world scenarios where players are boundedly rational.
2. NEs in zero-sum games may not be unique. While rare in practice, these pathological cases may lead to difficulties when resolving which NE to select.
3. NEs are discontinuous with respect to P – a small change in P can lead to jumps in u^*, v^*. This precludes integrating the technique into differentiable learning procedures.

To address these issues, in our learning setting we propose to model the player's action with the quantal response equilibria [130] instead. In general, QRE models situations where payoff matrices are injected with some noise. Specifically, we consider the *logit* equilibrium, where payoffs are perturbed by samples from a Gumbel distribution. The smoothness of the QRE with respect to P makes gradient-based approaches feasible [5]. It is known that for zero-sum games, the logit equilibrium obeys the unique fixed point

$$u_i^* = \frac{\exp(-Pv^*)_i}{\sum_{q \in [n]} \exp(-Pv^*)_q}, \quad v_j^* = \frac{\exp(P^T u^*)_j}{\sum_{q \in [m]} \exp(P^T u^*)_q}, \tag{3.4}$$

that is, u^* and v^* are the softmax of the reward vectors $-Pv^*$ and $P^T u$. It is further known that for a fixed opponent strategy, the logit equilibrium corresponds to a strategy regularized by the Gibbs entropy [132]. Since the Gibbs entropy is strictly convex (and in fact, strongly convex in $|| \cdot ||_1$), the regularized best response is unique.

Remark 4: In the QRE, there is an additional rationality (or temperature) parameter λ capturing the strength of regularization. In this section, we fix $\lambda = 1$ throughout. This is not a huge restriction when learning payoffs, since scaling rationality parameters in the QRE is the same as scaling payoff matrices by the same amount. This assumption will be lifted later on in Section 3.3 when introducing richer models for bounded rationality.

3.1.3 End-to-End Learning

In order to integrate zero-sum game solvers into an end-to-end learning framework, we need a method for "differentiating through" the game solution itself; that is, we need to compute the Jacobian (or more precisely, compute the Jacobian-vector products needed for backpropagation) of the quantal equilibrium solution with respect to the payoff matrix. Our method for doing so relies on techniques from differential calculus, and is a relatively straightforward extension of similar approaches to differentiating through optimization problems [6, 75] However, as a prelude to the more involved extensive form solution that we will discuss shortly, we describe our method in some detail, which involves both a particular approach to solving the QRE and to differentiating through its solution.

QRE Solver We observe that finding the fixed point in (3.4) is equivalent to solving the regularized min-max game

$$\begin{aligned} \min_{u\in\mathbb{R}^n_+} \max_{v\in\mathbb{R}^m_+} \quad & u^T P v - H(v) + H(u) \\ \text{subject to} \quad & 1^T u = 1, \quad 1^T v = 1, \end{aligned} \tag{3.5}$$

where $H(y)$ is the Gibbs entropy $\sum_i y_i \log y_i$. Notice that the non-negative constraints are implicit from the entropy term, and that the entropy regularization renders the equilibrium continuous with respect to P. Intuitively, entropy regularization encourages players to play more randomly, and no action has probability 0. Furthermore, since the objective is strictly a convex-concave problem, it has a unique saddle point which corresponds to (u^*, v^*).

This formulation leads to a solver for the QRE for two-player zero-sum games, using a primal-dual Newton Method. To begin, the KKT conditions for the above problem are

$$\begin{aligned} Pv + \log(u) + 1 + \mu 1 &= 0 \\ P^T u - \log(v) - 1 + \nu 1 &= 0 \\ 1^T u = 1, \qquad 1^T v &= 1, \end{aligned} \tag{3.6}$$

where μ, ν are Lagrange multipliers for the equality constraints on u, v respectively. Following Newton's method, we get the following update rule, which provides a convergent method for computing the QRE for 2 player zero-sum games

$$Q \begin{bmatrix} \Delta u \\ \Delta v \\ \Delta \mu \\ \Delta \nu \end{bmatrix} = - \begin{bmatrix} Pv + \log u + 1 + \mu 1 \\ P^T u - \log v - 1 + \nu 1 \\ 1^T u - 1 \\ 1^v - 1 \end{bmatrix}, \tag{3.7}$$

where Q is the Hessian of the Lagrangian, given by

$$Q = \begin{bmatrix} \text{diag}(\frac{1}{u}) & P & 1 & 0 \\ P^T & -\text{diag}(\frac{1}{v}) & 0 & 1 \\ 1^T & 0 & 0 & 0 \\ 0 & 1^T & 0 & 0 \end{bmatrix}. \tag{3.8}$$

Differentiating Through QRE Solutions The QRE solver also provides a method for computing the necessary Jacobian-vector products. The derivation follows in a similar manner to recent work in differentiating equality-constrained optimization problems [6, 75, 94] (the only difference being the min-max objective instead of a pure minimization objective, but since we compute differentials via the KKT conditions, the differences are minor). Specifically, given the solution (u^*, v^*) to the QRE, and considering some loss function $L(u^*, v^*)$ (for example, the log-likelihood of some observed data given this equilibrium probabilities), we show here how to compute the gradient of the loss with respect to the payoff P. In particular, taking differentials of the KKT conditions and rearranging leads to the following expression

$$Q \begin{bmatrix} \mathsf{d}u & \mathsf{d}v & \mathsf{d}\mu & \mathsf{d}\nu \end{bmatrix}^T = \begin{bmatrix} -\mathsf{d}Pv & -\mathsf{d}P^T u & 0 & 0 \end{bmatrix}^T. \tag{3.9}$$

For small changes denoted by $\mathsf{d}u, \mathsf{d}v$, we have

$$\begin{aligned} \mathsf{d}L &= \begin{bmatrix} \nabla_u L & \nabla_v L & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathsf{d}u & \mathsf{d}v & \mathsf{d}\mu & \mathsf{d}\nu \end{bmatrix}^T \\ &= \begin{bmatrix} \nabla_u L & \nabla_v L & 0 & 0 \end{bmatrix} Q^{-1} \begin{bmatrix} -\mathsf{d}Pv & -\mathsf{d}P^T u & 0 & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} v^T \mathsf{d}P^T & u^T \mathsf{d}P & 0 & 0 \end{bmatrix} Q^{-1} \begin{bmatrix} -\nabla_u L & -\nabla_v L & 0 & 0 \end{bmatrix}^T, \end{aligned}$$

where the last step is from symmetry of Q. This expression governs how small changes in $\mathsf{d}P$ affect L. For example, we may obtain the change in L after perturbing a single entry in P. Applying this procedure to all entries in P, simplifying and taking limits as $\mathsf{d}P$ is small yields

$$\nabla_P L = y_u v^T + u y_v^T, \tag{3.10}$$

where

$$\begin{bmatrix} y_u & y_v & y_\mu & y_\nu \end{bmatrix}^T = Q^{-1} \begin{bmatrix} -\nabla_u L & -\nabla_v L & 0 & 0 \end{bmatrix}^T. \tag{3.11}$$

Hence, the forward and backward passes with our module are respectively given by: 1) Using the expression in (3.7), solve for the logit equilibrium given P, and 2) Using $\nabla_u L$ and $\nabla_v L$, obtain $\nabla_P L$ using (3.10). It is stressed that the module is sufficiently general to be included in any existing architecture where having a zero-sum game module is appropriate.

Remark 5: It is natural to ask if the games are identifiable – that is, is there a unique P which under the logit QRE, generates u^*, v^*? The answer is no, in general. Assuming u^*, v^* are fixed, we can rewrite the KKT conditions in (3.6) as a system of linear equations in P. This system has $\mathcal{O}(nm)$ unknowns but only $\mathcal{O}(n + m)$ constraints. This implies that without a sufficiently compact parametrization, there will be infinitely many payoff matrices leading to identical equilibria.

For example, one can add a constant to all entries in P without changing the QRE. Another example arises when in rock paper scissors, but with payoffs for wins and losses swapped—both have uniform distribution as QREs. When under-constrained, one cannot expect to recover P. We believe characterizing the set of possible P's that agree with available data is an interesting but diffuclt problem that we leave to future work. However, in this thesis we show empirically that there are many settings where it *is* possible to recover underlying parameters.

Remark 6: We can use this differentiable game solver within an automatic differentiation framework to easily obtain gradients of virtually any loss with respect to any of the game parameters. Specifically we used the PyTorch library [144] to backpropagate gradients.

Remark 7: The derivation for $\nabla_P L$ may equivalently be derived using the implicit function theorem. See [14] for a detailed discussion on how this is done with convex optimization problems. The general technique of "differentiating through" optimization problems is known as *differentiable optimization*. Today, differentiable optimization is sufficiently advanced to be integrated into generic convex optimization packages [3, 4, 56], allowing them to be used off-the-shelf without having to explicitly perform the above derivations. Recently, this automated process has been extended to saddle-point problems. These generic methods are not optimized for EFGs, since generic solvers do not take into account structure and can be prohibitively slow for large EFGs. Scalability is addressed in Section 3.4.

3.2 Learning Extensive Form Games

In practice, many games are more naturally represented in extensive form. Can we learn game parameters in a similar fashion to payoff matrices in normal form? Does this have implications when modeling bounded rationality? How do we ensure that forward and backward passes are performed efficiently? This section addresses these issues.

3.2.1 Dilated Entropy Regularization in the Sequence Form

Learning payoff matrices of their equivalent normal form representation is computationally infeasible even for small games. For example, one-card poker has 2^{26} pure strategies per player. To facilitate learning of EFGs, we turn to the *sequence form* representation [178], which is sufficiently rich to represent all strategic behaviors given perfect recall (see Section 2.3.4).

Recall that sequence form strategies are given by *realization plans* $u \in \mathbb{R}^n_+, v \in \mathbb{R}^m_+$, where n and m are equal to the size of the game tree's action nodes for each player plus 1, where $u(\sigma)$ and $v(\sigma)$ represent probabilities of performing a sequence of actions σ in isolation from chance and other player's moves. As shown in (2.1), these are represented by sequence form constraints $Eu = e, Fv = f$. The expected payoff is given by $u^T Pv$, where P is the sequence form payoff matrix as defined in 2.3.4. Similar to the normal form representation, we propose solving the regularized min-max problem

$$\min_{u \in \mathbb{R}^n} \max_{v \in \mathbb{R}^m} u^T Pv + \sum_{I \in \mathcal{I}_1} \sum_{a \in \mathcal{A}(I)} u(Ia) \cdot \log \frac{u(Ia)}{u(\textbf{Seq}(I))} - \sum_{I \in \mathcal{I}_2} \sum_{a \in \mathcal{A}(I)} v(Ia) \cdot \log \frac{v(Ia)}{v(\textbf{Seq}(I))} \quad (3.12)$$

$$\text{such that } Eu = e, u \geqslant 0, \qquad Fv = f, v \geqslant 0 \tag{3.13}$$

This form of regularization is known as dilated or normalized entropy [24, 103], and is strictly convex/concave in u and v respectively. Roughly speaking, (3.12) adds in entropy regularization for each information set I based on the behavioral strategy at that I but weighed according to the probability of the parent sequence of I. Observe that this formulation operates in $\mathcal{O}(m+n)$ dimensions. This is bounded by the size of the game tree and is normally much smaller than the number of pure strategies.

One of our primary results is the fact that this particular form of regularization applied to the sequence form recovers the QRE as applied to the equivalent reduced normal form representation 2.3, i.e., the normal form representation but with unattainable strategies removed.

Theorem 1. *The solution to the optimization problem in* (3.12) *is realization equivalent to the QRE of the game in reduced normal form.*

Proof. (Sketch, see Section 3.6.1 for details) For each player, traverse and transform the treeplex bottom up. When encountering parallel infosets, take cartesian products and show that applying the softmax operator over the joint action space is equivalent to performing softmax independently for each infoset (since for a distribution with fixed marginals, entropy is maximized with independence; this is known as the maximum entropy coupling problem). When encountering a sequence, append the children actions (this explains why we get the reduced but not the vanilla normal form). □

Theorem 1 shows dilated entropy regularization leads to a well-accepted solution concept, even if it differs slightly from the more traditional definition of the QRE for extensive form games[1] [131]. A side consequence is that under certain regimes, the excessive gap technique [88, 103] used to quickly solve zero-sum EFGs converges to the NE specified by QRE in reduced normal form, as the rationality-parameter tends to ∞. Though not the focus of our work, solving (3.12) is a much faster (not to mention cleaner) alternative to explicitly computing the reduced normal form of the QRE.

3.2.2 Differentiable Learning in the Sequence Form

Here we derive a differentiable formulation of the sequence form QRE, mirroring our derivation for the normal form case, but admittedly with significantly more complex notation due to the more involved entropy term. The KKT conditions of our optimization problem are,

$$[Pv](\sigma_1) + 1 + \log \frac{u(\sigma_1)}{u\left(\textbf{Par}(\sigma_1)\right)} - J(\sigma_1) + \sum_{c \in \mathcal{C}(\sigma_1)} \mu(c) - \mu\left(\textbf{Inf}(\sigma_1)\right) = 0, \quad \forall \sigma_1 \in \Sigma_1$$

$$[P^T u](\sigma_2) - 1 - \log \frac{v(\sigma_2)}{v\left(\textbf{Par}(\sigma_2)\right)} + J(\sigma_2) + \sum_{c \in \mathcal{C}(\sigma_2)} \nu(c) - \nu\left(\textbf{Inf}(\sigma_2)\right) = 0, \quad \forall \sigma_2 \in \Sigma_2$$

[1]McKelvey and Palfrey [131] propose the agent form QRE (AQRE), which roughly speaking treats each infoset controlled by a separate player, each of which perform their own internal softmax.

$$Eu - e = 0, \quad Fv - f = 0, \quad u \geqslant 0, \quad v \geqslant 0$$

where $\mathcal{C}(\sigma_1), \mathcal{C}(\sigma_2)$ are the set of parallel infosets following sequences σ_1 or σ_2 and $J(\sigma_1), J(\sigma_2)$ are their respective sizes $|\mathcal{C}(\sigma_1)|, |\mathcal{C}(\sigma_2)|$. The dual variables μ and ν are vectors indexed by the infosets in $\mathcal{I}_1$ and $\mathcal{I}_2$. We stack and write the terms on the left hand side as a vector function $g(u, v, \mu, \nu)$. Taking derivatives again yields the updates for Newton's method.

$$\begin{bmatrix} -\Xi(u) & P & E^T & 0 \\ P^T & \Xi(v) & 0 & F^T \\ E & 0 & 0 & 0 \\ 0 & F & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \\ \Delta \mu \\ \Delta \nu \end{bmatrix} = -g(u, v, \mu, \nu)$$

where $\Xi(u) \in \mathbb{R}^{n \times n}$ and $\Xi(v) \in \mathbb{R}^{m \times m}$ with entries indexed by *pairs* of sequences in $(a, b) \in \Sigma_1 \times \Sigma_1$ and $(a', b') \in \Sigma_2 \times \Sigma_2$

$$\Xi(u)_{ab} = \begin{cases} -\frac{1+J(a)}{u(a)}, & a = b \\ \frac{1}{u(b)}, & \mathbf{Par}(a) = b \\ \frac{1}{u(a)}, & \mathbf{Par}(b) = a \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \Xi(v)_{a'b'} = \begin{cases} -\frac{1+J(a')}{v(a')}, & a' = b' \\ \frac{1}{v(b')}, & \mathbf{Par}(a') = b' \\ \frac{1}{v(a')}, & \mathbf{Par}(b') = a' \\ 0, & \text{otherwise} \end{cases}.$$

The updates are done in exactly the same manner as (3.10)

$$\nabla_P L = y_u v^T + u y_v^T, \tag{3.14}$$

where

$$\begin{bmatrix} y_u \\ y_v \\ y_\mu \\ y_\nu \end{bmatrix} = \begin{bmatrix} -\Xi(u) & P & E^T & 0 \\ P^T & \Xi(v) & 0 & F^T \\ E & 0 & 0 & 0 \\ 0 & F & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\nabla_u L \\ -\nabla_v L \\ 0 \\ 0 \end{bmatrix}. \tag{3.15}$$

As expected, when there is only one infoset for each player, the expressions in (3.14) and (3.15) are identical to (3.11) and (3.10).

3.3 Nested Logit Behavioral Models for Two-Player Games

One of the fundamental research problems in behavioral science is to mathematically model seemingly irrational (or non-utility maximizing) human behavior. Indeed, the assumption that players behave in accordance to the QRE of the reduced normal form (Theorem 1) severely limits the space of player strategies, and is known to exhibit pathological behavior even in one-player settings. To remedy this, we turn to models for bounded rationality in the *single-player* setting and propose natural extensions in the general-sum setting. This culminates in the Nested Logit QRE (NLQRE), which encompasses a wide range of rationality models and possesses rationality parameters which can be likewise learnt using gradient descent.

3.3.1 Nested Logit Models for One-Player Games

The class of the random utility models (RUM) [170] contain some of the most well-studied models of bounded rationality. The most commonly studied RUM is the Logit model[2], where given a set of alternatives $\mathcal{A}$ each with (known) utility U_a, the probability that alternative a is picked is $u^*_a = \frac{\exp(U_a)}{\sum_{a' \in \mathcal{A}} \exp(U_{a'})}$. It is equivalent to the probability that alternative a has highest utility under Gumbel noise, i.e., $u^*_a = \mathbb{P}\left(\arg\max_{a'}(U_{a'} + \epsilon_{a'}) = a\right)$, where $\epsilon_{a'}$ are i.i.d. Gumbel distributed. However, the standard logit model suffers from limitations.

Example 3 (Red-Bus, Blue-Bus paradox[3] McFadden et al. [129]): Suppose there are 3 alternatives for transport – a red bus, a blue bus, and a car. The player derives the same utility for each alternative, $x_{\text{car}} = x_{\text{red}} = x_{\text{blue}}$. Applying the logit model gives an equal probability of choosing each vehicle. However, one could very reasonably expect the car to be taken with probability $1/2$ and each bus to be chosen with probability $1/4$, since color of buses should have no material impact on decisions—each bus is a perfect substitute for the other.

The problem in Example 3 arises because logit models obey the strong property of *independence of irrelevant alternatives*, meaning that it does not take into account cases when alternatives are 'qualitatively' similar. *Nested-logit* (NL) models [171] address this limitation by grouping fundamentally similar alternatives together and allows for correlations between ϵ's belonging to the same group. In Example 3, we can split the decision process into two levels—in the first level, we decide whether to take a car or a bus, and in the second level (if reached), we choose which bus to take. Each level includes one round of dilated entropy regularization. If we have an extremely small amount of regularization at the second level compared to the first, then we end up with equal probabilities of taking buses and cars. If we use a standard logit model, then all 3 vehicles will be taken with probability $1/3$. This suggests that we can vary the amount of regularization smoothly to recover a range of models of bounded rationality.

In a general two-level NL model, $\mathcal{A}$ is divided into K disjoint clusters, with alternatives a belonging to cluster $k(a)$ chosen with probability

$$u^*_a \propto \exp\left(U_a / \lambda_{k(a)}\right) \left(\sum_{a' \in k(a)} \exp\left(U_{a'} / \lambda_{k(a)}\right) \right)^{\lambda_{k(a)} - 1},$$

where λ's are parameters governing noise correlation. These probabilities may be interpreted as a two-stage decision making process: in the first stage, a cluster is chosen, and in the second stage, the specific action is selected based on (scaled) softmax on U_a within the cluster. The probability of choosing each cluster in the first stage is given by the softmax over the (scaled) log-sum-exp of each cluster. When $\lambda = 1$, the standard logit model is recovered, and when $\lambda \to 0^+$, the 'elimination by aspects' model is obtained [173]. NL models can have multiple layers, leading to a NL tree representing the nested grouping. The reader is directed to Train [171] for background about nested logits and their various interpretations.

[2]Logits are more commonly known by the machine learning community as the 'softmax' operator.

[3]This was earlier known as the Beethoven/Debussy [126] or the bicycle/pony [166] example.

3.3.2 Nested-Logit Quantal Response Equilibria

Our proposed Nested-logit QRE (NLQRE) is a generalization of both the QRE (in zero-sum games) and NL models. That is, it generalizes NL models to two player zero-sum games, or equivalently, extends the QRE by permitting a more general nested logit structure. This allows us to model a far wider range of player behaviors, and in particular, cases where player rationality varies between stages of the game. We assume that the grouping of actions within each information set is known a-priori. The NLQRE is given by the unique solution to the following optimization problem

$$\begin{aligned} \min_{u \in \mathbb{R}^n} \max_{v \in \mathbb{R}^m} \quad & u^T P v + \sum_{I \in \mathcal{I}_1} \lambda_I \sum_{a \in \mathcal{A}(I)} u(Ia) \log \frac{u(Ia)}{u\left(\mathbf{Seq}(I)\right)} - \sum_{I \in \mathcal{I}_2} \lambda_I \sum_{a \in \mathcal{A}(I)} v(Ia) \log \frac{v(Ia)}{v\left(\mathbf{Seq}(I)\right)} \\ \text{such that} \quad & Eu = e, u \geqslant 0, \quad Fv = f, v \geqslant 0, \end{aligned} \tag{3.16}$$

where $\lambda_I > 0$ is a different regularization parameter for every information set in $\mathcal{I}$. The NL model is recovered in a one-player setting (i.e., Pv is a constant vector) and the QRE is recovered when there is no nesting and $\lambda = 1$ everywhere. Here, we do not assume λ's are known in our solution concept and instead treat them as parameters to be learned.

Remark 8: Being a generalization of NL models implies that NLQREs also take care of the same pathologies seen in single player settings, including the red-bus blue-bus paradox of McFadden et al. [129]. In fact, the additional representation power brought by introducing λ's to (3.16) cannot be achieved by a simple scaling of the payoff matrix in (3.12), even in the non-nested normal form games. A trivial counterexample is seen in a variant of rock-paper-scissors with non-uniform rewards (i.e., the payoffs for winners depend on their specific action). Suppose the game is played between 'strong' and 'weak' players reflected by low and high λ parameters respectively. Due to differing λ's for each player, the strategies of the two players in equilibrium are different. However, scaling P, or even changing individual payoffs for winners (while maintaining symmetry) can only result in symmetric equilibrium.

Remark 9: Readers familiar with nested logits may recall that the most common form of nested logits do not admit chance nodes (or in our 2-player setting, parallel information sets). It may be shown that there is a natural way of doing so by considering representing each alternative as a pure strategy in the *reduced normal form*, and by nesting each action based on the information sets which have a non-zero probability of being reached. We discuss these distinctions in Appendix 3.6.2.

Remark 10: The expression in (3.16) is fairly general. Broadly speaking, our framework allows for 2 types of nesting. First we allow for nesting via information sets (i.e., each information set gets its own λ, see Remark 1), and second, by clustering actions within an information set, which is achieved by introducing intermediate information sets (e.g., the 'red and blue bus' example). Our experiments in Section 3.5 focus on the former. However (3.16) and our proposed solver is able to handle the latter case, assuming that the nesting structure is known a-priori.

3.3.3 Differentiably Solving for NLQRE

Using the approach in Sections 3.1 and 3.2, we arrive at a naive solver for the NLQRE based on Newton's method. Recall $\mathcal{C}(\sigma_1)$ and $\mathcal{C}(\sigma_2)$ as sets of possible information sets immediately following $\sigma_1 \in \Sigma_1$ or $\sigma_2 \in \Sigma_2$. Define $J(\sigma) = \sum_{h \in C(\sigma)} \lambda_h$ — note that this is exactly the same as before if all λ's were equal to 0. Following the same procedures, the KKT conditions for (3.16) are, for all $h' \in \mathcal{I}_v, a' \in \mathcal{A}_{h'}$, and for all $h \in \mathcal{I}_u, a \in \mathcal{A}_h$,

$$[Pv](\sigma_1) + \lambda_{\textbf{Inf}(\sigma_1)} \cdot \left(1 + \log \frac{u(\sigma_1)}{u(\textbf{Par}(\sigma_1))}\right) - J(\sigma_1) + \sum_{c \in \mathcal{C}(\sigma)} \mu(c) - \mu(\textbf{Inf}(\sigma_1)) = 0 \quad \forall \sigma_1 \in \Sigma_1$$

$$[P^T u](\sigma_2) - \lambda_{\textbf{Inf}(\sigma_2)} \cdot \left(1 + \log \frac{v(\sigma_2)}{v(\textbf{Par}(\sigma_2))}\right) + J(\sigma_2) + \sum_{c \in \mathcal{C}(\sigma_2)} \nu(c) - \nu(\textbf{Inf}(\sigma_2)) = 0 \quad \forall \sigma_2 \in \Sigma_2$$

$$Eu - e = 0 \qquad Fv - f = 0. \tag{3.17}$$

These are necessary and sufficient conditions for NLQRE, implying that the NLQRE can be found by applying Newton's method to (3.17), yielding the following updates

$$\begin{bmatrix} \Xi(u) & P & E & 0 \\ P^T & -\Xi(v) & 0 & F \\ E^T & 0 & 0 & 0 \\ 0 & F^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \\ \Delta \mu \\ \Delta \nu \end{bmatrix} = -g(u, v, \mu, \nu), \quad \Xi(u)_{ab} = \begin{cases} \frac{\lambda_{\rho_a} + \sum_{h' \in \mathcal{C}(a)} \lambda_{h'}}{u(a)}, & a = b \\ -\frac{\lambda_{\rho_a}}{u(b)}, & \textbf{Par}(a) = b \\ -\frac{\lambda_{\rho_b}}{u(a)}, & \textbf{Par}(b) = a \end{cases}, \tag{3.18}$$

where $g(u, v, \mu, \nu)$ contains terms in (3.17) and $\Xi(v)$ is defined analogously in terms of the appropriate v and λ's. Observe that $\Xi(u)$ and $\Xi(v)$ are diagonally dominant and symmetric, implying that they are positive definite. In the backward pass, we require the gradients of the loss L with respect to P and λ. Similar to prior work [6, 75, 121], this may be done by applying the implicit function theorem or by simply manipulating differentials. This yields the gradients

$$\nabla_P L = y_u v^T + u y_v^T, \tag{3.19}$$

$$\nabla_{\lambda_h} L = \begin{cases} \kappa_h^T y_u & \forall h \in \mathcal{I}_1 \\ -K_h^T y_v & \forall h \in \mathcal{I}_2 \end{cases} \tag{3.20}$$

where

$$[\kappa_h](\sigma_1) = \begin{cases} 1 + \log\left(u(\sigma_1)/u(\textbf{Par}(\sigma_1))\right), & \textbf{Inf}(\sigma_1) = h \\ -1, & h \in \mathcal{C}(\sigma_1) \end{cases}, \tag{3.21}$$

$$[K_h](\sigma_2) = \begin{cases} 1 + \log\left(v(\sigma_2)/v(\textbf{Par}(\sigma_2))\right), & \textbf{Inf}(\sigma_2) = h \\ -1, & h \in \mathcal{C}(\sigma_2) \end{cases} \tag{3.22}$$

$$\text{and,} \begin{bmatrix} y_u \\ y_v \\ y_\mu \\ y_\nu \end{bmatrix} = \begin{bmatrix} -\Xi(u) & P & E & 0 \\ P^T & \Xi(v) & 0 & F \\ E^T & 0 & 0 & 0 \\ 0 & F^T & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\nabla_u L \\ -\nabla_v L \\ 0 \\ 0 \end{bmatrix}. \tag{3.23}$$

3.4 Efficiently Performing Forward and Backward Passes

Each gradient step involves solving optimization problems. Thus, having efficient solvers is crucial in scaling up. We focus on the two most expensive steps—the forward and backward passes. For the forward pass, we have so far used Newton's method, which requires solving linear equations such as those in (3.18) in each Newton step. When the game tree is large, solving (3.18) multiple times for a single gradient step dramatically slows down training. Similarly for the backward pass, one needs to solve a single linear system shown in (3.7), (3.15), (3.23). Again, when the game is large, naively solving the linear system is also prohibitively slow, even when utilizing sparse matrices. In this section, we show how first-order iterative methods (FOM), which do not require the solution of a linear system as a subroutine can be used to speed up computation in both the forward and backward passes. FOMs are also computationally attractive for solving extensive form games—unlike Newton's method, FOMs easily exploit the underlying tree structures in EFGs.

3.4.1 A Useful Subproblem

For both the forward and backward problems, the following regularized bilinear saddle point problems of the following form is of interest:

$$\min_{Ex=x_0} \max_{Fy=y_0} x^T P y + \mathcal{E}(x) - \mathcal{F}(y), \tag{3.24}$$

where $\mathcal{E}(x)$ and $\mathcal{F}(y)$ are strictly convex functions and P is a sequence form payoff matrix. Note that x and y are "almost" sequence form strategies — the primary difference is that we have replaced e and f by general vectors x_0 and y_0 and no longer have non-negativity constraints. It is obvious from (3.16) that the forward pass in our problem solves a problem in this form (the dilated entropy regularization also serves as a natural barrier for the domain of x and y). As we will show later, the linear system in the backward pass (3.23) can also be seen as solving a problem of this form. Hence, solving (3.24) efficiently will make each gradient step much faster.

3.4.2 FOM for Solving Regularized Bilinear Saddle Point Problems

Many methods to efficiently solve problems of the form (3.24) have been proposed. In our work, we adapt the method proposed by Chambolle and Pock [44][4], which alongside many other first order methods, apply *best response* subroutines towards smoothed versions of the min or max original problem taken in isolation. Their algorithm proceeds by alternating between regularized best-responses to minimization and maximization. Their algorithm, shown in Algorithm 1 gives the high-level overview of the optimization procedure, where BR are smoothed best responses with appropriately chosen Bregman divergences D_1, D_2 , their associated convex functions Ψ_1, Ψ_2, and step sizes $\tau_1, \tau_2 > 0$.

$$\mathsf{BR}_1(\bar{x}, \tilde{y}; P, \tau_1) = \arg\min_{Ex=x_0} x^T P \tilde{y} + \mathcal{E}(x) + \frac{1}{\tau} D_x(x, \bar{x})$$

[4]Their algorithm is more general and applies beyond game solving.

$$\mathsf{BR}_2(\bar{y}, \tilde{x}; P, \tau_2) = \operatorname*{arg\,min}_{Fy=y_0} -\tilde{x}^T P y + \mathcal{F}(y) + \frac{1}{\sigma} D_y(y, \bar{y}).$$

We first set $\tau_1 = \tau_2$ for convenience. For Algorithm 1 to be practical, we will require the best response oracles $\mathsf{BR}_1, \mathsf{BR}_2$ to be computed efficiently. By setting $D_1(x, \bar{x})$ and $D_2(y, \bar{y})$ to be of specific form similar to $\mathcal{E}(x)$ and $\mathcal{F}(y)$ respectively will simplify BR_1 and BR_2 to be (up to a factor) of the forms $\arg\min_{x:Ex=x_0} x^T c_1 + \mathcal{E}(x)$ and $\arg\min_{y:Fy=y_0} y^T c_2 + \mathcal{F}(y)$, for some c_1, c_2 as functions of $\bar{x}, \tilde{y}$ (or $\bar{y}, \tilde{x}$ respectively). These smoothed best responses are then efficiently computed by exploiting the tree-structure in extensive form games inherent in the $Ex = x_0$ and $Fy = y_0$ constraints. The remainder of this section outlines in turn the procedures required for both forward and backward passes.

Algorithm 1: FOM method of Chambolle and Pock [44] for solving (3.24)

```
**Input:** $x^{(0)}, y^{(0)}, P, \tau, \sigma$
**for** $i$ *in* $\{0, \dots\}$ **do**
    $\tilde{y} = y^{(i)}$;
    $x^{(i+1)} = \mathsf{BR}_1(x^{(i)}, \tilde{y}; P, \tau)$;
    $\tilde{x} = 2x^{(i+1)} - x^{(i)}$;
    $y^{(i+1)} = \mathsf{BR}_2(y^{(i)}, \tilde{x}; P, \sigma)$;
**end**
```

3.4.3 Efficient Forward Passes using FOM

For this section, the $u = x, v = y$ when referring Algorithm 1. Setting $\mathcal{E}(u), \mathcal{F}(v)$ to be the entropy terms in (3.16) and $u_0 = e, v_0 = f$ gives the expression in the form of (3.24). The natural divergence to be chosen is the standard entropy divergence adapted to the dilated setting (dropping terms in D_u which do not contain u).

$$\begin{aligned}
\Psi_1(u) = \mathcal{E}(u) &= \sum_{h \in \mathcal{I}_1} \lambda_h \sum_{a \in \mathcal{A}(h)} u(ha) \log \frac{u(ha)}{u(\textbf{Seq}(h))} \\
D_1(u, \bar{u}) &= \Psi_1(u) - u^T \Psi_1'(\bar{u}) \\
[\Psi_1'(\bar{u})](\sigma_1) &= \lambda_{\textbf{Inf}(\sigma_1)} - \sum_{h' \in \mathcal{C}(\sigma_1)} \lambda_{h'} + \lambda_{\textbf{Inf}(\sigma_1)} \log \frac{\bar{u}(\sigma_1)}{\bar{u}(\textbf{Par}(\sigma_1))}
\end{aligned}$$

where a similar expression holds for $D_v(v, \bar{v})$. Plugging into the expression for BR_1 gives

$$\mathsf{BR}_1(\bar{u}, \tilde{v}) = \operatorname*{arg\,min}_{Eu=e} \frac{\tau}{1+\tau} u^T \left(P\tilde{v} + \Psi_1'(\bar{u})\right) + \mathcal{E}(u).$$

It is known that, $\mathsf{BR}_1(\bar{u}, \tilde{v})$ may be solved by a bottom-up and top-down traversal of the treeplex, followed by a single sparse matrix-vector multiplication [88]. At each information set , we solve for the best response in behavioral form, i.e., assuming that information set was the root. Each of these sub-problems may be expressed in closed form using log-sum-exp and softmax functions. The sequence form is recovered from behavioral strategies with a single downwards traversal of the tree. Details can be found in Section 3.6.3.

Remark 11: In recent years, there have been many new methods for solving general min-max convex-concave problems, which includes the regularized bilinear saddle point problems in our setting. Naturally, algorithms tailored for treeplexes would be more efficient [53, 62]. We note that since publication, even more efficient methods to solve for QRE such as *regret circuits* [62] have been proposed. This method appears to be more efficient than ours, however, it is noteworthy that it relies on Theorem 1 and dilated entropy regularization.

3.4.4 Efficient Backward Passes using FOM

Recall that the backward pass also requires solving a linear system to obtain $[y_u \;\; y_v \;\; y_\mu \;\; y_\nu]$. We first begin by making the crucial observation that the solution to the (necessary and sufficient) KKT conditions of the following optimization problem is precisely the solution to the linear system in (3.23).

$$\begin{aligned} \min_x \max_y \quad & x^T P y + \frac{1}{2} x^T \Xi(u) x - \frac{1}{2} y^T \Xi(v) y + \nabla_u L^T x + \nabla_v L^T y \\ \text{such that} \quad & Ex = 0 \qquad Fy = 0. \end{aligned} \tag{3.25}$$

Note that in the backward pass, u and v are constants, while we are optimizing over x, y, which in turn are *not* probabilities (they are unbounded and do not satisfy sequence form constraints).

We show that (3.25) can be solved efficiently—more efficiently than a system of linear equations. This dramatically speeds up the backward pass. First, observe that since $\Xi(u)$ and $\Xi(v)$ are positive definite, this is a convex-concave problem of the form required by Algorithm 1. We select the natural distance generating function $\Psi_1 = \frac{1}{2} x^T \Xi(u) x$ which yields (ignoring terms containing only $\bar{x}$),

$$D_1(x, \bar{x}) = \frac{1}{2} x^T \Xi(u) x - x^T \Xi(u) \bar{x}$$

Plugging this into the expression for $\mathsf{BR}_1(\bar{x}, \tilde{y})$ and rearranging gives

$$\operatorname*{arg\,min}_{Ex=0} \frac{\tau}{1+\tau} x^T (\nabla_u L + P\tilde{y} - \frac{1}{\tau} \Xi(u) \bar{x}) + \frac{1}{2} x^T \Xi(u) x \tag{3.26}$$

Letting $c = \frac{\tau}{1+\tau} \left(\nabla_u L + P\tilde{y} - \frac{1}{\tau} \Xi(u) \bar{x} \right)$ in (3.26) gives the KKT conditions

$$c + E^T \gamma + \Xi(u) x = 0, \qquad Ex = 0$$

where γ are Lagrange multipliers. Multiplying by $E\Xi^{-1}(u)$ gives a linear system in γ

$$E\Xi^{-1}(u) c + E\Xi^{-1}(u) E^T \gamma = 0. \tag{3.27}$$

Note that we should not have introduced new roots in doing so, since these are linear systems and there is a unique solution to γ both before and after the multiplication. After solving for γ, one may solve for x

$$x = \Xi^{-1}(u) \left(-c - E^T \gamma \right). \tag{3.28}$$

Section	Setting	Purpose
3.5.1	Rock, Paper, Scissors with side info.	Learning matrix games with side information
3.5.2	One-Card Poker	Learning in EFGs
3.5.3	Security Resource Allocation	Learning with incomplete observations
3.5.4	Synthetic Multistage Game	Scalability of FOM
3.5.5	Larger One-Card Poker	Scalability of FOM, Learning NLQRE
3.5.6	Information Gathering Game	Learning nested logits

Table 3.1: Summary of experiments performed for Chapter 3.

Theorem 2. *Solving for γ and x in Equations* (3.27) *and* (3.28) *require linear time (in the size of the game tree).*

The derivation involves exploiting the tree-structure inherent in extensive form games. Details may be found in Section 3.6.3. The entire derivation can be similarly performed to efficiently compute BR_2.

3.5 Experiments

We empirically demonstrate our module's novel aspects – learning *extensive form games* in the presence of *side information*, with *partial observations*. We also evaluate the ability of using FOM in our effforts to scale up, as well as learning NLQRE. Due to space constraints, details such as hyperparameters and experimental setups are omitted. They may be found in Appendix 3.5.7 or the full papers [121, 122]. Table 3.1 contains a list of the experiments conducted.

3.5.1 Learning Payoffs in Rock, Paper, Scissors with Side Information

Rock Paper Scissors (RPS) is among the most well-studied 2-player zero-sum game. It is well known that playing uniformly is an NE and QRE for RPS. In this experiment, we consider the following variant (Figure 3.2), which breaks symmetry between the 3 actions. Notice that the traditional RPS is recovered when b_1, b_2, b_3 are all 1.

We assume that each of the b's is a linear function of some features $x \in \mathbb{R}^2$, i.e., $b_y = x^T w_y, y \in \{1, 2, 3\}$, where $w_y \in \mathbb{R}^2$ are to be learned. Features x in the dataset and ground-truth weights w are drawn uniformly from $[0, 1]$, and $[0, 10]$ respectively. Experiments were evaluated with a fixed test set of size 2000. The results presented in Figure 3.3 illustrate 2 key points. The first plot shows that accuracy dramatically improves with larger datasets, both in terms of MSE of learned parameters and predicted strategies. The second plot shows that with a reasonably sized dataset, convergence is stable and is fairly quick. As expected, the quality of learned parameters improves as the number of data points increases. We also observe that predicted strategies improve significantly when going from 2000 to 5000 samples, showing that despite not converging to better parameters, the network still demonstrates a marked improvement in predicting player strategies.

	R	P	S
R	0	$-b_1$	b_2
P	b_1	0	$-b_3$
S	$-b_2$	b_3	0

Figure 3.2: Payoff matrix of modified Rock-Paper-Scissors. Values shown are for the row player.

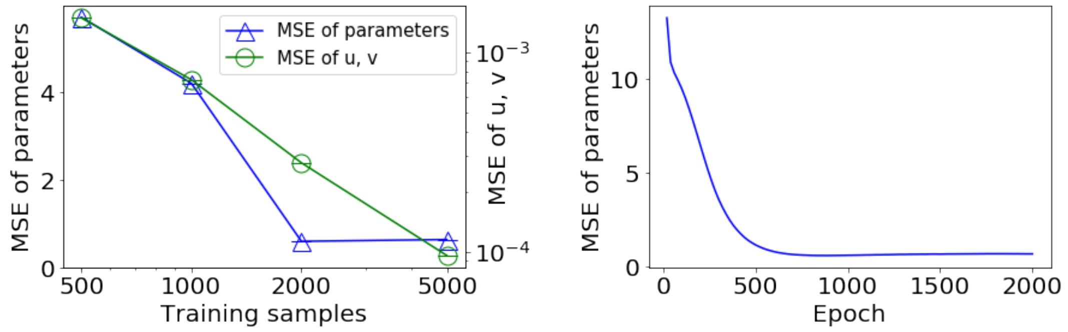


Figure 3.3: Results for Rock, Paper, Scissors. Left: MSE of parameters and predicted mixed strategies. Right: Convergence over 2000 epochs, using a dataset of size 2000.

3.5.2 Learning Card Distributions in One-Card Poker

We consider a simple variant of Kuhn poker with n cards. Here, players are dealt a single card labelled from 1 to n. Players have bets of 10, and there are two stages of betting for the first player. Both players begin with an ante of 10. Player 1 decides whether to bet an additional 10, followed by Player 2 (who folds if he does not call). Lastly, Player 1 may choose to bet if Player 2 raises. Both players are obliged to reveal their card at the end of each game.

While relatively simple, the game contains the key elements in extensive-form games and the strategic concepts in poker such as slow playing (e.g. not betting even if Player 1 holds the high card) and bluffing (e.g. betting even if the player holds a low card). Despite its simple structure, the game with n cards has 2^{2n} normal form pure strategies for each player. However, in sequence form, we only need to work with realization plans of size $4n$.

We assume that the deck is stacked non-uniformly. Our goal is to learn this distribution of cards after observing many rounds of play. Let $d \in \mathbb{R}^n, d \geqslant 0, \sum_i d_i = 1$ be the *weights* of cards. The probability that the players are dealt cards (i, j) $(i \neq j)$ is $d_i \times \frac{d_j}{1-d_i}$. Note that this distribution is asymmetric between players.

Instantiating the sequence form constraint matrices . We have the following possible instantiation of the sequence form constraints when $n = 4$.

$$E = \begin{bmatrix} I & I & 0 & 0 \\ -I & 0 & I & I \end{bmatrix}, F = \begin{bmatrix} I & 0 & I & 0 \\ 0 & I & 0 & I \end{bmatrix}, e = [11110000]^T, f = [11111111]^T,$$

where $E, F \in \mathbb{R}^{2n \times 4n}$. (For simplicity we have pushed the "empty sequence $= 1$" constraint into the leading infosets.) When the card distribution is uniform, i.e. $d = [0.25, 0.25, 0.25, 0.25]$ we define the sequence form payoff matrix P in terms of the *showdown* and *resign* matrices

$$S = \frac{1}{12}\begin{bmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix}, R = \frac{1}{12}\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, P = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & 0 & -R & 2S \\ 0 & R & 0 & 0 \\ 0 & 2S & 0 & 0 \end{bmatrix}$$

where by our convention, the row player is the minimizing player. Each block in P is of size 4×4, and gives possible outcomes for each of the 4^2 possible ways of dealing cards. The 4 blocks for the row player correspond (in order) to the actions 'fold on first move', 'raise on first move', 'fold on second move', 'raise on second move'. For the column player, the 16 actions are 'fold after first player folded', 'raise after first player folded', 'fold after first player raised', 'raise after first player raised'. The normalizing factor $1/12$ arises from the fact there are 12 possible ways of distributing cards to players. When d is not uniform, the sequence form payoff matrix can be written in terms of the $n \times n$ distribution matrix D,

$$P = \begin{bmatrix} D \circ S & 0 & 0 & 0 \\ 0 & 0 & -1 \circ D & 2D \circ S \\ 0 & 1 \circ D & 0 & 0 \\ 0 & 2D \circ S & 0 & 0 \end{bmatrix}, \quad \text{where } D_{ij} = \frac{d_i d_j}{1 - d_i},$$

i.e. every 4×4 block is pointwise weighted by the chance of being dealt the relevant cards. It may be seen that the resign matrix R is really just the all-ones matrix multiplied pointwise by D.

Remark 12: While counting cards seems to be a straightforward way to learn the card distribution when d does not change over time, our method is suited to learn the player's *perceived* or *believed* distribution of cards, which may be different from the distribution of cards dealt. This may even be a function of contextual features such as demographics of players.

A total of three experiments were run with $n = 4$. For each experiment, $d \sim \text{Dir}(1, 1, 1, 1)$, that is, the card distribution is sampled from a Dirichlet distribution. Each experiment comprises 5 runs of training, with same weights but different training sets. Training was for 2500 epochs, which was observed to be after convergence. The mean squared error of learned parameters are averaged over all runs and are presented in Figure 3.4.

3.5.3 Learning Target Payoffs Security Resource Allocation Game

In this set of experiments, we demonstrate the ability to learn from *incomplete observations* in a setting that abstracts attacks in a cybersecurity domain. The defender possesses k indistinguishable and indivisible defensive resources, e.g., cyber analysts, which he splits among n targets, $\{T_1, ..., T_n\}$. In an attacking attempt, the attacker (row player) chooses one target. In the event an attack on T_i succeeds, the attacker obtains a reward of R_i (and the defender $-R_i$), otherwise, the payoffs to both parties are 0. Each defensive resource independently prevents an intrusion with probability 0.5. For example, if there are two defenders guarding T_1, the chance of a successful attack on T_1 is $\frac{1}{2^2}$. This creates a scenario where the marginal benefit of each defensive resources decreases, thus requiring the defender to strike a balance. The matrix of expected payoffs when $n = 2, k = 3$ is shown in Figure 3.5.

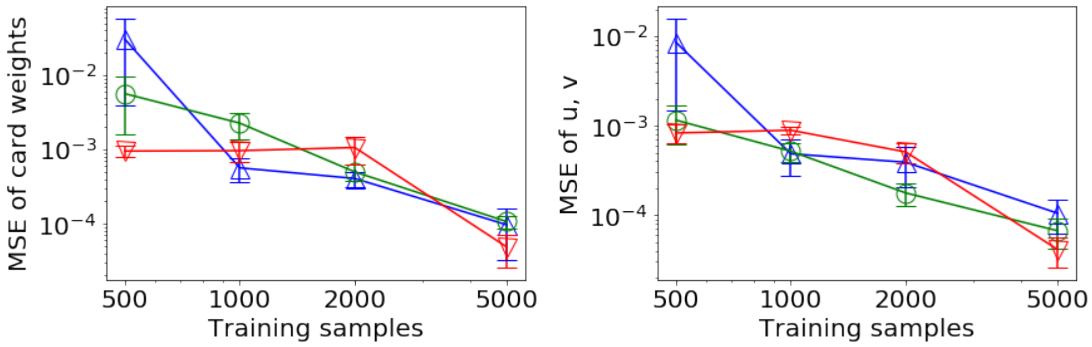


Figure 3.4: MSE of card weights (left) and realization plans (right). Each colored plot illustrates a different ground truth to be learned.

	{0, 3}	{1, 2}	{2, 1}	{3, 0}
T_1	$-R_1$	$-\frac{1}{2}R_1$	$-\frac{1}{4}R_1$	$-\frac{1}{8}R_1$
T_2	$-\frac{1}{8}R_2$	$-\frac{1}{4}R_2$	$-\frac{1}{2}R_2$	$-R_2$

Figure 3.5: Security Resource Allocation Game: Payoff matrix with $n = 2, k = 3$.

We also consider a multi-stage game where the attacker can launch t attacks, one in each stage while the defender chooses his allocation of resources in stage 1 and cannot change it in later stages. On the other hand, the attacker has the *option* of changing his target between stages. This describes a setting where analysts are deployed to specific network assets on a daily basis, while attackers are sufficiently nimble to make multiple attacks in a single day. To understand why the attacker may change target, consider that target T_i is attacked in stage 1 and the attack is unsuccessful. It may be inferred that it is more likely that T_i is better guarded, prompting the attacker to switch targets.

Three experiments are run with $n = 2, k = 5$ for games with single attack and double attack, i.e, $t = 1$ and $t = 2$. Crucially, in this set of experiments, we learn R_i only based on observations of the defender's actions. This setting yields a 10×6 sequence form payoff matrix. For each experiment, R_1 and R_2 are drawn uniformly in $[0, 2]$. Each experiment is run 10 times for at least 2000 epochs per run. The mean and standard error over each run is presented in Figure 3.6. The results show that our algorithm can still recover the game setting by only observing defender's actions.

3.5.4 Scaling up in Synthetic Multistage Games

Here we use randomly generated extensive form games to illustrate the computational efficiency of our proposed first order method compared to Newton's method. We evaluate the solvers for the forward and backward passes in isolation. The experiments are run over several depths d, where normal form games have $d = 1$. For $d > 1$, players play d distinct simultaneous sub-games in succession, where each simultaneous sub-game has $\hat{n}$ actions and transitions to the next sub-game is governed by the joint action by both players, i.e., the size of P will be exponential in d. The payoff matrices P were generated with each non-zero entry uniformly chosen from

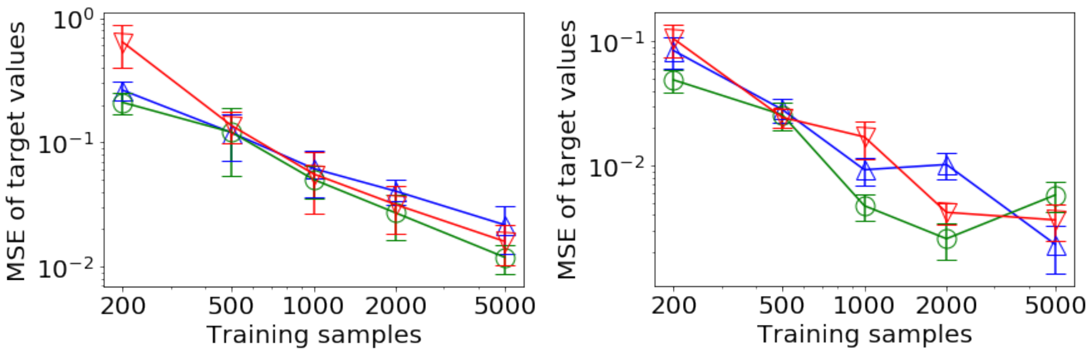


Figure 3.6: Security Resource Allocation Game: MSE of target values over 3 runs. Left: $t = 1$, Right: $t = 2$. Each colored plot illustrates a different ground truth to be learned.

$[-10, 10]$, and rationality parameter λ for *each* information set uniformly and independently chosen between $[0.9, 1.1]$.

Evaluation of Forward Passes In the forward pass, we compared the baseline Newton solver to our proposed first-order method. However, the termination criterion for the 2 methods are non-identical; as Newton's method minimizes the residual rather than duality gap. To strike a fair comparison, we evaluated the 2 methods by first running Newton's method till a residual of less than 10^{-3} is achieved. The duality gap of that solution is computed and subsequently used as the termination criterion for the FOM[5]. The timings and speedup are averaged over 50 trials and presented in Figure 3.7.

Evaluation of Backward Passes In the backward pass, the comparison for our proposed FOM is against solving the linear system in (3.23) directly. In the loss function, we will concern ourselves with the setting where the true matrix P and λ parameters are used in computing u, v for the forward pass. This corresponds to the case the model is already fairly well trained. The results over 50 trials are presented in Figure 3.8.

It is clear from both figures that our method scales much better than Newton's method for randomly generated matrices. Speedups of more than an order of magnitude are fairly common, and the improvement increases with problem size. Furthermore, it was also observed that our method consumed far less memory than sparse solvers. In fact, solving the sparse system when $d = 3, \hat{n} = 17$ (not plotted) required more than 10GB of memory. On the other hand, our FOM was able to solve such instances in less than a minute, and with no noticeable increase in memory usage. Note that P contains more than 1.4 million rows and columns in this setting.

3.5.5 Learning Rationality Parameters and Scaling up in One-Card Poker

Here, we operate in a slightly different setting. Instead of trying to learn underlying card distributions, we learn player rationality parameters. We assume that player rationality is independent

[5]On occasion, the Newton solver gave a gap extremely close to numerical precision. In these cases, we apply to a termination criterion of 10^{-12}.

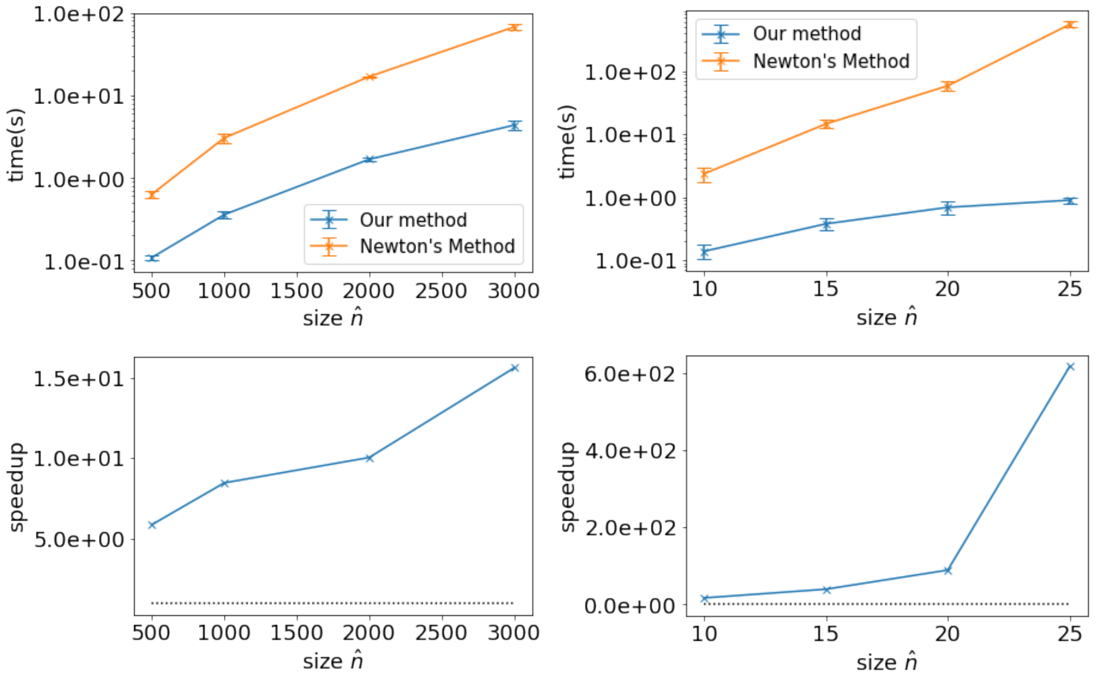


Figure 3.7: Timings (top) and speedup (bottom) for forward passes for $d = 1$ (left) and 2 (right). Error bars represent 1 standard deviation.

of the cards being drawn, and only depends on the past actions of (both) players. In this setting, there are just 4 parameters to be learned (independent of the size of the deck). To demonstrate the ability of FOM in scaling up, we use a deck size of 200 as opposed to just 4 in Section 3.5.2

We generate our data assuming that player rationality is some linear function of a scalar feature, i.e., there are 4 weights to be learned. The weight vector is drawn uniformly from $[0, 0.01]$. Feature vectors are drawn between $[0, 1]$. Our model is $\lambda_h = w_i \times f + \epsilon_\lambda$. The addition of a small ϵ_λ ensures that the λ's will always remain positive; in our experiments, $\epsilon_\lambda = 0.001$. For each feature, we compute the λ's and find its corresponding equilibrium from which we sample player actions. The results are plotted in Figure 3.9.

In all 3 cases, the log-loss is close to optimal given around 30 epochs. As expected, our exact solver exhibits behavior almost identical to that of Newton's method on a per-epoch basis. Our solver is significantly faster than the baseline. It was observed that at almost all stages of training, Newton's method took almost 2 orders of magnitude longer to learn a model of similar performance. In fact, a single epoch using Newton's method takes as much time as training the entire model using our solver.

3.5.6 Learning Nested Logit Models in the Information Gathering Game

Here we demonstrate the applicability of the nested logit model (i.e., a one player game) using a publicly available dataset [89]. The game proceeds as follows. Suppose there are 4 faced-down cards ranging from 1-10 placed in a 2×2 matrix (with potential repetitions). The goal of the game is to select the row containing cards with the largest sum. The game proceeds in 4 stages.

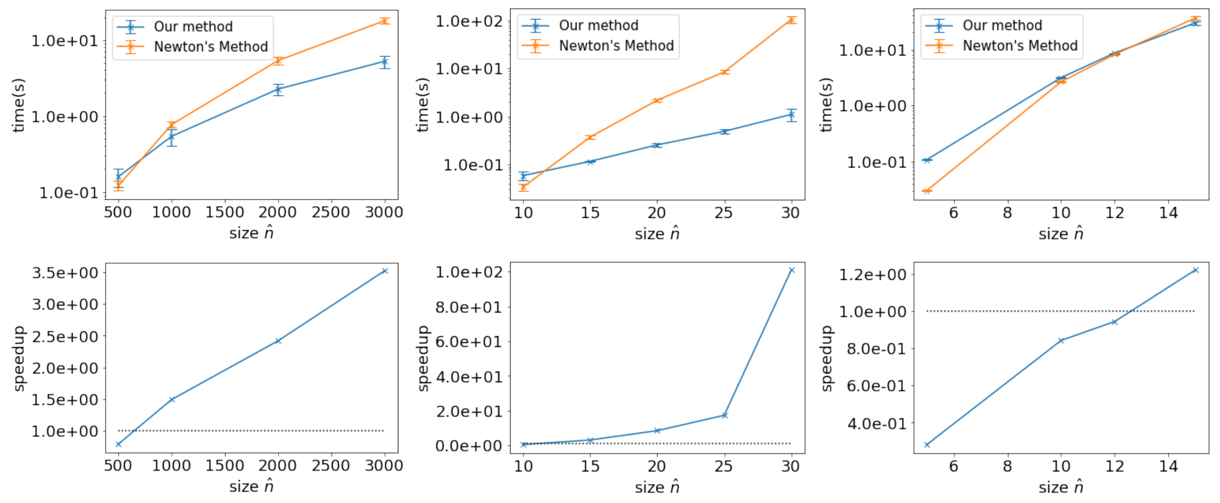


Figure 3.8: Timings (top) and speedup (bottom) for backward passes. From left to right, $d = 1, 2, 3$. Error bars represent 1 standard deviation.

At each stage of the game, the player may make a guess prematurely, or spend some points in revealing a new card. At the fourth and final stage, the player has to make a guess. The player obtains a reward of 60 and -50 points for correct and incorrect guesses, and may only guess once. The challenge is for the player to judge if it is worth paying to gather more information. Computationally, the optimal policy may be easily obtained using dynamic programming.

We model bounded rationality using the nested logit model. It is assumed that the level of rationality should be a function of a) how many cards are already open, and b) side information such as one's educational qualifications. This leads to a natural description of the game with 4 different λ's, each of which is some function of features, which we describe below.

Two models are trained for this experiment. NOFEAT refers to the case when there are no features (i.e., we are simply learning λ) and FEAT when we are exploiting demographic information. In this case, features comprise the player's academic qualifications and age, both with one-hot encodings. A player's age is split into 8 age ranges, and education levels follow that of the UK (i.e., GCSE, A levels, Undergraduate, Graduate). Our model employs a neural network with 3 hidden layers of width 100, interspersed by rectified linear activations. To ensure λ's are positive, all inputs were exponentiated before being fed into the solver. Figure 3.10 shows the log loss over the overall game as well as the loss at each individual stage. For comparison, we also provide the results for a player who picks a random action at every stage of the game. The learned λ parameters for each configuration of features is presented in Figure 3.11.

From Figure 3.10, we can see that both trained models significantly outperform UNIFORM. Log losses at each stage appear to decrease with the stage number. This is unsurprising since players behave more rationally (and hence predictably) as more information is revealed. However, our model appears to perform worse at stage 4, which is in fact a problem with full observability. We suspect this higher loss is a consequence of our model overfitting to be overly confident at the final stage, incurring a huge loss on the rare occasion a player answers incorrectly. Several trends are observed from Figure 3.11. First, notice that λ decrease by approximately a

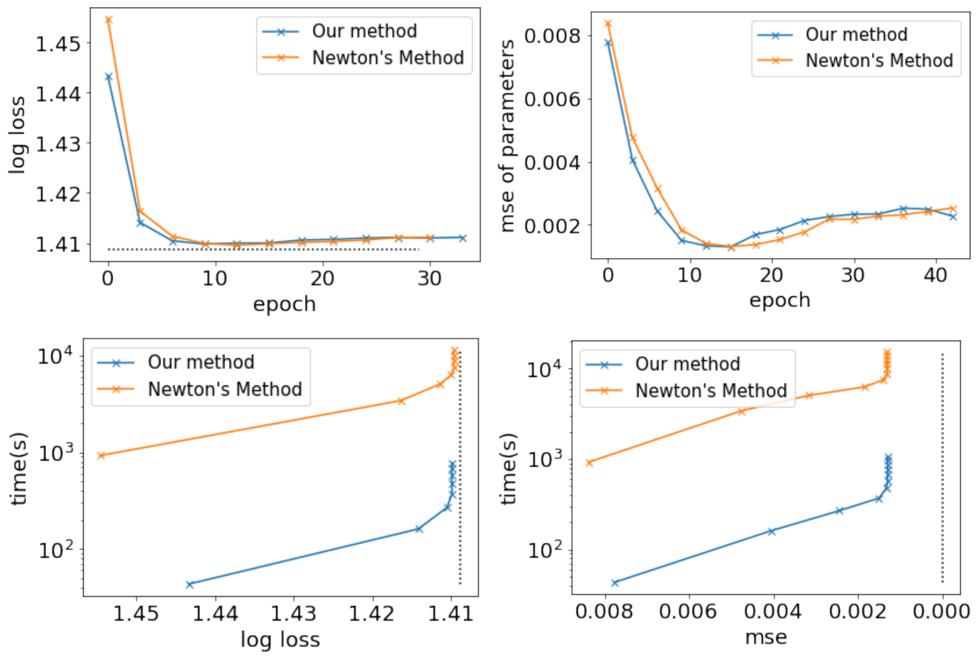


Figure 3.9: Results for Larger One-Card Poker. Top: log-loss and mse per epoch, Bottom: time required to obtain a particular log loss or mse.

	loss	loss(1)	loss(2)	loss(3)	loss(4)
UNIFORM	1.833	1.099	1.099	1.099	0.693
NOFEAT	1.422	0.878	**0.863**	0.826	**0.130**
FEAT	**1.419**	**0.874**	0.866	**0.818**	0.145

Figure 3.10: Log losses for the information gathering dataset. The first column shows losses over the whole game, other columns show losses for individual stages.

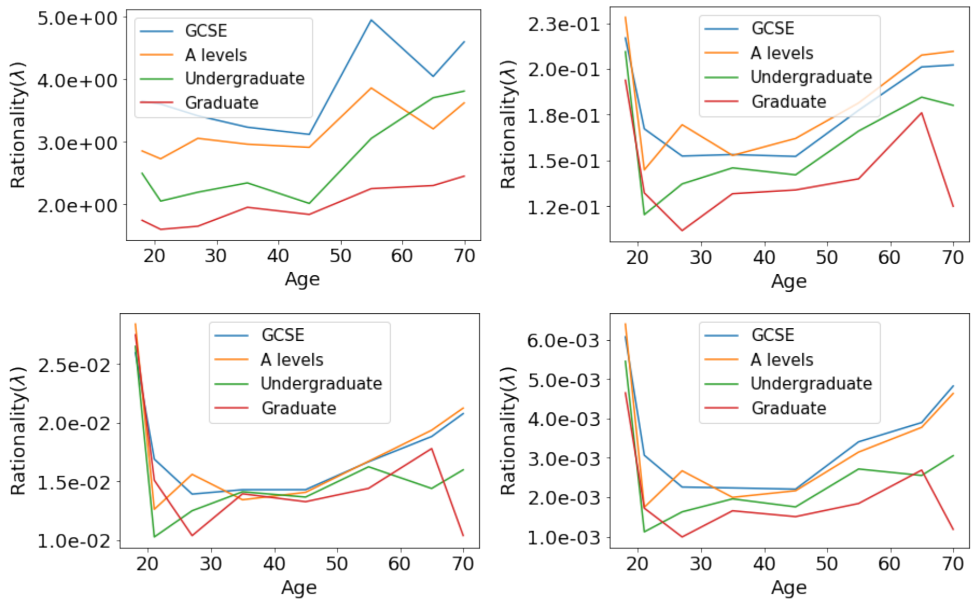


Figure 3.11: Rationality parameters λ for each stage of the game as a function of age and education (row major ordering). Ages are binned within age ranges. All respondents below the age of 18 were grouped together in a single age range.

factor of 10 between stages. This is fairly expected, since each information gathering leads to 10 other potential states. Also unsurprisingly, better educated respondents exhibit more rationality (recall a lower λ implies a more rational player). Interestingly, we can see a U-shaped trend in all stages, suggesting that people in the mid twenties and thirties are most rational. Both these observations agree with the findings by [89], where it was shown that higher educated and middle aged respondents obtained the most reward.

3.5.7 Implementation Details

All of the experiments are run using CPU cycles. The proposed first order method was implemented using Cython[15]. We chose to do so since the best-response subroutines require tree-traversals which are expensive in Python, while the second order method used the Numpy[81] and Scipy[175] libraries for the solution of linear systems. Where possible, we utilized the Scipy sparse matrix library. This was seen to provide a significant speedup for sparse P for both our method and Newton's method. The PyTorch automatic differentiation library [144] was used to automatically obtain gradients for components outside the game solving module.

1. **Rock, paper, scissors.** Experiments are run on an Amazon c4.2xlarge EC2 instance. The learning rate is 0.01, batch size of 128, using the Adam optimizer. The maximum number of epochs before termination is 1000.
2. **One card poker.** Experiments are run on an Amazon c4.2xlarge EC2 instance. The learning rate is 0.002, batch size of 128, using the RMSProp optimizer [87]. Weights are initialized to be uniform. The maximum number of epochs is 2500, although convergence

occurs significantly faster. Since there are no side-features, experiments may be sped up by computing the forward pass just once for each minibatch.

3. **Security resource allocation game.** The case where $t = 1$ was run on an Amazon c4.2xlarge EC2 instance. The experiment where $t = 2$ was performed on a 3.1 GHz Intel Core i5 with 16GB of RAM. In theory, this difference in environment should only affect wall-clock time, which is not the focus of this particular experiment.
4. **Synthetic multistage games.** All timings presented are wall-clock timings when run on identical Amazon EC2 instances. We set $\tau = 0.1$ when evaluating FOM.
5. **Scaling and learning rationality parameters in one card poker.** The training set of size 2000, with an independent test set of size 1000. We minimized the log loss using the Adam optimizer with a batch size of 64 and learning rate of 10^{-4}. We compared our solver against Newton's method, which terminates at a residual of 10^{-8}. We fixed $\tau = 1$ for the forward solver and $\tau = 0.1$ for the backward solver.

For experiments (1)-(3), the learning rate is 0.002 using the RMSProp optimizer (all other hyperparameters are left as the defaults in Pytorch). Each run was 5000 epochs. The weights are passed through $f(x) = (\tanh(x) + 1)$ to clip rewards to between $[0, 2]$ before parameterizing the payoff matrix.

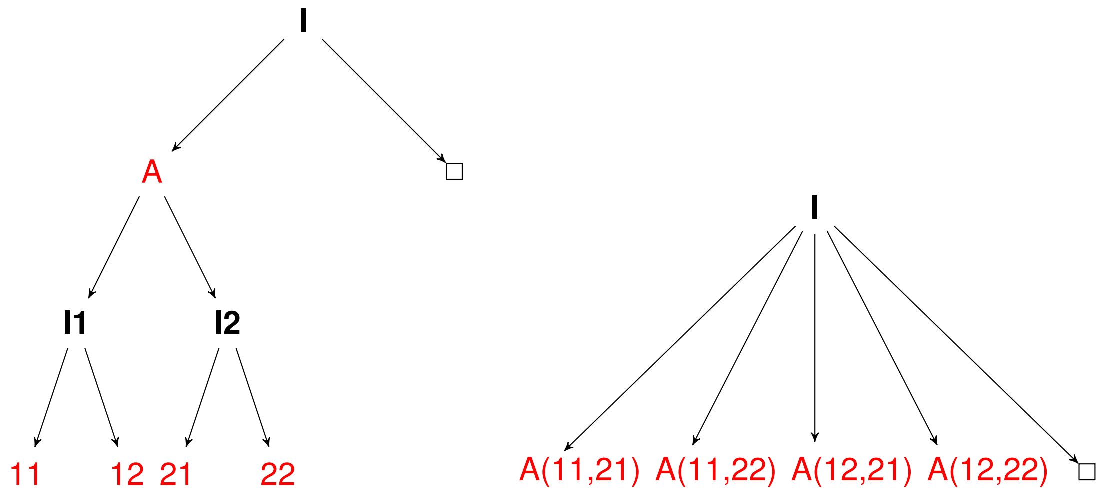
3.6 Technical Proofs and Experimental Details

In this section, we describe the proofs for Theorems 1 and 2. We also derive the representation of NLQRE as a nested-logit.

3.6.1 Proof of Theorem 1 (Dilated Entropy to Reduced Normal Form)

We first present the proof of one of our main technical results, that the dilated entropy regularization of a extensive form game is equivalent to the standard entropy-regularized QRE of the equivalent reduced normal-form game (Section 2.3.2).

Outline Consider the max player's treeplex shown in Figure 3.12a. Red nodes are vertices and black nodes are information sets. White squares represent sibling actions of A, which may in turn, contain other parts of the game tree. The information states associated with I_1, I_2 are parallel information states, i.e. both of which may be reached from A with non-zero probability (assuming suitable opponent/chance actions). We show that the optimal solution using dilated entropy regularization on the game in Figure 3.12a is realization equivalent to the optimum of the reduced game in Figure 3.12b. Essentially, this operation converts *part* of the sequences to the normal form, by replacing parts of the tree with all possibly contingencies. Repeated application of this operation eventually translates the sequence-form representation to the reduced normal form. This is performed on the tree in a bottom-up manner while maintaining realization equivalence at each stage.



(a) Original treeplex. (b) Treeplex after a single iteration.

Figure 3.12: Treeplexes before and after one iteration of a transformation which preserves realization equivalence under the QRE. Repeatedly performing this operation eventually yields a flat treeplex where the actions remaining correspond to those in the reduced normal form.

Part 1: A Simple Case

Let A be the an action such that all immediate child *actions* are leaves (e.g. Figure 3.12a). We condition on having taken the action A. We will first show that after conditioning the resultant reduced normal form is equivalent to its sequence form.

Notation. Recall $\mathcal{C}_A = \{I_k\}$ denotes the children *information sets* of A. The action set at I_k is denoted by $\mathcal{A}_k = \{A_{kj}\} = \{a \in \mathcal{A}_v | \rho_a = I_k\}$. For the purposes of this derivation, the opponent's strategy is fixed, and his strategy profile is combined to give P_1's payoff vector $P^T u = x$.

In normal form, all contingencies are accounted for, and the set of pure strategies are given by the Cartesian product of action sets $\hat{\mathcal{A}}_A = \prod_k \mathcal{A}_k$. We denote the normal form mixed strategy profile, payoff matrix, and normal form payoff vector as $\hat{v}$, $\hat{P}$ and $\hat{x} = \hat{P}^T \hat{u}$. Note that the sizes of these vectors/matrices are not equivalent to their counterparts in sequence form. For convenience, we define $f(\hat{v})$ to be a function mapping $\hat{v}$ to v by performing the appropriate marginalization. Let $g(v, j)$ be the distribution of actions extracted from v supposing an information state of I_j. For simplicity let H now define the Shannon entropy, $H(y) = \sum y_i \log(y_i)$.

Derivation. The resultant QRE in the reduced normal form is,

$$\begin{aligned}
&\max_{\hat{v}} \hat{v}^T \hat{x} + H(\hat{v}) \\
=&\max_{\hat{v}} f(\hat{v})^T x + H(\hat{v}) \\
=&\max_{\hat{v}} f(\hat{v})^T x + H\left(\bigotimes_j g(f(\hat{v}), j)\right) \\
=&\max_{\hat{v}} f(\hat{v})^T x + \sum_j H(g(f(\hat{v}), j))
\end{aligned}$$

$$= \max_v v^T x + \sum_j H(g(v, j)).$$

The first line is by definition. The second line is the most crucial, and holds from the fact that joint entropy is maximized by independent random variables (induced by the marginals $g(f(\hat{v}), j)$). That is, if this independence relationship does not hold, then we could construct a strictly better candidate solution. The third line is because the joint entropy of mutually independent random variables is equal to the sum of their entropies. The last line is true since every sequence form strategy is induced by at least one reduced normal form strategy (i.e. the image of f is equal to the domain of v).

Part 2: The General Case

In Part 1, we showed realization equivalence when the white square is non-existent, i.e. there are no other children of I. Now, we relax that assumption. Let $g(\hat{v})$ map $\hat{v}$ to the probability distribution of actions taken at the first step (together with all contingencies if A is chosen). Let $f(\hat{v})$ map $\hat{v}$ to the probability distribution of actions taken, *excluding* contingencies for A (i.e. all actions taking A are condensed) and denote $f(\hat{u})_0$ to be the probability that the first action is A. Let $\hat{Z}(\hat{v})$ be the dilated entropy terms *not* attributed to the root of A, i.e. $Z(\hat{v}) = \sum_{i, i \neq I} \sum_{a \in \mathcal{A}_i} \hat{v}_a \log \frac{\hat{v}_a}{\hat{v}_{p_i}}$. The entropy term associated with the root is given by $g(\hat{v})$. This may be decomposed by conditioning on $f(\hat{v})$, i.e. the first action in the pre-operation reduced form.

$$H(g(\hat{v})) = \sum_j f(\hat{v})_j H(g(\hat{v}) | f(\hat{v})_j) + H(f(\hat{v})) \tag{3.29}$$

$$= f(\hat{v})_0 H(g(\hat{v}) | f(\hat{v})_0) + H(f(\hat{v})) \tag{3.30}$$

Our objective is

$$\max_{\hat{v}} \hat{v}^T \hat{x} + f(\hat{v})_0 H(g(\hat{v}) | f(\hat{v})_0) + \hat{Z}(\hat{v}) + H(f(\hat{u})) \tag{3.31}$$

The objective may be written in 2 stages – first optimizing $f(\hat{v})$ and second, optimizing the other elements of $\hat{v}$, i.e. actions that are not rooted to I *or* contain A, while respecting all the flow constraints. Terms in the former group are written as $\hat{v}_{A_k}$ and the latter by $\hat{v}_r$. Their related payoffs are vectors $\hat{x}_A$ and $\hat{x}_r$.

Observe that given $\hat{f}(\hat{v})$, the first 3 terms may decomposed into terms dependent only on *either* $\hat{v}_A$ and $\hat{v}_r$, allowing us to perform maximization independently. We have

$$\hat{v}^T \hat{x} = \hat{v}_A^T \hat{x}_A + \hat{v}_r^T \hat{x}_r \tag{3.32}$$

since payoffs at non-terminal actions are 0. Furthermore, given $\hat{f}(\hat{v})$, $\hat{Z}(\hat{v})$ is only dependent on $\hat{v}_r$, since the only values of $\hat{f}(\hat{v})$ dictate the constraints that terms in $\hat{Z}$ must satisfy. Specifically, $Z(\hat{v}) = \sum_{i, i \neq I} \hat{v}_{p_i} \sum_{a \in \mathcal{A}_i} \frac{\hat{v}_a}{\hat{v}_{p_i}} \log \frac{\hat{v}_a}{\hat{v}_{p_i}}$. Lastly, the second term in (3.31) is equivalent to maximizing the entropy given A taken. Putting this together, we have the objective

$$\max_{\hat{f}(\hat{v})} \max_{\hat{v}_A} \max_{\hat{v}_r} \hat{v}_A^T \hat{x}_A + \hat{v}_r^T \hat{x}_r + f(\hat{v})_0 H(g(\hat{v}) | f(\hat{v})_0) + \hat{Z}(\hat{v}) + H(f(\hat{u})) \tag{3.33}$$

$$= \max_{\hat{f}(\hat{v})} (H(f(\hat{u})) + \max_{\hat{v}_A} \left(\hat{v}_A^T \hat{x}_A + f(\hat{v})_0 H(g(\hat{v}) | f(\hat{v})_0)\right) + \max_{\hat{v}_r} \left(\hat{v}_r^T \hat{x}_r + \hat{Z}(\hat{v})\right)) \tag{3.34}$$

Observe that $\frac{\hat{v}_A^T \hat{x}_A}{f(\hat{v})_0}$ is the payoff obtained conditioned on A being taken as the first move. Hence, the left inner maximization may be written as $f(\hat{v})_0 \cdot \max_{\hat{v}_A} \left(\frac{\hat{v}_A^T \hat{x}_A}{f(\hat{v})_0} + H(g(\hat{v}) | f(\hat{v})_0) \right)$, which is precisely $f(\hat{v})_0$ multiplied by the sequence form objective which was described in Part 1 (i.e. conditioned on A being taken).

Substituting the result from Part 1 and merging the maximization terms together recovers our desired result. The final piece of the proof is to relax the assumption that I is the root. This may be done repeatedly, by conditioning on reaching I and marginalizing probabilities as necessary.

3.6.2 Best response of the NLQRE as a nested logit

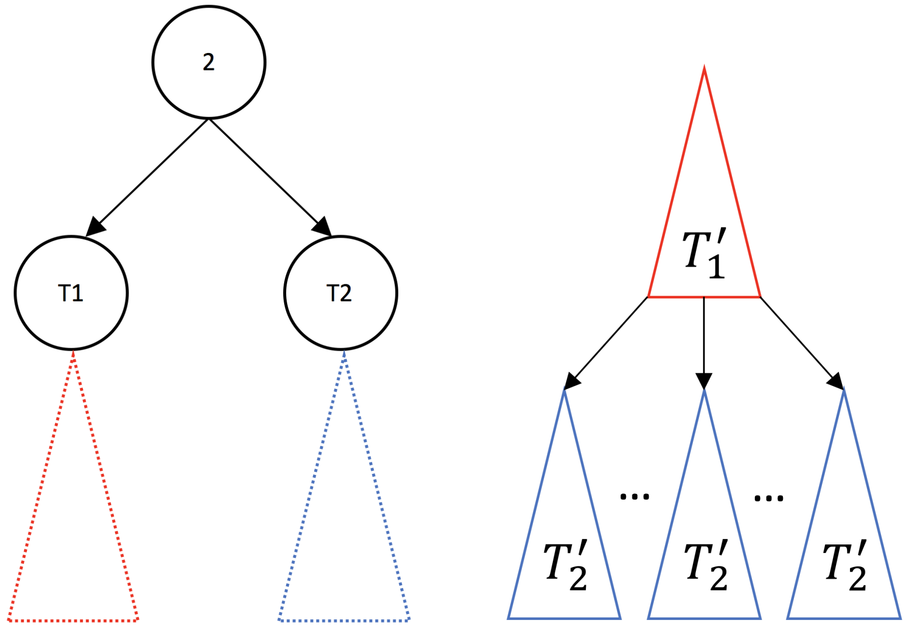
Here we show that the best-response in the NLQRE, which may contain parallel information sets (either due to chance or actions by other players) may be regarded as a nested logit. The idea is to express each strategy in the reduced normal form into a sequence of decisions, each describing what is to be done in each parallel information set.

Consider the following game in Figure 3.13a. Chance (or the other player), labelled as (2), first chooses out of 2 actions, which is made known to the player. For example, this could be the private cards which are dealt to a player in a game of poker. The nodes T_1 and T_2 are the subtrees following these 2 actions by the chance player.

Without loss of generality, let T_1' and T_2' be the tree representation of the player's strategies in subgame T_1 and T_2, i.e., T_1' and T_2' are decision trees for the player. Given that chance could have chosen either action to begin with, the pure strategies are the cross product of all strategies between T_1' and T_2', i.e., the player has to account for all possible contingencies. This may be written as a 2-stage decision process in Figure 3.13b, where the first and second stages are choices from T_1' and T_2' respectively, where T_2' is duplicated n_{leaves} times, where n_{leaves} is the number of possible leaves for T_1'.

The rewards are additive in each stage, implying that the best response for each of the duplicated trees are identical. (Note however that the actual payoffs are modulated by the probability of the chance player choosing the left or right action, but this factor is identical for each copy of T_2'). Furthermore, the leaves of T_1' form a probability vector (since T_1' is a decision tree). This implies that setting the rationality parameters for the roots of all copies T_2' to be λ_2 and the rationality parameter at T_1' to be λ_1 yields precisely one-player version of the optimization problem in (3.16), since the objective in each copy of T_2' is identical and their coefficients sum to 1. Recursively applying this process bottom up to each subtree (i.e., making duplicate copies of subtrees whenever we encounter parallel information sets) gives the desired result.

Observe that each pure strategy (path) in Figure 3.13b is a pure strategy in reduced normal form. However, each path may pass through different information sets (for example, when there is nesting of actions in the bus example), and hence different λ parameters. This is in line with what one would expect with nested logits.



(a) Full 2-player game tree. (b) Player's decision tree.

Figure 3.13: Example showing the equivalence The first action is performed by chance (or the other player). The player has to provide for contingencies in both the red and blue trees. Red and blue components are decision trees for the player. Each leaf (or path from the root to a leaf) is a pure strategy in the reduced normal form of the game in Figure 3.13a.

3.6.3 Efficient computation for forward and backward passes

In this section, we provide the complete computational details and proofs with regard of how to compute best responses for the forward and backward passes.

Forward Pass

Recall from Section 3.4.3 that we needed to find the expression for the following regularized best response using a single bottom-up traversal of the game tree and a single sparse matrix-vector multiplication.

$$\mathrm{BR}_1(\bar{u}, \tilde{v}) = \operatorname*{arg\,min}_{Eu=e} \frac{\tau}{1+\tau} u^T \left(P\tilde{v} + \Psi_1'(\bar{u})\right) + \mathcal{E}(u).$$

At each information set , we solve for the 'behavioral' best response (i.e., assuming that information set was the root). Each of these sub-problems may be conveniently expressed using log-sum-exp and softmax functions. Denoting $c(\bar{u}) = P\tilde{v} + \Psi_1'(\bar{u})$, we compute

$$\min_{u, Eu=e} u^T c(\bar{u}) + \sum_{I \in \mathcal{I}_1} \lambda_I u(\mathbf{Inf}(I)) \sum_{a \in \mathcal{A}(I)} \frac{u(Ia)}{u(\mathbf{Inf}(I))} \log \left(\frac{u(Ia)}{u(\mathbf{Inf}(I))} \right),$$

where the constraint that $u > 0$ is implicit from the log barrier.

Optimization of the inner summation, along with the relevant part of the inner product may be done in closed form using log-sum-exp. The tree constraints for u allows them to perform traversals bottom up. Throughout the traversal process, denote z_I as the 'value' of each infoset I and r_σ as the value of each action.

$$z_I = \lambda_I \log \left(\sum_{a \in \mathcal{A}(I)} \exp\left(\frac{r_{Ia}}{\lambda_I} \right) \right), \quad r_\sigma = -c(\bar{u})_\sigma + \sum_{I' : \textbf{Seq}(I') = \sigma} z_{I'}$$

The behavioral strategies $\hat{u}(a)$ are be expressed using the softmax function. For an action a belonging to infoset I,

$$\hat{u}(a) = \frac{\exp(r_{Ia}/\lambda_I)}{\sum_{a' \in \mathcal{A}(I)} \exp(r_{Ia'}/\lambda_I)} \tag{3.35}$$

The sequence form may be recovered from behavioral strategies using a single downwards traversal of the tree.

Backward Pass

Recall that we needed to prove Theorem 2, i.e., that solving for γ and x in Equations (3.27) and (3.28) require linear time. We reproduce both the equations here for convenience.

$$E\Xi^{-1}(u)c + E\Xi^{-1}(u)E^T\gamma = 0$$
$$x = \Xi^{-1}(u)\left(-c - E^T\gamma\right).$$

Proof. We first verify the following useful results by straightforward algebraic manipulation.
Theorem 3. $\Xi^{-1}(u)E^T = \left[\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{I}_u|}\right]$*, where* α_I*'s are column vectors of size equal to* Σ_1*, each containing* $u(\sigma)/\lambda_I$*'s when* σ *is some descendent of I, and 0 otherwise.*

Taking transposes gives the following, $E\Xi^{-1}(u)$ is equal to $\left[\beta_1, \ldots, \beta_{|\mathcal{A}_u|}\right]$, where the β_σ's are column vectors of length equal to $|I_1|$, and have entries equal to $\frac{u(\sigma)}{\lambda_I}$ if the infoset I is an ancestor of σ.
Theorem 4. $E\Xi^{-1}(u)E^T = diag(\{u(\textbf{\textit{Seq}}(I)\})$*, i.e. equal to a square matrix of size* $|I_1|$*, with diagonal entries equal to the* $\frac{u(\textbf{\textit{Seq}}(I))}{\lambda_I}$ *corresponding to a given information state's parent action.*
Theorem 5. *For any vector c,* $E\Xi^{-1}(u)c$ *may be computed in linear time by traversing the tree bottom-up.*

Now we are ready to prove our main theorem. We first solve for γ. Applying Theorem 3 and Theorem 4 gives an expression for γ without requiring any explicit multiplication. Then, we can obtain γ by applying Theorem 5 together with the fact that $E\Xi^{-1}E^T$ is diagonal. Note that the computation of c requires Py, which may be done in time linear in the size of the *extensive form* game tree. In the extreme case, the game could be a single-stage simultaneous move game, resulting in P being dense, though for typical EFGs, P should be fairly sparse.

Once γ has been computed, we may solve for x

$$x = \Xi^{-1}(u)\left(-c - E^T\gamma\right) \tag{3.36}$$

Since $\Xi(u)$ is tree-structured, the inversion may be done in linear time using Gaussian Elimination. Similarly, $E^T\gamma$ may be computed in linear time because of sparsity in E. That is, the number of non-zero elements in E is equal to the sum of the number of actions over all information sets (recall that each row in E has non-zero entries for the actions for a given information set and its parent). $\square$

3.7 Conclusion

In this chapter, we have described an fully differentiable, novel game solving module which facilitates the learning of unknown game parameters and utilities in an end-to-end fashion. In order to better model bounded rationality, we have also proposed the NLQRE, which can be seen as an extension of nested logits to the multiplayer setting. By utilizing more modern solvers in our game solving module, we are also able to scale learning to medium-sized games. Since then, our work has been extended in many directions [73, 84, 117, 182], demonstrating the broad applicability of differentiable game solvers.

The primary limitation of our proposed method is the strong assumptions we make, particularly the zero-sum and QRE (or NLQRE) assumptions. Many real world games are general-sum, which are difficult to compute and brings up tricky questions like choice of equilibrium concept and/or equilibrium selection. Similarly, QREs are smooth, which makes its computation easier. However, in many sequential settings the non-zero mass placed on each action makes it unrealistic in practice. The problem of non-identifiablity can be fairly severe, and we do not have a rigorous framework to analyze the space of game parameters (beyond simple transformations) with equilibrium matching the data collected. We now list several ideas for extending our work.

- One natural extension is to use other variants of the QRE, e.g., those arising from random utility models. It is unclear if these would be equivalent to adding strictly convex/concave regularizer like in (3.5) and 3.12 and hence unclear if resultant fixed points are unique. One could also define variants of equilibrium using different types of regularization directly, for example, by using Rényi or Tsallis entropies over Shannon entropies. For example, the Tsallis entropy has been used to regularize Bellman equations in single-player reinforcement learning and can be proven to yield policies with sparse support [111], which may more suitable for human behavior in large, sequential environments. It is unclear if our theoretical or algorithmic results will extend to these new settings.
- Another interesting avenue is to consider applying priors over the game parameters of interest. For example, we may have some prior belief over player rationality, game utilities, or chance nodes based on previous studies. One simple way to incorporate this into our framework is to add L2 regularization on those parameters — this would constitute MAP inference under a Gaussian prior. Can we do this in a fully Bayesian manner?
- Other interesting directions include alleviating the problem of non-identifiability by allowing us to learn sets of parameters which are consistent with data, characterizing such sets of parameters in special classes of games, extensions to other equilibrium, learning co-operation/information sharing structure in multiplayer games, and actively learning game parameters via interventions, drawing from the literature on causal inference.

Chapter 4

Subgame Resolving for Strong Stackelberg Equilibrium in Extensive Form Games

Strong Stackelberg equilibria (SSE) have found many uses in security domains, such as wildlife poaching protection [60] and airport patrols [148]. Many of these settings, particularly those involving patrolling, are sequential by nature and are best represented as extensive-form games (EFGs). Unfortunately, even though finding a SSE in general EFGs is provably intractable [114], existing methods for computing SSE in general-sum EFGs are entirely offline. That is, they compute a (often expensive) solution for the *entire* game ahead of time and always play according to that offline solution. In contrast, **subgame resolving** (also known as safe *search*) additionally leverages online computation to improve the strategy for the specific situations that come up during play. Subgame resolving has been a key component for AI in single-agent settings [82, 118], perfect-information games [37, 159, 160, 169], and zero-sum imperfect-information games [29, 31, 136]. In order to apply resolving to two-player zero-sum imperfect-information games in a way that would not do worse than simply playing an offline strategy, **safe resolving** techniques were developed [28, 35, 135]. Safe resolving begins with a **blueprint** strategy that is computed offline. The resolving algorithm then adds extra constraints to ensure that its solution is no worse than the blueprint (that is, that it approximates an equilibrium at least as closely as the blueprint).

However, safe resolving algorithms have so far been primarily developed for two-player zero-sum games. In this chapter, we introduce the key elements of subgame resolving and show how this can be extended to compute Strong Stackelberg equilibria (SSE) in general-sum games. Our algorithms represent the first instance of subgame resolving being applied in the general-sum Stackelberg games with formal guarantees on performance. We show that by using resolving, one can approximate leader-optimal SSEs in games much larger than purely offline methods. We also show that our resolving procedure is itself solving a smaller SSE, thus making our method complementary to other methods based on strategy generation. Experimental results show that in large games our resolving algorithm outperforms offline methods while requiring significantly less computation.

4.1 Background and Related Work

We consider a general-sum imperfect information EFG G with two players, P_1 and P_2. We now review the notation and ideas used in EFGs, the sequence form representation and treeplexes previously described Section 2.3.4, as well as the ideas behind Strong Stackelberg Equilibrium (SSE). We then introduce the key idea behind safe subgame resolving.

4.1.1 Review of EFGs and the Sequence Form

Recall that $\mathcal{S}$ is the set of all vertices in the game tree, which in turn belong to $\mathcal{S}_1$ (for P_1), $\mathcal{S}_2$ (for P_2) and $\mathcal{S}_\mathsf{C}$ (for the chance player P_C). $\mathcal{A}(s)$ is the set of actions available at s and $\mathsf{P}(s)$ is the player to act at s. Each leaf $\ell \in \mathcal{L} \subseteq \mathcal{S}$ is associated with payoff $r_i : \mathcal{L} \to \mathbb{R}$ for each player. The function $C : \mathcal{S} \to [0, 1]$ is the probability of reaching a state s if both players play to do so (i.e., the contribution of P_C along the path to s. Nodes belonging to each $\mathcal{S}_i$ are further partitioned into infosets $I_i \in \mathcal{I}_i$, where nodes are indistinguishable, meaning that any (possibly randomized) strategy for P_i must behave the same way for all nodes in I_i. Since nodes in the same infoset have the same actions, we overload $\mathcal{A}(I)$ to mean the set of actions in any state $s \in I$. We assume that G has perfect recall, meaning that players do not forget their past actions and observations.

Since G has perfect recall, we utilize the sequence form representation. A sequence σ_i is an ordered list of actions taken by P_i before reaching some state s. The empty sequence $\varnothing$ is a special sequence without any actions. The set of possible sequences is denoted Σ_i. We write $\sigma_i a = \sigma_i'$ if a sequence $\sigma_i' \in \Sigma_i$ is obtained by appending action a to σ_i, which we denote by $\mathbf{Seq}_i(I_i)$ or $\mathbf{Seq}_i(s)$. Conversely, $\mathbf{Inf}_i(\sigma_i')$ denotes the information set containing the last action taken in σ_i'. We drop the subscript i when the player is clear from infoset/state. Using the sequence form, mixed strategies are given by *realization plans*, $u : \Sigma_1 \to \mathbb{R}$ and $v : \Sigma_2 \to \mathbb{R}$, which represent distributions over sequences. The realization plans $u(\sigma_1)$ and $v(\sigma_2)$ give the probability that σ_i will be played, assuming all other players (including P_C) plays such as to reach $\mathbf{Inf}(\sigma_i)$. Concretely, they obey nonnegativity and the sequence form constraints, i.e., $u(\varnothing) = 1$, $\forall I \in \mathcal{I}_1$, $u\left(\mathbf{Seq}(I)\right) = \sum_{a \in \mathcal{A}(I)} u\left(\sigma_1 a\right)$ and similarly for P_2 with realization plan v. The polytope of realization plans for each player may be visualized using *treeplexes* (Section 2.3.4). Informally, a treeplex is a tree rooted at $\varnothing$ with subsequent nodes alternating between information sets and sequences, and are operationally useful for providing recursive implementations for common operations in EFGs such as finding best responses.

4.1.2 Stackelberg Equilibria in EFGs

Strong Stackelberg Equilibria (SSE) describe games in which there is asymmetry in the commitment powers of players. By convention, P_1 and P_2 play the role of *leader* and *follower* respectively. The leader P_1 is able to commit to a (potentially mixed) strategy. The follower best-responds to this strategy, while breaking ties by favoring the leader. By carefully committing to a mixed strategy, the leader implicitly issues threats and incentives such that followers best-respond in a manner favorable to the leader. Our goal is to find one such commitment for the leader. SSE are guaranteed to exist and the value of the game for each player is unique. In matrix games, an SSE can be found in polynomial time using the multiple-LP approach [47].

Unfortunately, solving for SSE in EFGs in general-sum games with either chance or imperfect information is known to be NP-hard in the size of the game tree [114] due to there being exponentially many pure strategies. Bosansky and Cermak [18] avoid transformation to normal form and formulate a compact mixed-integer linear program (MILP) which uses a binary sequence-form follower best response variable to modestly-sized problems. More recently, Černý et al. [41] propose heuristically guided incremental strategy generation. All of these methods are fully offline—that is, computation is performed before the game even begins.

4.1.3 Review: Solving SSEs in EFGs using Mixed Integer Linear Programs

In this section, we review the Mixed Integer Linear Program (MILP) proposed by Bosansky and Cermak [18]. This is not necessarily the fastest solver today, but is a fairly efficient baseline to compare against. The MILP is linear in the size of the game tree, and typically runs much faster than solving the equivalent normal form game.

$$\max_{p,u,v,V,q} \sum_{\ell\in\mathcal{L}} p(\ell)\cdot r_1(\ell)\cdot C(\ell) \tag{4.1}$$

$$V_{\textbf{Inf}(\sigma_2)} = q(\sigma_2) + \sum_{I'\in\mathcal{I}_2:\textbf{Seq}_2(I')=\sigma_2} V(I') + \sum_{\sigma_1\in\Sigma_1} u(\sigma_1)\cdot g_2(\sigma_1,\sigma_2) \qquad \forall\sigma_2\in\Sigma_2 \tag{4.2}$$

$$u(\varnothing)=1,\ v(\varnothing)=1 \tag{4.3}$$

$$u(\sigma_1) = \sum_{a\in\mathcal{A}_1(I)} u(\sigma_1 a) \qquad \forall I\in\mathcal{I}_1, \sigma_1=\textbf{Seq}_1(I) \tag{4.4}$$

$$v(\sigma_2) = \sum_{a\in\mathcal{A}_2(I)} v(\sigma_2 a) \qquad \forall I\in\mathcal{I}_2, \sigma_2=\textbf{Seq}_2(I) \tag{4.5}$$

$$0\leqslant q(\sigma_2)\leqslant (1-v(\sigma_2))\cdot M \qquad \forall\sigma_2\in\Sigma_2 \tag{4.6}$$

$$0\leqslant p(\ell)\leqslant v(\textbf{Seq}_2(\ell)) \qquad \forall\ell\in\mathcal{L} \tag{4.7}$$

$$0\leqslant p(\ell)\leqslant u(\textbf{Seq}_1(\ell)) \qquad \forall\ell\in\mathcal{L} \tag{4.8}$$

$$\sum_{\ell\in\mathcal{L}} p(\ell)\cdot C(\ell) = 1 \tag{4.9}$$

$$v(\sigma_2)\in\{0,1\} \qquad \forall\sigma_2\in\Sigma_2 \tag{4.10}$$

$$0\leqslant u(\sigma_1)\leqslant 1 \qquad \forall\sigma_1\in\Sigma_1 \tag{4.11}$$

Conceptually, $p(\ell)$ is the product of player probabilities to reach leaf $\ell\in\mathcal{L}$, such that the final probability of reaching ℓ is $p(\ell)\cdot C(\ell)$. The variables u and v are the leader and follower strategies in sequence form respectively, while V is the EV of each follower information set when strategies u, v are adopted. q is the (non-negative) slack for each sequence/action in each information set, i.e., the difference of the value of an information set and the value of a particular sequence/action within that information set. The term $g_i(\sigma_i,\sigma_{-i})$ is the EV of player i over all nodes reached when executing a pair of sequences (σ_i,σ_{-i}), i.e., $g_i(\sigma_i,\sigma_{-i}) = \sum_{\ell\in\mathcal{L}:\sigma_k=\textbf{Seq}_k(\ell)} r_i(\ell)\cdot C(\ell)$.

Constraint (4.2) ties in the values of the information set V to the slack variables q and payoffs. That is, for every sequence σ_2 of the follower, the value of its preceding information set is equal to the EV of all information sets I' immediately following σ_2 (second term) added with the payoffs from all leaf sequences terminating with σ_2 (third term), compensated by the slack of σ_2 (first term). Constraints (4.3), (4.4),(4.5) are the sequence form constraints [178]. Constraint (4.6) ensures that, for large enough values of M, if the follower's sequence form strategy is 1 for some sequence, then the slack for that sequence cannot be positive, i.e., the follower must be choosing the best action for himself. Constraints (4.7), (4.8), and (4.9) ensure that $p(\ell) \cdot C(\ell)$ is indeed the probability of reaching each leaf. Constraints (4.10) and (4.11) enforce that the follower's best response is pure, and that sequence form strategies must lie in $[0, 1]$ for all sequences. The objective (4.1) is the expected utility of the leader, which is linear in $p(\ell)$.

4.1.4 Safe Subgame Resolving

As its name suggests, much of the literature surrounding subgame resolving is the idea of *subgames*. We suppose that the game tree is can be decomposed into disjoint *subgames*.

Definition 1. A set of states $\mathcal{S}_{\text{sub}} \subseteq S$ is a subgame if (a) if $s \sqsubset s'$ and $s \in \mathcal{S}_{\text{sub}}$ then $s' \in \mathcal{S}_{\text{sub}}$, and (b) if $s \in I_i$ for some $I_i \in \mathcal{I}_i$ and $s \in \mathcal{S}_{\text{sub}}$, then for all $s' \in I_i$ we have $s' \in \mathcal{S}_{\text{sub}}$.

Condition (a) implies that one cannot leave a subgame after entering it, while (b) ensures that information sets are 'contained' within subgames—if any history in an information set belongs to a subgame, then every history in that information set belongs to that subgame. For the j-th subgame $\mathcal{S}^j_{\text{sub}}$, $\mathcal{I}^j_i \subseteq \mathcal{I}_i$ is the set of information sets belonging to P_i within subgame j. Furthermore, let $\mathcal{I}^j_{i,\text{head}} \subseteq \mathcal{I}^j_i$ be the 'head' information sets of P_i in subgame j, i.e., $I_i \in \mathcal{I}^j_{i,\text{head}}$ if and only if $\textbf{Inf}_i(\textbf{Seq}_i(I_i))$ does not exist or does not belong to $\mathcal{I}^j_i$. With a slight abuse of notation, let $I^j_{i,\text{head}}(\ell)$ be the (unique, if existent) information set in $\mathcal{I}^j_{i,\text{head}}$ preceding leaf $\ell \in \mathcal{S}^j_{\text{sub}} \cap \mathcal{L}$. Intuitively, if such an information set does not exist, then the probability of reaching ℓ is only dependent on one player's actions inside the the subgame j (though it could well depend on either or both player's strategy prior to entering the subgame). For simplicity in notation, we will assume that this edge case does not occur.

Remark 13: A more common definition of subgame is in terms of public states. For example, all states with the same public cards (which constitute all the public information there is) will be grouped as to the same public states. Using the definition based on public states will satisfy the requirements of Definition 1

At the beginning, we are given a *blueprint* strategy for the leader. This is typically best possible strategy we can compute using purely offline methods. The leader follows the blueprint strategy in actual play until reaching some subgame. Upon reaching the subgame, the leader computes a refined strategy for that particular subgame and follows it thereafter. The pseudocode is given in Algorithm 2. The computational advantage stems from us only having to compute a refinement for a single subgame, and not all of them.

Our goal is to develop effective algorithms for the refinement step (*). Algorithm 2 implicitly defines a leader strategy distinct from the blueprint. Crucially, this implies that the follower best responds to this implicit strategy and not just the blueprint. Functionally, it is as though the leader uploaded the blueprint alongside the refinement algorithm (*); from the follower's perspective it

Algorithm 2: Generic subgame resolving template for SSE

```
Input: EFG specification, leader blueprint
while game is not over do
    if currently in some subgame $j$ then
        if first time in this subgame then
            (*) Refine leader strategy for subgame $j$
        end
        Play action according to refined strategy
    else
        Play action according to blueprint
    end
end
```

is best responding the combination of both of them. Resolving is said to be *safe* if after applying Algorithm 2, the leader's expected payoff is no less than the blueprint assuming the follower best responds to the algorithm. We design refinement algorithms that give rise to strategies that are (i) safe, and (ii) give the leader a high expected reward under the follower's best response.

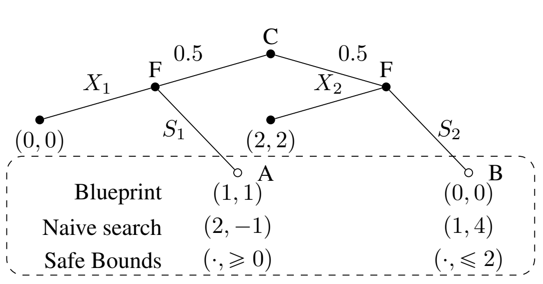
Remark 14: The focus of this work is on the resolving step (*). To this end, we will assume that the blueprint is given to use a-priori. Naturally, the quality of blueprint will affect the leader's expected payoff; indeed, one cannot expect a high leader payoff for arbitrary blueprints. In practice, there are ways to specify reasonable blueprints. The most common is to solve a small, abstracted game. For example, in no-limit poker, the blueprint can be obtained by restricting bet sizes, and hence the size of action spaces. The resultant abstracted game is small enough to be solved offline and used as the basis for refinements. Such an abstraction was applied the poker bot *Libratus* [27].

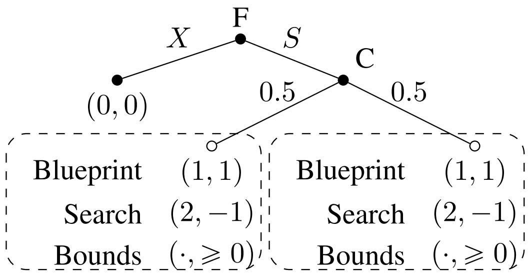
Remark 15: The bulk of our effort is to guarantee safety in our algorithms. Performing unsafe search is possible, and has been done for zero-sum games [189]. These tend to perform better in practice, but often has no performance guarantees. Another advantage of unsafe search is that one need not specify the blueprint strategy beyond the pre-subgame portion of the game.

Remark 16: In zero-sum games, resolving/search is often done in a nested manner (also known as *continual resolving*). This was employed in the poker bots *Libratus* and *DeepStack* [27, 136]. For example, Libratus performed refinements in increasingly finer abstractions of the game as the game is played. Nested resolving gives an algorithm that is closer in spirit to depth limited search used in perfect information games like chess (see [157] for one such example) We do not explore nested resolving in this thesis. This could be an interesting avenue for future work.

4.2 Causes of Unsafe Subgame Resolving

To motivate our algorithm, we first explore how unsafe behavior may arise using a method known as **naïve resolving**. Naïve resolving assumes that prior to entering a subgame, the follower plays the best-response to the blueprint. For each subgame, the leader computes a normalized distribution of initial (subgame) states and solves a new game with initial states in obeying this distribution. Consider the 2-player EFG in Figure 4.1a. After the initial chance node, the follower


C
0.5
0.5
F
F
X_1
X_2
S_1
S_2
$(0,0)$
$(2,2)$
A
B
Blueprint $(1,1)$ $(0,0)$
Naive search $(2,-1)$ $(1,4)$
Safe Bounds $(\cdot,\geqslant 0)$ $(\cdot,\leqslant 2)$
(a) Insufficient incentives and threats



(b) Solving multiple subgames

Figure 4.1: Causes of unsafe naïve search. Boxed regions denote subgames. Expected values for each player under (i) the blueprint strategy and its best response and (ii) under naïve search is shown in the box, as are bounds guaranteeing no change of follower strategies after refinement.

decides to e(**X**)it, or (**S**)tay; the latter brings the game into a subgame denoted by the dotted box. Upon reaching A (or B), the follower receives an expected value (EV) of 1 (resp. 0) when best responding to the blueprint. Thus, under the blueprint, the follower chooses to stay (exit) on the left (right) branches. The expected payoff per player is $(1.5, 1.5)$.

4.2.1 Insufficient Incentives and Threats

Suppose the leader performs naïve resolving in Figure 4.1a, which improves the leader's EV in A from 1 to 2 but reduces the follower's EV in A from 1 to -1. The follower is aware that the leader will perform resolving and thus chooses X_1 over S_1 even before entering A, since exiting gives a payoff of 0. Conversely, suppose resolving improves the leader's EV in B from 0 to 1 and also improves the follower's EV from 0 to 4. Now, the higher post-resolving payoff in B causes the follower to switch from X_2 to S_2. These changes together cause the leader's EV to drop from 1.5 to 0.5, which is worse than the blueprint, implying that naïve resolving is *unsafe*.[1]

Insight: Naïve resolving may induce changes in the follower's strategy *before* the subgame, which adjusts the probability of entering each state *within* the subgame. If one could enforce that in the refined subgame, payoffs to the follower in A remain no less than 0, then the follower would continue to stay, but possibly with leader payoffs greater than the blueprint. Similarly, we may avoid entering B by enforcing that follower payoff in B not exceed 2.

4.2.2 Resolving with Multiple Subgames

Consider the game in Figure 4.1b. Here, the follower chooses to exit or stay before the chance node is reached. If the follower chooses stay, then the chance node determines which of two

[1]This counterexample arises from the general-sum nature of this game, and does not occur in zero-sum games.

identical subgames is entered. Under the blueprint, the follower receives an EV of 1 for choosing stay and an EV of 0 for choosing exit. We consider 2 cases. Case (i): Resolving is performed *only* in the left subgame, which decreases the follower's EV in that subgame from 1 to -1. Then, the expected payoff for staying is $(1.5, 0)$. The follower continues to favor staying (breaking ties in favor of the leader) and the leader's EV increases from 1.0 to 1.5. Now consider Case (ii), where resolving is performed on whichever subgame is encountered during play. Here the follower knows that his EV for staying will be -1 regardless of which subgame is reached. Thus, he will exit, which decreases the leader's payoff to 0, which is less than the blueprint value of 1, and hence *unsafe*.[2]

Insight: Performing resolving using Algorithm 2 is equivalent to performing resolving for *all* subgames. Even if conducting resolving only in a *single* subgame does not cause a shift in the follower's strategy, the combined effect of applying resolving to multiple subgames may. Again, one could remedy this by carefully selecting constraints. If we bound the follower post-resolving EVs for each of the 2 subgames to be $\geqslant 0$, then we can guarantee that X would never be chosen. Note that this is not the only scheme which ensures safety, e.g., a lower bound of 1 and -1 for the left and right subgame is safe too.

4.3 A Safe Algorithm for Subgame Resolving

The crux of our method is to modify naïve resolving such that the follower's pre-subgame best response remains the same even when resolving is applied. This, in turn, can be achieved by enforcing bounds on the follower's EV in any subgame strategies computed via resolving. Concretely, our resolving method comprises 3 steps:

1. Preprocess the follower's best response to the blueprint and its values.
2. Identify a set of non-trivial safety bounds on follower payoffs [3].
3. Solve for the SSE in the subgame reached in actual play constrained to respect the safety bounds computed.

Steps 1 and 2 are independent of which subgame was reached in practice. Thus, they may computed offline (though they do not have to be) and reused if the game is played repeatedly.

4.3.1 Preprocessing of Blueprint

Denote the leader's sequence form blueprint strategy as u^{bp}. We will assume that the game is small enough such that the follower's best response (pure, tiebreaks leader-favored) to the blueprint may be computed, which we denote by v^{bp}. We call the set of information sets which, based on v^{bp} have non-zero probability of being reached the *trunk*, $T \subseteq \mathcal{I}_2 : v^{\text{bp}}(\mathbf{Seq}_2(T)) = 1$. Next, we traverse the follower's treeplex bottom up and compute the payoffs at each information set and sequence (accounting for chance factors $C(z)$ for each leaf). We term these as best-response values (BRVs) under the blueprint. These are computed for both $\sigma_2 \in \Sigma_2$ and $I_2 \in \mathcal{I}_2$

[2]This issue occurs in zero-sum games as well [28].

[3]One could trivially achieve safety by sticking to the blueprint.

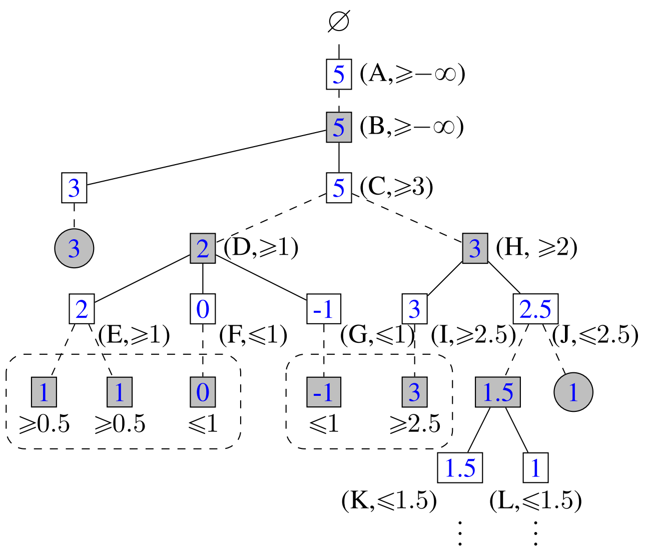


Figure 4.2: Example of bounds computation. Filled boxes represent information sets, circled nodes are terminal payoff entries, hollow boxes are sequences which may be followed by parallel information sets, which are in turn preceded by dashed lines. The dashed rectangle indicates subgames, of which we only show the head information sets of. BRVs of sequences and information sets are within the boxes and the (labels, computed bounds) are placed next to them.

recursively via the following

$$\begin{aligned} BRV(I_2) &= \max_{a \in \mathcal{A}(I_2)} BRV(I_2 a) \\ BRV(\sigma_2) &= \sum_{I' : \mathbf{Seq}_2(I') = \sigma_2} BRV(I') + \sum_{\sigma_1 \in \Sigma_1} u(\sigma_1) \cdot g_2(\sigma_1, \sigma_2), \end{aligned}$$

where $g_2(\sigma_1, \sigma_2)$ is the expected utility of player 2 over all nodes reached when executing the sequence pair (σ_1, σ_2), i.e.,

$$g_2(\sigma_1, \sigma_2) = \sum_{\ell \in \mathcal{L} : \sigma_k = \mathbf{Seq}_k(\ell)} r_2(\ell) \cdot C(\ell).$$

This processing step involves just a single traversal of the game tree.

4.3.2 Generating Safety Bounds

We traverse the follower's treeplex top down while propagating follower payoffs bounds which guarantee that the follower's best response remains v^{bp}. This is recursively done until we reach an information set I belonging to some subgame j. The EV of I is then required to satisfy its associated bound for future steps of the algorithm. Example 4 illustrates these key ideas.

Example 4: Consider the treeplex in Figure 4.2. Values of information sets and sequences are in blue and annotated in order of traversal alongside their bounds. The bounds are generated as follows. The reader should verify that under these generated bounds, the best response of P_2 prior to entering subgames remains equal to v^{bp}.

- The empty sequence $\varnothing$ requires a value greater than $-\infty$.
- For each information set (in this case, B) which follows $\varnothing$, we require (vacuously) for their values to be $\geqslant -\infty$.
- We want the sequence C to be chosen. Hence, the value of C has to be $\geqslant 3$, which, with the lower bound of $-\infty$ gives a final bound of $\geqslant 3$.
- Sum of values for parallel information sets D and H must be greater than C. Under the blueprint, their sum is 5. This gives a 'slack' of 2, split evenly between D and H, yielding bounds of $2 - 1 = 1$ and $3 - 1 = 2$ respectively.
- Sequence E requires a value no smaller than F, G, and the bound for by the D, which contains it. Other actions have follower payoffs smaller than 1. We set a *lower* bound of 1 for E and an *upper* bound of 1 for F and G.
- Sequence I should be chosen over J. Furthermore, the value of sequence I should be $\geqslant 2$—this was the bound propagated into H. We choose the tighter of the J's blueprint value and the propagated bound of 2, yielding a bound of $\geqslant 2.5$ for I and a bound of $\leqslant 2.5$ for J.
- Sequences K and L should not be reached if the follower's best response to the blueprint is followed—we cannot make this portion too appealing. Hence, we apply upper bounds of 1.5 for sequences K and L.

The formal procedure shown in Algorithm 3 is recursive and mirrors the worked example. It takes as input the game G, blueprint u^{bp}, best response v^{bp}, follower BRVs and returns upper and lower bounds $\mathcal{B}(I)$ for all head information sets of subgame j, $\mathcal{I}^j_{2,\text{head}}$. Since the blueprint strategy and its best response satisfies these bounds, feasibility is guaranteed. By construction, lower and upper bounds are obtained for information sets within and outside the trunk respectively. The bounds computation require only a single traversal of the follower's treeplex, which is smaller than the game tree.

The COMPUTEBOUNDS function is the starting point of the bounds generation algorithm. It takes in an EFG specification, a blueprint given, and the BRVs (of sequences $\sigma \in \Sigma_2$ and information sets $I \in \mathcal{I}_2$ computed while preprocessing the blueprint. Our goal is to populate the function $\mathcal{B}(I)$ which maps information sets $I \in \mathcal{I}^j_{2,\text{head}}$ to upper/lower bounds on their values. We begin the recursive procedure by calling EXPSEQTRUNK on the empty sequence $\varnothing$ and vacuous lower bound $-\infty$. Note that $\varnothing$ is by definition in the trunk. This recursive process comprises 4 sub-procedures, one for each infoset and sequence in and outside the trunk. While expanding an infoset in the trunk, we may remain or leave the trunk. Once leaving the trunk, we will never return.

- EXPSEQTRUNK takes in some sequence $\sigma \in \Sigma_2$ and a lower bound lb. Note that we are guaranteed that $lb \leqslant BRV(\sigma)$. The function compute a set of lower bounds on payoffs of information sets I_2 following σ such that (a) the best response to the blueprint satisfies these suggested bounds and (b) under the given bounds on values of I_2, the follower can at least expect a payoff of lb. This is achieved by computing the *slack*, the excess of the blueprint with respect to lb split equally between all I_2 following σ. For each of these I_2, we require their value be no smaller than the lower bound given by their BRVs minus the slack. Naturally, this bound is weaker than the BRV itself.

Algorithm 3: Bounds generation procedure.

```
Function ComputeBounds
    Input  : EFG, Blueprint and its BRVs
    Output: Bounds $\mathcal{B}(I)$ for all $I \in \mathcal{I}^j_{2,\text{head}}$
    ExpSeqTrunk($\varnothing$, $-\infty$)
end
Function ExpSeqTrunk($\sigma$, lb)
    for $I_2 \in \{\mathcal{I}_2 | \textbf{\textit{Seq}}_2(I_2) = \sigma\}$ do
        slack ← $\frac{BRV(\sigma)-lb}{|\{\mathcal{I}_2 | \textbf{Seq}_2(I_2)=\sigma\}|}$
        ExpInfTrunk($I_2$, $BRV(I_2)$-slack)
    end
end
Function ExpSeqNonTrunk($\sigma$, ub)
    for $I_2 \in \{\mathcal{I}_2 | \textbf{\textit{Seq}}_2(I_2) = \sigma\}$ do
        slack ← $\frac{ub-BRV(\sigma)}{|\{\mathcal{I}_2 | \textbf{Seq}_2(I_2)=\sigma\}|}$
        ExpInfNonTrunk($I_2$,
          $BRV(I_2)$+slack)
    end
end
Function ExpInfTrunk($I$, lb)
    if $I \in \mathcal{I}^j_{2,head}$ then
        $\mathcal{B}(I)$ ← lb
        return
    end
    $\sigma^*, \sigma'$ ← best, second best sequences in $I$
      under blueprint
    $v^*, v'$ ← $BRV(\sigma^*), BRV(\sigma')$
    bound ← $\max\left(\frac{v^*+v'}{2}, lb\right)$
    for $\sigma \in \{\Sigma_2 | \textbf{\textit{Inf}}_2(\sigma) = I\}$ do
        if $\sigma = \sigma^*$ is in best response then
            ExpSeqTrunk($\sigma$, bound)
        else
            ExpSeqNonTrunk($\sigma$, bound)
        end
    end
end
Function ExpInfNonTrunk($I$, ub)
    if $I \in \mathcal{I}^j_{2,head}$ then
        $\mathcal{B}(I)$ ← $ub$; return
    end
    for $\sigma \in \{\Sigma_2 | \textbf{\textit{Inf}}_2(\sigma) = I\}$ do
        ExpSeqNonTrunk($\sigma, ub$)
    end
end
```

- The function ExpInfTrunk does the same bound generation process for a given information set I inside the trunk, given a lower bound lb. First, if I is part of the head of a game in subgame j, then we simply store lb into $\mathcal{B}(I)$. If not, then we look at all sequences immediately following I—specifically, we compare the best and second best sequences, given by σ^* and σ'. To ensure that the best sequence still remains the best response, we need to decide on a threshold *bound* such that (i) all sequences other than σ^* does not exceed *bound*, and the value of σ^* is no less than *bound* and (ii) the blueprint itself must obey *bound*. One way to specify *bound* is to take the average of the BRVs of σ^* and σ'. For σ^* we recursively compute bounds by calling ExpSeqTrunk. For all other sequences, we enter a new recursive procedure which generates *upper bounds*.
- The function ExpSeqNonTrunk is similar in implementation to its counterpart ExpSeqTrunk, except that we compute upper instead of lower bounds. Likewise, ExpInfoNonTrunk stores an upper bound if I is in the head of subgame j, otherwise, it uses recursive calls to ExpInfNonTrunk to make sure that all immediate sequences following I does not have value greater than ub.

Alternative safe bounds. These are not the only bounds to guarantee safety. In particular, there are at least two methods to generate safe bounds.

1. Suppose we are splitting lower bounds at an information set I between child sequences (e.g., the way bounds for sequences E, F, G under information set D were computed). Let I have a lower bound of $\mathcal{B}(I)$ and the best and second best actions σ^* and σ' under the blueprint is w^* and w' respectively. Our implementation sets lower and upper bounds for σ^*, σ' to be $\max\{(w^* + w')/2, \mathcal{B}(I)\}$. However, any bound of the form $\max\{\alpha \cdot w^* + (1-\alpha) \cdot w', \mathcal{B}(I)\}, \alpha \in [0,1]$ achieves safety.
2. Splitting lower bounds at sequences σ between parallel information sets under σ (e.g., when splitting the slack at C between D and H, or in Example 2.). Our implementation splits slack evenly though any non-negative split suffices.

We explore the empirical effect of these design decisions in our experiments (Section 4.4).

Alternative unsafe bounds. Bounds can also be generated in unsafe ways.In Section 4.4, we experiment with increasing the slack by some factor $\beta \geqslant 1$. That is, we alter the computation of slack in ExpSeqTrunk by multiplying it by β. This can potentially lead to unsafe behavior, since the follower's payoff under this sequence may possibly be strictly less than lb, but can lead to better performance in practice. A similar unsafe procedure was first explored by Brown and Sandholm [28] in solving zero-sum games.

Remark 17: Safe bounds guarantee safety by ensuring the follower's best response does not deviate from v^{bp} in pre-subgame infostates. This may not be the only way to do so—for a given blueprint, it may be possible to yield a higher payoff by performing refinements which induce the follower to change its best response even for infostates outside of subgames, yet yield a higher payoff for the leader.

4.3.3 MILP Formulation for Constrained SSE

Once safety bounds are generated, we can include them in a MILP similar to that of Bosansky and Cermak [18]. The solution of this MILP is the strategy of the leader, normalized such that $u(\mathbf{Seq}_1(I_1)) = 1$ for all $I_1 \in \mathcal{I}^j_{1,\text{head}}$. Let $\mathcal{L}^j$ be the set of terminal states which lie within subgame j, $\mathcal{L}^j = \mathcal{L} \cap \mathcal{S}^j_{\text{sub}}$. Let $C^j(\ell)$ be the new chance probability when all actions (for both players) taken prior to the subgame are converted to be by chance, according to the blueprint. That is, $C^j(\ell) = C(\ell) \cdot u^{\text{bp}}\left(\mathbf{Seq}_1(I_{1,\text{head}}(\ell))\right) \cdot v^{\text{bp}}\left(\mathbf{Seq}_2(I_{2,\text{head}}(\ell))\right)$. Similarly, we set $g^j_2(\sigma_1, \sigma_2) = \sum_{\ell \in \mathcal{L}^j : \sigma_k = \mathbf{Seq}_k(\ell)} r_2(\ell) \cdot C^j(\ell)$. Let $\mathcal{M}(j)$ be the total probability mass entering subgame j in the original game when the blueprint strategy and best response, $\mathcal{M}(j) = \sum_{\ell \in \mathcal{L}^j} C(\ell) \cdot u^{\text{bp}}(\mathbf{Seq}_1(\ell)) \cdot v^{\text{bp}}(\mathbf{Seq}_2(\ell))$.

$$\max_{p,u,v,V,q} \sum_{\ell \in \mathcal{L}^j} p(\ell) \cdot r_1(\ell) \cdot C^j(\ell) \tag{4.12}$$

$$V(\mathbf{Inf}_2(\sigma_2)) = q(\sigma_2) + \sum_{\substack{I' \in \mathcal{I}_2; \\ \mathbf{Seq}_2(I') = \sigma_2}} V(I') + \sum_{\sigma_1 \in \Sigma_1} u(\sigma_1)\, g^j_2(\sigma_1, \sigma_2) \qquad \forall \sigma_2 \in \Sigma^j_2 \tag{4.13}$$

$$u(\sigma_1) = 1 \quad \exists I_1 \in \mathcal{I}^j_{1,\text{head}} : \mathbf{Seq}_1(I_1) = \sigma_1 \tag{4.14}$$

$$v(\sigma_2) = 1 \quad \exists I_2 \in \mathcal{I}^j_{2,\text{head}} : \mathbf{Seq}_2(I_2) = \sigma_2 \tag{4.15}$$

$$u(\sigma_i) = \sum_{a \in \mathcal{A}_i(I_i)} u(\sigma_i a) \qquad \forall I_1 \in \mathcal{I}^j_1 : \sigma_1 = \mathbf{Seq}_1(I_1) \tag{4.16}$$

$$v(\sigma_2) = \sum_{a \in \mathcal{A}_2(I_2)} v(\sigma_2 a) \qquad \forall I_2 \in \mathcal{I}^j_2 : \sigma_2 = \mathbf{Seq}_2(I_2) \tag{4.17}$$

$$0 \leqslant q(\sigma_2) \leqslant (1 - v(\sigma_2)) \cdot M \qquad \forall \sigma_2 \in \Sigma^j_2 \tag{4.18}$$

$$0 \leqslant p(\ell) \leqslant v(\mathbf{Seq}_2(\ell)) \qquad \forall \ell \in \mathcal{L}^j \tag{4.19}$$

$$0 \leqslant p(\ell) \leqslant u(\mathbf{Seq}_1(\ell)) \qquad \forall \ell \in \mathcal{L}^j \tag{4.20}$$

$$\sum_{\ell \in \mathcal{L}^j} p(\ell) \cdot C^j(\ell) = \mathcal{M}(j) \tag{4.21}$$

$$V(I_2) \geqslant \mathcal{B}(I_2) \qquad \forall I_2 \in \mathcal{I}^j_{2,\text{head}} \cap T \tag{4.22}$$

$$V(I_2) \leqslant \mathcal{B}(I_2) \qquad \forall I_2 \in \mathcal{I}^j_{2,\text{head}} \cap \overline{T} \tag{4.23}$$

$$v(\sigma_2) \in \{0, 1\} \qquad \forall \sigma_2 \in \Sigma^j_2 \tag{4.24}$$

$$0 \leqslant u(\sigma_1) \leqslant 1 \qquad \forall \sigma_1 \in \Sigma^j_1 \tag{4.25}$$

Conceptually, $p(\ell)$ is such that the probability of reaching leaf ℓ is $p(\ell) \cdot C(\ell)$. Variables u and v are the leader and follower sequence form strategies, V is the value of information set for P_2 when u and v are adopted and q is the slack for each sequence.

Objective (4.12) is the expected payoff *in the full game* that the leader gets from subgame j, (4.14), (4.15), (4.16), (4.17), (4.24) and (4.25) are sequence form constraints for each player, where head information sets of each subgame are constrained to 1. (Note the follower is only allowed to have pure best responses (hence the binary constraints (4.24).) Constraints (4.18), (4.19), (4.20), and ensure the follower is best responding, and (4.21) ensures that the probability mass entering j is identical to the blueprint. Constraints (4.22) and (4.23) are bounds previously generated and are the ones which ensure the follower does not deviate from v^{bp} pre-subgame in the refined strategy. This MILP is roughly the size of the subgame, rather than the full G. This decrease in size is substantial, given that there are often thousands of subgames in real-world problems and the fact that solving SSEs in EFGs is NP-hard.

Remark 18: The MILP we propose for solving the constrained subgame is similar in spirit to that of [18] which we presented in Section 4.1.3. Constraints (4.13)-(4.21), (4.24) and (4.25) are analogous to constraints (4.2)-(4.9), (4.10), (4.11) except that they apply to the subgame j instead of the full game. Similarly, the objective (4.12) is to maximize the payoffs from within subgame j. The key addition is constraint (4.22) and (4.23), which are precisely the bounds computed earlier when traversing the treeplex.

4.3.4 Constrained SSEs Interpreted as Solutions to Transformed SSE

Here we show that the the solution to the constrained SSE is the solution of a transformed SSE problem. This implies that besides the MILP, we can employ other SSE solvers, such as those involving strategy generation [41]. This transformation is achieved by means of *gadgets*. Our construction enforces lower bounds in a manner similar to the original gadget of Burch et al. [35], while the upper bounds are enforced in a novel way. Let us assume that the tranformation is performed on the j-th subgame, under the mild assumption that follower head information sets $\mathcal{I}^j_{2,\text{head}}$ are the initial states in $\mathcal{S}^j_{\text{sub}}$. More detail is provided in Section 4.5. Figure 4.3 shows an example construction based on the game in Figure 4.1a.

For every state $h \in \mathcal{I}^j_{2,\text{head}}$, we compute the probability ω_h of reaching h under the blueprint, assuming the follower plays to reach it. The transformed game begins with chance leading to a normalized distribution of ω over these states. Now, recall that we need to enforce bounds on follower payoffs for head information sets $I_2 \in \mathcal{I}^j_{2,\text{head}}$. To enforce a lower bound $BRV(I_2) \geqslant \mathcal{B}(I_2)$, we use a technique described by Burch et al. [35]. Before each state $h \in I_2$, insert an auxiliary state h' belonging to a new information set I'_2, where the follower may opt to terminate the game with a payoff of $(-\infty, \mathcal{B}(I_2)/(\omega_h|I_2|))$ or continue to h, whose subsequent states are unchanged.[4] If the leader's strategy has $BRV(I_2) < \mathcal{B}(I_2)$, the follower would do better by terminating the game, leaving the leader with $-\infty$ payoff.

Enforcing upper bounds $BRV(I_2) \leqslant \mathcal{B}(I_2)$ may be done analogously. First, we reduce the payoffs to the leader for all leaves underneath I_2 to $-\infty$. Second, the follower has an additional action at I_2 to terminate the game with a payoff of $(0, \mathcal{B}(I_2)/(\omega_h|I_2|))$. If the follower's response to the leader's strategy gives $BRV(I_2) > \mathcal{B}(I_2)$, then the follower would choose some action other than to terminate the game, which nets the leader $-\infty$. If the bounds are satisfied, then the leader gets a payoff of 0, which is expected given that an upper bound implies that I_2 is not part of the trunk.

[4]The factor $\omega_h|I_2|$ arises since $\mathcal{B}$ was computed in treeplexes, which already takes into account chance.

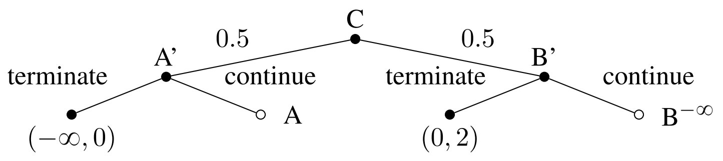


Figure 4.3: The transformed tree for solving the constrained SSE with the safety bounds of Figure 4.1a. A' and B' are auxiliary states introduced for the follower. $B^{-\infty}$ is identical to B, except that leader payoffs are $-\infty$.

4.4 Experiments

In this section we show experimental results for our resolving algorithm (based on the MILP in Section 4.3.3) in synthetic 2-stage games, Goofspiel (where no player wins in a tie) and Leduc hold'em poker (with rake). Experiments were conducted on a Intel i7-7700K @ 4.20GHz with 4 cores and 64GB of RAM. We use the commercial solver Gurobi [79] to solve all instances of MILPs.

We show that even if Stackelberg equilibrium computation for the entire game (using the MILP of Bosansky and Cermak [18]) is warm started using the blueprint strategy u^{bp} and follower's best response v^{bp}, then in large games it is still intractable to compute a strategy. In fact, in some cases it is intractable to even generate the model, let alone solve it. In contrast, our safe resolving algorithm can be done at a far lower computational cost and with far less memory. Since our games are larger than what Gurobi is able to solve to completion in reasonable time, we instead constrain the time allowed to solve each (sub)game and report the incumbent solution. We consider only the time taken by Gurobi in solving the MILP, which dominates preprocessing and bounds generation, both of which only require a constant number of passes over the game tree. In all cases, we warm-start Gurobi with the blueprint strategy.

To properly evaluate the benefits of resolving, we perform resolving on *every* subgame and combine the resulting subgame strategies to obtain the implicit full-game strategy prescribed by Algorithm 2. The follower's best response to this strategy is computed and used to evaluate the leader's payoff. Note that *this is only done to measure how closely the algorithm approximates a SSE*—in practice, resolving is applied only to the subgame reached in actual play and is performed just once. Hence, the worst-case time for a *single* playthrough is no worse than the longest time required for resolving over a *single* subgame (and not the sum over all subgames).

We compare our method against the MILP proposed by Bosansky and Cermak [18] rather the more recent incremental strategy generation method proposed by Černỳ et al. [41]. The former is flexible and applies to all EFGs with perfect recall, while the latter involves the Stackelberg Extensive Form Correlated Equilibrium (SEFCE) as a subroutine for strategy generation. Computing an SEFCE is itself computationally difficult except in games with no chance, in which case finding an SEFCE can be written as a linear program.

4.4.1 Synthetic Two-Stage Games

The two-stage game closely resembles a 2-step Markov game. In the first stage, both players play a general-sum matrix game G_{main} of size $n \times n$, after which, actions are made public. In the second stage, one out of M secondary games $\{G^j_{\text{sec}}\}$, each general-sum and of size $m \times m$ is chosen and played. Each player obtains payoffs equal to the sum of their payoffs for each stage. Given that the leader played action a_1, the probability of transitioning to game j is given by the mixture, $\mathbb{P}(G^{(j)}_{\text{sec}}|a_1) = \kappa \cdot X_{j,a_1} + (1-\kappa) \cdot q_j$, where X_{j,a_1} is a $M \times n$ transition matrix non-negative entries and columns summing to 1 and q_j lies on the M dimensional probability simplex. Here, κ governs the level of influence the leader's strategy has on the next stage. [5]

The columns of X are chosen by independently drawing weights uniformly from $[0, 1]$ and re-normalizing, while q is uniform. We generate 10 games each for different settings of M, m, n and κ. A subgame was defined for each action pair played in the first stage, together with the secondary game transitioned into. The blueprint was chosen to be the SSE of the first stage *alone*, with actions chosen uniformly at random for the second stage. The SSE for the first stage was solved using the multiple LP method and runs in negligible time (< 5 seconds). For full-game solving, we allowed Gurobi to run for a maximum of 1000s. For resolving, we allowed 100 seconds—in practice, this never exceeds more than 20 seconds for *any* subgame.

We report the average quality of solutions in Table 4.1. The full-game solver reports the optimal solution if converges. This occurs in the smaller game settings where ($M = m \leqslant 10$). In these cases resolving performs near-optimally. In larger games ($M = m \geqslant 100$), full-game resolving fails to converge and barely outperforms the blueprint strategy. In fact, in the largest setting only 2 out of 10 cases resulted in *any* improvement from the blueprint, and even so, still performed worse than our method. Our method yields substantial improvements from the blueprint regardless of κ.

4.4.2 Goofspiel

Goofspiel [152] is a game where 2 players simultaneously bid over a sequence of n prizes, valued at $0, \cdots, n-1$. Each player owns cards worth $1, \cdots, n$, which are used in closed bids for prizes auctioned over a span of n rounds. Bids are public after each round. Cards bid are discarded regardless of the auction outcome. The player with the higher bid wins the prize. In a tie, neither player wins and the prize is discarded. Hence, Goofspiel is not zero-sum, players can benefit by coordinating to avoid ties.

In our setting, the n prizes are ordered uniformly in an order unknown to players. Subgames are selected to be all states which have the same bids and prizes after first m rounds are resolved. As m grows, there are fewer but larger subgames. When $m = n$, the only subgame is the entire game. The blueprint was chosen to be the NE under a zero (constant)-sum version of Goofspiel, where players split the prize evenly in ties. The NE of a zero-sum game may be computed efficiently using the sequence form representation [178]. Under the blueprint, the leader obtains a utility of 3.02 and 5.03 for $n = 4$ and $n = 5$ respectively.

[5] One may be tempted to first solve the M Stackelberg games independently, and then apply backward induction, solving the first stage with payoffs adjusted for the second. This intuition is incorrect—the leader can issue non-credible threats in the second stage, inducing the follower to behave favorably in the first.

n	M	m	κ	Blueprint	Ours	Full-game
			0	1.2945	1.4472	**1.4778**
2	2	2	0.1	1.2945	1.4477	**1.4779**
			0.9	1.2951	1.4519	**1.4790**
			0	1.1684	1.6179	**1.6186**
2	10	10	0.1	1.1689	1.6180	**1.6186**
			0.9	1.1723	1.6183	**1.6190**
			0	1.1730	**1.6696**	1.3125
2	100	100	0.1	1.1729	**1.6696**	1.2652
			0.9	1.1722	**1.6696**	1.4055
			0	1.3756	**1.8722**	1.4074
5	100	100	0.1	1.3756	**1.8723**	1.4073
			0.9	1.3752	**1.8723**	1.4534

Table 4.1: Average leader payoffs for two-stage games.

Table 4.2 summarizes the solution quality and running time as we vary n, m. When $n = m$, Gurobi struggles to solve the program to optimality and we report the best incumbent solution found within a shorter time frame. As a sanity check, observe the leader's utility is never worse than the blueprint. When $n = 4$, the incumbent solution for solving the full game has improved significantly from the blueprint in fewer than 5 seconds. This indicates that the game is sufficiently small that performing resolving is not a good idea. However, when $n = 5$, solving the full game ($m = 5$) required 180 times longer compared to resolving ($m \in \{3, 4\}$) in order to obtain *any* improvement over the blueprint, while resolving only needed 100 seconds in order to improve upon the blueprint in a subgame. Furthermore, full-game solving required more than 50 GB of memory while resolving required less than 5 GB.

4.4.3 Leduc Hold'em

Leduc hold'em [163] is a simplified form of Texas hold'em. Players are dealt a single card in the beginning. In our variant there are n cards with 2 suits, 2 betting rounds, an initial bet of 1 per player, and a maximum of 5 bets per round. The bet sizes for the first and second round are 2 and 4. In the second round, a public card is revealed. If a player's card matches the number of the public card, then he/she wins in a showdown, else the higher card wins (a tie is also possible).

Our variant of Leduc includes *rake*, which is a commission fee to the house. We assume for simplicity a fixed rake $\rho = 0.1$. This means that the winner receives a payoff of $(1 - \rho)x$ instead of x. The loser still receives a payoff of $-x$. When $\rho > 0$, the game is not zero-sum. Player 1 assumes the role of leader. Subgames are defined to be all states with the same public information from the second round onward. The blueprint strategy was obtained using the unraked ($\rho = 0$, zero-sum) variant and is solved efficiently using a linear program. We limited the full-game method to a maximum of 5000 seconds and 200 seconds *per subgame* for our method. We reiterate that since we perform resolving only on subgames encountered in actual play, 200 seconds is an *upper bound* on the time taken for a single playthrough when employing

n	$(\lvert\Sigma\rvert, \lvert\mathcal{I}\rvert)$	m	Num. of sub-games	Max. time per sub-game (s)	Leader utility
4	$(2.1, 1.7) \cdot 10^4$	2	1728	5	3.02
		3	64	5	3.07
		4[‡]	1	5	4.06
				$1.0 \cdot 10^2$	4.15
				$5.5 \cdot 10^2$[†]	4.23
5	$(2.7, 2.2) \cdot 10^6$	3	8000	$1.0 \cdot 10^2$	5.19
		4	125	$1.0 \cdot 10^2$	5.29
		5[‡]	1	$1.0 \cdot 10^3$	5.03
				$1.0 \cdot 10^4$	5.03
				$1.8 \cdot 10^4$[†]	5.65

Table 4.2: Results for Goofspiel. [†]This is the earliest time that the incumbent solution achieves the given utility. [‡]This is equivalent to full-game resolving.

n	$\lvert\Sigma\rvert$	$\lvert\mathcal{I}\rvert$	Blueprint	Ours	Full-game
3	5377	2016	-0.1738	-0.1686	**-0.1335**
4	9985	3744	-0.1905	**-0.1862**	-0.1882
5	16001	6000	-0.2028	**-0.2003**	-0.2028
6	23425	8784	-0.1832	**-0.1780**	-0.1832
8	42497	15936	-0.1670	**-0.1609**	N/A

Table 4.3: Leader payoffs for Leduc hold'em with n cards.

resolving (some SSE are easier than others to solve).

The results are summarized in Table 4.3. For large games, the full-game method struggles with improving on the blueprint. In fact, when $n = 8$ the number of terminal states is so large that the Gurobi model could not be created even after 3 hours. Even when $n = 6$, model construction took an hour—it had near $7 \cdot 10^5$ constraints and $4 \cdot 10^5$ variables, of which $2.3 \cdot 10^4$ are binary. Even when the model was successfully built, no progress beyond the blueprint was made.

Varying Bound Generation Parameters.

We now explore how varying α affects solution quality. Furthermore, we experiment with multiplying the slack (see information sets D and H in Section 4.3) by a constant $\beta \geqslant 1$. This results in weaker but potentially unsafe bounds. Results on Goofspiel and Leduc are summarized in Figure 4.4. We observe that lower values of α yield slightly better performance in Leduc, but did not see any clear trend for Goofspiel. As β increases, we observe significant improvements initially. However, when β is too large, performance suffers and even becomes unsafe in the case of Leduc. These results suggest that resolving may be more effective with principled selections of α and β, which we leave for future work.

α	0.1	0.25	$0.5^{\dagger\dagger}$	0.75	0.9
Goofspiel	5.32	5.34	5.29	5.31	5.30
Leduc	-0.184	-0.185	-0.186	-0.188	-0.189
β	$1^{\dagger\dagger}$	2	4	8	16
Goofspiel	5.29	5.35	5.55	5.56	5.50
Leduc	-0.186	-0.182	-0.178	-0.212	-0.212

Table 4.4: Leader payoffs for varying α and β. We consider Goofspiel with $n = 5, m = 4$ and Leduc Hold'em with $n = 4$. Time constraints are the same as previous experiments. ††These are the default values for α and β.

4.5 Technical Proofs

In this section we provide more details on how the constrained SSE can be cast as another SSE problem. The general idea is loosely related to the *subgame resolving* method of Burch et al. [35], although our method extends to general sum games, and allows for the inclusion of both upper and lower bounds as is needed for our search operation.

The broad idea behind Burch et al. [35] is to (i) create an initial chance node leading to all leading states in the subgame (i.e., all states $h \in H^j_{\text{sub}}$ such that there are no states $h' \in H^j_{\text{sub}}$ such that $h' \prec h$) based on the normalized probability of encountering those states under r_i^{bp} and (ii) enforce the constraints using a *gadget*; specifically, by adding a small number of auxiliary information sets/actions to help coax the solution to obey the required bounds.

4.5.1 Restricted case: initial states h in head information sets

To make exposition easier, we begin with the assumption that the $\mathcal{I}^j_{2,\text{head}}$ is a subset of the initial states in subgame j. This assumption will be relaxed later.

Preliminaries For some sequence form strategy pair u, v for leader and follower respectively, the expected payoff to P_i is given by $\sum_{\ell \in \mathcal{L}: \sigma_i = \mathbf{Seq}_i(\ell)} u(\sigma_1) \cdot v(\sigma_2) \cdot r_i(\ell) \cdot C(\ell)$, i.e., the summation of the utilities $r_i(\ell)$ of each leaf of the game, multiplied by the probability that both players play the required sequences, u (and v) and the chance factor $C(\ell)$. That is, the utility from each leaf z is weighed by the probability of reaching it $u(\sigma_1) \cdot v(\sigma_2) \cdot C(\ell)$. The value of an information set $I_i \in \mathcal{I}_i$ is the contribution from all leaves under I_i, i.e., $V_i(I_i) = \sum_{\ell \in \mathcal{L}: \sigma_k = \mathbf{Seq}_k(\ell)} u(\sigma_1) \cdot v(\sigma_2) \cdot r_i(\ell) \cdot C(\ell)$, taking into account the effect of chance for each leaf. Now let $b_i(\sigma_i)$ be the behavioral strategy associated with σ_i, i.e.,

$$b_1(\sigma_1) = \begin{cases} u(\sigma_1)/u(\mathbf{Par}(\sigma_1))) & \text{if } u(\sigma_1) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

and likewise for P_2 and sequence form strategy v. The sequence form $u(\sigma_1)$ is the product of behavioral strategies in previous information sets. Hence, each of these terms in V_i (be it from leader, follower, or chance) can be separated into products involving those *before* or *after*

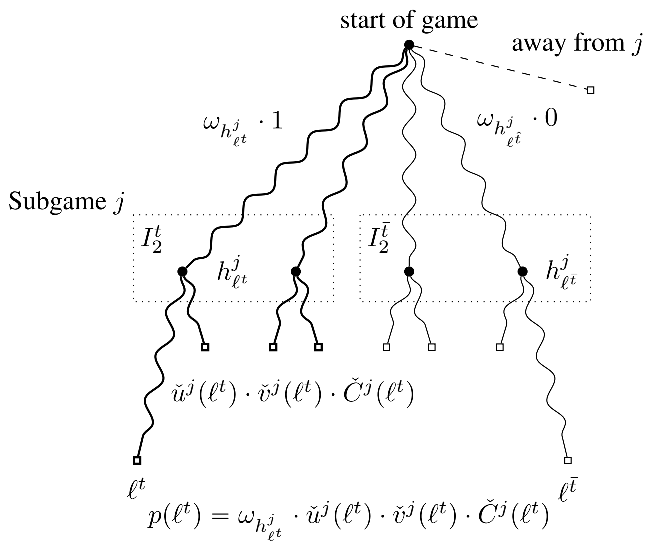

start of game
away from j
$\omega_{h^j_{\ell^t}} \cdot 1$
$\omega_{h^j_{\ell^{\hat{t}}}} \cdot 0$
Subgame j
I_2^t
$h^j_{\ell^t}$
$I_2^{\bar{t}}$
$h^j_{\ell^{\bar{t}}}$
$\check{u}^j(\ell^t) \cdot \check{v}^j(\ell^t) \cdot \check{C}^j(\ell^t)$
ℓ^t
$\ell^{\bar{t}}$
$p(\ell^t) = \omega_{h^j_{\ell^t}} \cdot \check{u}^j(\ell^t) \cdot \check{v}^j(\ell^t) \cdot \check{C}^j(\ell^t)$

Figure 4.4: Decomposition of probabilities for subgame j. Curly lines indicate a series of actions from either player or chance. The dashed box shows subgame j, while dotting boxes are head information sets in $\mathcal{I}^j_{2,\text{head}}$. Thick lines belong to states that are below information sets belonging to the trunk. States that do not lead to subgame j are omitted. Note that the subtrees under the 2 information sets are *not* disjoint, as information sets of the *leader* can span over both subtrees.

subgame j. That is, for a leaf $\ell \in \mathcal{L}^j$, the probability of reaching it can be written as $p(\ell) = \hat{u}^j(\ell) \cdot \hat{v}^j(\ell) \cdot \hat{C}^j(\ell) \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot \check{C}^j(\ell)$, where $\hat{(\cdot)}^j$ and $\check{(\cdot)}^j$ represent probabilities accrued before and after subgame j respectively.

The original game. Figure 4.4 illustrates our setting. For head information set $h \in \mathcal{I}^j_{2,\text{head}}$, define ω^{bp}_h to be the probability of reaching h following the leader's blueprint assuming the follower plays to reach h. Now denote by h^j_ℓ the first state in subgame j leading to leaf ℓ such that $\omega^{\text{bp}}_{h^j_\ell} = \hat{u}^{\text{bp}}(\ell) \cdot \hat{C}^j(\ell)$ is the product of the contributions from the leader and chance, but not the follower. The probability of reaching leaf ℓ is given by $p(\ell) = \omega^{\text{bp}}_{h^j_\ell} \cdot \hat{v}^j(z) \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot \check{C}^j(\ell)$. Observe that if ℓ^t lies beneath an infoset $I_2^t \in \mathcal{I}^j_{2,\text{head}} \cap T$ (i.e., it lies in the trunk and the follower under the blueprint plays to I), $\hat{v}^j(\ell^t) = 1$ (since $v^{\text{bp}}(\textbf{Seq}(I_2)) = 1$). Conversely, if $z^{\bar{t}}$ lies under $I_2^{\bar{t}} \in \mathcal{I}^j_{2,\text{head}} \cap \bar{T}$, i.e., not part of the trunk, then $\hat{v}^j(\ell^{\bar{t}}) = 0$, and the probability of reaching the leaf (under $\hat{u}^{\text{bp}}$ and $\hat{v}^{\text{bp}}$) is 0. From now onward, we will drop the superscript $(\cdot)^{\text{bp}}$ from ω when it is clear we are basing it on the blueprint strategy. This is consistent with the notation used in in Section 4.3.

We want to find a strategy $\check{u}^j$ such that for every information state $I_2 \in \mathcal{I}^j_{2,\text{head}}$, when $\hat{u} = \hat{u}^{\text{bp}}$ and $\hat{v} = \hat{v}^{\text{bp}}$, the best response $\check{u}^j_2$ ensures that $V_2(I_2)$ obeys some upper or lower bounds. That is, value of the information set V_2 in this game, given by

$$V_2(I_2) = \sum_{\substack{\ell \in \mathcal{L}, h \in I_2, \\ h \prec \ell}} \omega_{h^j_\ell} \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot \check{C}^j(\ell) \cdot r_2(\ell), \tag{4.26}$$

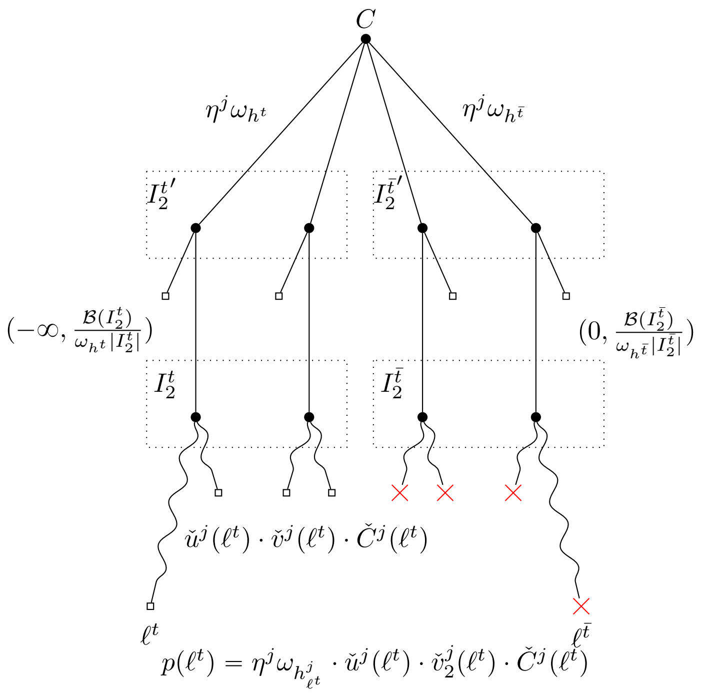

C
$\eta^j \omega_{h^t}$
$\eta^j \omega_{h^{\bar{t}}}$
$I_2^{t\prime}$
$I_2^{\bar{t}\prime}$
$(-\infty, \frac{\mathcal{B}(I_2^t)}{\omega_{h^t}|I_2^t|})$
$(0, \frac{\mathcal{B}(I_2^{\bar{t}})}{\omega_{h^{\bar{t}}}|I_2^{\bar{t}}|})$
I_2^t
$I_2^{\bar{t}}$
$\check{u}^j(\ell^t) \cdot \check{v}^j(\ell^t) \cdot \check{C}^j(\ell^t)$
ℓ^t
$\ell^{\bar{t}}$
$p(\ell^t) = \eta^j \omega_{h^j_{\ell^t}} \cdot \check{u}^j(\ell^t) \cdot \check{v}_2^j(\ell^t) \cdot \check{C}^j(\ell^t)$

Figure 4.5: An example of the transformed game of Figure 4.4. Information sets $I_2^{t\prime}$ and $I_2^{\bar{t}\prime}$ are newly added information sets. Auxiliary actions are in blue, all belonging to newly added information sets. Leaves which are descendants of head information sets not belonging to the trunk are given by red crosses—their payoff to the leader is set to $-\infty$, while keeping the follower payoffs the same.

should be no greater/less than some $\mathcal{B}(I_2)$.

The transformed game. Now consider the transformed subgame described in Section 4.3. Figure 4.5 illustrates how this transformation may look like and the corresponding probabilities. We look at all possible initial states in subgame j, and start the game with chance leading to head states h with a distribution proportional to ω_h. For subgame j, let the normalizing constant over initial states be $\eta^j > 0$. Note that since we are *including* states outside of the trunk, η^j may be greater or less than 1. We duplicate every initial state and head information set, giving the follower an option of terminating or continuing on with the game, where terminating yields an immediate payoff of $\left(-\infty, \frac{\mathcal{B}(I_2^t)}{\omega_h|I_2^t|}\right)$ when the information set containing h belongs to the trunk, and $\left(0, \frac{\mathcal{B}(I_2^{\bar{t}})}{\omega_h|I_2^{\bar{t}}|}\right)$ otherwise. For leaves which are descendants of non-trunk information sets, i.e., $\ell \in \mathcal{L}, h \in I_2 \in \mathcal{I}_{2,\text{head}}^j \cap \bar{T}, h \prec \ell$, the payoffs for the leaders are adjusted to $-\infty$. There is a one-to-one correspondence between the behavioral strategies in the modified subgame and the original game simply by using $\check{u}^j$ (as well as $\check{v}^j$) interchangeably. Next, we show that (i) the bounds $\mathcal{B}$ are satisfied by the solution to the transformed game, (ii) for head information sets in the trunk, any solution satisfying $\mathcal{B}$ will never achieve a higher payoff by selecting an auxiliary action, and (iii) for head information sets outside of the trunk, the solutions satisfying

$\mathcal{B}$ will, by selecting the auxiliary action, achieve a payoff greater or equal to continuing with the game. For (i), we first consider information set I_2^t, which is a head infoset also within the trunk. The *terminate* action will result in a follower payoff (taking into account the initial chance node which was added) independent of the leader's subgame strategy $\check{u}^j$,

$$\sum_{h \in I_2^t} \frac{\mathcal{B}(I_2^t)}{\omega_h |I_2^t|} \cdot \eta^j \omega_{h^t} = \eta^j \mathcal{B}(I_2^t). \tag{4.27}$$

If the follower chooses to continue the game, then his payoff (now dependent on the leader's refined strategy $\check{u}_1^j$ is obtained by performing weighted sums over leaves

$$\eta^j \sum_{\substack{\ell \in \mathcal{L}, h \in I_2^t \\ h < \ell}} \omega_{h^t} \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot \check{C}^j(\ell) \cdot r_2(\ell). \tag{4.28}$$

If the leader is to avoid obtaining $-\infty$, then the follower must choose to remain in the game, which will only happen when (4.28) $\geqslant$ (4.27), i.e.,

$$\sum_{\substack{\ell \in \mathcal{L}, h \in I_2^t \\ h < \ell}} \omega_{h^t} \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot \check{C}^j(\ell) \cdot r_2(\ell) \geqslant \mathcal{B}(I_2^t). \tag{4.29}$$

The expression on the left hand side of the inequality is precisely the expression in (4.26). Since the leader can always avoid the $-\infty$ payoff by selecting $\check{u}$ and $\check{v}$ in accordance with the blueprint, the auxiliary action is never chosen and hence, the lower bounds for the value of trunk information sets is always satisfied.

Similar expressions can be found for non-trunk head-infosets. (4.27) holds completely analogously. The solution to the transformed game needs to make sure the follower always selects the auxiliary action is *always* chosen for information sets not belonging to the trunk, so as to avoid $-\infty$ payoffs from continuing. Therefore, the solution to the transformed game guarantees that (4.29) holds, except that the direction of the inequality is reversed. Again, the left hand side of the expression corresponds to (4.26). Hence, the SSE for the transformed game satisfies our reqired bounds. Furthermore, by starting from (4.26) and working backward, we can also show that any solution $\check{u}^j$ satisfying the constrained SSE does not lead to a best response of $-\infty$ for the leader.

Finally, we show that the objective function of the game is identical up to a positive constant. In the original constrained SSE problem, we sum over all leaf descendants of the trunk and compute the leader's utilities weighed by the probability of reaching those leaves.

$$\sum_{I_2 \in \mathcal{I}^j_{2,\text{head}} \cap T} \sum_{\substack{h \in \mathcal{L}, h' \in I_2 \\ h' < h}} \omega_{h^j_{\ell t}} \cdot \check{u}^j(\ell) \cdot \check{v}^j(\ell) \cdot C^j(\ell) \cdot r_1(\ell)$$

the same expression, except for an additional factor of η. Unlike the constrained SSE setting, the initial distribution has a non-zero probability of starting in a non-trunk state $h \in I_2 \in \mathcal{I}_{2,\text{head}} \cap \bar{T}$. However, since the auxiliary action is always taken under optimality, the leader payoffs from those branches will be 0.

4.5.2 The general case

The general case is slightly more complicated. Now, the initial states in subgames may not belong to the follower. The issue with trying to add auxiliary states the same way as before is that there could be leader actions lying between the start of the subgame and the (follower) head information set. These leader actions have probabilities which are not yet fixed during the start of the search process. To over come this, instead of enforcing bounds on information sets, we enforce bounds on parts of their parent sequences (which will lie outside the subgame).

We first partition the head information sets into *groups* based on their parent sequence. Groups can contain singletons. Observe that information sets in the same group are either all in the trunk or all are not. Let the groups be $G_k = \{I^k_{2,1}, I^k_{2,2}, ..., I^k_{2,m_k}\}, I^k_{2,q} \in \mathcal{I}^j_{\text{head}}$, and the group's heads be the initial states which contain a path to some state in the group, i.e, they are: $G_{k,\text{head}} = \{h | h \prec h', h' \in I^k_{2,q} \in \mathcal{I}^j_{2,\text{head}} \text{ and } \nexists h'' \in H^j_{\text{sub}}, h'' \prec h\}$. Crucially, note that for two distinct groups $G_i, G_k, i \neq k$, their heads $G_{i,\text{head}}$ and $G_{k,\text{head}}$ are disjoint. This because (i) there must be some difference in prior actions that player 2 took (prior to reaching the head information sets) that caused them to be in different groups, and (ii) this action must be taken prior to the subgame by the definition of a head information set.

If 2 information sets $I_{2,1}, I_{2,2} \in \mathcal{I}^j_{2,\text{head}}$ have the same parent sequence $\sigma_2 = \textbf{Seq}_2(I_{2,1}) = \textbf{Seq}_2(I_{2,2})$, i.e., they belong to the same group, G_k, it follows that their individual bounds $\mathcal{B}(I_{2,1}), \mathcal{B}(I_{2,2})$ must have come from some split on some bound (upper or lower) on the value of σ_2. Instead of trying to enforce that the bounds for $I_{2,1}$ and $I_{2,2}$ are satisfied, we try to enforce bounds on the *sum* of the values of $I_{2,1}, I_{2,2}$, since the sum is what is truly important in the bounds for σ_2 when we perform the bounds generation procedure.

The transformation then proceeds in the same way as the restricted case, except that we operate on the heads of each *group*, rather than on the head information sets. The bounds for heads of each group is the *sum* of the bounds of head information sets in that group, and that the factor containing the size of the head information set is replaced by the number of heads for that group.

Upper and lower bounds are enforced using the same gadget as the restricted case, depending on whether the bound is an upper or lower bound. Figure 4.6 shows an example of a lower bound in group G_k. Note that the follower payoff in the auxiliary actions contains a sum over bounds over all information sets belonging to the group. Technically, we are performing safe search while respecting weaker (but still safe) bounds. Upper bounds are done the same way analogous to the restricted case.

4.6 Conclusion

In this chapter, we have shown how to extend safe resolving to the realm of SSE. We show that safety may be achieved by adding a few straightforward bounds on the value of follower information sets. We showed it is possible to cast the bounded resolving problem as another SSE, which makes our approach complementary to other offline methods. Our experimental results on Leduc hold'em demonstrate the ability of our method to scale to large games beyond those which MILPs may solve.

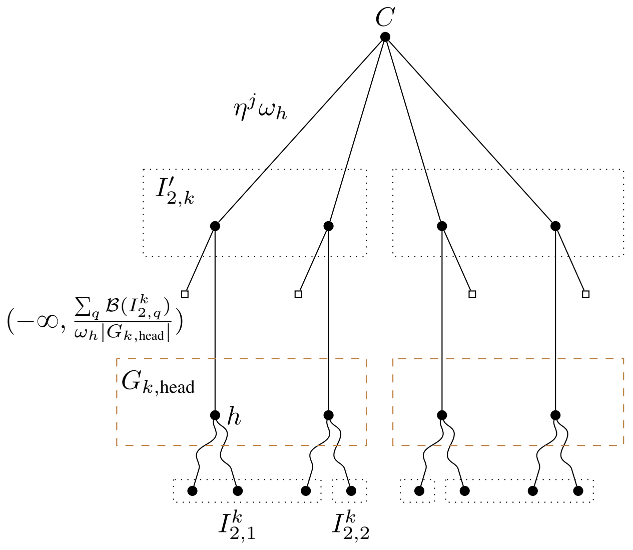

C
$\eta^j \omega_h$
$I'_{2,k}$
$(-\infty, \frac{\sum_q \mathcal{B}(I^k_{2,q})}{\omega_h |G_{k,\text{head}}|})$
$G_{k,\text{head}}$
h
$I^k_{2,1}$
$I^k_{2,2}$

Figure 4.6: An example of a general transformation. Brown dashed lines are the heads of individual groups. $I^k_{2,1}$ and $I^k_{2,2}$ belong to the same group G_k with the heads $G_{k,\text{head}}$. The newly created auxiliary states are in the new information set $I'_{2,k}$. In this case, G_k is in the trunk, hence we enforce a lower bound being enforced for G_k.

There are several shortcomings in our framework. The first is scalability. An ideal algorithm is one which does applies to games larger than we can possibly traverse, as is the case when subgame solving is applied to zero-sum games. However, the current algorithm requires us to compute a set of safe bounds, which in turn requires us to traverse the follower's treeplex. The second would be the strong assumption on follower's behavioral model. In our work (and Chapter 6), we assumed that the follower best responds to any commitment of the leader. This is a fundamental difference between equilibrium in general-sum games compared to Nash equilibrium in zero-games. There, we would do no worse than our exploitability even if our opponent is boundedly rational. Here, assuming our opponent is fully rational (including tiebreaking) when in fact it was not, can lead to "unboundedly" worse outcomes. The problem is that we do not have an accurate follower behavioral model.

One future direction is to include incorporating function approximation and machine learning to further scale (we touch on this a little in Chapter 6). We also believe there is a lot to be explored (both empirically and theoretically) as to what constitutes a good blueprint, whether we can avoid full traversal of the game tree to compute safe bounds, or to avoid explicit computation of these bounds in the first place. We suspect that with a better understanding of subgame solving there is huge potential for unsafe search to perform well in practice with the right tuning of parameters. We are also looking at developing algorithms to exploit special structures in games (e.g., security and patrolling games), as well as more general solution concepts, e.g., multiple leaders or followers. Dealing with bounded rationality of the follower is one that plagues most applications employing the Stackelberg equilibrium (or for the matter, most equilibrium concepts in general-sum games). For example, much work has been done in investigating the use of quantal-response like responses [1, 7, 42, 142, 162]. We believe there is more to be done here in terms of robustness to entire classes of bounded rationality. For example, we believe that our safety guarantees would apply even if the follower's change in best response (from the best response of the blueprint) only applied to any (unknown to the leader) collection of "sub-treeplexes", as opposed to the entire game. Can we do better?

Chapter 5

Subgame Resolving for Extensive Form Correlated Equilibrium

Correlation between players is a powerful tool in game theory. The *Correlated Equilibrium* (CE) is an equilibrium that allows for players to coordinate actions with the aid of a mediator or a randomized correlation device[1], and is known to allow for outcomes which lead to a significantly higher social welfare as compared to solution concepts which require independent play, such as Nash Equilibrium (NE), on top of being computationally more tractable. In a CE, the mediator recommend actions privately to the players according to a probability distribution over joint actions that is known to all players, and the players have no incentive to deviate from the recommended action if they are perfectly rational.

A natural extension of CE to extensive form games (EFG) is the *Extensive Form Correlated Equilibrium* (EFCE), where players are recommended actions at each decision point [179]. Farina et al. [63] examine behavior observed in EFCEs; for example, in peaceful conflict resolution in a variant of the board game *Battleship* and a bargaining game based on the *Sheriff of Nottingham*. The ability to correlate strategies allow outcomes which (in expectation) Pareto dominate the best possible Nash equilibrium. Furthermore, Von Stengel and Forges [179] showed that 2-player games without chance (and later generalized by [61] to *triangle-free* games), EFCEs can be found in polynomial time. Unfortunately, solving and storing an EFCE possibly still requires space that is quadratic in the size of the game tree. This is a significant barrier towards solving large games: for example, storing an EFCE for a simplified variant of Battleship [63] with a grid size of 3×2 and 4 timesteps requires storing a vector with more than 10^8 entries.

In this chapter, we extend subgame resolving to approximate EFCE in 2-player games *without chance*. Instead of announcing the full correlation plan that specifies the probability of recommending different actions at each decision point, the mediator computes the EFCE strategies online. Conceptually, it can be viewed as having the mediator publish the algorithm of choosing recommended actions, and the algorithm is designed in a way such that the rational players will have no incentive to deviate from the recommended actions. One potential application is in ridesharing applications, where a central mediator can recommend which routes drivers take (it

[1]One can also define CE (alongside similar classes of equilibrium [137]) in terms of average strategies between no-regret agents. We avoid this definition in this paper, although we do employ no-regret algorithms as a means of equilibrium *computation*.

is non ideal to have players go to the same area). Here, recommendations must ensure that every driver is incentivized to continue using the application.

This chapter presents three key contributions.

1. We lay out the framework for safe subgame resolving for EFCE in terms of the exploitability of a correlation plan with respect to a correlation blueprint.
2. We show that for games without chance, the structure of the polytope of correlation plans contains a sufficient level of independence between subgames to facilitate independent solving.
3. Third, we provide two refinement algorithms, the first based on a modification of the linear program (LP) of Von Stengel and Forges [179], and the second utilizing a recent and more efficient method based on regret minimization [64]. To the best of our knowledge, this is the first application of subgame resolving to the correlated setting. We experimentally show its scalability in benchmark games.

5.1 Background and Related Work

We first review EFGs and the sequence form (Section 2.3.4), followed by subgames and subgame resolving from Chapter 4. Note that there are some additions and simplifications in notation. We then introduce extensive-form correlated equilibrium (EFCE), the polytope of correlation plans which represents the space of joint strategies. We then briefly describe some existing methods for efficiently solving EFCE.

5.1.1 EFGs and the Sequence Form

In this chapter, restrict our focus to 2-player EFGs G *without chance*. As before, this is represented by a finite game tree: nodes represent game states, belonging to either player P_1 or P_2, while actions are represented by edges directed down the tree. To represent imperfect information, G is supplemented with *information sets* (infosets) $I_i \in \mathcal{I}_i$, $i \in [2]$, which are collection of states belonging to but are indistinguishable to P_i. States in the same infoset contain the same actions $a_i \in \mathcal{A}(I_i)$. We denote by ha the state that is reached immediately after taking action a at state h. We say that state h precedes h', denoted by $h \prec h'$ if $h \neq h'$ and h' is a descendent of h in the game tree, and use the notation $h \preceq h'$ when allowing $h = h'$. We assume players have perfect recall, that is, players never forget past observations and past actions. The set of terminal states $\mathcal{L}$ are known as *leaves*. Each leaf is associated with utilities received by P_i, given by $r_i(h)$. For a given leaf $\ell \in \mathcal{L}$, the *social welfare* is given by $r_1(\ell) + r_2(\ell)$.

We define the set of *sequences* for P_i as the set $\Sigma_i := \{(I, a) : I \in \mathcal{I}_i, a \in \mathcal{A}(I)\} \cup \{\varnothing\}$, where $\varnothing$ is known as the *empty sequence*. For any infoset $I_i \in \mathcal{I}_i$, we denote by $\textbf{Seq}(I_i)$ the *parent sequence* of I, which is defined as the (unique) sequence which precedes I from the root to any node in I; if no such sequence exists, then $\sigma(I) = \varnothing$. Sequences in Σ_i form a partial order; for sequences $\tau = (I, a), \tau' = (I', a') \in \Sigma_i$, we write $\tau \sqsubset \tau'$ if there exists states ha and $h' \in I'$ belonging to P_i such that $ha \preceq h'$, and write $\tau \sqsubseteq \tau'$ if allowing $\tau = \tau'$. If in addition, $\textbf{Seq}(I') = \tau$, we say that τ' is an immediate successor of τ and write $\tau \sqsubset_1 \tau'$. Since the game has no chance, each leaf $h \in \mathcal{L}$ is uniquely identified by a pair of sequences (σ_1, σ_2). With a slight abuse of notation we write $(\sigma_1, \sigma_2) \in \mathcal{L}$, and denote corresponding player payoffs and social welfare by $u_i(\sigma_1, \sigma_2)$ and $u(\sigma_1, \sigma_2)$ respectively.

Sequence-form strategies. In the sequence form, a (mixed) strategy for P_i is represented by a vector x_i, indexed by the sequences $\sigma = (I, a) \in \Sigma_i$. The entry $x_i[\sigma]$ contains the *product* of the probabilities of P_i taking actions from the root to information set I. , including a itself, with the base case given by $x_i[\varnothing] = 1$.[2] Hence, valid sequence-form strategies must satisfy the 'flow' constraints; for every $I \in \mathcal{I}_i$, we have $\sum_{a \in \mathcal{A}(I)} x_i[(I, a)] = \textbf{Seq}(I)$. Sequence-form strategies have size roughly equal to the number of actions (summed over all infosets) of the player, while flow constraints can be seen as a generalization of the sum-to-one constraints for strategies in the simplex.

5.1.2 Subgames

Our definition of a subgame is identical to that of Chapter 4. Some additional notation is in order.
Definition 2. (Subgame) Let G be an EFG with perfect recall. Let H be a subset of nodes in G and $\check{\mathsf{G}}_H$ be the subgraph induced by H. We call $\check{\mathsf{G}}_H$ a subgame of G when: (i) if state $s \in H$, then $s' \sqsupset s$ implies $s' \in H$, and (ii) for all information sets $I \in \mathcal{I}_1 \cup \mathcal{I}_2$, we have $H \cap I = I$ or $\varnothing$.[3]
Definition 3. (Subgame Decomposition) Let $\mathcal{H} = \{H_j\}$ be sets of vertices of G. We call $\mathcal{H}$ a valid subgame decomposition if (i) $\mathcal{H}$ contains non-intersecting sets, (ii) each $H_j \in \mathcal{H}$ induces a valid subgame $\check{\mathsf{G}}_{H_j}$ ($\check{\mathsf{G}}_j$ for short).

For this chapter, we will assume that we are equipped with a valid subgame decomposition $\mathcal{H}$, which induces J disjoint subgames $\{\check{\mathsf{G}}_j\}$. There are many possible ways to obtain subgame decomposition, but by far the most natural and common one is based on *public information*. In this chapter, we make no additional assumptions on subgames apart from those in the definition. We call nodes that are not included in any $\mathcal{H}$ as *pre-subgame*, with an induced subtree $\hat{\mathsf{G}}$. Note that $\hat{\mathsf{G}}$ obeys property (ii) of a subgame; if some infoset is only partially contained in $\hat{\mathsf{G}}$, then it must be partially contained in some subgame, which is disallowed. Consequently, leaves, infosets, and sequences may be likewise partitioned. We denote these sets by $\check{\mathcal{L}}_j, \hat{\mathcal{L}}, \check{\mathcal{I}}_{i,j}, \hat{\mathcal{I}}_i$, and $\check{\Sigma}_{i,j}, \hat{\Sigma}_i$. As an example, the game in Figure 5.1 has two subgames, both starting off with P_2 making his move. Here, P_2's infosets belong to separate subgames, while P_1's infosets all lie in $\hat{\mathsf{G}}$. Similarly, all of P_2's non-empty sequences lie in a subgame, while all of P_1's sequences do not. Another valid subgame decomposition is to have all but the root be in a single subgame.

5.1.3 Extensive-Form Correlated Equilibria

Extensive-form correlated equilibria (EFCE) is a natural extension of CE to EFGs. Unlike regular CEs, players do not receive recommendations for the full game upfront; instead, recommendations are received sequentially, and only for infosets the players are currently in. In the original paper by Von Stengel and Forges [179], this is achieved by means of *sealed recommendations*,

[2]Perfect recall implies there is only one such series of actions.
[3]Alternatively if $h \in \check{\mathsf{G}}$ and belongs to some infoset I, then all states $h' \in I$ are contained in $\check{\mathsf{G}}$.

while Farina et al. [63] have the mediator generating recommendations over the course of the game, but ceasing all future recommendations if a player deviates from a recommendation. We call the recommended actions *trigger sequences* $\sigma^!$ [58, 63]. Trigger sequences contain the last recommended action from the mediator before any deviation, and implicitly contains information about all previous recommendations (due to perfect recall). EFCEs are *incentive-compatible*, players do not expect to benefit by unilaterally deviating from recommended actions.

Polytope of correlation plans A significant benefit of EFCEs over regular CEs is computational cost: computing a CE that achieves maximum social welfare is NP-complete [179], while in 2-player perfect recall games without chance[4], the constraints that define an EFCE may be expressed in a polynomial number of linear constraints and hence may be solved using a linear program. Crucial to these positive results is a theorem by Von Stengel and Forges [179] which characterizes Ξ, the *polytope of correlation plans* which compactly represents the space of joint (reduced) normal-form strategies up to strategic equivalence.

Definition 4. (Connected infosets, $I \rightleftharpoons I'$) Let I, I' be infosets from either player. We say that I, I' are *connected* and write $I \rightleftharpoons I'$ if there exists nodes $u \in I, v \in I'$ in G lying on a path starting from the root, i.e., $u \preceq v$ or $v \preceq u$.

Definition 5. (Relevant sequences, $\sigma_1 \bowtie \sigma_2$) Let $\sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2$. We say that the sequence pair (σ_1, σ_2) is relevant, denoted by $\sigma_1 \bowtie \sigma_2$ if (i) either σ_1 or σ_2 is $\varnothing$ or (ii) $\sigma_1 = (I_1, a_1), \sigma_2 = (I_2, a_2)$ for $I_1 \rightleftharpoons I_2$ and some actions a_1, a_2. For convenience, we use the same notation $\sigma_1 \bowtie I_2$ when either $\sigma_1 = \varnothing$ or if $\sigma_1 = (I_1, a_1)$ and $I_1 \rightleftharpoons I_2$, with a symmetric definition for $I_1 \rightleftharpoons \sigma_2$.

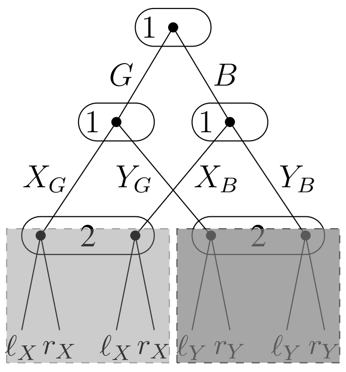
Definition 6. (Von Stengel and Forges) Let G be a perfect recall game without chance. Then, Ξ is a convex polytope of correlation plans which contains non-negative vectors indexed by relevant sequence pairs, with constraints

$$\Xi := \left\{ \boldsymbol{\xi} \geqslant 0 : \begin{array}{c} \xi[\varnothing, \varnothing] = 1, \\ \sum_{a \in \mathcal{A}(I)} \xi[(I_1, a), \sigma_2] = \xi[\mathbf{Seq}(I_1), \sigma_2], \\ \sum_{a \in \mathcal{A}(I)} \xi[\sigma_1, (I_2, a)] = \xi[\sigma_1, \mathbf{Seq}(I_2)] \end{array} \right\}$$

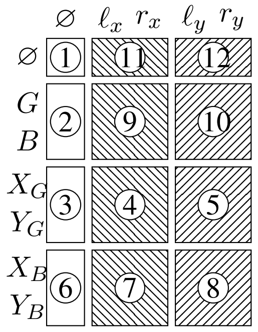
where the second and third constraints are over all $I_1 \bowtie \sigma_2$ and $\sigma_1 \rightleftharpoons I_2$ respectively.

For clarity, we use boldface (e.g., $\boldsymbol{\xi}$) to refer to an entire correlation plan and $\xi[\sigma_1, \sigma_2]$ to reference the entry entry given by relevant sequences σ_1 and σ_2. Visually, one can view Ξ as a 2-dimensional 'checkerboard' of size $|\Sigma_1| \cdot |\Sigma_2|$ with entries to be filled in indices where $\sigma_1 \bowtie \sigma_2$. The second and third constraints are simply the sequence-form constraints [179] applied to each row and column of the checkerboard. For example, for the game in Figure 5.1, all sequence pairs are relevant, and we have row constraints $\xi[\sigma_1, \ell_x] + \xi[\sigma_1, r_x] = \xi[\sigma_1, \varnothing]$ and $\xi[\sigma_1, \ell_y] + \xi[\sigma_1, r_y] = \xi[\sigma_1, \varnothing]$ for all sequences $\sigma_1 \in \Sigma_1$, and column constraints $\xi[G, \sigma_2] + \xi[B, \sigma_2] = \xi[\varnothing, \sigma_2]$, $\xi[X_G, \sigma_2] + \xi[Y_G, \sigma_2] = \xi[G, \sigma_2]$, and $\xi[X_B, \sigma_2] + \xi[Y_B, \sigma_2] = \xi[B, \sigma_2]$ for all $\sigma_2 \in \Sigma_2$.

[4]and more generally in games that are triangle-free[61]



(a) Modified signaling game



(b) Polytope of correlation plans

Figure 5.1: Left: Modified signaling game used in [179] with 2 subgames. Subgames are shaded in different colors. Right: A correlation plan $\boldsymbol{\xi}$ of Figure 5.1a indexed by sequence pairs (σ_1, σ_2). Circles denote fill-in order under the decomposition of Farina et al. [64]. Dashed rectangles show sequence pairs in different subgames.

LP-based EFCE solvers. Observe that Ξ contains a polynomial number of unknowns and linear constraints. A correlation plan in Ξ is an EFCE if it also satisfies incentive constraints that enforce incentive compatibility such that it is optimal for a player to follow the recommendation. Von Stengel and Forges show that the incentive constraints can be also expressed in a polynomial number of linear constraints over $\boldsymbol{\xi}$. Specifically, incentive constraints when $\sigma^! = (I, a^!)$ is recommended for P_1 (the case for P_2 follows naturally) are expressed by[5]

$$\mu(\sigma^!) \geqslant \beta(\sigma'; \sigma^!) \qquad \sigma' = (I, a'), a' \in \mathcal{A}(I)\backslash\{a^!\} \tag{5.1}$$

$$\mu(\sigma) = \sum_{\sigma_2 : (\sigma, \sigma_2) \in \mathcal{L}} r_1(\sigma, \sigma_2)\xi[\sigma, \sigma_2] + \sum_{\sigma' : \sigma' >_1 \sigma} \mu(\sigma') \tag{5.2}$$

$$\beta(\sigma_1; \sigma^!) = \sum_{(\sigma_1, \sigma_2) \in \mathcal{L}} r_1(\sigma_1, \sigma_2)\xi[\sigma^!, \sigma_2] + \sum_{I' : \sigma(I') = \sigma_1} \nu(I'; \sigma^!) \tag{5.3}$$

$$\nu(I; \sigma^!) \geqslant \beta(\sigma; \sigma^!) \qquad a \in \mathcal{I}(I) \tag{5.4}$$

Here, $\mu(\sigma)$ gives the expected utility of P_1 if he abides to this and all following recommendations. Together, (5.3) and (5.4) recursively define the values of the best response of P_1 for deviating to σ' given $\sigma^!$ was recommended. The term $\xi[\sigma^!, \sigma_2]$ essentially contains the (unnormalized) posterior of P_2's sequence given that $\sigma^!$ was recommended.

Bilinear saddle-point problems and regret minimization More recent work by [63, 64] show that the problem of finding an EFCE can be formulated as a bilinear saddle point problem, i.e., an optimization problem of the form $\min_{\mathbf{x} \in \mathcal{X}} \max_{\mathbf{y} \in \mathcal{Y}} \mathbf{x}^T A \mathbf{y}$. Conceptually, this can be seen a *zero-sum* game between two entities, (i) a *mediator* who optimizes $\boldsymbol{\xi} \in \Xi$, and (ii) a *deviator* who selects for each sequence $\sigma^! \in \Sigma_i$ the strategy (for all $\sigma \sqsupset \sigma^!$) that is to be taken after

[5]Readers familiar with the work of Von Stengel and Forges [179] will notice that we use a slightly different LP. This is to make our future definition of exploitability more convenient.

deviating from $\sigma^!$ given the mediator's choice of $\boldsymbol{\xi}$. Essentially, the mediator tries to increase the value of μ, while the deviator seeks to increase ν, which makes the inequality in (5.1) more difficult to achieve. Farina et al. characterize Ξ in terms of a series of convexity-preserving operations known as *scaled extensions* and provide a regret minimizer for sets constructed via scaled extensions. This construction leads to an efficient EFCE solver that runs in linear space, which we adapt in one of our resolving algorithms.

Quality of correlation plans The quality of any correlation plan $\boldsymbol{\xi}$ is measured by (i) its expected *social welfare*, $\sum_{(\sigma_1,\sigma_2)\in\mathcal{L}} \xi[\sigma_1,\sigma_2]r(\sigma_1,\sigma_2)$, where $r = r_1 + r_2$, and (ii) the degree to which the ξ violates the incentive constraints.

Definition 7. (Exploitability) Given a trigger sequence $\sigma^!$ of P_1, and a strategy $\boldsymbol{\xi} \in \Xi$, the value of the best-deviating response to $\sigma^! = (I, a^!)$ is given by

$$\begin{aligned}
\beta^*(\boldsymbol{\xi};\sigma^!) &= \max_{a\in\mathcal{A}(I)\setminus\{a^!\}} \beta((I,a),\boldsymbol{\xi};\sigma^!) \\
\beta(\sigma,\boldsymbol{\xi};\sigma^!) &= \sum_{\sigma_2:(\sigma,\sigma_2)\in\mathcal{L}} r_1(\sigma,\sigma_2)\xi[\sigma^!,\sigma_2] + \sum_{I:\textbf{Seq}(I)=\sigma} \nu(I,\boldsymbol{\xi};\sigma^!) \\
\nu(I,\boldsymbol{\xi};\sigma^!) &= \max_{a\in\mathcal{I}\{a'\}} \beta((I,a),\boldsymbol{\xi};\sigma^!)
\end{aligned}$$

with a similar definition for P_2. The exploitability of $\boldsymbol{\xi}$ for a trigger sequence $\sigma^!$ is given by

$$\delta^*(\boldsymbol{\xi},\sigma^!) = \beta^*(\boldsymbol{\xi};\sigma^!) - \mu(\boldsymbol{\xi},\sigma^!) \tag{5.5}$$

where $\mu(\boldsymbol{\xi},\sigma^!)$ is the value of the $\sigma^!$ if it and all future recommendations are followed, as defined in (5.2).

$\beta^*(\boldsymbol{\xi};\sigma^!)$ is the highest reward a player can get from deviating from the trigger sequence $\sigma^!$, while δ^* measures the gain from doing so. If $\delta^*(\boldsymbol{\xi};\sigma^!) \leqslant 0$ for all $\sigma^! \in \mathcal{I}_1 \cup \mathcal{I}_2$, then $\boldsymbol{\xi}$ is an EFCE. In LP-based solvers, the social welfare is maximized through the objective function, while exploitability is $\leqslant 0$ using linear constraints. In the regret minimization method, exploitability is bounded by the average regret incurred by the solvers, which goes to 0 at a rate of $1/\sqrt{T}$. Maximizing social welfare with regret minimizers can be done by including another constraint lower bounding the social welfare and wrapping this around a binary search to maximize this lower bound.

5.2 Subgame Resolving for Extensive Form Correlated Equilibrium

Subgame resolving exploits the sequential nature of EFGs to refine strategies online. We begin with an analog of the blueprint used in Chapter 4 known as a *correlation blueprint*, typically a guess or approximation of an EFCE.

Definition 8. (Correlation blueprint) A correlation blueprint $\boldsymbol{\xi}_0 \in \Xi$ for the game G is an oracle $\xi_0[\sigma_1,\sigma_2]$ which can be accessed in constant time for all $\sigma_1 \bowtie \sigma_2$.

Remark 19: Blueprint strategies $\boldsymbol{\xi}_0$ may not necessarily be stored explicitly: all we require is that its entries may be accessed efficiently. For example, a blueprint may have players playing independently according to sequence form strategies $\xi^{(i)}(\sigma)$, such that $\xi_0[\sigma_1, \sigma_2] = \xi^{(1)}(\sigma_1) \cdot \xi^{(2)}(\sigma_2)$, i.e., no correlation between players' actions in this special blueprint. In the Battleship benchmark of Farina et al. [63], the blueprint could recommend players to place ships and fire uniformly at random. For games that are close to zero-sum (e.g., Goofspiel), we can solve the zero-sum variant efficiently and then report each player's independent strategy.

At the beginning of the game, players receive recommendations from the blueprint strategy. Once the game enters a subgame, an equilibrium refinement step is performed *only for that subgame entered*, and recommended actions are instead drawn from that refined correlation plan for the rest of the game. Subgame resolving is an *online* method; instead of solving for the equilibrium upfront, it defers part of its computation to when the game is being played. A generic algorithm is shown in Algorithm 4.

5.2.1 Refinements of Correlation Blueprint

Subgame resolving for EFCEs differs significantly from prior work for zero-sum and Stackelberg games. This is because we are now updating relevant sequence pairs of $\boldsymbol{\xi}_0$ in the correlation polytope Ξ, which unlike the space of sequence form strategies, has no obvious hierarchical structure. Fortunately, Definitions 6 and 2 provide enough structure to perform resolving.

Theorem 6. (Independence between subgames) *For $j \in [J]$, let the set S_j contain relevant sequences $(\sigma_1 \bowtie \sigma_2)$ where either σ_1 or σ_2 is belongs to $\check{\mathsf{G}}_j$, i.e., (i) $\sigma_1, \sigma_2 \in \check{\mathsf{G}}_j$, (ii) $\sigma_1 \in \hat{\mathsf{G}}, \sigma_2 \in \check{\mathsf{G}}_j$ or (iii) $\sigma_1 \in \check{\mathsf{G}}_j, \sigma_2 \in \hat{\mathsf{G}}$. Furthermore, let S_0 be the set of relevant sequence pairs such that $\sigma_1, \sigma_2 \in \hat{\mathsf{G}}$. Then $\{S_0, \cdots, S_J\}$ forms a partition of relevant sequence pairs.*

Proof. The cases are disjoint. Hence, it suffices to show that there does not exist $\sigma_1 \bowtie \sigma_2$, where $\sigma_1 = (I_1, a_1) \in \check{\mathsf{G}}_j, \sigma_2 = (I_2, a_2) \in \check{\mathsf{G}}_k$ for $j \neq k$. By definition $\sigma_1 \rightleftharpoons \sigma_2$ implies that there exists a path from the root passing through vertices $v_1 \in I_1$ and $v_2 \in I_2$. WLOG suppose that $v_1 \prec v_2$ in this path. Then v_2 must lie in $\check{\mathsf{G}}_j$ by property (i) of the subgame. This is a contradiction. □

A relevant sequence pair (σ_1, σ_2) is *pre-subgame*, written shorthand by $(\sigma_1, \sigma_2) \in \hat{\mathsf{G}}$ if $(\sigma_1, \sigma_2) \in S_0$. Similarly, we write $(\sigma_1, \sigma_2) \in \check{\mathsf{G}}_j$ if $(\sigma_1, \sigma_2) \in S_j$. Theorem 6 shows exactly one of these must hold.

Definition 9. (Refinements) For a given blueprint $\boldsymbol{\xi}_0 \in \Xi$ and subgame decomposition $\mathcal{H}$, a correlation plan $\tilde{\boldsymbol{\xi}} \in \Xi$ is called a complete refinement if $\tilde{\xi}[\sigma_1, \sigma_2] = \xi_0[\sigma_1, \sigma_2]$ for all $(\sigma_1, \sigma_2) \in \hat{\mathsf{G}}$. Let Ξ_j be Ξ but restricted to sequence pairs $(\sigma_1, \sigma_2) \in \check{\mathsf{G}}_j \cup \hat{\mathsf{G}}$. We call $\tilde{\boldsymbol{\xi}}_j \in \Xi_j$ a refinement of subgame j if $\tilde{\xi}_j[\sigma_1, \sigma_2] = \xi_0[\sigma_1, \sigma_2]$ for all $(\sigma_1, \sigma_2) \in \hat{\mathsf{G}}$.

Example 5: In Figure 5.1, a complete refinement involves updating all but the first column of Figure 5.1b, since for P_2, all but the empty sequence is in some subgame. For the left subgame, we have Ξ_j being the first 3 columns; finding a refinement involves updating the columns containing ℓ_x, r_x (sequences which are contained in the subgame) and dropping the last 2 columns, while respecting the constraints in Definition 6. For the right subgame, a refinement will update

Algorithm 4: Subgame Resolving for EFCE

```
Input: EFG, blueprint $\boldsymbol{\xi}_0$
1: while game is not over do
2:     if currently in some subgame $j$ then
3:         if first time in subgame then
4:             (*) Refine $\boldsymbol{\xi}_0 \to \tilde{\boldsymbol{\xi}}_j$
5:         end if
6:         Recommend action according to $\tilde{\boldsymbol{\xi}}_j$
7:     else
8:         Recommend action according to $\boldsymbol{\xi}_0$
9:     end if
10: end while
```

ℓ_y and r_y. A complete refinement updates all sequence pairs without modifying the first column corresponding to the empty sequence $\varnothing$, i.e., the shaded entries in Figure 5.1b.

Theorem 6 implies that updated entries in a complete refinement (shaded entries in Figure 5.1b) for each refinement do not overlap. This implies refined correlation plans can be combined to form complete refinements. Let $\{\tilde{\boldsymbol{\xi}}_j\}$ contain a refinement for each subgame. Then, $\{\tilde{\boldsymbol{\xi}}_j\} = \{\tilde{\boldsymbol{\xi}}_1, \ldots, \tilde{\boldsymbol{\xi}}_J\}$ *induces* a natural complete refinement $\tilde{\boldsymbol{\xi}}$ where

$$\tilde{\xi}[\sigma_1, \sigma_2] = \begin{cases} \xi_0[\sigma_1, \sigma_2] & (\sigma_1, \sigma_2) \in \hat{\mathsf{G}} \\ \tilde{\xi}_j[\sigma_1, \sigma_2] & (\sigma_1, \sigma_2) \in \check{\mathsf{G}}_j \end{cases}.$$

Observe that $\tilde{\boldsymbol{\xi}} \in \Xi$ since no constraint of Ξ involves sequences pairs *across* different subgames.

The independence property of sequence pairs extends to EFCE incentive constraints for trigger sequences within subgames. For every trigger sequence $\sigma^!$ in $\check{\mathsf{G}}_j$, the best-deviating response (see Definition 7) will never have to reference sequence pairs containing any sequence outside $\check{\mathsf{G}}_j$. This is intuitively true, since once inside $\check{\mathsf{G}}_j$, a potential deviating player will never encounter states outside of $\check{\mathsf{G}}_j$ in the future, and hence need not consider them. This suggests may be possible to perform refinements of subgames *independently* without solving other entries containing sequences from other subgames. However, this independence of incentive constraints does not apply to pre-subgame trigger sequences $\sigma^! \in \hat{\Sigma}_i$. For those sequences, $\delta^*(\boldsymbol{\xi}, \sigma^!)$ will in general depend on refined solutions from multiple, distinct subgames. Handling these constraints is a primary challenge addressed in this chapter.

5.2.2 Safe Refining Algorithms

An important property when performing subgame-resolving for independent, uncorrelated strategies is that of safety, and was the central issue discussed extensively in solving NE in zero-sum games [28]. There, it was observed naive application of resolving algorithms can result in solutions which are of lower quality than the blueprint. The fundamental problem is that when P_1 performed resolving, the best-response of P_2 in the pre-subgame portion differs from the blueprint, hence whatever initial distribution over states at the beginning of the subgame no longer holds. This phenomenon is known as *unsafe* resolving. A similar phenomenon quantified in terms of exploitability holds for EFCEs.

Definition 10. (Safe refinements) A complete refinement $\tilde{\boldsymbol{\xi}}$ of $\boldsymbol{\xi}_0$ is safe if for all trigger sequences $\sigma^!$

$$\delta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) \leqslant \max\left(0, \delta^*(\boldsymbol{\xi}_0; \sigma^!)\right),$$

i.e., the exploitability of $\tilde{\boldsymbol{\xi}}$ for all $\sigma^!$ is 0 or less than the blueprint. We say that $\tilde{\boldsymbol{\xi}}$ is fully safe if in addition, the social welfare (assuming no deviations) under $\tilde{\boldsymbol{\xi}}$ is no less than $\boldsymbol{\xi}_0$. A resolving algorithm is said to be (fully) safe if the complete refinement induced by all j refinements $\tilde{\boldsymbol{\xi}}_j$ is (fully) safe.

In safe refinements, players are at least as incentivized to follow the resolved strategy than the blueprint. Fully safe refinements ensures further that the social welfare will not be diminished. Apart from incurring additional computing costs, there can be no harm in applying fully safe resolving. Clearly, a fully safe resolving algorithm exists in the form of one that trivially returns the blueprint.

Remark 20: We point out that the notion of safety in this chapter is slightly different from the one in Chapter 4. There, safety was defined in terms of the leader's payoff after the follower best responds. Here, safety (and full safety) is defined in terms of the degree that players have to deviate (and the social welfare). There are two reasons for these distinctions. First, in SSEs, our goal is to refine the leader's blueprint under the assumption that the follower best responds. The leader's strategy (blueprint or refinement) does not have to obey any explicit incentive constraints, only the follower's best response does. Here, our goal for EFCEs is to propose a joint strategy between players—this implies *both* players have distinct incentive constraints. Indeed, we reiterate that $\boldsymbol{\xi}_0$ may not even satisfy incentive constraints. Second, there are two objectives here — that of social welfare and incentive compatibility. This explains distinction between safe and fully safe refinements. Fully safe refinements "Pareto dominate" the blueprint in both objectives.

Resolving with multiple subgames. In Definition 10, we required that the induced complete refinement $\tilde{\boldsymbol{\xi}}$ be used to measure safety, and not just the refined strategy of a subgame $\tilde{\boldsymbol{\xi}}_j$. This may seem odd at first, since the primary advantages of resolving was that it did not require computing strategies for subgames not reached in actual play. However, it turns out that this is necessary. Consider the perspective of P_i who in the pre-subgame portion of G was recommended a sequence $\sigma^!$ and was considering deviation. At that point of decision making, P_i does not know which subgame will be reached in the future; however, it knows that whichever subgame is encountered (if at all), refinement will be performed. Thus, when contemplating deviation, P_i in fact computes the value of a best-deviating response to the complete refinement $\tilde{\boldsymbol{\xi}}$. This is despite the fact that in a single playthrough of the game, at most one subgame can be encountered in reality. Another interpretation is that the mediator publishes the refinement *algorithm* which implicitly defines the complete $\tilde{\boldsymbol{\xi}}$, which players contemplate best responses to. Hence, even though the resolving algorithm does not explicitly compute a complete refinement, it should still guarantee safety *as if* it did. This is analogous to the insight made for SSEs, discussed in Section 4.2.2.

5.3 Algorithms for Safe Subgame Resolving

Suppose the mediator has thus far given recommendations based on $\boldsymbol{\xi}_0$ and the players have just entered subgame j. Following Algorithm 4, the mediator computes a refinement $\tilde{\xi}_j$ which he uses for all future recommendations. Our approach mirrors that of Chapter 4, we (i) begin by computing a set of bounds which guarantee safety if enforced and (ii) run a "naïve" optimization which additionally, enforces those bounds. Step (i) involves a search process reminiscent of Chapter 4, except that there are now 2 players and a correlation blueprint $\boldsymbol{\xi}_0$. We again provide 2 algorithms for step (ii), the first uses a linear program, while the second uses regret minimization.

5.3.1 Computing a Set of Safety Bounds

On top of the structural constraints of Ξ_j, we have 3 categories (A-C) of additional constraints that ensure safety. Constraint set (A) enforces safety for trigger sequences $\sigma^! \in \check{\mathsf{G}}_j$, in an manner identical to (5.1), while constraint sets (B-C) ensures that the complete refinement $\tilde{\boldsymbol{\xi}}$ is safe; loosely speaking, (B) contains lower bounds that ensure that following recommendations will yield a high enough payoff to a player contemplating deviation, while (C) contains upper bounds which ensure that players which have deviated do not get rewarded too much.

(A) Safety for in-subgame triggers For each P_i and each sequence in subgame j, i.e., $\sigma^! = (I^!, a^!) \in \check{\Sigma}_{i,j}$, we require

$$\mu(\tilde{\boldsymbol{\xi}}; \sigma^!) \geqslant \beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) - \delta^*(\boldsymbol{\xi}_0; \sigma^!) \tag{5.6}$$

where the μ, ν have constraints identical to (5.2), (5.4). These constraints only involve sequence (pairs) that lie within $\check{\mathsf{G}}_j$ and not other subgames, so no modifications are needed.

Computing safe infoset value bounds Now we turn to constraints (B) and (C), which guarantee safety for trigger sequences in $\hat{\mathsf{G}}$. Our approach is to, for each trigger-sequence $\sigma^! = (I, a^!)$, generate a set of linear constraints which guarantee that the safety for $\sigma^!$ is satisfied, in accordance to Definition 10. What are some sufficient conditions on $\mu(\tilde{\boldsymbol{\xi}}; \sigma^!)$ and $\beta(\sigma', \tilde{\boldsymbol{\xi}}; \sigma^!)$ where $\sigma' \in \{Ia | a \neq a^!\}$ such that the safety condition in Definition 10 is satisfied for $\sigma^!$? To answer this, let us consider $\alpha = \max(0, \delta^*(\boldsymbol{\xi}_0; \sigma^!))$. There are 2 cases. (i) If $\alpha = 0$, then the blueprint was already sufficiently unexploitable for $\sigma^!$. Thus we could afford to decrease $\mu(\tilde{\boldsymbol{\xi}}; \sigma^!)$ and increase $\beta(\sigma', \tilde{\boldsymbol{\xi}}; \sigma^!)$ relative to the blueprint, if it would lead to higher social welfare. (ii) If $\alpha \geqslant 0$, then $\sigma^!$ was exploitable and we do not want to worsen exploitability. This can be avoided if we could somehow ensure $\mu(\boldsymbol{\xi}, \sigma^!)$ and $\beta(\sigma', \tilde{\boldsymbol{\xi}}; \sigma^!)$ do not decrease or increase respectively. Concretely, in case (i), we can require $\beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) \leqslant \hat{\beta}(\sigma; \sigma^!) = \beta^*(\boldsymbol{\xi}_0; \sigma^!) - \delta^*(\boldsymbol{\xi}_0; \sigma^!)/2$, and $\mu(\tilde{\boldsymbol{\xi}}; \sigma^!) \geqslant \check{\mu}(\sigma; \sigma^!) = \mu(\boldsymbol{\xi}_0, \sigma^!) + \delta^*(\boldsymbol{\xi}_0; \sigma^!)/2$. In case (ii), we can require $\beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) \leqslant \hat{\beta}(\sigma; \sigma^!) = \beta^*(\boldsymbol{\xi}_0; \sigma^!)$, and $\mu(\tilde{\boldsymbol{\xi}}; \sigma^!) \geqslant \check{\mu}(\sigma; \sigma^!) = \mu(\boldsymbol{\xi}_0, \sigma^!)$. These are sufficient conditions to guarantee that safety is maintained for $\sigma^!$. Yet, enforcing this is not possible, since $\mu(\tilde{\boldsymbol{\xi}}; \sigma^!)$ and $\beta(\sigma', \tilde{\boldsymbol{\xi}}; \sigma^!)$ can depend on relevant sequence pairs belonging to other subgames. The trick is to recursively unroll μ and β, maintaining bounds which guarantee for safety at each step. This is repeated until we reach infosets belonging to subgames.

Definition 11. (Head infosets) For a player i and subgame j, $I \in \mathcal{I}_i$ is a head infoset of subgame j if $I \in \hat{\mathcal{I}}_{i,j}$ and there does not exist I' preceding I such that $I' \notin \hat{\mathcal{I}}_i$. The set of head infosets for player i in subgame j is denoted by $\mathcal{I}_{i,j}^{\text{head}} \subseteq \check{\mathcal{I}}_{i,j}$. I is called a head infoset if it is a head infoset of some subgame. The set of all head infosets for player i is denoted $\mathcal{I}_i^{\text{head}}$.

(B) Lower bounds on $\mu(\tilde{\boldsymbol{\xi}}, \sigma^!)$ Recall that $\mu(\tilde{\boldsymbol{\xi}}, \sigma^!)$ is the expected utility accrued from leaves $(\sigma_1, \sigma_2) \in \mathcal{L}$, where $\sigma_1 \succeq \sigma^!$. We can recursively decompose $\mu(\tilde{\boldsymbol{\xi}}, \sigma^!)$ into values of infosets, sequences and their summations.

$$d(\sigma; \sigma^!) = \left(\mu(\boldsymbol{\xi}_0, \sigma) - \breve{\mu}(\sigma; \sigma^!)\right) / \left|\{I | \sigma(I) = \sigma\}\right| \tag{5.7}$$

$$\breve{v}(I; \sigma^!) = v(I) - d(\sigma(I); \sigma^!) \tag{5.8}$$

$$f(I; \sigma^!) = (v(\boldsymbol{\xi}_0, I) - \breve{v}(I; \sigma^!)) / \left|\mathcal{A}(I)\right| \tag{5.9}$$

$$\breve{\mu}(\sigma; \sigma^!) = \mu(\boldsymbol{\xi}_0, \sigma) - f(I; \sigma^!) \qquad I : \sigma = (I, a) \tag{5.10}$$

where $v(\boldsymbol{\xi}_0, I) = \sum_{\sigma:(I,a), a \in \mathcal{A}(I)} \mu(\boldsymbol{\xi}_0, \sigma)$, For every σ, starting from $\sigma^!$, we compute in (5.7) the *slack*, i.e., the difference between our desired lower bound $\breve{\mu}(\sigma)$ and what was achieved with the blueprint. In (5.8), this slack is split equally between all infosets which have σ as the parent sequence. A similar process is repeated for infosets in (5.9) and (5.10). We alternate between computing lower bounds for sequences and infosets until we have computed $\breve{v}(I)$ for $I \in \mathcal{I}_i^{\text{head}}$ in (5.8). We repeat this for all $\sigma^! \in \hat{\mathsf{G}}$ and take the tighter of the bounds to obtain $\breve{v}(I^{\text{head}})$ for all $I^{\text{head}} \in \mathcal{I}_i^{\text{head}}$.

(C) Upper bounds on $\beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!)$ Recall that β^* is the value of the best-deviating response. We can unroll the inequalities using Definition 7 and stop once a head infoset is reached, i.e., when we encounter a term $\nu(I, \tilde{\boldsymbol{\xi}}; \sigma^!)$ for some $I \in \mathcal{I}_i^{\text{head}}$. If these terms were upper-bounded appropriately, then $\beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!)$ would be upper-bounded. One possible way is to compute upper bounds recursively

$$s(\sigma; \sigma^!) = \left(\hat{\beta}(\sigma; \sigma^!) - \beta(\sigma, \boldsymbol{\xi}_0; \sigma^!)\right) / \left|\{I | \sigma(I) = \sigma\}\right| \tag{5.11}$$

$$\hat{\nu}(I; \sigma^!) = \nu(I, \boldsymbol{\xi}_0; \sigma^!) + s \tag{5.12}$$

$$\hat{\beta}(\sigma; \sigma^!) = \hat{\nu}(I; \sigma^!). \tag{5.13}$$

At the end of the bounds computation step, we have sets $\hat{\mathcal{B}}_{i,j} = \{(I, \sigma^!, \hat{\nu}(I; \sigma^!)\}$ and $\breve{\mathcal{B}}_{i,j} = \{(I, \breve{v}(I))\}$, each containing constraints of the form $v(\tilde{\boldsymbol{\xi}}; I^{\text{head}}) \geqslant \breve{v}(I^{\text{head}})$ and $\nu(I^{\text{head}}, \tilde{\boldsymbol{\xi}}; \sigma^!) \leqslant \hat{\nu}(I; \sigma^!)$ for some $I \in \mathcal{I}_{i,j}^{\text{head}}$.

As one might expect, a complete refinement is safe when the computed bounds $\breve{\mathcal{B}}_{i,j}$ and $\hat{\mathcal{B}}_{i,j}$ are obeyed. This is shown in the following the following theorem.

Theorem 7 (Bounds imply safety). *If a correlation plan $\tilde{\boldsymbol{\xi}}$ satisfies all constraints in $\breve{\mathcal{B}}_{i,j}$ and $\hat{\mathcal{B}}_{i,j}$ for all $i \in \{1, 2\}$ and $j \in [J]$, then $\delta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) \leqslant \max\left(0, \delta^*(\boldsymbol{\xi}_0; \sigma^!)\right)$ for all $\sigma^! \in \hat{\mathsf{G}}$.*

Similar to the bounds generation process in Chapter 4, it's proof is largely by construction and its derivation can be found in [120].

5.3.2 Safe Refinement Based on Linear Programs

Once the bounds $\breve{\mathcal{B}}_{i,j}$ and $\hat{\mathcal{B}}_{i,j}$ have computed, enforcing them is simply a matter of placing them on top of the constraints for trigger sequences in $\breve{\mathsf{G}}_j$ in (5.6). For lower bounds of the form $(I, \breve{v}(I)) \in \breve{\mathcal{B}}_{i,j}$, we introduce variables $v(I)$, where $v(I) = \sum_{\sigma:(I,a),a\in\mathcal{A}(I)} \mu(\sigma)$ and enforce $v(I) \geqslant \breve{v}(I)$. Note that the auxiliary variables $\mu(\sigma)$ has already been introduced as part of exploitability of $\breve{\mathsf{G}}_j$ when enforcing (5.6). For upper bounds $(I, \sigma^!, \hat{\nu}(I;\sigma^!)) \in \hat{\mathcal{B}}_{i,j}$, we introduce variables $\nu(I;\sigma^!)$ and enforce $\nu(I;\sigma^!) \leqslant \hat{\nu}(I;\sigma^!)$. To ensure that $\nu(I;\sigma^!)$ is indeed the value of the infoset given a trigger sequence $\sigma^!$, we will have to introduce auxiliary variables similar to (5.3), (5.4) recursively. This LP is always feasible, since the blueprint would trivially satisfy all bounds constraints. To achieve full safety, we simply set the objective to be the component of social welfare culminating from subgame j.

The rest of this section describes the LP in detail and **may be skipped without affecting the rest of the chapter**. There are 2 classes of constraints in the LP, (i) structural constraints and (ii) the incentive constraints. Structural constraints enforce that $\tilde{\boldsymbol{\xi}}_j$ lies in Ξ_j. Incentive constraints consist of three sets of constraints (A-C) which ensure safety. (A) ensures that safety is achieved for in-subgame triggers, while (B) and (C) ensures safety for triggers that lie in $\hat{\mathsf{G}}$.

Structural Constraints

Let us first describe the structure of Ξ_j. The components of Ξ_j are indexed by sequence pairs (σ_1, σ_2), where σ_i is either $\varnothing$ or (I_i, a_i). As with Ξ, we require $\sigma_1 \bowtie \sigma_2$. Furthermore, we also require that none of σ_i lies in another G_k, $k \neq j$, i.e., S_k in Theorem 6. That is, if σ_i is non-empty and $\sigma_i = (I_i, a_i)$, then I_i lies in subgame j. Naturally, we require that $\tilde{\boldsymbol{\xi}}_j \geqslant 0$, and that $\tilde{\xi}_j[\varnothing, \varnothing] = 1$. We also need the following two constraints to hold.

- **Sequence-form constraints on rows and columns of $\tilde{\boldsymbol{\xi}}_j$.** These are similar to the flow conservation constraints, but are for the probability of sequence pairs. For example, in the left subgame game in Figure 5.1, Ξ_j is determined by the first 3 columns, and we have row constraint $\xi[\sigma_1, \ell_x] + \xi[\sigma_1, r_x] = \xi[\sigma_1, \varnothing]$, and column constraints $\xi[G, \sigma_2] + \xi[B, \sigma_2] = \xi[\varnothing, \sigma_2]$, $\xi[X_G, \sigma_2] + \xi[Y_G, \sigma_2] = \xi[G, \sigma_2]$, and $\xi[X_B, \sigma_2] + \xi[Y_B, \sigma_2] = \xi[B, \sigma_2]$ for $\sigma_2 \in \{\varnothing, \ell_x, r_x\}$. Generally, we have

$$\Xi_j := \left\{ \boldsymbol{\xi} \geqslant 0 : \begin{array}{c} \boldsymbol{\xi}[\varnothing, \varnothing] = 1, \\ \sum_{a\in\mathcal{A}(I)} \xi[(I_1, a), \sigma_2] = \xi[\sigma(I_1), \sigma_2], \\ \sum_{a\in\mathcal{A}(I)} \xi[\sigma_1, (I_2, a)] = \xi[\sigma_1, \sigma(I_2)] \end{array} \right\},$$

  The constraints defining Ξ_j are essentially identical to that of Ξ — the convex polytope of correlation plans in the original game, except we now work with a restricted index set.

- **Equality-to-blueprint constraints.** These ensures that $\tilde{\boldsymbol{\xi}}_j$ is indeed a refinement of ξ_0. In Figure 5.1, this would mean that the entries in $\hat{\mathsf{G}}$ (entries in the first column) is equal to the blueprint, i.e., $\tilde{\xi}_j[\sigma_1, \varnothing] = \xi_0[\sigma_1, \varnothing]$ for all $\sigma_1 \in \Sigma_1$. More generally, for all $(\sigma_1, \sigma_2) \in S_0$ (i.e., σ_1, σ_2 are both either equal to $\varnothing$ *both* or do not belong to a subgame), we require that $\tilde{\xi}_j[\sigma_1, \sigma_2] = \xi_0[\sigma_1, \sigma_2]$.

Auxiliary Variables

Enforcing the constraint set (A-C) described in Section 5.3.1 using $\breve{\mathcal{B}}_{i,j}$ and $\hat{\mathcal{B}}_{i,j}$ requires the introduction of numerous variables, which can be simplified by the introduction of auxiliary variables. For this part, unless otherwise stated, these variables are with respect to $\tilde{\boldsymbol{\xi}}_j$. We will make it explicit if we need to reference the blueprint $\boldsymbol{\xi}_0$.

- **Values of sequences assuming no deviation.** The first are the variables $\mu(\sigma)$, which exist for each player for all $\sigma = (I, a) \in \Sigma_i$, where $I \in \check{I}_j$. These capture the value of contribution of payoffs for P_i *assuming both players abide to all recommendations* under $\tilde{\xi}_j$ for all leaves involving sequences $\sigma' \sqsupseteq \sigma$. These can be computed recursively the same way as the original LP of Von Stengel and Forges [179] in (5.2).

$$\mu(\sigma) = \sum_{\sigma_2;(\sigma,\sigma_2)\in\mathcal{L}} u_1(\sigma,\sigma_2)\tilde{\xi}_j[\sigma,\sigma_2] + \sum_{\sigma' \succ_1 \sigma} \mu(\sigma'),$$

  that is, value of each sequence σ_i is given by the rewards for P_i from leaves containing σ_i, plus the rewards from future sequences (computed recursively). Note that by the definition of a subgame (Definition 2), during the recursive process, we will never have to 'address' a $\mu(\sigma')$ where σ' lies outside of $\check{\Sigma}_j$. The number of variables and constraints is approximately $|\check{\Sigma}_j|$. We will eventually use these in constraints for in-subgame triggers (A), as well as the lower bounds in (B).

- **Value of infosets assuming no deviations.** For convenience, we will write the value of infosets of I as $v(I) = \sum_{\sigma:(I,a),a\in\mathcal{A}(I)} \mu(\sigma)$. These are used for the lower bounds in (A). In our implementation, they are also used as auxiliary variables while recursively computing $\mu(\sigma)$. This is equivalent to the definition of $\mu(\sigma)$ provided above. The number of variables here is $|\check{\mathcal{I}}_j|$ (which is in turn upper bounded by $|\check{\Sigma}_j|$.

- **Values of infosets under deviations.** Next, we have the variables $\nu(I;\sigma^!)$. These represent the values of infoset I given that the player was triggered by $\sigma^!$, deviated and plays the best response after his deviation. Let $\sigma^! = (I^!, a^!)$ be a trigger sequence, and $\sigma' = (I^!, a'), a' \neq a^!$ is the sequence which was deviated to. $v(I;\sigma^!)$ exists for each $I \in \mathcal{I}_j$ where $\sigma(I) \sqsupseteq \sigma'$, i.e., I (which belongs in the subgame j) could be encountered after deviating to σ', which lies in the same infoset as $\sigma^!$. The variables in ν can be further split into 2 groups. If $\sigma^!$ lies in $\check{\Sigma}_j$, then this is similar to the variable in (5.4). If not, then note that these are only created for infosets within subgame j. The former is used in enforcing safety for in-subgame triggers (A) and the latter for constraint set (C). The total number of variables here for each P_i is no greater (and in practice, usually much smaller) than $(|\hat{\Sigma}_i| + |\check{\Sigma}_{i,j}|) \cdot |\check{\mathcal{I}}_{i,j}|$.

- **Value of sequences under deviations.** $\beta(\sigma;\sigma^!)$ is very similar to ν, in that it is the value of a sequence assuming the last recommendation received and deviated from was $\sigma^!$. Again, we are only concerned with sequences $\sigma \in \check{\Sigma}_j$, and those which could be reached after deviation from $\sigma^!$. For a fixed trigger sequence $\sigma^!$, $\beta(\sigma;\sigma^!)$ and $\nu(I;\sigma^!)$ can be computed together recursively using Equations (5.4) and (5.3). The inequalities ensure that the values in ν and β are such that these are no less than best-responses towards $\tilde{\boldsymbol{\xi}}_j$. The number of variables here is no greater than $(|\hat{\Sigma}_i| + |\check{\Sigma}_{i,j}|) \cdot |\check{\Sigma}_{i,j}|$.

Incentive Constraint Sets (A)-(C)

- **Incentive Constraint Set (A)** Recall that these ensure that safety is achieved for $\sigma^! \in \check{\Sigma}_j$. For each such $\sigma^!$, we have constraints of the form

$$\mu(\tilde{\boldsymbol{\xi}}; \sigma^!) \geqslant \beta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) - \delta^*(\boldsymbol{\xi}_0; \sigma^!). \tag{5.14}$$

- **Incentive Constraint Set (B)** These are intended to guarantee lower bounds on $\mu(\sigma^!)$, where $\sigma \in \hat{\Sigma}_i$. The sufficient conditions for doing so are in the set $\check{\mathcal{B}}_{i,j}$, each of the form $(I, \check{v}(I))$. We will need

$$v(I^{\text{head}}) \geqslant \check{v}(I^{\text{head}})$$

  for some $I^{\text{head}} \in \mathcal{I}^{\text{head}}_{i,j}$.

- **Incentive Constraint Set (C)** Similarly, (C) is intended to upper bound $\beta^*(\sigma^!)$, i.e., ensure that deviating would not be too beneficial. This is achieved by making sure $\beta^*(\sigma; \sigma^!)$ for all $\sigma \neq \sigma^!$ and $\sigma = (I^!, a)$. This can be achieved when $\nu(I^{\text{head}}; \sigma^!) \leqslant \hat{\nu}(I, \sigma^!)$ for each tuple $(I, \sigma^!, \hat{v}(I; \sigma^!)) \in \hat{\mathcal{B}}_{i,j}$.

5.3.3 Safe Refinements using Regret Minimization

Our second algorithm is based on regret minimization. We solve a saddle-point problem using self-play, utilizing the scaled extension operator of Farina et al. [64] to provide an efficient regret minimizer over Ξ_j. This leads to a significantly more efficient algorithm, which we now give a brief overview of.

The first step is to rewrite the refinement LP as a bilinear saddle point problem, similar to what Farina et al. [63] did. Observe that a refinement $\tilde{\boldsymbol{\xi}}_j$ is safe if and only if the greatest violation of the safety constraints to be equal to 0. Building on this intuition, we introduce for each safety constraint, multipliers $\lambda^{\delta}_{i,\sigma^!}$, $\lambda^{\nu}_{i,I,\sigma^!}$ and $\lambda^{v}_{i,I}$ — for exploitability (in $\check{\mathsf{G}}_j$), upper bounds, and lower bounds respectively. These multipliers are non-negative and sum to 1. Additionally, we introduce variables $\check{y}_{i,\sigma^!} \in \check{Y}_{i,\sigma^!}$ for $(I^!, a^!) = \sigma^! \in \check{\Sigma}_{i,j}$. Similarly, for trigger sequences $(I^!, a^!) = \sigma^! \in \hat{\Sigma}_i$, we introduce $\hat{y}_{i,\sigma^!} \in \hat{Y}_{i,\sigma^!}$. These y's represent the components of the best-deviating responses to trigger sequences $\sigma^!$, and whose polytopes can be easily represented using the sequence-form representation of Von Stengel [178]. Resolving is equivalent to solving the following bilinear saddle point problem:

$$\min_{\tilde{\boldsymbol{\xi}}_j} \max_{i,\lambda,y} \left\{ \begin{array}{c} \displaystyle\sum_{i,\sigma^! \in \check{\Sigma}_{i,j}} \left[\tilde{\boldsymbol{\xi}}_j^T R^{\delta} z^{\delta}_{i,\sigma^!} + \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{\delta}_{i,\sigma^!} b^{\delta}_{i,\sigma^!} \right) \right] + \\ \displaystyle\sum_{\substack{i,(I,\sigma^!,\cdot) \\ \in \hat{\mathcal{B}}_{i,j}}} \left[\tilde{\boldsymbol{\xi}}_j^T R^{\nu} z^{\nu}_{i,\sigma^!} + \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{\nu}_{i,\sigma^!} b^{\nu}_{i,\sigma^!} \right) \right] + \\ \displaystyle\sum_{i,(I,\cdot) \in \check{B}_j} \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{v}_{i,I} b^{v}_{i,I} \right) \end{array} \right\}, \tag{5.15}$$

where $z^{\delta}_{i,\sigma^!} = \lambda^{\delta}_{i,\sigma^!} \check{y}_{i,\sigma^!}$ and $z^{\delta}_{i,\sigma^!} = \lambda^{\nu}_{i,\sigma^!} \hat{y}_{i,\sigma^!}$, for appropriately chosen constants R, b (which may vary on $\boldsymbol{\xi}_0$).

Algorithm 5: Refinement with Regret Minimization

```
Input: EFG, blueprint $\boldsymbol{\xi}_0$
1: Decompose $\Xi_j$ into series of scaled extensions.
2: Construct regret RM'er $\mathcal{X}$ over $\Xi_j$.
3: Construct regret RM'er $\mathcal{Y}$ over deviators.
4: while saddle point gap $\geqslant \epsilon$ do
5:     $\tilde{\boldsymbol{\xi}}_j^{(t)} \leftarrow \mathcal{X}$.recommend; $y^{(t)} \leftarrow \mathcal{Y}$.recommend
6:     $\mathcal{X}$.observeLoss($y_t$); $\mathcal{Y}$.observeLoss($\xi_t$)
7: end while
```

The second step is to treat the refinement problem as a *zero-sum* game between a *mediator*, who chooses a refinement $\tilde{\boldsymbol{\xi}}_j$ and *deviator*, who chooses multipliers and best-deviating responses. This zero-sum game can be solved by running self-play between two Hannan-consistent regret minimizers and taking average strategies. A regret minimizer for the deviator is constructed efficiently using counterfactual regret minimization [193]. A regret minimizer over Ξ_j is constructed using the decomposition technique used by Farina et al. [64] with some additional tiebreaking rules to ensure we do not have to "fill-in" sequence pairs in $\hat{\mathsf{G}}$.

In the following, we go into some details of the saddle-point reformulation and construction of regret minimizers. These details are not necessary for to understand the rest of this chapter.

Refinements as a Bilinear Saddle-point Problem

The LP *without the objective* (and by extension, safe resolving) can be written as a bilinear saddle point problem. Consider a refinement $\tilde{\boldsymbol{\xi}}_j$. The largest violation of a constraint is:

$$\max_{i \in \{1,2\}} \left\{ \begin{array}{c} \max\limits_{\sigma^! \in \check{\Sigma}_{i,j}} \delta^*(\tilde{\boldsymbol{\xi}}; \sigma^!) - \delta^*(\boldsymbol{\xi}_0; \sigma^!) \\ \max\limits_{(I, \sigma^!, \hat{\nu}(I;\sigma^!)) \in \hat{\mathcal{B}}_{i,j}} \nu(I, \tilde{\boldsymbol{\xi}}_j; \sigma^!) - \hat{\nu}(I; \sigma^!) \\ \max\limits_{(I, \breve{v}(I)) \in \breve{\mathcal{B}}_{i,j}} \breve{v}(I) - v(\tilde{\boldsymbol{\xi}}_j; I) \end{array} \right\}.$$

The inner maximizations are for (A) safety for trigger sequences $\sigma^! \in \check{\mathsf{G}}_j$ (C) upper bounds on values head infoset I under the best deviating response to a the trigger sequences $\sigma^!$, and (B) lower bounds on values of head infosets assuming no deviation occurs. If the $\tilde{\boldsymbol{\xi}}_j$ satisfies the LP, then the above expression is non-positive. These nested maximizations can be rewritten as the maximization of a linear function over a polytope with a polynomial number of constraints. For each $i \in \{1,2\}$ we introduce multipliers for each of the maximizations: $\lambda^{\delta}_{i,\sigma^!} \forall \sigma^! \in \check{\Sigma}_{i,j}$, $\lambda^{\nu}_{i,I,\sigma^!}$, $\lambda^{v}_{i,I}$.

$$\max_{\substack{i, \lambda^{\delta}_{i,\sigma^!}, \lambda^{\nu}_{i,I,\sigma^!}, \lambda^{v}_{i,I} \\ \sum(\sum \lambda^{\delta} + \sum \lambda^{\nu} + \lambda^{v} = 1}} \left\{ \begin{array}{c} \lambda^{\delta}_{i,\sigma^!}(\delta^*(\tilde{\boldsymbol{\xi}}_j; \sigma^!) - \delta^*(\boldsymbol{\xi}_0; \sigma^!)) + \\ \lambda^{\nu}_{i,I,\sigma^!}(\nu(I, \tilde{\boldsymbol{\xi}}_j; \sigma^!) - \hat{\nu}(I; \sigma^!)) + \\ \lambda^{v}_{i,I}(\breve{v}(I) - v(\tilde{\boldsymbol{\xi}}_j; I)) \end{array} \right\}$$

Optimizing over ν is simply finding the best-deviating response to a trigger sequence $\sigma^!$ over a polytope $Y_{i,\sigma^!}$ using the sequence form representation [178]. Given μ as well as the bounds are constants, the expression can be written as a single *linear* maximization over the multipliers and $y \in Y_{i,\sigma^!}$. Thus, we can rewrite the entire expression into a single bilinear maximization problem over $\tilde{\boldsymbol{\xi}}$ and the multipliers:

$$\min_{\tilde{\boldsymbol{\xi}}_j} \max_{i,\lambda,y} \left\{ \begin{array}{c} \sum\limits_{i,\sigma^! \in \check{\Sigma}_{i,j}} \left[\tilde{\boldsymbol{\xi}}_j^T R^{\delta}_{\sigma^!} z^{\delta}_{i,\sigma^!} + \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{\delta}_{i,\sigma^!} b^{\delta}_{i,\sigma^!} \right) \right] + \\ \sum\limits_{\substack{i,(I,\sigma^!,\cdot) \\ \in \hat{\mathcal{B}}_{i,j}}} \left[\tilde{\boldsymbol{\xi}}_j^T R^{\nu}_{I,\sigma^!} z^{\nu}_{i,\sigma^!} + \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{\nu}_{i,\sigma^!} b^{\nu}_{i,\sigma^!} \right) \right] + \\ \sum\limits_{i,(I,\cdot) \in \breve{B}_j} \tilde{\boldsymbol{\xi}}_j^T \left(\lambda^{v}_{i,I} b^{v}_{i,I} \right) \end{array} \right\},$$

where $z^{\delta}_{i,\sigma^!} = \lambda^{\delta}_{i,\sigma^!} \check{y}_{i,\sigma^!}$ and $z^{\delta}_{i,\sigma^!} = \lambda^{\nu}_{i,\sigma^!} \hat{y}_{i,\sigma^!}$, for appropriately chosen constants R, b (which may vary on $\boldsymbol{\xi}_0$). Hence, we can treat the refinement problem as a *zero-sum* game between a *mediator*, who chooses a refinement $\tilde{\boldsymbol{\xi}}_j$ and *deviator*, who chooses multipliers and best-deviating responses. This zero-sum game can be solved by running self-play between two Hannan-consistent regret minimizers and taking average strategies. A regret minimizer for the deviator can be constructed efficiently using existing techniques [62] or simply CFR [193].

We now briefly describe what R and b contain. R^{δ} and R^{ν} are constants for each trigger sequence $\sigma^!$, such that for a best response (weighted by λ^{δ} and λ^{ν}), $\tilde{\boldsymbol{\xi}}_j^T R^{\delta} z^{\delta}_{i,\sigma^!}$ gives the largest possible reward for deviating, and in the case of ν the value of the head infoset. b contains two components (i) the bound (either upper/lower or safety bounds for in-subgame deviations), and (ii) the value of sequences /infosets assuming best-responses.

Decomposition of Ξ_j Using Scaled Extensions

We briefly describe the decomposition algorithm of Farina et al. [64]. The reader is directed there for more details. Our first component is the *scaled extension operator*, introduced in [64] is a convexity preserving operation between sets. It was used to incrementally extend the strategy space Ξ in a top-down, rather than bottom-up fashion.

Definition 12. Scaled Extension (Farina et al. [64]) Let $\mathcal{X}$ and $\mathcal{Y}$ be non-empty, compact, and convex sets, and let $h : \mathcal{X} \to \mathbb{R}_+$ be a nonnegative affine real function. The scaled extension of $\mathcal{X}$ with $\mathcal{Y}$ via h is defined as the set

$$\mathcal{X} \lhd_h \mathcal{Y} := \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \in \mathcal{X}, y \in h(\boldsymbol{x})\mathcal{Y}\}$$

Scaled extensions preserve convexity, non-emptiness, and compactness of sets.

Expressing Ξ using scaled extensions. The idea is that some of the structural constraints of Ξ were redundant.

Example 6: Suppose we were trying to express Ξ in Figure 5.1. In top-down fashion, we begin with $\xi[\varnothing, \varnothing]$. Now, we look at the constraints that sum to $\xi[\varnothing, \varnothing]$. This gives 3 options: expand block 2, or blocks 11 or 12. Let us expand block 2. By 'expand' what we do is to apply the scaled extension using a set $\mathcal{Y}$ which is a simplex of size 2. The function h is chosen to be 0 everywhere except for the index which we are expanding from (in this case $\xi[\varnothing, \varnothing]$. It increases the 'size' of the set by 2 dimensions. We can see that the scaled extension have handled the structural constraints so far. It turns out we can repeatedly apply this and fill in blocks 3-8 the same way, and in that order. Now, we need to fill-in blocks 9-12. It turns out, however, that the constraints which involve blocks 9-12 already explicitly fix those entries. That is, there is no longer any degree of freedom for those entries: they can be inferred from the other entries. Crucially, note that block 9 is uniquely determined by blocks 4 and 7, without any inconsistencies or clashes. Since this is the case, we will express 9-12 as a sum of entries in the partially built Ξ. Again, this can be done using the scaled extension, by setting $\mathcal{Y}$ to be a singleton, but choosing $h = 0$ except for the indices we want to sum *from*.

The order of expansion in Example 6 was carefully chosen. For example, if we had expanded 11 and 12 first, followed by blocks 9-10. However, in almost all cases, we have painted ourselves into the corner: block 2's constraints are such that it needs to be summed to by both 9 and 10. For example we set columns $r_x, r_y = 0$, and expanded $[\varnothing, \ell_x]$, $[\varnothing, \ell_y] = 1$. Next, we expanded downwards we set $[G, \ell_x] = 1$ and $[B, \ell_y] = 1$. The, we try to 'backfill' 2 using the constraints that involve 2. This would end up as having $[G, \varnothing]$ and $[B, \varnothing]$ both being equal to 1. But this in turn means that they sum to 1 (the structural constraints of $[\varnothing, \varnothing]$). This shows that the order of expansion is crucial: not all will work well. In this example, the choice of expanding block 2 rather than blocks 11, and 12 was crucial. It turns out that a 'good' order of decomposition always exists in games without chance.

Definition 13. (Critical infosets Farina et al. [64]) Let (σ_1, σ_2) be a relevant sequence pair and let $I_1 \in \mathcal{I}_1$ be an infoset for P_1 such that $\sigma(I_1) = \sigma_1$. Inofset I_1 is called critical for σ_2 if there exists at least one $I_2 \in \mathcal{I}_2$ with $\sigma(I_2) = \sigma_2$ such that $I_1 \rightleftharpoons I_2$. A symmetric definition holds for $I_2 \in \mathcal{I}_2$.

A key result of [64] was that in games without chance, for any relevant sequence pair (σ_1, σ_2) at least one player has at most one critical infoset for the opponent's sequence. That player is called the *critical player*.

The DECOMPOSE subroutine This expands a sequence pair (σ_1, σ_2) and adds it into $\mathcal{X}$. It is recursive in nature and comprises 3 main steps. The reader is directed to Farina et al. [64] for a more detailed explanation.

1. Find a critical player from (σ_1, σ_2), where $\sigma_i = (I_i, a_i)$. This is guaranteed to exist. WLOG let that player be P_1. Then, expand all infosets I if $\sigma_2 = \varnothing$ or $I_1 \rightleftharpoons (I_2, a_2)$, and $\sigma(I) = \sigma_1$. Each expansion is done using the scaled extension, with h being 0 everywhere and a 1 in the index of admission.
2. For each sequence σ' immediately under I, call DECOMPOSE(σ', σ_2). After this step, all indices in $\{(\sigma_1, \sigma_2') | \sigma_2' \sqsupset \sigma_2\}$.
3. We perform backfilling of structural constraints $\{(\sigma_1, \sigma_2' | \sigma_2' \sqsupset \sigma_2\}$. First, if the critical player does have a critical infoset, then we will backfill—there are no longer any degrees of freedom. We will assign the value based on the constraint. If there is no critical infoset, then this we split by attaching more scaled extensions to simplexes.

A key tiebreaking rule. We perform the decomposition of Ξ in the same order one would if we were performing for the full game [64], except that we terminate whenever encountering a sequence pair $(\sigma_1, \sigma_2) \notin \hat{\mathsf{G}} \cup \check{\mathsf{G}}_j$. We show that this process expresses Ξ_j and can be viewed as a series of scaled extensions. Consequently, there exists a regret minimizer over Ξ_j.

However, there is an additional tiebreaking element which we may encounter in the decomposition algorithm: when faced with a choice to expand a sequence pair, we should prioritize sequence pairs which do *not* lie in subgames over those which do. This prioritization is natural, since we do not want to fill in sequence pairs in the subgame before pre-subgame sequence pairs. This tiebreaking rule will not interfere with the rest of the decomposition algorithm.

Example 7: Consider a 2×2 matrix game. Here, actions are simply sequences. We call the sequences for P_i, $\sigma_{i,1}$ and $\sigma_{i,2}$. Now the matrix game can be expressed in terms of an EFG. We assume player moves first, and takes an action. After that, P_2 takes his action, but without knowledge of P_1's action. This is achieved using an information set that spans over the 2 states after P_1 took his action. Now, let us suppose a single subgame which contains only P_2's actions, i.e., P_1 making his move was pre-subgame. This in turn means that the relevant sequence pairs $(\sigma_{1,1}, \varnothing)$ and $(\sigma_{1,2}, \varnothing)$ are not in a subgame (i.e., in a refinement, they are supposed to follow the blueprint), while all other sequence pairs (except for $(\varnothing, \varnothing)$) are in the subgame.

Now, if were to just apply the expansion order of Farina et al. [64], we would have two options: either expand $\sigma_{1,1}$ and $\sigma_{1,2}$ first, or there other way round. The reader may verify that expanding $\sigma_{1,1}$ and $\sigma_{1,2}$ first works fine. That is, we can expand $\sigma_{1,1}$ and $\sigma_{1,2}$, followed by (σ_1, σ_2), since neither σ_1 and σ_2 are the empty sequence. Then, we can backfill $\sigma_{2,1}$ and $\sigma_{2,2}$. However, if we were to expand alongside P_2, i.e., $\sigma_{2,1}$ and $\sigma_{2,2}$, then a nasty situation occurs. After filling up the correlated non-empty sequence pairs, we have to backfill $\sigma_{1,1}$ and $\sigma_{1,2}$. This is not possible, since $\sigma_{1,1}$ and $\sigma_{1,2}$ were part of the blueprint! In general, we are not permitted to perform the backfill when the 'source' is in a subgame and the 'destination' is pre-subgame.

The solution to the problem in Example 7 is simple: when there is a tie as to which sequence pairs should be expanded, always select the one that is pre-subgame. Can this simple solution always work? If we expanded the infosets that are not in subgames first before the other player's infoset (which is in a subgame), then we can be sure that in the backfill step we will *never* fill in a sequence pair in $\hat{\mathsf{G}}$ with sum of sequence pairs in $\check{\mathsf{G}}_j$. However, can this expansion rule ever conflict with the expansion rule of [64]—i.e., expanding infosets belonging to the critical player? It turns out this clash will never occur.

Theorem 8. *Let (σ_1, σ_2) be a relevant sequence pair and I_1, I_2, I_2' be infosets such that* ***Seq**$(I_1) = \sigma_1$,* ***Seq**$(I_2) = \sigma_2$,* ***Seq**$(I_2') = \sigma_2$, $I_1 \rightleftharpoons I_2$ and $I_1 \rightleftharpoons I_2'$ (implying that I_1 is a critical infoset for (σ_1, σ_2)). It cannot be that I_1 belongs to some subgame but either or both of I_2, I_2' do not.*

Proof. First, note that if any of I_2 or I_2' lies in a subgame, it must be the same one as I_1, there exists a path starting from the root passing through I_1 and that infoset (I_2 or I_2'). Now, suppose WLOG that I_2 is not in a subgame but I_1 is. We will demonstrate a contradiction.

First, we know that there exists a state $h_2 \in I_2$ and a state $h_1 \in I_1$ where $h_2 \prec h_1$. This is because $I_2 \rightleftharpoons I_1$. It cannot be that $h_1 \prec h_2$, since that would imply that I_2 must belong to a a subgame (which contradicts our assumption). Since $I_2' \rightleftharpoons I_1$, there exists a state $h_2' \in I_2$ and $h_1' \in I_1$ which are along a path from the root (though this time, we do not know which precedes the other). Let w be the lowest-common-ancestor of h_2 and h_2'. The state w cannot be a chance node (since G has no chance). It also cannot belong to P_1, since this would mean that h_1 and h_1'

are in different infosets (they must have been the result of different actions of P_1 at w). Hence, w must belong to P_2. But that is also impossible, since we assumed from the beginning that $\mathbf{Seq}(I_2) = \mathbf{Seq}(I_2') = \sigma_2$. Clearly this cannot be the case since they have different preceding sequences starting from w. $\square$

Put together, this means that the rules of expansion will not conflict. If there ever needs to be backfilling, it will be either entirely within or outside a subgame, or backfilling entries within a subgame from entries before a subgame, and not the other way around (which will violate the equality-to-blueprint constraints).

5.4 Experiments

We evaluate our algorithms using the LP-based and regret minimization-based refining. We use the benchmark game of EFCE called *Battleship*, introduced by Farina et al. [63]. This game is played in 2 stages. In the *placement* stage, players privately place their ship(s) of size 1 by m on $W \times H$ grid. In the *firing* stage, players take turns firing at each other over T timesteps, or until a player's ship is destroyed. Each shot is at a single tile, and a ship is considered destroyed when all tiles in the ship are shot at least once. Locations of shots are known to both players, and cannot be made at the same location more than once. A player gets 1 point for destroying the opponent's ship, but loses $\gamma > 1$ points if its ship is destroyed (i.e., players are afraid of losing more than they desire to win). If no ship is destroyed by the end of the game, the game ends in a tie and both players get 0. Naturally, the NE is for players to place and fire their ships uniformly at random[6]. It was found by Farina et al. [63] that in a SW-maximizing EFCE, mediators can recommend players to miss their shots (i.e., shoot into the unoccupied areas), leading to more frequent peaceful resolutions *while remaining incentive compatible*. The intuitive explanation given is that when players who deviate from the recommendation to fire in the sea will be "punished" by the mediator by having their ship's location revealed to the opponent.

We use 2 different correlation blueprints for our experiments, *Uniform* and *Jittered*. Both correlation plans are based on *independent* player strategies stored using the sequence form. That is, $\xi_0[\sigma_1, \sigma_2] = \xi_0^{(1)}(\sigma_1) \cdot \xi_0^{(2)}(\sigma_2)$, where $\xi_0^{(1)}, \xi_0^{(2)}$ are sequence form strategies for each player. In *Uniform*, $\xi_0^{(i)}$ have actions uniformly at random at each infoset. In *Jittered*, each player has randomly generated behavioral strategies. Here, for infoset $I \in \mathcal{I}_i$, action $a_j \in \mathcal{A}(I)$ is played with probability $p(a_j; I) = \kappa_{I,j} / \sum_k \kappa_{I,k}$, with $\kappa_{I,k} = 1 + w \cdot \varepsilon_{I,k}$, where each $\varepsilon_{I,k}$ is drawn independently and uniformly from $[-1, 1]$ and $w \in [0, 1]$ is a width parameter governing the level of deviation from uniform strategies. Note that ξ_0 need not be independent in general. However, independent strategies are more compactly represented (see Remark 19). Indeed, since $\boldsymbol{\xi}_0$ is large, it is unrealistic to represent a blueprint explicitly without exploiting some form of game abstraction.

Subgames are defined based on public information, which at the k-th step of firing are precisely the locations fired by each player. We base subgames on the shot history up till timestep $T' < T$. T' balances the trade-off between accuracy versus computational costs. For a grid of size n, we have $J = \prod_{k=T-T'+1}^{T} k^2$ subgames. When T' is small, we have fewer subgames, but

[6]at least when ships are of size 1.

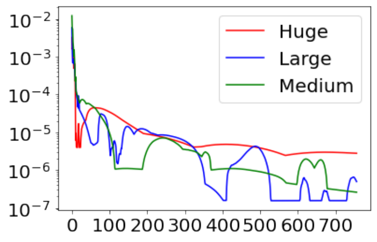
n, T J	$\|\Xi_j\|$	γ	*Uniform* BP	*Uniform* Refined	*Jittered* BP	*Jittered* Refined
$3, 2,$ 9	382	2	-3.70	-3.70	-3.55	-3.55
		5	-14.8	-14.8	-14.2	-14.2
$4, 3,$ 16	3.2e3	2	-3.13	-2.95	-3.24	-3.10
		5	-12.5	-11.4	-13.0	-11.8
$5, 3,$ 25	2.3e4	2	-1.92	-1.34	-1.95	-1.25
		5	-7.68	-4.80	-7.82	-4.32
$6, 3,$ 36	1.2e5	2	-1.23	-.772	-1.25	-.627
		5	-4.94	-2.47	-4.99	-1.95

Table 5.1: Comparison of social welfare between blueprint (BP) and SW-maximizing safe refinement with ships of size 1. Social welfare is reported at a scale of 1e-2.

can achieve better social welfare. All experiments are run on an Apple M1 Chip with 16GB of RAM with 8 cores. LPs are solved using Gurobi [79].

Safe resolving with SW maximization We first show using our LP-based method that ensures fully safe resolving can lead to significantly higher social welfare as compared to the blueprint. We set $T' = 1$ and we use ships with $m = 1$, i.e., the game is over once any ship is hit. Consequently, the game is entirely symmetric in terms of location. The NE here is to play and shoot uniformly at random. Hence, *Uniform* is a valid, though not SW-optimal EFCE. Under *Uniform*, the exploitability δ^* under the blueprint is 0, implying that the complete refinement $\tilde{\boldsymbol{\xi}}$ is also an EFCE. We perform refinement on the first subgame (this without loss of generality due to symmetry) and compare the SW accumulated from the subgame under the blueprint and refinement. For *Jittered*, we repeated the experiment 10 times with different seeds and report the mean. The results are reported in Table 5.1. In all our experiments, our refined strategy $\tilde{\boldsymbol{\xi}}_j$ gives a much higher SW. For example, in the largest example with $\gamma = 2$, SW increases by 4.6e-3. *This is not a negligible improvement*; since this is applied to all 36 subgames, the expected improvement in SW of the complete refinement $\tilde{\boldsymbol{\xi}}$ is actually 0.167. $|\Xi_j|$ is significantly smaller than $|\Xi|$, such that each refinement is computed in no more than 10 seconds.

Safe resolving using regret minimization We now demonstrate the scalability of refinement based on regret minimization. Our goal here is to demonstrate that subgame resolving can be performed efficiently for games that are too large for $\boldsymbol{\xi}$ to even be stored in memory. We run refinement using our regret minimization algorithm and report the "pseudo"-exploitability of $\tilde{\boldsymbol{\xi}}_j$ (i.e., the value of the inner maximization over (i, λ, y), (5.15), or the most violated incentive constraint of the LP). We use $T' = 1, \gamma = 2$ and the *Uniform* blueprint. The results are reported in Figure 5.2. Our huge instance is several times larger than the largest instance in Farina et al. [64], and it would require a significant amount of memory to store a full correlation plan $\tilde{\boldsymbol{\xi}}$. We find that in practice, resolving requires less than 0.5 seconds per iteration, while using no more than 2GB of memory.



	Med.	Large	Huge
W, H	3,2	3,2	3,2
T	4	4	5
m	1	2	2
$\lvert\Xi\rvert$	3.89M	111M	360M

Figure 5.2: Left: Most violated incentive constraint of $\tilde{\xi}$ plot against iteration number. Right: Parameters of game.

5.5 Conclusion

In this chapter, we have proposed a novel subgame resolving technique for EFCE. We offer two algorithms, the first based on LPs and the second uses regret minimization, both of which consume significantly less compute than full-game solvers. Our technique is, to the best of our knowledge, the first *online* algorithm towards solving EFCE.

As another variant of subgame solving for general-sum EFGs, many of the shortcomings in Chapter 4 hold, especially with respect to bounded rationality and scalability. The latter is a particularly serious problem, since correlation plans are much larger than independent strategies.

Other future directions include relaxing the assumption that the game is without chance, as well as extensions to other variants of correlated equilibria [138]. It would also be interesting to extend subgame resolving (with similar types of guarantees) to multiplayer ($>$ 2 players) EFCE [39, 65], where more modern solvers are *decentralized* and often more efficient than the methods in [64]. It is also worth exploring (perhaps in certain important classes of games), as to how suboptimal subgame solving is compared to an optimal solution, either with respect to incentive compatibility or social welfare. Another important direction is to explore using function approximation in a fashion similar to DeepStack [136], in fact, we explore this direction a little in Chapter 6.

Chapter 6

Function Approximation for Stackelberg Equilibrium

6.1 Introduction

A central challenge in modern game solving is to handle large game trees, particularly those too large to traverse or explicitly specify. These include board games like Chess, Poker [13, 27, 31, 76, 136, 159, 160] and modern video games with large state and action spaces [174]. Today, scalable game solving is frequently achieved via function approximation (FA), typically by using neural networks to model state values and harnessing the network's ability to generalize its evaluation to states never encountered before [136, 157, 159, 160]. Methods employing FA have achieved not only state-of-the-art performance, but also exhibit more human-like behavior [97].

Surprisingly, FA is rarely applied to solution concepts used in general-sum games such as Stackelberg equilibrium, which are generally regarded as being more difficult to solve than the perfectly cooperative/competitive Nash equilibrium. Indeed, the bulk of existing literature centers around methods such as exact backward induction [19, 20], incremental strategy generation [40, 41, 91, 96], and mathematical programming [18].[1] While exact, these methods rarely scale to large game trees, especially those too large to traverse, severely limiting our ability to tackle general-sum games that are of practical interest, such as those in security domains like wildlife poaching prevention [60] and airport patrols [149]. However, for many equilibrium concepts in general-sum games, the value of a state often *cannot* be summarized as a scalar (or fixed sized vector), rendering the direct application of FA-based zero-sum solvers like [160] infeasible.

In this paper, we propose applying FA to model the *Enforceable Payoff Frontier* (EPF) for each state and using it to solve for the *Stackelberg extensive-form correlated equilibrium* (SEFCE) in two-player games of perfect information. Introduced in [19, 20, 114], EPFs capture the tradeoff between player payoffs and is analogous to the state value in zero-sum games.[2] Specifically, we (i) study the pitfalls that can occur with using FA in general-sum games, (ii) propose

[1]Meta-game solving [110, 183] is used in zero-sum games, but not general-sum Stackelberg games.

[2]The idea of an EPF was initially used by [114] to give a polynomial time solution for SSEs. However, they (as well as other work) do not propose any naming.

a method for solving SEFCEs by modeling EPFs using neural networks and minimizing an appropriately designed Bellman-like loss, and (iii) provide guarantees on incentive compatibility and performance of our method. Our approach is the first application of FA in Stackelberg settings without relying on best-response oracles for performance guarantees. Experimental results show that our method can (a) approximate solutions in games too large to explicitly traverse, and (b) generalize learned EPFs over states in a repeated setting where game payoffs vary based on features.

6.2 Preliminaries and Notation

A 2-player *perfect information* game without chance G is represented by a finite game tree with game states $s \in \mathcal{S}$ given by vertices and action space $\mathcal{A}(s)$ given by directed edges starting from s. Each state belongs to either player P_1 or P_2; we denote these disjoint sets by $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. Every leaf (terminal state) $\ell \in \mathcal{L} \subseteq \mathcal{S}$ of G is associated with payoffs, given by $r_i(\ell)$ for each player i. Taking action $a \in \mathcal{A}(s)$ at state $s \notin \mathcal{L}$ leads to $s' = T(a; s)$, where $s' \in \mathcal{S}$ is the next state and T is the deterministic transition function. Let $\mathcal{C}(s) = \{s' \mid T(a; s) = s', a \in \mathcal{A}(s)\}$ denote the immediate children of s. We say that state s precedes ($\sqsubset$) state s' if $s \neq s'$ and s is an ancestor of s' in G, and write $\sqsubseteq$ if allowing $s = s'$. An action $a \in \mathcal{A}(s)$ *leads to* s' if $s \sqsubset s'$ and $T(a; s) \sqsubseteq s'$. With a slight abuse of notation, we denote $T(a; s) \sqsubseteq s'$ by $a \sqsubset s'$ or $(s, a) \sqsubset s'$. Since G is a tree, for states s, s' where $s \sqsubset s'$, exactly one $a \in \mathcal{A}(s)$ such that $(s, a) \sqsubseteq s'$. We use the notation $\sqsupseteq$ and $\sqsupset$ when the relationships are reversed. Since G has perfect information, there is no meaningful distinction between the relationships $\sqsubseteq$ and $\preceq$, and we stick to $\sqsubseteq$ throughout this chapter.

Behavioral Strategies. In this chapter, it will be convenient to use the behavioral representation for strategies. A strategy $\pi_i, i \in \{1, 2\}$, is a mapping from state $s \in \mathcal{S}_i$ to a distribution over actions $\mathcal{A}(s)$, i.e., $\pi_i(a, s) \geqslant 0$ and $\sum_{a \in \mathcal{A}(s)} \pi_i(a; s) = 1$ or equivalently, $\pi_i(\cdot, s) \in \Delta_{\mathcal{A}(s)}$. Given strategies π_1 and π_2, the probability of reaching $\ell \in \mathcal{L}$ starting from s is given by the product $p(\ell|s; \pi_1, \pi_2) = \prod_{i \in \{1,2\}} \prod_{(s',a); s \sqsubseteq s', (s',a) \sqsubset \ell, s' \in \mathcal{S}_i} \pi_i(a; s')$, and player i's expected payoff starting from s is $R_i(s; \pi_1, \pi_2) = \sum_{\ell \in \mathcal{L}} p(\ell|s; \pi_1, \pi_2) r_i(\ell)$. We use as shorthand $p(\ell; \pi_1, \pi_2)$ and $R_i(\pi_1, \pi_2)$ if s is the root. A strategy π_2 is a *best response* to a strategy π_1 if $R_2(\pi_1, \pi_2) \geqslant R_2(\pi_1, \pi_2')$ for all strategies π_2'. The set of best responses to π_1 is written as $\mathsf{BR}_2(\pi_1)$.

The *grim strategy* $\arg\min_{\pi_1} \max_{\pi_2} R_2(\pi_1, \pi_2)$ of P_1 towards P_2 is one which guarantees the lowest payoff for P_2. Conversely, the *joint altruistic strategy* $\arg\max_{\pi_1, \pi_2} R_2(\pi_1, \pi_2)$ is one which maximizes P_2's payoff. We restrict grim and altruistic strategies to those which are subgame-perfect, i.e., they remain the optimal if the game was rooted at some other state.[3] Grim and altruistic strategies ignore P_1's own payoffs and can be computed by backward induction. For each state, we denote by $\underline{V}(s)$ and $\overline{V}(s)$ the internal values of P_2 for grim and altruistic strategies obtained via backward induction.

[3]This is to avoid strategies which play arbitrarily at states which have 0 probability of being reached.

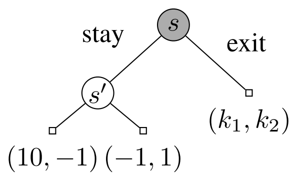
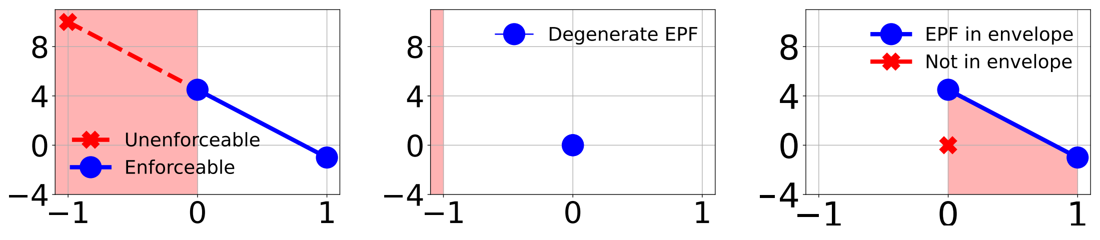


Figure 6.1: Toy example. Leader ○, follower ● and leaf ▫ states are vertices while edges are actions. Payoffs at leaves $\ell \in \mathcal{L}$ are given by $(r_1(\ell), r_2(\ell))$, i.e., the leader's payoff comes first.



(a) EPF at vertex s'. (b) EPF after exiting. (c) EPF at root vertex s.

Figure 6.2: Example EPFs based on the game in Figure 6.1. The x and y axes are follower (μ_2) and leader payoffs ($U_s(\mu)$). In (a) and (b) the pink regions give P_2 too little reward and are truncated. In (c), the pink region is not part of the upper concave envelope and hence removed.

6.2.1 Stackelberg Equilibrium in Perfect Information Games

In a Strong Stackelberg equilibrium (SSE), there is a distinguished *leader* and *follower*, which we assume are P_1 and P_2 respectively. The leader *commits* to any strategy π_1 and the follower best responds to the leader, breaking ties by selecting $\pi_2 \in \mathsf{BR}_2(\pi_1)$ such as to benefit the leader. [4] Solving for the SSE entails finding the optimal commitment for the *leader*, i.e., a pair $\pi = (\pi_1, \pi_2)$ such that $\pi_2 \in \mathsf{BR}_2(\pi_1)$ and $R_1(\pi_1, \pi_2)$ is to be maximized.

It is well-known that the optimal SSE will perform no worse (for the leader) than Nash equilibrium, and often much better. Consider the game in Figure 6.1 with $k_1 = k_2 = 0$. If the expected follower payoff from staying is less than 0, then it would exit immediately. Hence, solutions such as the subgame perfect Nash gives a leader payoff of 0. The optimal Stackelberg solution is for the leader to commit to a uniform strategy—this ensures that staying yields the follower a payoff of 0, which under the tie-breaking rules of SSE nets the leader a payoff of 4.5.

Stackelberg Extensive-Form Correlated Equilibrium For this paper, we will focus on a relaxation of the SSE known as the Stackelberg extensive-form correlated equilibirum (SEFCE) [20], which allows the leader to explicitly recommend actions to the follower *at the time of decision making*. If the follower deviates from the recommendation, the leader is free to retaliate—typically with the grim strategy. In a SEFCE, P_1 takes and recommends actions to maximize its reward, subject to the constraints that the recommendations are sufficiently appealing to P_2 relative to threat of P_2 facing the grim strategy after deviating.

[4]Commitment rights are justified by repeated interactions. If the P_1 reneges on its commitment, P_2 plays another best response, which is detrimental to the leader. This setting is unlike [54] which uses binding agreements.

Definition 14 (Minimum required incentives)**.** Given $s \in \mathcal{S}_2$, $s' \in \mathcal{C}(s)$, we define the *minimum required incentive* $\tau(s') = \max_{s^! \in \mathcal{C}(s); s^! \neq s'} \underline{V}(s^!)$, i.e., the minimum amount that P_1 needs to *promise* P_2 under s' for it to be reached.

Definition 15 (Stackelberg Extensive Form Correlated Equilibrium)**.** A strategy pair $\pi = (\pi_1, \pi_2)$ is a Stackelberg Extensive Form Correlated equilibrium (SEFCE) if it is *incentive compatible*, i.e., for all $s \in \mathcal{S}_2$, $a \in \mathcal{A}(s)$, $\pi_2(a; s) > 0 \implies R_2(T(a; s); \pi_1, \pi_2) \geqslant \tau(T(a; s))$. Additionally, π is *optimal* if $R_1(\pi_1, \pi_2)$ is maximized.

In Section 6.3, we describe how optimal SEFCE can be computed in polynomial time for perfect information games by backward induction.

Remark 21: In Definition 15 we have chosen to distinguish between incentive compatibility and optimality. This is unlike the definition used in Section 2.4.3. This distinction is useful, since we will want to find (π_1, π_2) which are incentive compatible but not necessarily optimal (due to approximation errors). Note that the optimality of SEFCE implies the leader-favored "strong" tie-breaking aspect seen in SSE.

Remark 22 (Connection to other equilibrium concepts): Stackelberg equilibrium can be seen as a modification of the Nash equilibrium, where the leader's incentive constraints are removed in lieu of having an objective to maximize the leader's payoff. Conitzer and Korzhyk [46] introduced the analogous idea of committing to *correlated strategies* for normal form game. One begins with correlated equilibrium constraints then, omits the leader incentive constraints and then maximizes the leader's utility. The SEFCE follows the exact same process but for EFCE constraints (which were introduced in Chapter 5). The complexity of SEFCE is significantly lower in this chapter because of our restriction to perfect information games.

Remark 23: A rule of thumb is that chance and imperfect information each make equilibrium-finding more difficult. On the other hand, correlation makes the problem easier. If G has imperfect information and chance, solving for SEFCE and SSE is NP-hard. If G has perfect information but with chance, then finding SSE remains NP-hard, but SEFCE may be found in polynomial time. When the game has neither perfect information nor chance, then both SSE and SEFCE may be found in polynomial time. Finer grained analysis of computational complexities (for example, in stochastic games) can be found in [19] and our full paper [123].

6.2.2 Function Approximation of State Values

When finding Nash equilibrium in perfect information games, the *value* v_s of a state is a crucial quantity which summarizes the utility obtained from s onward, assuming optimal play from all players. It contains sufficient information for one to obtain an optimal solution after using them to 'replace' subtrees. Typically v_s should only rely on states $s' \sqsupseteq s$. In zero-sum games, $v_s = \underline{V}_s$ while in cooperative games, $v_s = \overline{V}_s$. Knowing the true value of each state immediately enables the optimal policy via one-step lookahead. While v_s can be computed over all states by backward induction, this is not feasible when G is large. A standard workaround is to replace v_s with an approximate $\tilde{v}_s$ which is then used in tandem with some search algorithm (depth-limited search, Monte-Carlo tree search, etc.) to obtain an approximate solution. Today, $\tilde{v}_s$ is often *learned*. By representing $\tilde{v}$ with a rich function class over state features (typically using a neural network), modern solvers are able to generalize $\tilde{v}$ across large state spaces without explicitly visiting every state, thus scaling to much larger games.

Fitted Value Iteration. A class of methods closely related to ours is Fitted Value Iteration (FVI) [57, 108, 139]. The idea behind FVI is to optimize for parameters such as to minimize the *Bellman loss* over sampled states by treating it as a regular regression problem. [5] The Bellman loss measures the distance between $\tilde{v}_s$ and the estimated value using one-step lookahead based on $\tilde{v}$. If this distance is 0 for all s, then $\tilde{v}$ matches the optimal v. However, small errors in FA accumulate and cascade across states. Thus, it is important to bound performance as a function of the Bellman loss over all s.

6.2.3 Related Work

Some work has been done in generalizing state values in general-sum games, but few involve learning them. Related to ours is [55, 127, 140], which approximate the achievable set of payoffs for correlated equilibrium, and eventually SSE [116] in stochastic games. These methods are analytical in nature and scale poorly with the state space. [78, 146] propose a Q-learning-like algorithm over general-sum Markov games, but do not apply FA and only consider stationary strategies which preclude strategies involving long range threats like the SSE. [192] show a class of general-sum Markov games where value-iteration like methods will necessarily fail. [190] study reinforcement learning in the Stackelberg setting, but only consider followers with myopic best responses. [38] apply FVI in a multiobjective setting, but do not consider the issue of incentive compatibility. Another approach is to apply reinforcement learning and self-play [112]. Recent methods account for the nonstationary environment each player faces during training [67, 147]; however they have little game theoretical guarantees in terms of incentive compatibility, particularly in non zero-sum games.

6.3 Review: Solving Stackelberg Equilibrium via Enforceable Payoff Frontiers

In Section 6.2, we emphasized the importance of the value function v in solving zero-sum games. In this section, we review the analogue for SEFCE in the general-sum games, which we term as *Enforceable Payoff Frontiers* (EPF), although they were introduced by Letchford and Conitzer [114] while studying the complexity of solving Stackelberg equilibrium in EFGs,

Definition 16 (Informal). The Enforceable Payoff Frontier at state s is a *function* $U_s : \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty\}$, such that $U_s(\mu_2)$ gives the maximum leader payoff for a SEFCE (possibly suboptimal!) for a game rooted at s, on condition that P_2 obtains a payoff of μ_2. By convention, $U_s(\mu_2) = -\infty$ if P_2 cannot obtain a payoff of μ_2.

[5]We distinguish RL and FVI in that the transition function is known explicitly and made used of in FVI.

All leaves $s \in \mathcal{L}$ have degenerate EPFs $U_s(r_2(s)) = r_1(s)$ and $-\infty$ everywhere else. EPFs capture the tradeoff in payoffs between P_1 and P_2, making them ideal candidates for solving SEFCEs. We now review the two-phase algorithm of [20] using the example game in Figure 6.1 with $k_1 = k_2 = 0$. This approach forms the basis for our proposed FA method.

Phase 1: Computing EPF by Backward Induction. The EPF at s' is given by the line segment connecting payoffs of its children EPF and $-\infty$ everywhere else. This is because the leader is able to freely mix over actions. To compute U_s, we first consider in turn the EPFs after staying or exiting.

- Case 1: P_1 is recommending P_2 to stay. For incentive compatibility, it needs to *promise* P_2 a payoff of at least 0 under $T(\text{stay}; s) = s'$. Thus, we **left-truncate** the regions of the EPF at s' which violate this promise, leaving behind the blue segment (Figure 6.2a). This represents the payoffs at s' that are *enforceable* by P_1.
- Case 2: P_1 is recommending P_2 to exit. To discourage P_2 from staying, it commits (i.e., threatens) to the grim strategy at s' if P_2 chooses to stay. This threat yields P_2 a payoff of $-1 \leqslant k_2 = 0$. Hence, no truncation is needed and the set of enforceable payoffs is the (degenerate) blue line segment (Figure 6.2b).

Finally, we recover U_s, using the two children EPFs of s. Observe that we could achieve payoffs on any line segment connecting point *across* the EPFs of s's children. This union of points on such lines (ignoring those leader-dominated) is given by the **upper concave envelope** of the blue segments in Figure 6.2a and 6.2b; this removes $\{(0, 0)\}$, resulting in the EPF shown in Figure 6.2c.

More generally, let g_1 and g_2 be functions such that $g_j : \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty\}$. We denote by $g_1 \bigwedge g_2$ their upper concave envelope, i.e., $\inf\{h(\mu) \mid h \text{ is concave and } h \geqslant \max\{g_1, g_2\} \text{ over } \mathbb{R}\}$. Since $\bigwedge$ is associative and commutative, we use as shorthand $\bigwedge_{\{\cdot\}}$ when applying $\bigwedge$ repeatedly over a finite set of functions. In addition, we denote $g \rhd t$ as the left-truncation of the g with threshold $t \in \mathbb{R}$, i.e., $[g \rhd t](\mu) = g(\mu)$ if $\mu \geqslant t$ and $-\infty$ otherwise. Note that both $\bigwedge$ and $\rhd$ are closed over concave functions. For any $s \in \mathcal{S}$, its EPF U_s can be concisely written in terms of its children EPF $U_{s'}$ (where $s' \in \mathcal{C}(s)$) using $\bigwedge$, $\rhd$ and $\tau(s')$.

$$U_s(\mu) = \begin{cases} \left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'}\right](\mu) & \text{if } s \in \mathcal{S}_1 \\ \left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'} \rhd \tau(s')\right](\mu) & \text{if } s \in \mathcal{S}_2 \end{cases}, \tag{6.1}$$

which we apply in a bottom-up fashion to complete Phase 1.

Phase 2: Extracting Strategies from EPF. Once U_s has been computed for all $s \in \mathcal{S}$, we can recover the optimal strategy π_1 by applying one-step lookahead starting from the root. First, we extract $(\mathsf{OPT}_2, \mathsf{OPT}_1)$, the coordinates of the maximum point in U_{root}, which contain payoffs under the optimal π. Here, this is $(0, 4.5)$. We initialize $\mu_2 = \mathsf{OPT}_2$, which represents P_1's promised payoff to P_2 at the current state s. Next, we traverse G depth-first. By construction, $U_s(\mu_2) > -\infty$ and the point $(\mu_2, U_s(\mu_2))$ is the convex combination of either 1 or 2 points belonging to its children EPFs. The mixing factors correspond to the optimal strategy $\pi(a; s)$. If there are 2 distinct children s', s'' with mixing factor α', α'', we repeat this process for s', s'' with

Algorithm 6: Training Pipeline

```
1: Sample trajectory $s_{\text{new}}^{(1)}, \ldots, s_{\text{new}}^{(t)}$
2: Update replay buffer $\mathcal{B}$ with $s^{(1)}, \ldots, s^{(t)}$
3: for $i \in \{1, \ldots, t\}$ do
4:     Sample batch $S = \{s^{(1)}, \ldots s^{(n)}\} \subseteq \mathcal{B}$
5:     $\ell \leftarrow \text{ComputeLoss}(S; E_\phi)$
6:     Update $\phi$ using $\partial\ell/\partial\phi$
7: end for
```

$\mu_2' = \mu_2/\alpha', \mu_2'' = \mu_2/\alpha''$, otherwise we repeat the process for s' and $\mu_2' = \mu_2$. For our example, we start at s, $\mu_2 = 0$, which was obtained by P_2 playing 'stay' exclusively, so we keep μ_2 and move to s'. At s', $\mu = 0$ by mixing uniformly, which gives us the result in Section 6.2.

Theorem 9 (Structure of EPFs [19, 20]). *Let s be a state in a perfect information EFG. Then, we have U_s is piecewise linear concave with number of knots[6] no greater than the number of leaves beneath s. Using backward induction, EPFs (and hence SEFCE) for all states can be computed in time (poly $(|\mathcal{S}|)$).*

Markovian Property. Just like state values v_s in zero-sum games, we can replace any internal vertex s in G with its EPF while not affecting the optimal strategy in all other branches of the game. This can done by adding a single leader vertex with actions leading to terminal states with payoffs corresponding to the coordinates of the knots of U_s. Since U_s is obtained via backward induction, it only depends on states beneath s. In fact, if two games G and G' (which could be equal to G) shared a common subgame rooted in s and s' respectively, we could reuse the U_s found in G for $U_{s'}$ in G'. This observation underpins the inspiration for our work—if s and s' are similar in some features, then U_s and $U_{s'}$ are likely similar and it should be possible to *learn* and *generalize* EPFs over states.

6.4 Challenges in Function Approximation for Enforceable Payoff Frontiers

We now return to our original problem of applying FA to find SEFCE. Our idea, outlined in Algorithm 6 and 7 is a straightforward extension of FVI. Suppose each state has features $f(s)$—in the simplest case this could be a state's history. We design a neural network $E_\phi(f)$ parameterized by ϕ. This network maps state features $f(s)$ to some representation of $\widetilde{U}_s$, the approximated EPFs. To achieve a good approximation, we optimize ϕ by minimizing an appropriate Bellman-like loss (over EPFs) based on Equation (6.1) while using our approximation $\widetilde{U}_s$ in lieu of U_s. Despite its simplicity, there remain several design considerations.

[6]Knots are where the slope of the EPF changes.

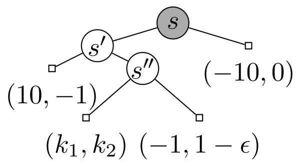
Algorithm 7: COMPUTELOSS$(S; E_\phi)$

```
1: for $i \in \{1 \dots n\}$ do
2:     $\widetilde{U}_{s^{(i)}} \leftarrow E_\phi(f(s^{(i)}))$
3:     $\widetilde{U}_{s_{\text{next}}^{(j)}} \leftarrow E_\phi(f(s_{\text{next}}^{(j)}))$   for all $s_{\text{next}}^{(j)} \in \mathcal{C}(s^{(i)})$
4:     Compute $\widetilde{U}_{s^{(i)}}^{\text{target}}$ using Equation (6.1) and $\{\widetilde{U}_{s_{\text{next}}^{(j)}}\}$
5: end for
6: return $\sum_i L(\widetilde{U}_{s^{(i)}}, \widetilde{U}_{s^{(i)}}^{\text{target}})$
```

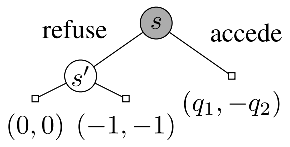
EPFs are the 'right' object to learn. Unlike state values, representing an exact EPF at a state s could require more than constant memory since the number of knots could be linear in the number of leaves underneath it (Theorem 9). Can we get away with summarizing a state with a scalar or a small vector? Unfortunately, any 'lossless summary' which enjoys the Markovian property necessarily encapsulates the EPF. To see why, consider the class of games G_k in Figure 6.1 with $k_1 = -2$ and $k = k_2 \in [-1, 1]$. The optimal leader payoff for any G_k is $\frac{9-11k}{2}$, which is precisely $U_{s'}(k)$ (Figure 6.2a). Now consider any lossless summary for s' and use it to solve *every* G_k. The resultant optimal leader payoffs can recover $U_{s'}(\mu_2)$ between $\mu_2 \in [-1, 1]$. This implies that no lossless summary more compact than the EPF exists.

Unfulfillable Promises Arising from FA Error. Consider the game in Figure 6.3a with $k_1 = -10$, $k_2 = -1$. The exact $U_{s'}$ is the line segment combining the points $(-1, 10)$ and $(1 - \epsilon, -1)$, shown in green in Figure 6.4a. Now suppose that due to function approximation errors we instead learned the blue line segment containing $(-1, 10)$ and $(1, -1)$. Performing Phase 2 using $\widetilde{U}$, the policy extracted at s' is once again the uniform policy and requires us to promise the follower a utility of 1 in s''. However, achieving a payoff of 1 is *impossible* regardless of how much the leader is willing to sacrifice, since the maximum outcome under s'' is $1 - \epsilon$. Since this is an *unfulfillable promise*, the follower's best responds by exiting in s, which gives the leader a payoff of -10. In general, unfulfillable promises due to small FA error can lead to arbitrarily low payoffs. Indeed, the problem arises because $\widetilde{U}$ does not even *define* an incentive compatible policy in G.

Costly Promises. Consider the case where $k_1 = -30, k_2 = 1$ while keeping $\widetilde{U}_{s'}$ the same. Here, the promise of 1 at s'' *is* fulfillable, but involves incurring a cost of -30, which is even lower than having follower staying (Figure 6.4b). In general, this problem of *costly promises* stems from the EPF being wrongly estimated, even for a small range of μ_2. We can see how costly promises arise even from small ϵ is. The underlying issue is that in general, U_s can have large Lipschitz constants (e.g., proportionate to $(\max_s r_1(s) - \min_s r_1(s)) / (\min |r_2(s) - r_2(s)|)$). The existence of costly payoffs rules out EPF representations based on discretizing the space of μ_2, since small errors incurred by discretization could lead to huge drops in performance.

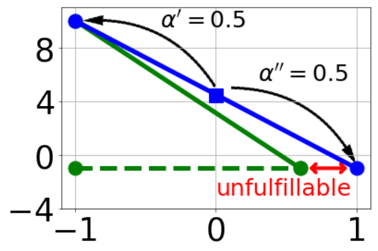


(a) Game illustrating unfulfillable and costly promises.

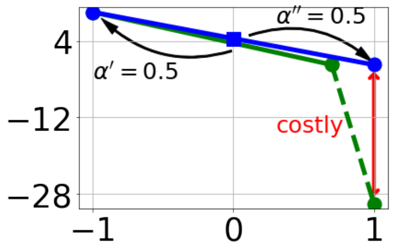


(b) Stage game used in the TANTRUM game.

Figure 6.3: Game trees used in Sections 6.4 and 6.6.2. Leader ○, follower ● and leaf ▫ states are vertices while edges are actions. Payoffs at leaves $\ell \in \mathcal{L}$ are given by $(r_1(\ell), r_2(\ell))$.



(a) Unfulfillable promise.



(b) Costly promise.

Figure 6.4: EPFs for (a) unfulfillable promises and (b) costly promises. Blue lines are estimated EPFs $\widetilde{U}_{s'}$, solid and dotted green lines are true EPFs $U_{s'}$, $U_{s''}$. In both cases, FA error leads us to believe that the payoff given by the blue square at $(0, 4.5)$ can be achieved by mixing the endpoints of $\widetilde{U}_{s'}$ with probability $\alpha' = \alpha'' = 0.5$ (black arrows).

6.5 Function Approximation of Enforceable Payoff Frontiers

We now design our method using the insights from Section 6.4. We learn EPFs without relying on discretization over P_2 payoffs μ_2. Unfulfillable promises are avoided entirely by ensuring that the set of μ_2 where $\widetilde{U}_s(\mu_2) > -\infty$ lies within some known set of achievable P_2 payoffs, while costly promises are mitigated by suitable loss functions.

6.5.1 Representing EPFs using Neural Networks

Our proposed network architecture represents EPFs by a small set of $m \geqslant 2$ points $P_\phi(s) = \{(x_j, y_j)\}$, for $j \in [m]$. Here, m is a hyperparameter trading off complexity of the neural network E_ϕ with its representation power. The approximated EPF $\widetilde{U}_s$ is the linear interpolation of these m points; and $\widetilde{U}_s = -\infty$ if $\mu_2 > \max_j x_j$ or $\mu_2 < \min_j x_j$. For now, we make the assumption that follower payoffs under the altruistic and grim strategy ($\overline{V}(s)$ and $\underline{V}(s)$) are known *exactly* for all states. Through the architecture of E_ϕ that for all $j \in [m]$, we have $\underline{V}(s) \leqslant x_j \leqslant \overline{V}(s)$. As we will see, this helps avoid unfulfillable promises and allows for convenient loss functions.

Concretely, our network $E_\phi(f(s); \underline{V}(s), \overline{V}(s))$ takes in as inputs state features $f(s)$, lower and upper bounds $\underline{V}(s) \leqslant \overline{V}(s)$ and outputs a matrix in $\mathbb{R}^{m \times 2}$ representing $\{(x_j, y_j)\}$ where $x_1 = \underline{V}(s)$ and $x_m = \overline{V}(s)$. For simplicity, we use a multilayer feedforward network with depth

d, width w and ReLU activations for each layer. Serious applications should utilize domain specific architectures. Denoting the output of the last fully connected layer by $h^{(d)}(f(s)) \in \mathbb{R}^w$, for $j \in \{2 \ldots m-1\}$ and $k \in [m]$ we set

$$\begin{aligned} x_j &= \sigma\left(z_{x,j}^T h^{(d)}(f(s)) + b_{x,j}\right) \cdot \left(\overline{V}(s) - \underline{V}(s)\right) + \underline{V}(s), \\ y_k &= z_{y,k}^T h^{(d)}(f(s)) + b_{y,k}, \end{aligned}$$

and $x_1 = \underline{V}(s)$ and $x_m = \overline{V}(s)$, where $\sigma(x) = 1/(1+\exp(-x))$. Here, $z_{x,j}, z_{y,k} \in \mathbb{R}^w$ and $b_{x,j}, b_{y,k} \in \mathbb{R}$ are weights and biases, which alongside the parameters from feedforward network form the network parameters ϕ to be optimized. Since $\widetilde{U}_s$ is represented by its knots (given by $P_\phi(s)$), $\bigwedge$ and consequently, (6.1) may be performed *explicitly* and *efficiently*, returning an entire EPF represented by its knots (as opposed to the EPF evaluated at a single point). This is crucial, since the computation is performed every state every iteration (Line 4 of Algorithm 7).

6.5.2 Loss Functions for Learning EPFs

Given 2 EPFs $\widetilde{U}_s, \widetilde{U}'_s$ we minimize the following loss to mitigate costly promises,

$$L_\infty(\widetilde{U}_s, \widetilde{U}'_s) = \max_{\mu_2} |\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2)|.$$

L_∞ was chosen specifically to incur a large loss if the approximation is wildly inaccurate in a small range of μ_2 (e.g., Figure 6.4b). Achieving a small loss requires that $\widetilde{U}_s(\mu_2)$ approximates $\widetilde{U}'_s(\mu_2))$ well for all μ_2. This design decision is particularly important. For example, contrast L_∞ with another intuitive loss $L_2(\widetilde{U}_s, \widetilde{U}'_s) = \int_{\mu_2} (\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2))^2 d\mu_2$. Observe that L_2 is exceedingly small in the example of Figure 6.4b — in fact, when ϵ is small enough leads to almost no loss, even though the policy as discussed in Section 6.4 is highly suboptimal. This phenomena leads to costly promises, which was indeed observed in our tests.

6.5.3 Theoretical Guarantees

Any learned $\widetilde{U}$ implicitly defines a policy $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2)$ by one-step lookahead using Equation (6.1) and the method described in Phase 2 (Section 6.3). Extracting $\tilde{\pi}$ need not be done offline for all $s \in \mathcal{S}$; in fact, when G is too large it is necessary that we only extract $\tilde{\pi}(\cdot; s)$ on-demand. Nonetheless, $\tilde{\pi}$ enjoys some important properties.

Theorem 10 (Incentive Compatibility). *For any policy $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2)$ obtained using our method, any $s \in \mathcal{S}_2$, $a \in \mathcal{A}(s)$, we have $\tilde{\pi}_2(a; s) > 0 \implies R_2(T(a; s); \tilde{\pi}_1, \tilde{\pi}_2) \geqslant \tau(T(a; s))$.*
Theorem 11 (FA Error). *If $L_\infty(\widetilde{U}_s, \widetilde{U}_s^{target}) \leqslant \epsilon$ for all $s \in \mathcal{S}$, then $|R_1(\tilde{\pi}) - R_1(\pi^*)| = \mathcal{O}(D\epsilon)$ where D is the depth of G and $\pi^* = (\pi_1^*, \pi_2^*)$ is the optimal strategy.*

The proofs of Theorems 10 and 11 are based on backward induction. Since the details are fairly laborious, we defer them to Section 6.7. Here, $T(a; s)$ is transition function (Section 6.2). Recall from Section 6.2 that for π to be an optimal SEFCE, we require (i) incentive compatibility and (ii) $R_1(\pi)$ to be maximized. Theorems 10 and 11 illustrate how our approach disentangles these criteria. Theorem 10 guarantees that P_2 will always be incentivized to follow P_1's recommendations, i.e., there will be no unexpected outcomes arising from unfulfillable promises. Crucially, this is a hard constraint which is satisfied solely due to our choice of network architecture, which ensures that $\widetilde{U}_s(\mu_2) = -\infty$ when $\mu_2 > \overline{V}_s$ for *any* $\tilde{\pi}$ obtained from $\widetilde{U}$. Conversely, Theorem 11 shows that the goal of maximizing R_1 *subject to incentive compatibility* is achieved by attaining a small FA error across all states. This distinction is important. Most notably, incentive compatibility is no longer dependent on convergence during training. This *explicit* guarantee stands in contrast with methods employing self-play reinforcement learning agents; there, incentive compatibility follows *implicitly* from the apparent convergence of a player's strategy. This guarantee has practical implications, for example, evaluating the quality of $\tilde{\pi}$ can be done by estimating $R_1(\tilde{\pi})$ based on sampled trajectories, while implicit guarantees requires incentive compatibility to be demonstrated using some approximate best-response oracle and usually involves expensive training of a RL agent.

The primary limitation of our method is when $\overline{V}$ and $\underline{V}$ (and hence τ) are not known exactly. As it turns out, we can instead use upper and lower approximations while still retaining incentive compatibility. Let $\tilde{\pi}_1^{\text{grim}}$ be an *approximate grim strategy*. Define $\underset{\sim}{V}(s)$ to be the expected follower payoffs at s when faced best-responding to $\tilde{\pi}_1^{\text{grim}}$, i.e., $R_2(s; \tilde{\pi}_1^{\text{grim}}, \pi_2)$, where $\pi_2 \in BRS_2(\tilde{\pi}_1^{\text{grim}})$. Following Definition 14, the *approximate minimum required incentive* is $\tilde{\tau}(s') = \max_{s^! \in \mathcal{C}(s); s^! \neq s'} \underset{\sim}{V}(s^!)$ for all $s \in \mathcal{S}_2$, $s' \in \mathcal{C}(s)$. Similarly, let $\tilde{\pi}^{\text{alt}}$ be an *approximate joint altruistic strategy* and its resultant internal payoffs in each state be $\widetilde{V}(s)$.

Under the mild assumption that $\tilde{\pi}^{\text{alt}}$ always benefits P_2 more than the $\tilde{\pi}_1^{\text{grim}}$, i.e., $\widetilde{V}(s) \geqslant \underset{\sim}{V}(s)$ for all s, we can replace the $\tau, \underline{V}$ and $\overline{V}$ with $\tilde{\tau}, \underset{\sim}{V}$ and $\widetilde{V}$ and maintain incentive compatibility (Theorem 10). The intuition is straightforward: if P_2's threats are 'good enough', parts of the EPF will still be enforceable. Furthermore, promises will always be fulfillable since EPFs domains are now limited to be no greater than $\widetilde{V}(s)$, which we know can be achieved by definition. Unfortunately, Theorem 11 no longer holds, not even in terms of $\max_s |\tilde{\tau}(s) - \tau(s)|$. This is again due to the large Lipschitz constants of U_s. However, we have the weaker guarantee (whose proof follows that of Theorem 11) that performance is close to that predicted at the root.

Theorem 12 (FA Error with Weaker Bounds). *If $L_\infty(\widetilde{U}_s, \widetilde{U}_s^{target}) \leqslant \epsilon$ for all $s \in \mathcal{S}$, then $|R_1(\tilde{\pi}) - \widetilde{\mathsf{OPT}}_2| = \mathcal{O}(D\epsilon)$ where D is the depth of* G *and $\widetilde{\mathsf{OPT}}_2 = \max_{\mu_2} \widetilde{U}_{root}(\mu_2)$.*

Remark 24: The key technical difficulty here is finding $\widetilde{V}$. In our experiments, $\tilde{\pi}_1^{\text{grim}}$ can be found analytically. In general large games, we can approximate $\tilde{\pi}_1^{\text{grim}}$, $\widetilde{V}$ by searching over $\mathcal{S}_2$, but use heuristics when expanding nodes in $\mathcal{S}_1$.

6.5.4 Implementation Details

In this section we describe some of the techniques used to improve and speed up training.

- Employing techniques from reinforcement learning used to stabilize learning, e.g., target networks and prioritized experience replay [8, 133, 155].
- Using a modified loss function based on the sum of squared differences evaluated at the knots, i.e., $L = \sum_{\mu_2 \in \{\text{knots}\}} [\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2)]^2$.

- Speeding up training by implementing the upper concave envelope and left truncation by using GPU-friendly operations in PyTorch [144]. This allows the training pipeline to be run end-to-end efficiently
- Training on a variant of $\widetilde{U}_s$ which accounts only for the decreasing portion of EPFs—intuitively, the increasing region (i.e., portions to the left of the decreasing parts) is Pareto-dominated and we do not want to "waste" effort learning them.
- Schemes for sampling trajectories used to populate the replay buffer.

We stress that these techniques appeared to improve performance (training stability, speed, quality of policy), at least on surface. However, these implementation details are not the core contribution of our work, and we did not perform extensive experiments to verify their importance.

Borrowing Training Techniques from RL and FVI

We employ target networks [133]. Instead of performing gradient descent on the 'true' loss, we create a 'frozen' copy of the network which we use the compute $\widetilde{U}'_s$ (i.e., $\widetilde{U}_s^{\text{target}}$). However, $\widetilde{U}$ is still computed from the main network (with weights to be updated in gradient descent). The key idea is that the target $\widetilde{U}'_s$ is no longer update at every epoch, which can destabilize training. We update the target network with the main network once every 2000 episodes.

For larger games, we noticed that the bulk of loss was attributed to a small fraction of states. To focus attention on these states, we employ prioritized replay [155]. We set the probability of selecting each state s in the replay buffer to be proportionate to the square root of the last loss, i.e., $L(U_s, U'_s)^\alpha$ observed. We used $\alpha = 0.5$ for convenience ([155] suggest a value of 0.7).

Finding out the best hyperparameters for target networks, experience and prioritized replay is beyond the scope of this paper and left as future work.

Modified Loss Function

Our experience is that L_∞ does manage to learn EPFs well, however, learning can be slow and sometimes unstable. Our hypothesis is that slow learning is due to the fact only the point responsible for the loss, as well as its neighbors has its coordinates updated during training. This 'local' learning of $\widetilde{U}_s$ makes learning slow, particularly at the start of training. Second, we found that rather than L_∞, using the *square* of the largest absolute pointwise difference tends to stabilize training (though Theorem 11 would have to be modified to be in terms of $\sqrt{\epsilon}$ instead).

Let U_s and U'_s be two EPFs represented by k_1 and k_2 knots. Let $X_1 = \{x_1, \ldots x_{k_1}\}$ and $X_2 = \{x'_1, \ldots x'_{k_2}\}$ be the x-coordinates of the knots in U_s and U'_s respectively. Then, we use the following loss

$$L(U_s, U'_s) = \sum_{x \in X_1 \cup X_2} \left(U_s(x) - U'_s(x)\right)^2. \tag{6.2}$$

This loss still avoids costly promises since we are still taking pointwise differences (rather than over an integral).

Practical Implementation of Upper Concave Envelope and Left-Truncation

Theoretically, finding the upper concave envelope of k points can be found in linear time. However, this algorithm requires a significant number of backtracking and if-else statements, making this implementation unsuitable for batch operations on a GPU. The alternative which we employ runs in $\mathcal{O}(k^3)$ time which is in practice much faster when run on a GPU. For every distinct pairs of points (x_i, y_i), and (x_j, y_j), we check, for every point (x_a, y_a) where $x_a \in [x_i, x_j]$ whether (x_a, y_a) lies below or above the line segment $(x_i, y_i), (x_j, y_j)$. If a point (x_a, y_a) is below *any* such line segment, we flag it as 'not included', indicating that the upper concave envelope will not include this point. The overall scheme is complicated (see attached code) but runs significantly faster than the linear time method when batch sizes are greater than 32. An important downside, however is (a) the amount of *GPU* memory used for intermediate calculations and (b) the poor scaling (cubic) in terms of number of knots (which is βm, where β is the branching factor and m the number of knots) per EPF. The actual implementation for these routines can be found in the main paper and the source code [123].

Dealing with different number of actions at each vertex and truncated points. Rather than removing points and padding them (to make each batch fit nicely in rectangular tensor), we maintain a 'mask' matrix which indicates that such a point is inactive. These points will not be used in computation of upper concave envelopes (both as potential points and as part of a line segment). Furthermore, this scheme makes it convenient to truncate points (simply mask those truncated points out and perform interpolation to get the new point on at $\tau(s')$).

Training only using decreasing portions of EPF Learning the increasing portion of an EPF is not useful, since points there are Pareto dominated. When extracting $\tilde{\pi}$, we will never select those points. As such, we instead consider a slight variation of the EPF $U_s : \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty\}$ that gives the maximum leader payoff given the follower gets a payoff of *at least* μ_2 (rather than exactly μ_2). This slight change ensures that EPFs are never increasing, while keeping all of the properties we proved earlier on. Omitting the increasing portions saves us from wasting any of the m knots on the increasing portions, and instead focus on the decreasing portion (where there is a real trade-off between payoffs between P_1 and P_2. One example of this is shown in Figure 6.6a and Figure 6.6b, where we showed the 'true' EPF and the modified EPFs that we use for our experiments.

Concretely, let $\widetilde{U}'_s$ be computed based on (6.1) with its representation given by the set of knots $\{(x_1, y_1), \ldots, (x_k, y_k)\}$, assumed to be sorted in ascending order of x-coordinates, and where $x_1 = \underline{V}(s)$. Let the $j = \arg\max_i y_i$. Then, the set of points which we use for training is the modified set $\{(\underline{V}(s), y_j), (x_j, y_j), (x_{j+1}, y_{j+1}), \ldots (x_k, y_k)\}$.

Sampling of Training Trajectories

One of the design decisions in FVI is how one should sample states, or trajectories. In the single-player setting, it is commonplace to use some form of ϵ-greedy sampling. In our work, we use an even simple sampling scheme which takes actions uniformly at random.

There are a few exceptions. For RC sampled *states* uniformly at random. This was made possible because the game was small and we could enumerate all states. The implication is that each leaf is sampled much more frequently than from actual trajectories. Our experience is that since the game is small, getting samples from uniform trajectories should still work well. For TANTRUM, the game has a depth of 50. In many cases, states in the middle are not learning anything meaningful because their children EPFs have not been learned well. As such, we adopt a 'layered' approach, where initially we only allow for states s at most d_{max} to be added to the replay buffer, d_{max} is gradually increased as training goes on. This helps EPFs to be learned for states deeper in the tree first before their parents. We find that for TANTRUM (the non-featurized version with $n = 25$), this was essential to get stable learning of EPFs (recall that in our setting, TANTRUM has a size of roughly $\sim 3^{25}$ and a depth of 50. A uniform trajectory leaves some states to be sampled with probability $1/2^{50}$). We start off at $d_{\text{max}} = 20$ and reduce d_{max} by 1 for every 50000 epochs. For other games, uniform trajectories work well enough since the game is not too deep.

6.6 Experiments

Our experiments are designed to the answer the following:

- Can EPFs be learned in a stable manner using our proposed FVI-like method? To what sizes can we scale, and what are the bottlenecks and caveats in our proposed method?
- What is the accuracy of learned EPFs compared to the ground truth? More importantly, do they yield good policies, relative to (i) the true SEFCE and (ii) other simple equilibrium, e.g., subgame perfect Nash.
- How useful are the learned EPFs. In particular, can EPFs be used to generalize not just across states in the same game, but states *across different games* with slightly varying parameterizations?

Our code is made publicly available at https://github.com/lingchunkai/learn-epf-sefce.

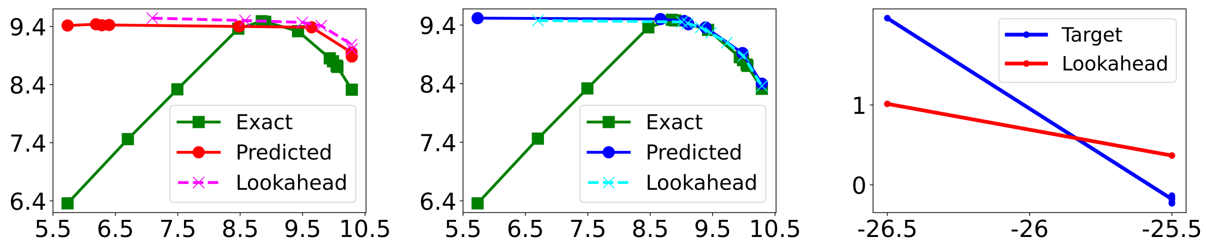
6.6.1 Games of Interest

There are few large general-sum EFGs available in the literature that exhibit interesting leader-follower behavior yet enjoy perfect information. As such, we created the following two synthetic games. An explanation of optimal strategy and precise experiment environments are deferred to Section 6.7.

Tantrum. TANTRUM is the game in Figure 6.3b repeated n times, with $q_1 > 0, q_2 \geqslant 1$, and rewards accumulated over stages. The only way P_1 can get positive payoffs is by threatening to throw a trantrum with the mutually destructive $(-1, -1)$ outcome. Since $q_2 > 1$, P_2 has to use threats spanning over stages to sufficiently entice P_2 to accede. Even though TRANTRUM has $\mathcal{O}(3^n)$ leaves, it is clear that the grim (resp. altruistic) strategy is to throw (resp. not throw) a tantrum at every step. Hence $\overline{V}$ and $\underline{V}$ are known even when n is large, making TANTRUM a good testbed. The raw features $f(s)$ is a 5-dimensional vector, the first 3 are the occurrences

	#	$\overline{\Delta_{\text{OPT}}}$	$\overline{\Delta_{\text{SP}}}$	$\overline{\Delta_{\text{non}}}$
RC	10	-.0247	.200	.265
TANTRUM	5	-.0262	8.89	N/A
RC+	3	N/A	N/A	.421

Figure 6.5: Results for games with fixed parameters averaged over # specifies # trials. $\overline{\Delta_{\text{OPT}}}$, $\overline{\Delta_{\text{SP}}}$, and $\overline{\Delta_{\text{non}}}$ is the average *difference* between our method and the optimal SEFCE, subgame perfect Nash, and non-strategic leader commitment respectively.



(a) EPF after 100k epochs (b) EPF after 2M epochs (c) Failure case

Figure 6.6: (a)-(b) Learned EPFs at the root for RC. (c) A failure case in TANTRUM, even though learned policies are still near-optimal.

count of outcomes for previous stages, and the last 2 being a one-hot vector indicating the current state.

Resource Collection (RC). RC is played on a $J \times J$ grid with a time horizon n. Each cell contains varying quantities of 2 different resources $\mathsf{r}_1(x,y), \mathsf{r}_2(x,y) \geqslant 0$, both of which are collected (at most once) by either players entering. Players begin in the center and alternately choose to either move to an adjacent cell or stay put. Each P_i is only interested in resource i, and players agree to pool together resources when the game ends. RC gives P_1 the opportunity to threaten P_2 with going 'on strike' if P_2 does not move to the cells that P_1 recommends. RC has approximately $\mathcal{O}(25^n)$ leaves. The grim strategy is for P_1 to stay put. However, unlike TANTRUM, computing $\underline{V}$ and $\overline{V}$ still requires search (at least for P_2) at each state, which is still computationally expensive. We use as features (a) one-hot vector showing past visited locations, (b) the current coordinates of each player and whose turn it is (c) the amount of each resource collected, and (d) the number of rounds remaining.

6.6.2 Experimental Setup

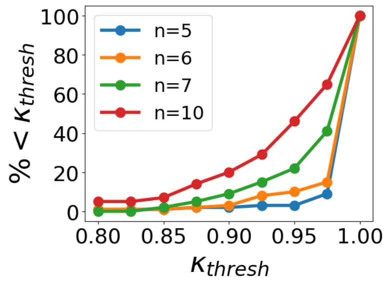
Experiments on Games with Fixed Parameters.

These are environments which are fixed: the training and testing environments are equivalent, and the goal is to demonstrate generalization over states. We run 3 sub-experiments.

- **[RC]** We experimented with RC with $J = 7, n = 4$ over 10 different games. Rewards r_i were generated using a log-Gaussian process over (x, y) to simulate spatial correlations

n	$\overline{\kappa}$	greedy-$\overline{\kappa}$	stress-$\overline{\kappa}$
5	.993	.828	.997
6	.982	.773	.982
7	.968	.778	.921
10	.938	.775	.898

(a) Results (fraction of optimum) for our method, a greedy baseline, a stress test based on training using out of sample distributions.



(b) Proportion of trials which give $\kappa < \kappa_{\text{thresh}}$.

Figure 6.7: Results for Featurized TRANTRUM as depth n varies.

(details in Appendix). We also report the payoffs from a 'non-strategic' P_1 which optimizes only for resources it collects, while letting P_2 best respond.

- **[TANTRUM]** We ran TANTRUM with $n = 25$, $q_1 = 1$ and q_2 chosen randomly. These games have $> 1e12$ states; however, we can still obtain the optimal strategy due to the special structure of the game (note the subgame perfect equilibrium gives P_1 zero payoff).
- **[RC+]** We ran RC with $J = 9$, $n = 6$. Since G is large, we use approximates ($\tilde{\tau}$, $\widetilde{V}$, $\underset{\sim}{V}$) obtained from $\tilde{\pi}_1^{\text{grim}}$ and $\tilde{\pi}^{\text{alt}}$. $\tilde{\pi}_1^{\text{grim}}$ is for P_2 to stay put, while $\underset{\sim}{V}$ is obtained by applying search *online* (i.e., when s appears in training) for P_2 starting from s. Thus $\tilde{\tau}(s)$ can also be computed online from $\underset{\sim}{V}$. $\tilde{\pi}^{\text{alt}}$ is obtained by running exact search to a depth of 4 (counted from the root) and then switching to a greedy algorithm. On the rare occasion that $\widetilde{V}(s) < \underset{\sim}{V}(s)$, we set $\widetilde{V}(s) \leftarrow \underset{\sim}{V}(s)$.

We report results in Figure 6.5, which show the *difference* between P_1's payoff for our method and (i) the optimal SEFCE, (ii) the subgame perfect Nash, and (iii) the non-strategic leader commitment.

Featurized TANTRUM.

Here, we see if we are able to generalize across games that have similar structures but differing parameters. We allow q_1, q_2 to *vary* between stages of G, giving vectors $\mathbf{q}_i \in [1, \infty]^n$. Each trajectory uses different $\mathbf{q}_i$, which we append as features to our network, alongside the payoffs already collected for each player. For training, we draw i.i.d. samples of $\mathbf{q}_i^j \sim \exp(1) + 1$. The evaluation metric is $\kappa = R_1(\tilde{\pi})/\mathsf{OPT}$, i.e., the ratio of P_1's payoffs under $\tilde{\pi}$ compared to the optimal π. For each n, we test on 100 $\mathbf{q}$-vectors not seen during training and compare their κ against a 'greedy' strategy which recommends P_2 to accede as long as there are sufficient threats in the remainder of the game for P_1 (details in Appendix). We also stress test $\tilde{\pi}$ on a different *test* distribution $\hat{\mathbf{q}}_i^j \sim \exp(1) + 4$. We report results in Figure 6.7a and 6.7b.

6.6.3 Results and Discussion

For fixed parameter games small enough such that the optimal strategy may be computed, we observe near optimal performance which significantly outperforms other baselines. In the case of [RC], the average value of each an improvement of .5 is approximately equal to moving an extra half move (on average). In [TANTRUM], the subgame perfect equilibrium is vacuous as P_1 is unable to issue threats and gets a payoff of 0. In [RC+], the game is too large we are unable to fully expand the game tree, however, we still significantly outperform the non-strategic baseline.

For featurized TANTRUM, we perform near-optimally for small n, even when stress tested with out-of-distribution **q**'s (Figure 6.7a). Performance drops as n becomes larger, which is natural as EPFs become more complex. While performance degrades as n increases, we still significantly outperform the greedy baseline. The stress test suggests that the network is not merely memorizing data.

Figures 6.6a and 6.6b shows the learned EPFs at the root for epochs 100k and 2M, obtained directly or from one-step lookahead. As explained in Section 6.5, we only learn the decreasing portions of EPFs. After 2M training epochs, the predicted EPFs and one-step lookahead mirrors the true EPF in the decreasing portions, which is not the case at the beginning. At the beginning of training, many knots (red markers) are wasted on learning the 'useless' increasing portions on the left. After 2M epochs, knots (blue markers) were learning the EPF at the 'useful' decreasing regions (see Section 6.5.4).

Figure 6.6c gives an state in TANTRUM whose EPF yields high loss even after training. This failure case is not rare since TANTRUM is large. Yet, the resultant action is still optimal—in this case the promise to P_2 was $\mu_2 = -25.5$ which is precisely $\overline{V}(s)$. Like MDPs, policies can be near-optimal even with high Bellman losses in some states.

6.7 Technical Proofs and Implementation Details

We end this section by presenting first presenting the technical proofs of Theorems 10, 11 and 12, Next, we discuss the properties of optimal SEFCE in the games we experimented on.

6.7.1 Preliminaries

Let $\widetilde{U}_s$ be our predicted EPF at state s. For $s \in \mathcal{S} \backslash \mathcal{L}$, we denote as shorthand

$$\widetilde{U}'_s = \widetilde{U}^{\text{target}}_s = \begin{cases} \left[\bigwedge_{s' \in \mathcal{C}(s)} \widetilde{U}_{s'} \right](\mu) & \text{if } s \in \mathcal{S}_1 \\ \left[\bigwedge_{s' \in \mathcal{C}(s)} \widetilde{U}_{s'} \rhd \tau(s') \right](\mu) & \text{if } s \in \mathcal{S}_2 \end{cases}$$

what is obtained from one-step lookahead using Section 6.3, and for $s \in \mathcal{L}$,

$$\widetilde{U}_s(\mu_2) = \widetilde{U}'_s(\mu_2) = \begin{cases} r_1(s) & \mu_2 = r_2(s) \\ -\infty & \text{otherwise} \end{cases}.$$

For clarity, we denote likewise for the exact EPFs U'_s (which will be equal by definition to U_s).

Domains of EPFs Given any function $h : \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty\}$, we denote its domain by $Dom[h] = \{x | h(x) > -\infty\}$. The following Theorem ensure that the required *domains* all match. This theorem is added for completeness; the reader can skip over this section if desired.

Theorem 13. *For all $s \in \mathcal{S}$, $Dom[U_s] = Dom[U'_s] = Dom[\widetilde{U}_s] = Dom[\widetilde{U}'_s] = [\underline{V}(s), \overline{V}(s)]$.*

The proof of Theorem 13 is straightforward and left to Section 6.7.5.

Depth of a state Let the *depth* of a state $D(s)$ be the longest path needed to reach a leaf, i.e.,

$$D(s) = \begin{cases} 0 & s \in \mathcal{L} \\ \max_{s' \in \mathcal{C}(s)} D(s') + 1 & \text{otherwise} \end{cases}.$$

Note that depth is defined in terms of the number of steps remaining, not the more natural notion of how many actions have already been taken. Since G is finite, $D = \max_{s \in \mathcal{S}} D(s)$ and is finite. Since errors accumulate in each Bellman backup, it is unsurprising that the D is involved in our FA guarantee.

Upper Concave Envelopes and Left Truncations Since U_s and $\widetilde{U}_s$ are one dimensional functions, $\bigwedge$ can be written alternatively as

$$\left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'}\right](\mu_2) = \max_{\substack{s',s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}} tU_{s'}(\mu') + (1-t)U_{s''}(\mu''), \tag{6.3}$$

that is, the maximum that could be obtained by interpolating between at most two points across 2 children states $s', s'' \in \mathcal{C}(s)$ [114].

Recall that $\tau(s')$ is defined only for $s' \in \mathcal{C}(s)$, where $s \in \mathcal{S}_2$. For convenience, we define $\beta(s') = \tau(s')$ when $s \in \mathcal{S}_2$ and $-\infty$ when $s \in \mathcal{S}_1$. β plays the same role as τ, since left truncating at $-\infty$ does not change anything, i.e., $f \rhd (-\infty) = f$ for any $f : \mathbb{R} \mapsto \mathbb{R} \cup \{-\infty\}$. Using β allows us to perform a 'dummy' left truncation when $s \in \mathcal{S}_1$ and avoid having to split into different cases.

6.7.2 Proof of Theorem 10

Recall that Theorem 10 asserts that all recommendations made to P_2 are incentive compatible:

Theorem 10 (Incentive Compatibility). *For any policy $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2)$ obtained using our method, any $s \in \mathcal{S}_2$, $a \in \mathcal{A}(s)$, we have $\tilde{\pi}_2(a; s) > 0 \implies R_2(T(a; s); \tilde{\pi}_1, \tilde{\pi}_2) \geqslant \tau(T(a; s))$.*

Proof. We know that for all $s \in \mathcal{S} \backslash \mathcal{L}$, $\mu_2 \in [\underline{V}(s), \overline{V}(s)]$, our policy $(\tilde{\pi}_1, \tilde{\pi}_2)$ is obtained by solving the optimization problem

$$\operatorname*{arg\,max}_{\substack{s',s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}} t[\widetilde{U}_{s'} \rhd \beta(s')](\mu') + (1-t)[\widetilde{U}_{s''} \rhd \beta(s'')](\mu''), \tag{6.4}$$

Clearly, $R_2(s; \tilde{\pi}_1, \tilde{\pi}_2)$ lies in $[\underline{V}(s), \overline{V}(s)]$. We also know that the value of the objective of (6.4) is greater than $-\infty$. Because of the left-truncation operator, we can see that any value of μ' and μ'' chosen must be greater than $\beta(s')$ and $\beta(s'')$ respectively (or that t or $1-t$ must be equal to 0). Recall that $\beta(s') = \tau(s')$ for $s' \in \mathcal{S}_2$, and that $s' = sa = T(a; s)$ for some action a (and similarly for s''). We conclude by noting that the actions leading to s' and s'' are the only ones that are possibly played with strictly non-zero probability. $\square$

6.7.3 Proof of Theorem 11

We prove that the estimate $\widetilde{U}_s$ is not too different from the optimal U_s. This is done by backward induction on the tree. Next, we prove that the extracted policy $\tilde{\pi}$ is near optimal as well. Roughly speaking, this is done by "top-down" traversal of the game tree.

Our Goal Our goal is to show that assuming that the L_∞ loss is low for all states s, i.e.,

$$\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2)| \leqslant \epsilon \tag{6.5}$$

then we will enjoy good performance, i.e, the leader gets a payoff of order $\mathcal{O}(\epsilon D)$ less than optimal (additively).

Learned EPFs are Approximately Optimal

The first half of the theorem is to show that our learned EPFs $\tilde{U}_s$ are for all s, close to the true U_s pointwise. This is done by strong induction on the states by increasing depth and applying (6.1). Our induction hypothesis is

$$\mathcal{K}_j : \quad \max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - U_s(\mu_2)| \leqslant j\epsilon \qquad \forall s \text{ where } D(s) = j.$$

By definition, K_0 satisfies our requirement since $s \in \mathcal{L}$. Thus, the base case is satisfied. Now we prove the inductive case. Assume that $\mathcal{K}_0, \cdots, \mathcal{K}_{j-1}$ are all satisfied. We want to show $\mathcal{K}_j$ using (6.5).

Theorem 14. *Let $s \in \mathcal{S}$ such that $D(s) = j$. Suppose $\mathcal{K}_0, \ldots \mathcal{K}_{j-1}$ are true. Then we have $|\widetilde{U}'_s(\mu_2) - U'_s(\mu_2)| \leqslant \epsilon(j-1)$ for all $\mu_2 \in [\underline{V}(s), \overline{V}(s)]$.*

Proof. Fix μ_2. We want to show $|\widetilde{U}'_s(\mu_2) - U'_s(\mu_2)| \leqslant \epsilon(j-1)$. Now, let $\hat{\sigma} = (\hat{s}', \hat{s}'', \hat{t}, \hat{\mu}', \hat{\mu}'')$ be the parameters which achieves the maximum

$$\operatorname*{arg\,max}_{\substack{s', s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}} \quad t[U_{s'} \rhd \beta(s')](\mu') + (1-t)[U_{s''} \rhd \beta(s'')](\mu'') \tag{6.6}$$

and similarly when we are working with learned EFPs, $\tilde{\sigma} = (\tilde{s}', \tilde{s}'', \tilde{t}, \tilde{\mu}', \tilde{\mu}'')$

$$\operatorname*{arg\,max}_{\substack{s', s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}} \quad t[\widetilde{U}_{s'} \rhd \beta(s')](\mu') + (1-t)[\widetilde{U}_{s''} \rhd \beta(s'')](\mu''), \tag{6.7}$$

which are the arguments that optimized give $U'_s(\mu_2)$ and $\widetilde{U}'_s(\mu_2)$ respectively. That is, $\hat{\sigma}$ and $\tilde{\sigma}$ gives how the point at the EPF with the x coordinate equal to μ_2 is obtained as a mixture of at most 2 points from the upper convex envelope (when $s' = s''$, we simply repeat the 2 points and set $t = 1/2$ for simplicity). We proceed by showing a contradiction. We have two cases.

Case 1. Suppose that $\widetilde{U}'_s(\mu_2) > U'_s(\mu_2) + \epsilon(j-1)$. Then we have

$$\begin{aligned}
&\left| \underbrace{\tilde{t}[\widetilde{U}_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') + (1-\tilde{t})[\widetilde{U}_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'')}_{=\widetilde{U}'_s(\mu_2) > U'_s(\mu_2)+\epsilon(j-1)} - \underbrace{\tilde{t}[U_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') + (1-\tilde{t})[U_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'')}_{\leqslant U'_s(\mu_2)} \right| \\
&= \left| \tilde{t}\left([\widetilde{U}_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') - [U_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') \right) + (1-\tilde{t})\left([\widetilde{U}_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'') - [U_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'') \right) \right| \\
&\leqslant \epsilon(j-1)
\end{aligned} \tag{6.8}$$

and where first inequality inside $|\cdot|$ follows from our assumption in case 1, and the second from the fact that $U'_s(\mu_2)$ was taken from an argmax, i.e., (6.6). The third line holds from our induction hypothesis $\mathcal{K}_i$ (6.5), where $i \in [0, j-1]$, the fact that $D(s') < D(s) = j$ and how the $\rhd$ operator cannot increase the absolute error of the difference.[7] However, these 3 inequalities cannot hold simultaneously, thus the assumption for Case 1 cannot be true, i.e., we must have $\widetilde{U}'_s(\mu_2) \leqslant U'_s(\mu_2) + \epsilon(j-1)$

Case 2. Suppose that $\widetilde{U}'_s(\mu_2) < U'_s(\mu_2) - \epsilon(j-1)$. Then using a similar derivation we have

$$\begin{aligned}
&\left| \underbrace{\hat{t}[U_{\hat{s}'} \rhd \beta(\hat{s}')](\hat{\mu}') + (1-\hat{t})[]U_{\hat{s}''} \rhd \beta(\hat{s}'')](\hat{\mu}'')}_{=U'_s(\mu_2) > \widetilde{U}'_s(\mu_2)+\epsilon(j-1)} - \underbrace{\hat{t}[\widetilde{U}_{\hat{s}'} \rhd \beta(\hat{s}')](\hat{\mu}') + (1-\hat{t})[\widetilde{U}_{\hat{s}''} \rhd \beta(\hat{s}'')](\hat{\mu}'')}_{\leqslant \widetilde{U}'_s(\mu_2)} \right| \\
&= \left| \hat{t}\left([U_{\hat{s}'} \rhd \beta(\hat{s}')](\hat{\mu}') - [\widetilde{U}_{\hat{s}'} \rhd \beta(\hat{s}')](\hat{\mu}') \right) + (1-\hat{t})\left([U_{\hat{s}''} \rhd \beta(\hat{s}'')](\hat{\mu}'') - [\widetilde{U}_{\hat{s}''} \rhd \beta(\hat{s}'')](\hat{\mu}'') \right) \right| \\
&\leqslant \epsilon(j-1)
\end{aligned} \tag{6.9}$$

where the first inequality inside $|\cdot|$ comes from case 2's assumption, the second is from the fact that $\widetilde{U}'_s(\mu_2)$ was taken from an argmax, i.e., (6.7). The third line follows from the induction hypothesis $\mathcal{K}_i$, where $i \in [0, j-1]$ and the depth of s. These 3 inequalities cannot hold simultaneously, so our assumption in case 2 cannot be true and $\widetilde{U}'_s(\mu_2) \geqslant U'_s(\mu_2) - \epsilon(j-1)$.

Combining the result from both cases gives the desired result. □

[7]Note that since $\tilde{\sigma}$ is the argmax, we are guaranteed to not have values of $\tilde{\mu}$ that lie outside the domains of the truncated EPFs.

Now, we consider the μ_2 which has the worst possible discrepancy, which gives

$$\begin{aligned}
&\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - U_s(\mu_2)| \\
\leqslant &\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2)| + |\widetilde{U}'_s(\mu_2) - U_s(\mu_2)| \\
= &\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - U'_s(\mu_2)| + |\widetilde{U}'_s(\mu_2) - U'_s(\mu_2)| \\
\leqslant &\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - \widetilde{U}'_s(\mu_2)| + \max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}'_s(\mu_2) - U'_s(\mu_2)| \\
\leqslant &\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}'_s(\mu_2) - U'_s(\mu_2)| + \epsilon \\
\leqslant &\epsilon j,
\end{aligned} \tag{6.10}$$

where the second line follows from the triangle inequality, the third line using the equality between $U_s(\mu_2)$ and $U'_s(\mu_2)$, the fourth from the fact that $\max_x |f(x) + g(x)| \leqslant \max_x |f(x)| + \max_y |g(y)|$, the fifth from the assumption (6.5), the the last line from Theorem 14. The main theorem for this part follows by induction on $D(s)$, the fact that $D(s)$ is bounded by the depth of the tree, and the fact that the base case $\mathcal{K}_0$ is trivially true.

Theorem 15. *If $L_\infty(\widetilde{U}_s, \widetilde{U}'_s) \leqslant \epsilon$ for all $s \in \mathcal{S}$, then $\max_{\mu_2 \in [\underline{V}(s), \overline{V}(s)]} |\widetilde{U}_s(\mu_2) - U_s(\mu_2)| \leqslant \epsilon D$, where D is the depth of the game.*

Proof. This follows from the definition of L_∞ and the above derivations. □

Leader's Payoffs from Induced Strategy is Close-To-Expected (Theorem 11)

Theorem 15 tells us that our EPF everywhere is close (pointwise) to the true EPF if ϵ is small. Now we need to establish the suboptimality when *playing* according to $\tilde{\pi}$, which is a joint policy implicit from $\widetilde{U}_s$. We begin with some notation.

Let $\widetilde{Q}_s(\mu_2) : [\underline{V}(s), \overline{V}(s)] \mapsto \mathbb{R}$ be the payoff to P_1 assuming we started at state s, promised a payoff of μ_2 to P_2 and used the approximate EPFs $\widetilde{U}_s$ for all descendent states $s' \sqsupseteq s$ (the domain precludes unfulfilled promises). That is, given $\tilde{\sigma} = (\tilde{s}', \tilde{s}'', \tilde{t}, \tilde{\mu}', \tilde{\mu}'')$ given by

$$\operatorname*{arg\,max}_{\substack{s', s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}} t[\widetilde{U}_{s'} \rhd \beta(s')](\mu') + (1-t)[\widetilde{U}_{s''} \rhd \beta(s'')](\mu''), \tag{6.11}$$

the induced policy is to play to $\tilde{s}'$ with probability $\tilde{t}$ and a consequent promise of $\tilde{\mu}'$, as well as playing to $\tilde{s}''$ with probability $(1 - \tilde{t})$ and a promised payoff of $\tilde{\mu}''$. By definition, if $s \in \mathcal{L}$, $\widetilde{Q}_s = \widetilde{U}_s = U_s$ trivially. We also have the following recursive equations for a given $s \notin \mathcal{L}$, μ_2

$$\widetilde{Q}_s(\mu_2) = \tilde{t}\widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') + (1 - \tilde{t})\widetilde{Q}_{\tilde{s}''}(\tilde{\mu}''). \tag{6.12}$$

Theorem 16. $\left|\widetilde{Q}_s(\mu_2) - \widetilde{U}_s(\mu_2)\right| \leqslant \epsilon D$ *for all $s \in \mathcal{S}$ and for all $\mu_2 \in [\underline{V}(s), \overline{V}(s)]$.*

Proof. The proof is given by strong induction on the $D(s)$ again. Let

$$\mathcal{H}_j : \quad \left|\widetilde{Q}_s(\mu_2) - \widetilde{U}_s(\mu_2)\right| \leqslant \epsilon j \quad \forall s \text{ where } D(s) = j \tag{6.13}$$

By definition $\mathcal{H}_0$ is true. Now let us suppose that $\mathcal{H}_0 \ldots \mathcal{H}_{j-1}$ is true. We have for $s \in \mathcal{S}, D(s) = j$,

$$\begin{aligned}
&\left|\widetilde{Q}_s(\mu_2) - \widetilde{U}_s(\mu_2)\right| \\
\leqslant &\left|\widetilde{Q}_s(\mu_2) - \widetilde{U}'_s(\mu_2)\right| + \left|\widetilde{U}'_s(\mu_2) - \widetilde{U}_s(\mu_2)\right| \\
\leqslant &\left|\widetilde{Q}_s(\mu_2) - \widetilde{U}'_s(\mu_2)\right| + \epsilon \\
= &\left|\tilde{t}\widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') + (1-\tilde{t})\widetilde{Q}_{\tilde{s}''}(\tilde{\mu}'') - \tilde{t}[\widetilde{U}_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') - (1-\tilde{t})[\widetilde{U}_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'')\right| + \epsilon \\
\leqslant &\tilde{t}\underbrace{\left|\widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') - \widetilde{U}_{\tilde{s}'}(\tilde{\mu}')\right|}_{\leqslant \epsilon(j-1)} + (1-\tilde{t})\underbrace{\left|\widetilde{Q}_{\tilde{s}''}(\tilde{\mu}'') - \widetilde{U}_{\tilde{s}''}(\tilde{\mu}'')\right|}_{\leqslant \epsilon(j-1)} + \epsilon \\
\leqslant &\epsilon j
\end{aligned}$$

The second line follows from the triangle inequality. The third line follows from our FA assumption (6.5). The fourth line follows from expansion of the definitions of $\widetilde{Q}_s$ and $\widetilde{U}'_s$, i.e., (6.12) and (6.7) The fifth line follows the induction hypothesis and the fact that $s', s'' \in \mathcal{C}(s)$ have at least one lower depth than s. Also, the truncation operator never causes any element to exceed domain bounds (which would give $-\infty$ values). By strong induction $\mathcal{H}_j$ is true for all $j \in [0, D]$ and the theorem follows through directly. □

Piecing Everything Together

Let $\mu_2^* = \arg\max_{\mu_2} U_{\text{root}}(\mu_2)$, i.e., the promise given to the follower *at the root* under the optimal policy π^*. Let $\tilde{\mu}_2 = \arg\max_{\mu_2} \widetilde{U}_{\text{root}}(\mu_2)$, which is the promise to be given to the follower *at the root* under $\tilde{\pi}$. We have

$$\begin{aligned}
&U_{\text{root}}(\mu_2^*) - \widetilde{Q}_{\text{root}}(\tilde{\mu}_2) \\
= &\underbrace{U_{\text{root}}(\mu_2^*) - \widetilde{U}_{\text{root}}(\mu_2^*)}_{|\cdot| \leqslant \epsilon D} + \underbrace{\widetilde{U}_{\text{root}}(\mu_2^*)}_{\leqslant \widetilde{U}_{\text{root}}(\tilde{\mu}_2)} - \widetilde{Q}_{\text{root}}(\tilde{\mu}_2) \\
\leqslant &\epsilon D + \left|\widetilde{U}_{\text{root}}(\tilde{\mu}_2) - \widetilde{Q}_{\text{root}}(\tilde{\mu}_2)\right| \\
\leqslant &2\epsilon D.
\end{aligned}$$

The inequalities in the second line come from Theorem 15 and the definition of $\tilde{\mu}_2$; specifically that it is taken over the argmax. The last line comes from Theorem 16. To complete the proof, we simply observe that $\widetilde{Q}_{\text{root}}(\tilde{\mu}_2)$ is precisely $R_1(\tilde{\pi}_1, \tilde{\pi}_2)$ by definition.

6.7.4 Proof of Theorem 12

The proof is essentially the same as presented in Section 6.7.3, except that we are working with approximate bounds rather than strict ones. We have, using the new bounds,

For $s \in \mathcal{S} \backslash \mathcal{L}$, we denote using shorthand

$$\widetilde{U}'_s = \widetilde{U}^{\text{target}}_s = \begin{cases} \left[\bigwedge_{s' \in \mathcal{C}(s)} \widetilde{U}_{s'} \right](\mu) & \text{if } s \in \mathcal{S}_1 \\ \left[\bigwedge_{s' \in \mathcal{C}(s)} \widetilde{U}_{s'} \rhd \tilde{\tau}(s') \right](\mu) & \text{if } s \in \mathcal{S}_2 \end{cases}$$

be what is obtained from one-step lookahead using Section 6.3, and for $s \in \mathcal{L}$,

$$\widetilde{U}_s(\mu_2) = \widetilde{U}'_s(\mu_2) = \begin{cases} r_1(s) & \mu_2 = r_2(s) \\ -\infty & \text{otherwise} \end{cases}.$$

This is the same as the case with exact $\underline{V}_s, \overline{V}_s$, but with a stricter truncation $\tilde{\tau}(s')$ for each $s' \in \mathcal{C}(s)$. We define $\tilde{\beta}$ just like before: $\tilde{\beta}(s') = \tilde{\tau}(s')$ when $s \in \mathcal{S}_2$ and $-\infty$ when $s \in \mathcal{S}_1$. We follow along the same way as Theorem 11 in Section 6.7.3.

Let $\widetilde{Q}_s(\mu_2) : [\underset{\sim}{V}(s), \widetilde{V}(s)] \mapsto \mathbb{R}$ be the payoff to P_1 assuming we started at state s, promised a payoff of μ_2 to P_2 and used the approximate EPFs $\widetilde{U}_s$ for all descendent states $s' \sqsupseteq s$ (the domain precludes unfulfilled promises). That is, given $\tilde{\sigma} = (\tilde{s}', \tilde{s}'', \tilde{t}, \tilde{\mu}', \tilde{\mu}'')$ given by

$$\underset{\substack{s', s'' \in \mathcal{C}(s) \\ t \in [0,1]; \mu', \mu'' \in \mathbb{R} \\ t\mu' + (1-t)\mu'' = \mu_2}}{\arg\max} \quad t[\widetilde{U}_{s'} \rhd \tilde{\beta}(s')](\mu') + (1-t)[\widetilde{U}_{s''} \rhd \tilde{\beta}(s'')](\mu''), \tag{6.14}$$

the induced policy is to play to $\tilde{s}'$ with probability $\tilde{t}$ and a consequent promise of $\tilde{\mu}'$, as well as playing to $\tilde{s}''$ with probability $(1 - \tilde{t})$ and a promised payoff of $\tilde{\mu}''$. By definition, if $s \in \mathcal{L}$, $\widetilde{Q}_s = \widetilde{U}_s = U_s$ trivially. Just like before, we also have the following recursive equations for a given $s \notin \mathcal{L}$, μ_2

$$\widetilde{Q}_s(\mu_2) = \tilde{t}\widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') + (1 - \tilde{t})\widetilde{Q}_{\tilde{s}''}(\tilde{\mu}''). \tag{6.15}$$

Let our induction hypothesis be

$$\mathcal{H}_j : \quad \left| \widetilde{Q}_s(\mu_2) - \widetilde{U}_s(\mu_2) \right| \leqslant \epsilon j \quad \forall s \text{ where } D(s) = j. \tag{6.16}$$

By definition $\mathcal{H}_0$ is true. Now let us suppose that $\mathcal{H}_0 \dots \mathcal{H}_{j-1}$ is true. We have for $s \in \mathcal{S}, D(s) = j$,

$$\begin{aligned} &\left| \widetilde{Q}_s(\mu_2) - \widetilde{U}_s(\mu_2) \right| \\ \leqslant &\left| \widetilde{Q}_s(\mu_2) - \widetilde{U}'_s(\mu_2) \right| + \left| \widetilde{U}'_s(\mu_2) - \widetilde{U}_s(\mu_2) \right| \\ \leqslant &\left| \widetilde{Q}_s(\mu_2) - \widetilde{U}'_s(\mu_2) \right| + \epsilon \end{aligned}$$

$$
\begin{aligned}
&= \left| \tilde{t}\widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') + (1-\tilde{t})\widetilde{Q}_{\tilde{s}''}(\tilde{\mu}'') - \tilde{t}[\widetilde{U}_{\tilde{s}'} \rhd \beta(\tilde{s}')](\tilde{\mu}') - (1-\tilde{t})[\widetilde{U}_{\tilde{s}''} \rhd \beta(\tilde{s}'')](\tilde{\mu}'') \right| + \epsilon \\
&\leqslant \tilde{t} \underbrace{\left| \widetilde{Q}_{\tilde{s}'}(\tilde{\mu}') - \widetilde{U}_{\tilde{s}'}(\tilde{\mu}') \right|}_{\leqslant \epsilon(j-1)} + (1-\tilde{t}) \underbrace{\left| \widetilde{Q}_{\tilde{s}''}(\tilde{\mu}'') - \widetilde{U}_{\tilde{s}''}(\tilde{\mu}'') \right|}_{\leqslant \epsilon(j-1)} + \epsilon \\
&\leqslant \epsilon j
\end{aligned}
$$

The second line follows from the triangle inequality. The third line follows from our FA assumption (6.5). The fourth line follows from expansion of the definitions of $\widetilde{Q}_s$ and $\widetilde{U}'_s$, i.e., (6.12) and (6.7) The fifth line follows the induction hypothesis and the fact that $s', s'' \in \mathcal{C}(s)$ have at least one lower depth than s. Also, the truncation operator never causes any element to exceed domain bounds (which would give $-\infty$ values). By strong induction $\mathcal{H}_j$ is true for all $j \in [0, D]$. Finally, we observe that $\widetilde{Q}_s(\mu_2) = R_1(\tilde{\pi})$ when $\tilde{\mu}_2 = \arg\max_{\mu_2} \widetilde{U}_s(\mu_2)$. This completes the proof.

6.7.5 Proof of Theorem 13

Theorem 17. *For all $s \in \mathcal{S}$, $Dom[U_s] = Dom[U's] = [\underline{V}(s), \overline{V}(s)]$.*

Proof. The first equality is by definition. We now show that $Dom[U_s] = [\underline{V}, \overline{V}]$ by definition. Consider the state s we are applying (6.1) to and the 2 possible cases.

Case 1: $s \in \mathcal{S}_1$**.** By definition $\overline{V}(s) = \max_{s' \in \mathcal{C}(s)} \overline{V}(s')$, and $\underline{V}(s) = \min_{s' \in \mathcal{C}(s)} \underline{V}(s)$. First, observe that

$$
\begin{aligned}
\max\left\{Dom\left[U'_s\right]\right\} &= \max\left\{ Dom\left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'} \right] \right\} \\
&= \max_{s' \in \mathcal{C}(s)} \max\left\{Dom\left[U_{s'}\right]\right\} \\
&= \max_{s' \in \mathcal{C}(s)} \overline{V}(s') \\
&= \overline{V}(s),
\end{aligned}
\tag{6.17}
$$

where the second line follows from the fact that the largest x-coordinate after taking the upper-concave-envelope is the largest of the largest-x coordinates over each $U_{s'}$. Similarly, we have

$$
\begin{aligned}
\min\left\{Dom\left[U'_s\right]\right\} &= \min\left\{ Dom\left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'} \right] \right\} \\
&= \min_{s' \in \mathcal{C}(s)} \min\left\{Dom\left[U_{s'}\right]\right\} \\
&= \min_{s' \in \mathcal{C}(s)} \underline{V}(s') \\
&= \underline{V}(s),
\end{aligned}
\tag{6.18}
$$

where the second line comes again from the fact that the lowest x-coordinate after taking upper concave envelopes is the smallest of all the smallest x-coordinates over each $U_{s'}$. Now, (6.17) and (6.18) established the lower and upper limits of U'_s. Since U'_s is concave, for every $\min\{Dom[U'_s]\} \leqslant \mu_2 \leqslant \max\{Dom[U'_s]\}$ we have

$$U'_s(\mu_2) \geqslant \min(U'_s(\min\{Dom[U'_s]\}), U'_s(\max\{Dom[U'_s]\})) > -\infty$$

. This completes Case 1.

Case 2: $s \in \mathcal{S}_2$**.** By definition, $\overline{V}(s) = \max_{s' \in \mathcal{C}(s)} \overline{V}(s')$ and $\underline{V}(s) = \max_{s' \in \mathcal{C}(s)} \underline{V}(s')$ (note the difference with Case 1, since P_2 decides the current action). We again work out upper and lower bounds of $Dom[U'_s]$.

$$\begin{aligned}
\max\{Dom\,[U'_s]\} &= \max\left\{Dom\left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'} \rhd \tau(s')\right]\right\} \\
&= \max_{s' \in \mathcal{C}(s)} \max\{Dom\,[U_{s'} \rhd \tau(s')]\} \\
&= \max_{s' \in \mathcal{C}(s)} \max\left\{[\underline{V}(s'), \overline{V}(s')] \cap [\tau(s'), \infty)\right\} \\
&= \max_{s' \in \mathcal{C}(s)} \overline{V}(s') \\
&= \overline{V}(s),
\end{aligned} \tag{6.19}$$

where the fourth line follows from the fact that $= \max_{s' \in \mathcal{C}(s)} \overline{V}(s') \geqslant \max_{s^! \in \mathcal{C}(s); s^! \neq s'} \overline{V}(s^!) \geqslant \max_{s^! \in \mathcal{C}(s); s^! \neq s'} \underline{V}(s^!) = \tau(s')$ (i.e., that the highest x coordinate in U'_s is never part of the left-truncation step). For any $s' \in \mathcal{C}(s)$,

$$Dom\,[U_{s'} \rhd \tau(s')] = \begin{cases} \varnothing & \max\{Dom[U_{s'}]\} < \tau(s') \\ Dom\,[U_{s'}] & \tau(s') < \min\{Dom\,[U_{s'}]\}\,. \\ [\tau(s'), \max\{Dom[U_{s'}]\}] & \text{otherwise} \end{cases} \tag{6.20}$$

For s' where $Dom\,[U_{s'} \rhd \tau(s')] \neq \varnothing$, we have

$$\min\{Dom\,[U_{s'} \rhd \tau(s')]\} = \max_{s'' \in \mathcal{C}(s)} \min\{Dom\,[U_{s''}]\}\,. \tag{6.21}$$

Note that this is not dependent on s'. Also, note that it cannot be the case that $Dom\,[U_{s'} \rhd \tau(s')] = \varnothing$ for *all* s'. In particular, consider $s^* = \arg\max_{s' \in \mathcal{C}(s)} \max\{Dom[U_{s'}]\}$, clearly, $\max\{Dom[U_{s'}]\} \geqslant \tau(s^*)$ so we do not wind up with the empty set in (6.20). For simplicity, let $\min\varnothing = \infty$. Hence, we can write

$$\begin{aligned}
\min\{Dom\,[U'_s \rhd \tau(s')]\} &= \min\left\{Dom\left[\bigwedge_{s' \in \mathcal{C}(s)} U_{s'} \rhd \tau(s')\right]\right\} \\
&= \min_{s' \in \mathcal{C}(s);} \min\{Dom\,[U_{s'} \rhd \tau(s')]\} \\
&= \min_{s' \in \mathcal{C}(s)} \max_{s'' \in \mathcal{C}(s)} \min\{Dom\,[U_{s''}]\} \\
&= \max_{s'' \in \mathcal{C}(s)} \min\{Dom\,[U_{s''}]\} \\
&= \underline{V}(s).
\end{aligned} \tag{6.22}$$

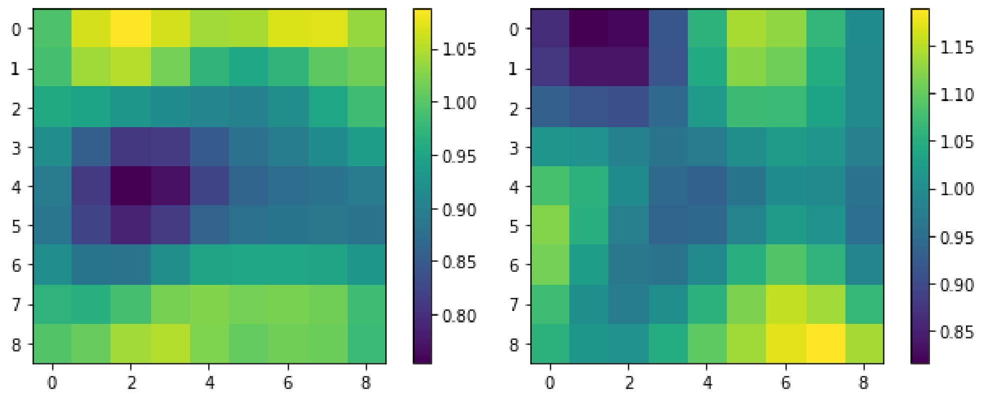


Figure 6.8: Left to right: An example of reward maps used for P_1 and P_2 in RC+.

The first line is by definition. The second line uses the same argument as in case 1. The second line follows (6.21) and the fact that at least one s^* exists. The last line follows from the definition of $\underline{V}(s)$. As with case 1, we use (6.19), (6.22) and the fact that U'_s is concave to show that $U'_s(\mu_2) > -\infty$ for $\mu_2 \in [\underline{V}(s), \overline{V}(s)]$. This completes the proof. □

We are now ready to tackle the proof of Theorem 13 (reproduced here): For all $s \in \mathcal{S}$,

$$Dom[U_s] = Dom[U'_s] = Dom[\widetilde{U}_s] = Dom[\widetilde{U}'_s] = [\underline{V}(s), \overline{V}(s)],$$

Proof. The first equality was shown in Theorem 17. We can, in fact reuse the proof of Theorem 17 by replacing U'_s and U_s with $\widetilde{U}'_s$ and $\widetilde{U}_s$. This completes this Theorem 13. □

6.7.6 Additional Details on Experimental Setup

Environment Details

For all our experiments, the network is a multilayer fully connected network of width 128, depth 8, ReLU activations and number of knots $m = 8$. We used the PyTorch library [144] and a GPU to accelerate training. No hyperparameter tuning was done.

RC Map Generation Details

Maps were generated with each reward map being drawn independently from a log Gaussian process (with query points given by the (x, y) coordinates on the grid). We use the square-exponential kernel, a length scale of 2.0 and a standard deviation of 0.1. This way of generating maps was to encourage spatial smoothness in rewards for more realism. Figure 6.8 give examples of maps generated using this procedure. From the figures, one cans see that good regions for P_1 may not be good for P_2 and vice versa.

TANTRUM Generation

For [TANTRUM], the values of q_2 were chosen (somewhat arbitrarily) to be in $\{1.5, 2.1, 3.4, 5.1, 6.7\}$.

	# Epochs	State sampling method	Prioritized replay	Traj. frequency
RC	2M	Random state	N/A	N/A
TANTRUM	4M	Unif. trajectory, layered	Yes	10
Feat. TANTRUM	2.7M	Uniform trajectory	Yes	10
RC+	1.7M	Uniform trajectory	Yes	20

Table 6.1: Differences in experimental setups over each game. Traj. frequency refers to how many epochs before we sample a new trajectory.

Training Hyperparameters

We used the Adam optimizer (Kingma and Ba) with AMSGrad (Reddi et. al.) with a learning rate of 1e-5 (except for RC, where we found using a learning rate of 1e-4 was more suitable). We use the implementation provided in the PyTorch [144] library. The replay buffer was of a size of 1M. The minibatch size was set to 128. The target network's parameters was updated once every 2000 training epochs.

Number of Epochs and Termination Criterion

Unfortunately, it is very rare to have a small loss for every single state. Hence, we terminate training at a fixed iteration. The number of epochs are given in Table 6.1.

6.7.7 Qualitative Discussion of Optimal Strategies in TANTRUM

We explore TANTRUM in the special case when $q_1 = 1$ and $q_2 > 1$. Intuitively, we should 'use' as many threats as possible. That is achieved by the leader committing to $(-1, -1)$ for all future states. If P_1 does that from the beginning, it will give P_2 $-n$ (the number of times the stage game is repeated) payoff to each player. Naturally, one upper-bound on how much P_1 can get is n/q_2, that is, P_2 chooses to accede on average of n/q_2 times per playthrough. P_1 cannot possible get more since that would lead to to P_2 losing more than n (which is the worst possible threat the leader can make from the beginning).

In our experiments, this was indeed true, and can be achieved by the following strategy. Let π be such that (a) the P_1 plays to $(0, 0)$ at all leader vertices and P_2 accedes for the first $j = \lfloor n/q_2 \rfloor$ stages with probability 1. At the $j + 1$-th vertex, it plays a mixed strategy (or rather, it receives the recommendation to mix) strategies, with probability $(n - jq_2)/q_2$ it accedes. Clearly, the expected payoff for P_2 is $-n$, and P_2 cannot do any better.

However, this result does not hold for all settings, typically when n is small. This is because we need to consider the threat is strong enough at every stage. Consider the case where $n = 3$ and $q_2 = 2$. Our derivation suggests that at the first stage, the follower accedes with probability 1 and at the second stage, it accedes with probability 0.5.

At the first stage, P_2 is indeed incentivized to accede. since if it will suffer from -3 if not, since acceding yields -2 payoff, which when combined with the expected payoff of -1 in the future, is equivalent to the threat of -3. At the second stage, it is just barely incentive compatible for the follower to accede. Specifically, if the player accedes, it will receive a payoff of -4 (it has already accumulated -2 from the previous stage). On the other hand, the grim trigger threat gives a payoff of only -4 (-2 from the past and -2 from the future). Hence, after receiving the recommendation P_2 is just incentivized to not deviate. However, when q_2 is increased by just a little (say to 2.1), this incentive is not sufficient. The **future** threat from not acceding is -2, but the follower already loses 2.1 from acceding. [8]

In general, for our derived bound to be tight, we will require

$$\underbrace{n - \lfloor n/q \rfloor}_{\text{threat from grim trigger}} \geqslant \underbrace{q}_{\text{cost from acceding this time round}},$$

that is, at the last accede recommendation (possibly with some probability), we still have enough rounds remaining as threats to maintain incentive compatibility. Technically we require this for all previous rounds; however this condition being satisfied for the last round implies that it is satisfied for all previous rounds.

We can also see from this discussion that in this repeated setting, it is always beneficial to recommend accede to the follower higher up the tree; this way, there is more room for the leader to threaten the follower with future $(-1, -1)$ actions.

Featurized TANTRUM As far as we know, there is no simple closed form solution for featurized TANTRUM.

6.8 Conclusion

We proposed a novel method of performing FA on EPFs that allows us to efficiently solve for SEFCE. This is to the best of our knowledge, the first time a such an object has been learned from state features, leading to a FA-based method of solving Stackelberg games with performance guarantees. We have explored some challenges faced when applying FA to solve SEFCE and showed how to overcome them with by using appropriate architectures and loss functions. Our result method is guaranteed to be incentive compatible and enjoys performance that is bounded by FA error. We believe that our approach will help to close the current gap between solving zero-sum and general-sum games.

Our approach suffers from several limitations, the largest of which would be the requirement to compute the grim trigger (or an approximate of it). Computing the grim trigger exactly requires solving for the minimax value of a zero-sum game exactly. This requires traversal of the game tree (in the absence of any special structure). We showed in our work that we can relax this requirement a little, by only requiring optimal play from the follower but not the leader in this grim trigger computation. However, this at best reduces the depth of the search tree by a factor of 2 (this effect is roughly equal to having the perfect action orderings when doing alpha-beta pruning). This is not enough to handle truly large games. Other limitations include not allowing for imperfect information, strong assumptions on perfect rationality from the follower (already discussed in Chapter 4), and the fairly restricted setting of SEFCE, which assumes correlation between players. We propose the following ideas as future work.

[8]This phenomena is very similar to the difference between a coarse correlated equilibrium and a regular CE. The difference is that a player has to decide to deviate before or after receiving its recommended actions.

- We believe that *learning* achievable payoffs (a generalization of EPFs introduced in [116]) may be possible in stochastic games in a similar manner. One advantage achievable sets of payoffs allows us to generate *nonstationary* policies, which traditional scalar-based values are not able to without including notion of memorizing history[192]. However, this poses fresh technical challenges, since learning sets is much more challenging than scalar functions: do we learn the extreme points of this set, or supporting hyperplanes (in the case of polytopes)? Can the Bellman backups be performed efficiently in these representations?
- A natural extension would be to combine both subgame solving and function approximation in general sum games, similar to what was done in DeepStack. This is in line with existing work in zero-sum games.
- For scalability, one could consider learning follower payoffs as well. However, care must be taken because of the potential for unfulfillable promises. A possible approach would be to learn follower payoffs with some kind of confidence bound. This would limit the probability that an unfulfillable promise is made, and allow us to bound (in a probabilistic sense) the quality of our policy. In fact, unfulfillable promises may not be all that bad: if the follower chooses a path not recommended, it could still yield a reasonable payoff for the leader. Thus, we could design algorithms where accurate learning of (follower) payoffs is focused on these "risky" states.
- Extensions to the reinforcement learning setting. In our work, we assumed that payoffs to both players were given in the game specification. What if we had to learn them while exploring the game? How do we come out with a good exploration strategy? A simple ϵ-greedy approach may not be ideal, since EPFs rarely dominate each other.
- Other nontrivial extensions are: handling imperfect information games, extensions to other general-sum equilibrium concepts, and studying simplifications in special cases (graphical, potential, mean field games etc.).

Chapter 7

Conclusion

This thesis tackles the problems of inverse game theory and scalable solving of general-sum extensive form games, with the goal of making game theory more applicable in real-world deployments. **Inverse game theory** is concerned with learning game parameters from player actions. We have designed an end-to-end machine learning algorithm based on differentiable optimization, allowing us to learn game parameters from samples of player data. Our algorithm is (relatively) scalable and allows us to efficiently learn game parameters assuming players play in accordance to the Quantal response equilibrium. **Scalable general-sum game solving** remains one of the most challenging problems in computational game theory. This thesis chips away at this problem in two separate directions: subgame solving and function approximation, both of which have been extremely successful in zero-sum games. We find that with some significant changes and restrictions, both approaches can be applied to general-sum games as well.

AI-driven decision making systems are rapidly gaining acceptance, driving the need for safe, sound, and efficient ways to handle multiagent interactions. This thesis is a step towards solving some of the practical problems that could be faced in real-world deployments. There are still many fertile grounds over and beyond those presented at the end of each chapter. These include dealing with a large number of agents, handling interactions with humans, lifelong learning, handling continuous time games, adaptively choosing appropriate equilibrium concepts. We hope that this thesis will inspire others to explore similar problems in the future.

Bibliography

[1] Yasaman Dehghani Abbasi, Martin Short, Arunesh Sinha, Nicole Sintov, Chao Zhang, and Milind Tambe. Human adversaries in opportunistic crime security games: Evaluating competing bounded rationality models. In *Proceedings of the third annual conference on advances in cognitive systems ACS*, page 2, 2015. 4.6

[2] Ilan Adler. The equivalence of linear programs and zero-sum games. *International Journal of Game Theory*, 42(1):165, 2013. 2.4.2

[3] 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. 7

[4] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019. 7

[5] Kareem Amin, Satinder Singh, and Michael P Wellman. Gradient methods for stackelberg security games. In *Conference on Uncertainty in Artificial Intelligence*, pages 2–11, 2016. 3.1.2

[6] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017. 3.1.3, 3.1.3, 3.3.3

[7] Bo An, Milind Tambe, and Arunesh Sinha. Stackelberg security games (ssg) basics and application overview. *Improving Homeland Security Decisions*, page 485, 2017. 4.6

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6): 26–38, 2017. 6.5.4

[9] Susan Athey and Philip A Haile. Identification of standard auction models. *Econometrica*, 70(6):2107–2140, 2002. 3

[10] Afraa Attiah, Mainak Chatterjee, and Cliff C Zou. A game theoretic approach to model cyber attack and defense strategies. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018. 1

[11] Robert J Aumann. Correlated equilibrium as an expression of bayesian rationality. *Econometrica: Journal of the Econometric Society*, pages 1–18, 1987. 2.4.4

[12] Robert J Aumann. 28. mixed and behavior strategies in infinite extensive games. In *Advances in Game Theory.(AM-52), Volume 52*, pages 627–650. Princeton University Press,

2016. 2.3.3

[13] Anton Bakhtin, David Wu, Adam Lerer, and Noam Brown. No-press diplomacy from scratch. *Advances in Neural Information Processing Systems*, 34, 2021. 6.1

[14] Shane Barratt. On the differentiability of the solution to convex optimization problems. *arXiv preprint arXiv:1804.05098*, 2018. 7

[15] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13 (2):31–39, 2011. 3.5.7

[16] David Blackwell. An analog of the minimax theorem for vector payoffs. 1956. 2.4.2

[17] Avrim Blum, Nika Haghtalab, and Ariel D Procaccia. Learning optimal commitment to overcome insecurity. In *Advances in Neural Information Processing Systems*, pages 1826–1834, 2014. 3

[18] Branislav Bosansky and Jiri Cermak. Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. 4.1.2, 4.1.3, 4.3.3, 18, 4.4, 6.1

[19] Branislav Bosanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Computation of stackelberg equilibria of finite sequential games. *CoRR*, abs/1507.07677, 2015. URL `http://arxiv.org/abs/1507.07677`. 6.1, 23, 9

[20] Branislav Bošanskỳ, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, and Peter Bro Miltersen. Computation of stackelberg equilibria of finite sequential games. *ACM Transactions on Economics and Computation (TEAC)*, 5(4):1–24, 2017. 6.1, 6.2.1, 6.3, 9

[21] Hartwig Bosse, Jaroslaw Byrka, and Evangelos Markakis. New algorithms for approximate nash equilibria in bimatrix games. In *Internet and Network Economics: Third International Workshop, WINE 2007, San Diego, CA, USA, December 12-14, 2007. Proceedings 3*, pages 17–29. Springer, 2007. 2.4.2

[22] Michael Bowling and Manuela Veloso. An analysis of stochastic game theory for multiagent reinforcement learning. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2000. 3

[23] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015. 1

[24] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 3.2.1

[25] Steven J Brams. Game theory: Pitfalls and opportunities in applying it to international relations. *International Studies Perspectives*, 1(3):221–232, 2000. 1.1

[26] Noam Brown. Equilibrium finding for large adversarial imperfect-information games. *PhD thesis*, 2020. 1

[27] Noam Brown and Tuomas Sandholm. Libratus: the superhuman ai for no-limit poker. In

Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017. 1, 2.4.2, 14, 16, 6.1

[28] Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *Advances in neural information processing systems*, pages 689–699, 2017. 1, 4, 2, 4.3.2, 5.2.2

[29] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733, 2017. 1, 4

[30] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1829–1836, 2019. 2.4.2

[31] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365 (6456):885–890, 2019. 1, 5, 4, 6.1

[32] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International conference on machine learning*, pages 793–802. PMLR, 2019. 1

[33] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems*, 33:17057–17069, 2020. 1

[34] Neil Burch. Time and space: Why imperfect information games are hard. 2018. 1, 1

[35] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. 4, 4.3.4, 4.5

[36] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008. 3

[37] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002. 4

[38] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Multi-objective fitted q-iteration: Pareto frontier approximation in one single run. In *2011 International Conference on Networking, Sensing and Control*, pages 260–265. IEEE, 2011. 6.2.3

[39] Andrea Celli, Alberto Marchesi, Gabriele Farina, and Nicola Gatti. No-regret learning dynamics for extensive-form correlated equilibrium. *Advances in Neural Information Processing Systems*, 33:7722–7732, 2020. 5.5

[40] Jiri Cermak, Branislav Bosansky, Karel Durkota, Viliam Lisy, and Christopher Kiekintveld. Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 6.1

[41] Jakub Černỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. Incremental strategy generation for stackelberg equilibria in extensive-form games. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 151–168. ACM, 2018. 4.1.2,

4.3.4, 4.4, 6.1

[42] Jakub Černỳ, Viliam Lisỳ, Branislav Bošanskỳ, and Bo An. Computing quantal stackelberg equilibrium in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5260–5268, 2021. 4.6

[43] Nicolo Cesa-Bianchi and Gábor Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51:239–261, 2003. 2.4.2

[44] Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1-2):253–287, 2016. 3.4.2, 1

[45] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009. 2.4.2

[46] Vincent Conitzer and Dmytro Korzhyk. Commitment to correlated strategies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 632–637, 2011. 22

[47] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 82–90. ACM, 2006. 1.2, 4.1.2

[48] Vincent Conitzer and Tuomas Sandholm. New complexity results about nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008. 2.4.2

[49] GB Dantzig. A proof of the equivalence of the programming problem and the game problem, in “activity analysis of production and allocation”(ed. tc koopmans), cowles commission monograph, no. 13, 1951. 2.4.2

[50] Constantinos Daskalakis, Aranyak Mehta, and Christos Papadimitriou. A note on approximate nash equilibria. In *Internet and Network Economics: Second International Workshop, WINE 2006, Patras, Greece, December 15-17, 2006. Proceedings 2*, pages 297–306. Springer, 2006. 2.4.2

[51] Constantinos Daskalakis, Aranyak Mehta, and Christos Papadimitriou. Progress in approximate nash equilibria. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 355–358, 2007. 2.4.2

[52] 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. 2.4.2

[53] Trevor Davis, Kevin Waugh, and Michael Bowling. Solving large extensive-form games with strategy constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1861–1868, 2019. 11

[54] Dave De Jonge and Dongmo Zhang. Strategic negotiations for extensive-form games. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–41, 2020. 4

[55] Mac Dermed and Liam Charles. *Value methods for efficiently solving stochastic games of complete and incomplete information*. PhD thesis, Georgia Institute of Technology, 2013. 6.2.3

[56] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016. 7

[57] Thomas Dietterich and Xin Wang. Batch value function approximation via support vectors. *Advances in neural information processing systems*, 14, 2001. 6.2.2

[58] Miroslav Dudík and Geoffrey J. Gordon. A sampling-based approach to computing equilibria in succinct extensive-form games. In *UAI*, 2009. 5.1.3

[59] Fei Fang, Thanh Hong Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Milind Tambe, and Andrew Lemieux. Deploying paws: Field optimization of the protection assistant for wildlife security. 2016. 1, 3.1.1

[60] Fei Fang, Thanh H Nguyen, Robert Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Brian C Schwedock, Milind Tambe, and Andrew Lemieux. Paws-a deployed game-theoretic application to combat poaching. *AI Magazine*, 38(1):23, 2017. 1, 1.1, 2.4.3, 4, 6.1

[61] Gabriele Farina and Tuomas Sandholm. Polynomial-time computation of optimal correlated equilibria in two-player extensive-form games with public chance moves and beyond. *arXiv preprint arXiv:2009.04336*, 2020. 5, 4

[62] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Regret circuits: Composability of regret minimizers. In *International conference on machine learning*, pages 1863–1872. PMLR, 2019. 11, 5.3.3

[63] Gabriele Farina, Chun Kai Ling, Fei Fang, and Tuomas Sandholm. Correlation in extensive-form games: Saddle-point formulation and benchmarks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/60ce36723c17bbac504f2ef4c8a46995-Paper.pdf`. 1.2, 1.3, 5, 5.1.3, 5.1.3, 19, 5.3.3, 5.4

[64] Gabriele Farina, Chun Kai Ling, Fei Fang, and Tuomas Sandholm. Efficient regret minimization algorithm for extensive-form correlated equilibrium. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/aee92f16efd522b9326c25cc3237ac15-Paper.pdf`. 1.2, 1.3, 3, 5.1, 5.1.3, 5.3.3, 5.3.3, 5.3.3, 12, 13, 5.3.3, 5.3.3, 7, 5.3.3, 5.4, 5.5

[65] Gabriele Farina, Andrea Celli, and Tuomas Sandholm. Efficient decentralized learning dynamics for extensive-form coarse correlated equilibrium: No expensive computation of stationary distributions required. *arXiv preprint arXiv:2109.08138*, 2021. 5.5

[66] John Fearnley, Martin Gairing, Paul W Goldberg, and Rahul Savani. Learning equilibria of games via payoff queries. *Journal of Machine Learning Research*, 16:1305–1344, 2015. 3

[67] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter

Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017. 6.2.3

[68] Dean P Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1-2):7–35, 1999. 2.4.2

[69] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 2.4.2

[70] David Gale, Harold W Kuhn, and Albert W Tucker. Linear programming and the theory of games. *Activity analysis of production and allocation*, 13:317–335, 1951. 2.4.2

[71] Andrea Galeotti, Sanjeev Goyal, Matthew O Jackson, Fernando Vega-Redondo, and Leeat Yariv. Network games. *The review of economic studies*, 77(1):218–244, 2010. 2.4.2

[72] Jiarui Gan, Edith Elkind, and Michael Wooldridge. Stackelberg security games with multiple uncoordinated defenders. 2018. 2.4.3

[73] Philipp Geiger and Christoph-Nikolas Straehle. Learning game-theoretic models of multiagent trajectories using implicit layers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4950–4958, 2021. 3.7

[74] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989. 2.4.2

[75] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016. 3.1.3, 3.1.3, 3.3.3

[76] Jonathan Gray, Adam Lerer, Anton Bakhtin, and Noam Brown. Human-level performance in no-press diplomacy via equilibrium search. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0-uUGPbIjD`. 6.1

[77] Amy Greenwald and Amir Jafari. A general class of no-regret learning algorithms and game-theoretic equilibria. In *COLT*, volume 3, pages 2–12, 2003. 2.4.4

[78] Amy Greenwald, Keith Hall, Roberto Serrano, et al. Correlated q-learning. In *ICML*, volume 3, pages 242–249, 2003. 6.2.3

[79] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL `http://www.gurobi.com`. 4.4, 5.4

[80] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957. 2.4.2

[81] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Na-*

ture, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`. 3.5.7

[82] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. 4

[83] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000. 2.4.2

[84] Howard Heaton, Daniel McKenzie, Qiuwei Li, Samy Wu Fung, Stanley Osher, and Wotao Yin. Learn to predict equilibria via fixed point networks. *arXiv preprint arXiv:2106.00906*, 2021. 3.7

[85] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016. 1

[86] P Jean-Jacques Herings and Ronald Peeters. Homotopy methods to compute equilibria in game theory. *Economic Theory*, 42(1):119–156, 2010. 2.4.2

[87] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012. 2

[88] Samid Hoda, Andrew Gilpin, Javier Pena, and Tuomas Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010. 2.3.4, 3.2.1, 3.4.3

[89] Laurence T Hunt, Robb B Rutledge, WM Nishantha Malalasekera, Steven W Kennerley, and Raymond J Dolan. Approach-induced biases in human information sampling. *PLoS biology*, 14(11):e2000638, 2016. 3.5.6, 3.5.6

[90] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *ICML*, volume 1, pages 226–233, 2001. 2.4.4

[91] 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 *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 327–334. Citeseer, 2011. 6.1

[92] Manish Jain, Bo An, and Milind Tambe. Security games applied to real-world: Research contributions and challenges. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, pages 15–39. Springer, 2013. 2.4.3

[93] Albert Xin Jiang, Kevin Leyton-Brown, and Navin AR Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, 2011. 2.4.2

[94] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016. 3.1.3

[95] Debarun Kar, Thanh H Nguyen, Fei Fang, Matthew Brown, Arunesh Sinha, Milind Tambe, and Albert Xin Jiang. Trends and applications in stackelberg security games. *Handbook of dynamic game theory*, pages 1–47, 2017. 2.4.3

[96] Jan Karwowski and Jacek Mańdziuk. Double-oracle sampling method for stackelberg equilibrium approximation in general-sum extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2054–2061, 2020. 6.1

[97] Garry Kasparov. Chess, a drosophila of reasoning. *Science*, 362(6419):1087–1087, 2018. 6.1

[98] Michael Kearns. Graphical games. *Algorithmic game theory*, 3:159–180, 2007. 2.4.2

[99] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 689–696. International Foundation for Autonomous Agents and Multiagent Systems, 2009. 2.4.3

[100] Spyros C Kontogiannis, Panagiota N Panagopoulou, and Paul G Spirakis. Polynomial algorithms for approximating nash equilibria of bimatrix games. In *WINE*, pages 286–296. Springer, 2006. 2.4.2

[101] Kavya Kopparapu, Edgar A Duéñez-Guzmán, Jayd Matyas, Alexander Sasha Vezhnevets, John P Agapiou, Kevin R McKee, Richard Everett, Janusz Marecki, Joel Z Leibo, and Thore Graepel. Hidden agenda: a social deduction game with diverse learned equilibria. *arXiv preprint arXiv:2201.01816*, 2022. 1

[102] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 805–810, 2010. 2.4.3

[103] Christian Kroer, Kevin Waugh, Fatma Kilinc-Karzan, and Tuomas Sandholm. Theoretical and practical advances on smoothing for extensive-form games. *arXiv preprint arXiv:1702.04849*, 2017. 1, 2.4.2, 3.2.1, 3.2.1

[104] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Solving large sequential games with the excessive gap technique. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 864–874. Curran Associates, Inc., 2018. 2.3.4

[105] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Robust stackelberg equilibria in extensive-form games and extension to limited lookahead. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 2

[106] Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950. 2.3.4, 2

[107] Volodymyr Kuleshov and Okke Schrijvers. Inverse game theory: Learning utilities in succinct games. In *Web and Internet Economics: 11th International Conference, WINE 2015, Amsterdam, The Netherlands, December 9-12, 2015, Proceedings 11*, pages 413–427. Springer, 2015. 3

[108] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003. 6.2.2

[109] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sam-

pling for regret minimization in extensive games. *Advances in neural information processing systems*, 22, 2009. 2.4.2

[110] 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. *Advances in neural information processing systems*, 30, 2017. 1, 2.4.2, 1

[111] Kyungjae Lee, Sungjoon Choi, and Songhwai Oh. Sparse markov decision processes with causal sparse tsallis entropy regularization for reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1466–1473, 2018. 3.7

[112] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017. 6.2.3

[113] 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. 2.4.2

[114] Joshua Letchford and Vincent Conitzer. Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 83–92. ACM, 2010. 2.4.3, 4, 4.1.2, 6.1, 2, 6.3, 6.7.1

[115] Joshua Letchford, Vincent Conitzer, and Kamesh Munagala. Learning and approximating the optimal strategy to commit to. In *International Symposium on Algorithmic Game Theory*, pages 250–262. Springer, 2009. 1.2, 2, 3

[116] 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. 6.2.3, 6.8

[117] Jiayang Li, Jing Yu, Yu Nie, and Zhaoran Wang. End-to-end learning and intervention in games. *Advances in Neural Information Processing Systems*, 33:16653–16665, 2020. 3.7

[118] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965. 4

[119] Chun Kai Ling and Noam Brown. Safe search for stackelberg equilibria in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5541–5548, 2021. 1.3

[120] Chun Kai Ling and Fei Fang. Safe subgame resolving for extensive form correlated equilibrium. 2022. 1.3, 5.3.1

[121] Chun Kai Ling, Fei Fang, and J Zico Kolter. What game are we playing? end-to-end learning in normal and extensive form games. *arXiv preprint arXiv:1805.02777*, 2018. 1.3, 3.3.3, 3.5

[122] Chun Kai Ling, Fei Fang, and J Zico Kolter. Large scale learning of agent rationality in two-player zero-sum games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6104–6111, 2019. 1.3, 3.5

[123] Chun Kai Ling, J Zico Kolter, and Fei Fang. Function approximation for solving stackelberg equilibrium in large perfect information games. *arXiv preprint arXiv:2212.14431*,

2022. 1.3, 23, 6.5.4

[124] Richard J Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, pages 36–41, 2003. 2.4.2

[125] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017. 2.4.2

[126] R Duncan Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2012. 3

[127] Liam MacDermed, Karthik S Narayan, Charles L Isbell, and Lora Weiss. Quick polytope approximation of all correlated equilibria in stochastic games. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011. 6.2.3

[128] Stephen McAleer, Kevin Wang, John B Lanier, Marc Lanctot, Pierre Baldi, Tuomas Sandholm, and Roy Fox. Anytime psro for two-player zero-sum games. *CoRR, abs/2201.07700, 2022b. URL https://arxiv. org/abs/2201.07700*, 2022. 2.4.2

[129] Daniel McFadden et al. Conditional logit analysis of qualitative choice behavior. 1973. 3, 8

[130] Richard D McKelvey and Thomas R Palfrey. Quantal response equilibria for normal form games. *Games and economic behavior*, 10(1):6–38, 1995. 2, 3.1.2

[131] Richard D McKelvey and Thomas R Palfrey. Quantal response equilibria for extensive form games. *Experimental economics*, 1(1):9–41, 1998. 3.2.1, 1

[132] Panayotis Mertikopoulos and William H Sandholm. Learning in games via reinforcement and regularization. *Mathematics of Operations Research*, 41(4):1297–1324, 2016. 3.1.2

[133] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 6.5.4, 6.5.4

[134] Dov Monderer and Lloyd S Shapley. Potential games. *Games and economic behavior*, 14 (1):124–143, 1996. 2.4.2

[135] Matej Moravcik, Martin Schmid, Karel Ha, Milan Hladik, and Stephen J Gaukrodger. Refining subgames in large imperfect information games. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. 4

[136] 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. 1, 1, 2.4.2, 4, 16, 5.5, 6.1

[137] Dustin Morrill, Ryan D'Orazio, Reca Sarfati, Marc Lanctot, James R Wright, Amy Greenwald, and Michael Bowling. Hindsight and sequential rationality of correlated play. *arXiv preprint arXiv:2012.05874*, 2020. 2.4.4, 1

[138] Dustin Morrill, Ryan D'Orazio, Marc Lanctot, James R Wright, Michael Bowling, and Amy Greenwald. Efficient deviation types and learning for hindsight rationality in extensive-form games. *arXiv preprint arXiv:2102.06973*, 2021. 5.5

[139] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008. 6.2.2

[140] Chris Murray and Geoff Gordon. *Finding correlated equilibria in general sum stochastic games*. 2007. 6.2.3

[141] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951. 2.4.2

[142] Thanh Nguyen, Rong Yang, Amos Azaria, Sarit Kraus, and Milind Tambe. Analyzing the effectiveness of adversary modeling in security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 718–724, 2013. 4.6

[143] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge university press, 2007. 2

[144] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 6, 3.5.7, 6.5.4, 6.7.6, 6.7.6

[145] Jeffrey Pawlick, Edward Colbert, and Quanyan Zhu. A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. *ACM Computing Surveys (CSUR)*, 52(4):1–28, 2019. 1

[146] Julien Pérolat, Florian Strub, Bilal Piot, and Olivier Pietquin. Learning nash equilibrium for general-sum markov games from batch data. In *Artificial Intelligence and Statistics*, pages 232–241. PMLR, 2017. 6.2.3

[147] Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *arXiv preprint arXiv:2206.15378*, 2022. 2.4.2, 6.2.3

[148] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. International Foundation for Autonomous Agents and Multiagent Systems, 2008. 2.4.3, 4

[149] James Pita, Manish Jain, Fernando Ordónez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Armor security for los angeles international airport. In *AAAI*, pages 1884–1885, 2008. 1.1, 6.1

[150] James Pita, Manish Jain, Fernando Ordónez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Using game theory for los angeles airport security. *Ai Magazine*, 30(1):43, 2009. 1

[151] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008. 2.4.2

[152] Sheldon M. Ross. Goofspiel — the game of pure strategy. *Journal of Applied Probability*, 8(3):621–625, 1971. doi: 10.2307/3212187. 4.4.2

[153] Tim Roughgarden. Routing games. *Algorithmic game theory*, 18:459–484, 2007. 2.4.2

[154] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding nash equilibria. In *AAAI*, pages 495–501, 2005. 2.4.2

[155] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 6.5.4, 6.5.4

[156] Martin Schmid. Search in imperfect information games. *arXiv preprint arXiv:2111.05884*, 2021. 1, 1

[157] Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, et al. Player of games. *arXiv preprint arXiv:2112.03178*, 2021. 16, 6.1

[158] Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. Finding friend and foe in multi-agent games. *Advances in Neural Information Processing Systems*, 32, 2019. 1

[159] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 4, 6.1

[160] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 4, 6.1

[161] Arunesh Sinha, Thanh H Nguyen, Debarun Kar, Matthew Brown, Milind Tambe, and Albert Xin Jiang. From physical security to cybersecurity. *Journal of Cybersecurity*, 1(1): 19–35, 2015. 2.4.3

[162] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. Stackelberg security games: Looking beyond a decade of success. IJCAI, 2018. 2.4.3, 4.6

[163] Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*, 2012. 4.4.3

[164] Eric Steinberger, Adam Lerer, and Noam Brown. Dream: Deep regret minimization with advantage baselines and model-free learning. *arXiv preprint arXiv:2006.10410*, 2020. 1

[165] Randall W Stone. The use and abuse of game theory in international relations: The theory of moves. *Journal of Conflict Resolution*, 45(2):216–244, 2001. 1.1

[166] P Suppes. Preference, utility and subjective probability. inhandbook of mathematical psychology, ed. rd luce, rr bush and eh galanter, 3, 249–410, 1965. 3

[167] Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011. 1, 2, 2.4.3

[168] Oskari Tammelin. Solving large imperfect information games using cfr+. *arXiv preprint arXiv:1407.5042*, 2014. 2.4.2

[169] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995. 4

[170] Louis L Thurstone. A law of comparative judgment. *Psychological review*, 34(4):273, 1927. 3.3.1

[171] Kenneth E Train. *Discrete choice methods with simulation*. Cambridge university press, 2009. 3.3.1

[172] Haralampos Tsaknakis and Paul G Spirakis. An optimization approach for approximate nash equilibria. In *Internet and Network Economics: Third International Workshop, WINE 2007, San Diego, CA, USA, December 12-14, 2007. Proceedings 3*, pages 42–56. Springer, 2007. 2.4.2

[173] Amos Tversky. Elimination by aspects: A theory of choice. *Psychological review*, 79(4): 281, 1972. 3.3.1

[174] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019. 1, 6.1

[175] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2. 3.5.7

[176] J. von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947. 1.1

[177] Heinrich Von Stackelberg. *Marktform und gleichgewicht*. J. springer, 1934. 2.4.3

[178] Bernhard Von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996. 2.4.2, 3.2.1, 4.1.3, 4.4.2, 5.3.3, 5.3.3

[179] Bernhard Von Stengel and Françoise Forges. Extensive-form correlated equilibrium: Def-

inition and computational complexity. *Mathematics of Operations Research*, 33(4):1002–1022, 2008. 1.2, 4, 2, 2.3.4, 2.4.2, 2.4.4, 5, 3, 5.1.3, 5.1.3, 6, 5.1.3, 5.1, 5.1.3, 5, 5.3.2

[180] Bernhard Von Stengel and Shmuel Zamir. Leadership games with convex strategy sets. *Games and Economic Behavior*, 69(2):446–457, 2010. 2.4.3, 2.4.3, 4

[181] Yevgeniy Vorobeychik, Michael P Wellman, and Satinder Singh. Learning payoff functions in infinite games. *Mach Learn*, 67:145–168, 2007. 3

[182] Kai Wang, Lily Xu, Andrew Perrault, Michael K Reiter, and Milind Tambe. Coordinating followers to reach better equilibria: End-to-end gradient descent for stackelberg games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5219–5227, 2022. 3.7

[183] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019. 1

[184] Kevin Waugh, Brian D Ziebart, and J Andrew Bagnell. Computational rationalization: the inverse equilibrium problem. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 1169–1176. Omnipress, 2011. 3

[185] Jörgen W Weibull. *Evolutionary game theory*. MIT press, 1997. 2

[186] Andrzej Wilczyński, Agnieszka Jakóbik, and Joanna Kołodziej. Stackelberg security games: Models, applications and computational aspects. *Journal of Telecommunications and Information Technology*, (3):70–79, 2016. 2.4.3

[187] Alireza Zarreh, Can Saygin, HungDa Wan, Yooneun Lee, and Alejandro Bracho. A game theory based cybersecurity assessment model for advanced manufacturing systems. *Procedia Manufacturing*, 26:1255–1264, 2018. 1

[188] Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376*, 2019. 1

[189] Brian Hu Zhang and Tuomas Sandholm. Subgame solving without common knowledge. *arXiv preprint arXiv:2106.06068*, 2021. 1, 15

[190] Han Zhong, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Can reinforcement learning find stackelberg-nash equilibria in general-sum markov games with myopic followers? *arXiv preprint arXiv:2112.13521*, 2021. 6.2.3

[191] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. icml'03, 2003. 2.4.2

[192] Martin Zinkevich, Amy Greenwald, and Michael Littman. Cyclic equilibria in markov games. *Advances in neural information processing systems*, 18, 2005. 6.2.3, 6.8

[193] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008. 1, 2.4.2, 5.3.3, 5.3.3