Equilibrium Finding for Large Adversarial Imperfect-Information Games

Noam Brown

CMU-CS-20-132

September 2020

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

Thesis Committee:
Tuomas Sandholm, Chair
Geoff Gordon
Ariel Procaccia
Satinder Singh, University of Michigan
Michael Wellman, University of Michigan

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

This research was sponsored by Facebook, the US army under grant number W911NF1610061, the Open Philanthropy Fellowship, the Tencent Fellowship, and the National Science Foundation under grant numbers IIS-0964579, CCF-1101668, IIS-1320620, IIS-1546752, IIS-1617590, and IIS-1718457. 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: Artificial Intelligence, Machine Learning, Game Theory, Equilibrium Finding, Regret Minimization, No-Regret Learning, Imperfect-Information Games, Reinforcement Learning, Deep Reinforcement Learning, Multi-Agent Reinforcement Learning

To my family.

Abstract

Imperfect-information games model strategic interactions involving multiple agents with private information. A typical goal in this setting is to approximate an equilibrium in which all agents' strategies are optimal. This thesis describes a number of advances in the computation of equilibria in large adversarial imperfect-information games. Together, these new techniques made it possible for the first time for an AI agent to defeat top human professionals in full-scale poker, which had for decades served as a grand challenge problem in the fields of AI and game theory.

We begin by introducing faster equilibrium-finding algorithms based on counterfactual regret minimization (CFR), an iterative algorithmic framework that converges to a Nash equilibrium in two-player zero-sum games. We describe new variants of CFR that use discounting to dramatically speed up convergence. These discounting variants are now the state-of-the-art equilibrium-finding algorithms for large adversarial imperfect-information games. We also describe theoretically sound pruning techniques that can speed up CFR by orders of magnitude in large games. Additionally, we introduce the first general technique for warm starting CFR.

Next, we describe new ways to scale CFR to extremely large games via automated abstraction and function approximation. In particular, we introduce the first provably locally optimal algorithm for discretizing continuous action spaces in imperfect-information games. We extend this into an algorithm that converges to an equilibrium even in games with continuous action spaces. We also introduce Deep CFR, a form of CFR that uses neural network function approximation rather than bucketing-based abstractions. Deep CFR was the first non-tabular form of CFR to scale to large games and enables CFR to be deployed in settings with little domain knowledge.

We also present new search techniques for imperfect-information games that ensure the agent's search strategy cannot be exploited by an adversary. These new forms of search outperform past approaches both in theory and in practice. We describe how these search techniques were used in the construction of Libratus to defeat top humans in two-player no-limit poker for the first time. Additionally, we introduce a method for depth-limited search that is orders of magnitude more efficient than prior approaches. We describe how this depth-limited search technique was incorporated into Pluribus, which defeated top human professionals in multiplayer poker for the first time. Finally, we present an algorithm that combines deep reinforcement learning with search at both training and test time, which takes a major step toward bridging the gap between research on perfect-information games and imperfect-information games.

Preface

Section 3.4 on dynamic thresholding was a collaboration with Christian Kroer. Christian Kroer conducted experiments on dynamic thresholding with the excessive gap technique.

Section 4.1 on Deep CFR was a collaboration with Adam Lerer and Sam Gross. Adam Lerer and Sam Gross designed the architecture for the neural network used in the experiments, handled the training of the neural network based on the generated data, contributed key improvements to the Deep CFR algorithm, proved Theorem 13, and helped with running the experiments.

Section 5.2 on depth-limited search via multi-valued states was a collaboration with Brandon Amos. Brandom Amos designed the architecture for the neural network used in the experiments and handled the training of the neural network based on the generated data.

Section 5.3 on ReBeL was a collaboration with Anton Bakhtin, Adam Lerer, and Qucheng Gong. Anton Bakhtin designed the architecture for the neural network used in the experiments, handled the training of the neural network for poker based on the generated data, developed the infrastructure for scaling the experiments to multiple machines, wrote the entirety of the open-sourced code for Liar's Dice, and contributed key improvements to the ReBeL algorithm. Adam Lerer contributed key improvements to the ReBeL algorithm and proved Theorem 23, Theorem 24, and Theorem 25. Qucheng Gong developed the initial neural network architecture and developed the initial neural network training framework for poker given the generated data.

Tartanian7 and Claudico were both collaborations with Sam Ganzfried. Sam Ganzfried developed the abstractions, the endgame solver, the client-server interface, and contributed to the planning for both agents.

Tartanian7 and Claudico were developed on the Blacklight Supercomputer at the Pittsburgh Supercomputing Center, with key support provided by John Urbanic. Baby Tartanian8 was developed using the Comet Supercomputer at the San Diego Supercomputer Center. Libratus and Pluribus were developed on the Bridges Supercomputer at the Pittsburgh Supercomputing Center.

All of the research in this thesis, with the exception of ReBeL in Section 5.3, was a collaboration with my advisor Tuomas Sandholm.

Acknowledgments

I would first like to thank my advisor Tuomas Sandholm. Tuomas patiently guided me through the research process, including several late-night pizza-fueled paper writing sessions. Without that guidance, my PhD would surely not have been as successful.

I am also fortunate to have collaborated with several others throughout my PhD: Sam Ganzfried, Christian Kroer, Gabriele Farina, Brandom Amos, Adam Lerer, Sam Gross, Anton Bakhtin, and Qucheng Gong. Without them, this research would not be possible.

I would also like to thank the wider computer poker research community, whose research forms the foundation on which this thesis is built and who have been more than happy to help me become familiar with the background material: Michael Bowling, Johnny Hawkin, Michael Johanson, Neil Burch, Nolan Bard, Kevin Waugh, Marc Lanctot, Eric Jackson, Tim Reiff, and many others. Thank you also to Jimmy Chou, Dong Kim, Jason Les, Bjorn Li, Daniel MacAulay, Doug Polk, Seth Davies, Michael Gagliano, Anthony Gregg, Linus Loeliger, Greg Merson, Nicholas Petrangelo, Sean Ruane, Trevor Savage, and Jacob Toole for your years of studying poker and effort in battling against Claudico, Libratus, and Pluribus, which allowed us to measure our bots against the best of what humanity has to offer.

I am grateful for the help that Rob Fergus, Geoff Gordon, Zico Kolter, Ariel Procaccia, Satinder Singth, and Michael Wellman provided in navigating the challenges of academia. I also greatly appreciate the flexibility and responsiveness of Deb Cavlovich, Catherine Copetas, and the rest of the CMU administrative staff during my time at CMU.

During my several years in Pittsburgh, I was lucky to have been surrounded by wonderful colleagues and friends. Thank you all for sharing afternoon tea with me, for the many games of Avalon and Hanabi, for the Canadian Thanksgiving celebrations, and for the summer twilight barbecues. All of you made my time in Pittsburgh a wonderful experience.

I was very fortunate to have worked with Jason Solomon and Matthew Urbaniak during and after my undergraduate education. The financial security I gained from that job allowed me to make the risky choice of pursuing a career in research, and allowed me to pay tuition for my first semester as a Masters student at Carnegie Mellon. Without that financial security, I would have chosen a different path.

Finally, a very special thank you to my parents Michael and Nurit, and to my whole family, for always supporting me and for always encouraging me to pursue my passion.

"And that's why there's never going to be a computer that can play world-class poker. It's a people game." –Doyle Brunson, *Super/System*, 1979

"The analysis of a more realistic poker game than our very simple model should be quite an interesting affair." –John Forbes Nash, 1951

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A primary goal for the field of artificial intelligence (AI) is developing agents capable of acting optimally in the real world. However, the real world is not stationary; there are humans and other agents with which the agent must interact. In order to be effective in multi-agent interactions, an AI agent must be able to reason strategically not just about its actions, but also about the possible actions of other agents. Game theory, which studies rational behavior in multi-agent interactions (i.e., "games"), provides a theoretical framework for determining strategies in these settings via computation of equilibria, in which each agent's strategy is an optimal response to every other agent's strategy.

This thesis describes new techniques for computing equilibria in large adversarial imperfect-information games. Perfect-information games, in which all agents know the exact state of the world, have been the subject of extensive AI research. However, most real-world interactions, such as negotiations, cybersecurity, and traffic navigation, involve one or more agents having access to information that is hidden from other agents. Such games are referred to as imperfect-information games, and have been studied far less despite their importance. Imperfect-information games exist on a spectrum from purely adversarial two-player zero-sum games, in which one agent's gain is the other agent's loss, to purely cooperative games, in which all agents share rewards. The theory for the techniques in this thesis applies primarily to two-player zero-sum games. However, in practice we observe these techniques produce competitive strategies in a broader set of adversarial games, such as six-player poker, and some of the theory extends to such settings as well.

We evaluate the methods described in this thesis on benchmark games, including the popular recreational game of poker. Since the very earliest research on AI, recreational games have been used as benchmarks for progress not just for strategic reasoning, but for the whole field of AI. The use of recreational games for this purpose is primarily motivated by three factors. First, the rules of a recreational game are well defined and have a clear scoring system, which allows performance to be objectively measured. Second, recreational games were developed by humans for humans, rather than being developed specifically for the evaluation of a particular AI algorithm. This mitigates the risk that AI researchers would choose a benchmark that sets the bar artificially low for their specific algorithm. Third, there are humans who have dedicated their lives toward reaching the pinnacle of human performance in a specific game. By playing against these humans, it is possible to measure whether AI has truly surpassed top human performance

in a particular benchmark. Of course, a likely additional motivation was the enjoyment of the games themselves.

Among imperfect-information games, poker has served for decades as the primary challenge problem for the fields of AI and game theory [8]. In fact, it is poker specifically that the foundational papers on game theory used to illustrate their concepts starting a century ago [112, 156]. The reason for this choice is simple: no other popular recreational game captures the challenges of hidden information as effectively and as elegantly as poker. In particular, researchers set the goal of surpassing top human performance in no-limit Texas hold'em poker, which is by far the most popular variant of poker in the world. While previous AI programs have defeated top humans in perfect-information games such as backgammon [152], checkers [130], chess [34], and Go [140], poker stubbornly proved resistant to those approaches.

This thesis describes a series of advances that form the new state-of-the-art approach to AI for adversarial imperfect-information games. Together, these techniques made it possible for the first time to defeat top humans in no-limit Texas hold'em poker. Moreover, the techniques are not specific to poker and can be widely applied to adversarial imperfect-information games in general.

Chapter 3 describes new variants of counterfactual regret minimization (CFR) [163], an iterative algorithmic framework for approximating equilibria in imperfect-information games. Discounted CFR and Linear CFR, introduced in Section 3.1, dramatically speed up the empirical convergence of CFR and are now the state-of-the-art equilibrium-finding algorithms for large adversarial imperfect-information games. Section 3.2 introduces the first theoretically sound technique for warm starting CFR from any arbitrary starting strategy. Sections 3.3-3.5 introduce pruning techniques that make CFR faster and use less memory asymptotically, both in theory and in practice, by allowing each iteration to traverse fewer nodes. In large games, this can improve the speed of CFR and reduce memory usage by an order of magnitude or more.

Chapter 4 describes new techniques for scaling CFR to games that are too large to be represented exactly, such as no-limit Texas hold'em poker which has more than 10^{161} decision points. In the past, this was accomplished using information abstraction, which buckets similar situations together based on chance outcomes, and action abstraction, which reduces continuous action spaces to a small discrete number. These techniques produce an abstract game, which is much smaller than the original game. The abstract game would be small enough to solve with tabular CFR and its solution used as an approximation of the solution for the full game. Section 4.1 introduces Deep CFR, a form of CFR that replaces information abstraction with deep neural network function approximation to estimate regrets for unseen situations. Deep CFR was the first non-tabular form of CFR to be successful in large games. Section 4.2 introduces a method for automated action abstraction in a way that is provably locally optimal. This was the first action abstraction algorithm with theoretical guarantees of local optimality. Section 4.3 takes this further and develops a technique for gradually expanding the action abstraction while also solving the game. This technique provably converges to an equilibrium even in games with continuous action spaces.

Chapter 5 describes new methods for conducting search in imperfect-information games. Search has been critical for achieving superhuman performance in perfect-information games such as backgammon [152], chess [34], and Go [140]. However, the search techniques used in perfect-information games are not theoretically sound and empirically ineffective in games with

hidden information. There were two main obstacles that needed to be overcome in order to make search theoretically sound in imperfect-information games: determining what the players' beliefs should be about the state of the world at the start of the search tree, and determining what the values should be of leaf nodes at the bottom of the search tree. Section 5.1 introduces new techniques that address the first obstacle in a theoretically sound way and that empirically greatly outperform past approaches. It also introduces a sound way for repeating search as play proceeds down the game tree. Section 5.2 introduces new techniques that address the second obstacle in a way that is theoretically sound and orders of magnitude less computationally expensive than past approaches. Section 5.3 introduces ReBeL, the first algorithm for combining reinforcement learning and search during training in a way that is guaranteed to converge to a Nash equilibrium in two-player zero-sum imperfect-information games (and also perfect-information games). ReBeL is a major step toward unifying the historically separate lines of research on perfect-information and imperfect-information games.

The research techniques introduced in this thesis led to Libratus, the first AI to defeat top human professionals in two-player no-limit Texas hold'em poker, and Pluribus, the first AI to defeat top humans in six-player no-limit Texas hold'em poker, which were long-standing grand challenge problems in the fields of AI and game theory. These agents, along with several other bots that were developed in order to evaluate the techniques described in this thesis, are described in detail in Chapter 6.

Nevertheless, there remains more to be done in order to develop even more scalable, efficient, and general algorithms. Chapter 7 concludes by discussing specific directions for future work in this space, both short-term and long-term.

Chapter 2

Notation and Background

This chapter covers notation and background information that will be used throughout the rest of this thesis.

2.1 Imperfect-Information Extensive-Form Games

Our notation is modified from that of Osborne and Rubinstein [119]. In an imperfect-information extensive-form (i.e., tree-form) game there is a finite set of players, $\mathcal{P}$. There is also a distinct "player" not in $\mathcal{P}$ that represents chance decisions and is denoted c. A **history** (i.e., node) h is defined by all information of the current situation, including private knowledge known to only one player. $A(h)$ denotes the actions available at a node and $P(h)$ is the player whose turn it is to act at that node. If chance acts at the node, then $P(h) = c$. We write $h \sqsubset h'$ if a sequence of actions leads from h to h'. We represent the node that follows h after choosing action a by $h \cdot a$. $\mathcal{H}$ is the set of all nodes. $\mathcal{Z} \subseteq \mathcal{H}$ are terminal nodes for which no actions are available and which award a value to each player. For each player $i \in \mathcal{P}$, there is a payoff function $u_i : \mathcal{Z} \to \mathbb{R}$. The range of payoffs is represented by L. If there are exactly two non-chance players and $u_1(z) + u_2(z) = 0$ for all $z \in \mathcal{Z}$, then the game is two-player zero-sum. If the game consists of only a single information set for each player, then the game is **normal form**.

Imperfect information is represented by **information sets** (infosets) for each player $i \in \mathcal{P}$. For any infoset I_i belonging to player i, all nodes $h, h' \in I_i$ are indistinguishable to i. Moreover, every node $h \in \mathcal{H}$ belongs to exactly one infoset for each player. The player who acts at I_i is denoted $P(I_i)$ and the available action set is denoted $A(I_i)$. The set of player i infosets in which player i acts is denoted $\mathcal{I}_i$ and the set of all infosets belonging to the acting player is denoted $\mathcal{I}$. The range of payoffs achievable from I_i is represented by $L(I_i)$ We may simply write I to represent the infoset I_i where i satisfies $I_i = I_{P(I_i)}$ (i.e., player i is the player who acts at I_i).

Throughout this thesis we assume players have **perfect recall**, which means they cannot forget any information they previously knew. More formally, let h, h' be histories such that $h \sqsubset h'$ and let g, g' be histories such that $g \sqsubset g'$. Perfect recall implies that if g and h do not share an infoset and $g \not\sqsubset h$ and $h \not\sqsubset g$, then h' and g' also do not share an infoset. Due to perfect recall, we can write $I_i \sqsubset I_i'$ if a sequence of actions leads from some history $h \in I_i$ to some history $h' \in I_i'$. We can also write $I_i \cdot a$ to represent the player i infoset $\{h \cdot a | h \in I_i\}$.

A **strategy** (i.e., a **policy**) $\sigma(I)$ is a probability vector over actions for acting player i in infoset I. Since all histories in an infoset belonging to i are indistinguishable, the strategies in each of them must be identical. The probability of a particular action a is denoted by $\sigma(I, a)$ or by $\sigma(h, a)$. We define σ_i to be a strategy for i in every infoset in the game where i acts. The strategy of all players other than i is σ_{-i}. A **strategy profile** σ is a tuple of strategies, one for each player.

We denote **reach** by $\pi^\sigma(h)$. This is the probability with which h is reached if all players play according to σ. Formally, $\pi^\sigma(h) = \Pi_{h' \cdot a' \sqsubseteq h} \sigma(h', a')$. We also use $\pi^\sigma(g, h)$ to represent the probability of reaching h given that history g has already been reached and all players play according to σ. Formally, $\pi^\sigma(g, h) = \Pi_{g \sqsubset h' \cdot a' \sqsubseteq h} \sigma(h', a')$.

The **player reach** $\pi_i^\sigma(h)$ of a history h is the product of the probabilities for all agent i actions leading to h. Formally, $\pi_i^\sigma(h) = \Pi_{h' \cdot a' \sqsubseteq h | P(h') = i} \sigma(h', a')$. Due to perfect recall, any two histories g, h in infoset I_i have the same player reach for player i. Thus, we similarly define the player reach $\pi_i^\sigma(I_i)$ of infoset I_i as $\pi_i^\sigma(I_i) = \Pi_{I_i' \cdot a' \sqsubseteq I_i | P(I_i) = i} \sigma(I_i', a')$.

The **external reach** (also called **opponent reach**) $\pi_{-i}^\sigma(h)$ of a history h is the contribution of chance and all players other than i. Formally, $\pi_{-i}^\sigma(h) = \Pi_{h' \cdot a' \sqsubseteq h | P(h') \neq i} \sigma(h', a')$. We also define the external reach of an infoset as $\pi_{-i}^\sigma(I_i) = \sum_{h \in I_i} \pi_{-i}^\sigma(h)$.

A **best response** for player i to σ_{-i} is a strategy $BR(\sigma_{-i}) = \text{argmax}_{\sigma_i'} u_i(\sigma_i', \sigma_{-i})$.

A **public state** K is defined as a set of histories in which it is common knowledge among all players, based on public observations, that the true state of the game is one of those histories. Formally, for any history $h \in K$, if $h, h' \in I_i$ for some infoset I_i, then $h' \in K$. An **imperfect-information subgame**, which we simply call a **subgame**, $\mathcal{S}$ is a union of public states where if any node A leads to any node B and both A and B are in $\mathcal{S}$, then every node between A and B is also in $\mathcal{S}$. Formally, for any node $h \in \mathcal{S}$, if $h, h' \in I_i$ for some infoset I_i then $h' \in \mathcal{S}$. Moreover, if $h \in \mathcal{S}$ and $h'' \in \mathcal{S}$ and there is a sequence of actions leading from h to h'' (i.e., $h \sqsubset h''$), then for every node h' such that $h \sqsubset h' \sqsubset h''$, $h' \in \mathcal{S}$. If $h \in S$ but no descendant of h is in S, then h is a **leaf node**. Additionally, the infosets containing h are **leaf infosets**. Finally, if $h \in S$ but no ancestor of h is in S, then h is a **root node** and the infosets containing h are **root infosets**.

2.2 Nash Equilibrium

Given a game, what is it we hope to compute? A natural answer is that we wish to compute a strategy that will maximize our expected value. However, a strategy's expected value ultimately depends on the opponents' strategies (or the strategies of the population from which the opponents are drawn). For example, if the opponents always throw Rock in Rock-Paper-Scissors, then it is optimal to always throw Paper, but if the opponents always throw Paper, then it is optimal to always throw Scissors.

If the population of opponent strategies is undefined then we typically wish to compute or approximate a **Nash equilibrium**. A Nash equilibrium σ^* is a strategy profile in which every agent's strategy is a best response: $\forall i,\ u_i(\sigma_i^*, \sigma_{-i}^*) = \max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i}^*)$ [112]. In other words, a Nash equilibrium is a strategy profile in which no player can improve by unilaterally shifting to a different strategy. Nash equilibria have been proven to exist in all finite games, and many infinite games, though finding an equilibrium may be difficult.

The **exploitability** $e(\sigma_i)$ of a strategy σ_i in a two-player zero-sum game is how much worse it does versus a best response compared to a Nash equilibrium strategy. Formally, $e(\sigma_i) = u_i\big(\sigma_i^*, BR(\sigma_i^*)\big) - u_i\big(\sigma_i, BR(\sigma_i)\big)$. In an ϵ-Nash equilibrium, no player has exploitability higher than ϵ. **NashConv** [98] generalizes the notion of exploitability to games with more than two players. The NashConv (or total exploitability) of a strategy profile is $e(\sigma) = \sum_{i \in \mathcal{P}} u_i(BR(\sigma_{-i}), \sigma_{-i}) - u_i(\sigma_i, \sigma_{-i})$. The **average exploitability** (or simply exploitability) of strategy profile σ is $e(\sigma)/|\mathcal{P}|$. If the average exploitability of a strategy profile σ is $e(\sigma) \leq \epsilon$, then σ is a **ϵ-Nash equilibrium**.

Two-player zero-sum games are a special class of games in which Nash equilibria have several useful additional properties. In particular, in two-player zero-sum games any player who chooses to play a Nash equilibrium strategy is guaranteed to not lose in expectation no matter what the opponent does (as long as one side does not have an intrinsic advantage under the game rules, or as long as the players alternate sides). In other words, a Nash equilibrium strategy is unbeatable in two-player zero-sum games. For this reason, to "solve" a two-player zero-sum game means to find an exact Nash equilibrium. For example, the Nash equilibrium strategy for Rock-Paper-Scissors is to randomly pick Rock, Paper, and Scissors with equal probability. Against such a strategy, the best that an opponent can do in expectation is tie.

In the simple case of Rock-Paper-Scissors, playing the Nash equilibrium also guarantees that the player will not *win* in expectation. However, in more complex games even determining how to tie against a Nash equilibrium may be difficult; if the opponent ever chooses suboptimal actions, then playing the Nash equilibrium will indeed result in victory in expectation.

In principle, playing the Nash equilibrium can be combined with opponent exploitation by initially playing the equilibrium strategy and then over time shifting to a strategy that exploits the opponent's observed weaknesses (for example, by switching to always playing Paper against an opponent that always plays Rock) [48]. However, except in certain restricted ways [52], shifting to an exploitative non-equilibrium strategy opens oneself up to exploitation because the opponent could also change strategies at any moment. Additionally, existing techniques for opponent exploitation require too many samples to be competitive with human ability outside of small games.

AI systems have reached superhuman performance in two-player zero-sum games such as checkers [130], chess [34], two-player limit poker [12], Go [140], and, as a result of the research described in this thesis, two-player no-limit poker [21]. In each of these cases, the AI system was generated by attempting to approximate a Nash equilibrium strategy rather than by, for example, trying to detect and exploit weaknesses in the opponent.

While a Nash equilibrium strategy is guaranteed to exist in any finite game, polynomial-time algorithms for finding one are only proven to exist for special classes of games, among which two-player zero-sum games are the most prominent. For two-player non-zero-sum games, no polynomial-time algorithm is known for finding a Nash equilibrium. Computing an equilibrium in such a game is in general PPAD-complete [39, 41]. Finding a Nash equilibrium in zero-sum games with three or more players is at least as hard (because a dummy player can be added to the two-player game to make it a three-player zero-sum game). Approximating a Nash equilibrium is also in general PPAD-complete [124]. In practice, even the best complete algorithm can only address games with a handful of possible strategies per player if there are more than two players [6].

Moreover, even if a Nash equilibrium could be computed efficiently in a game with more

than two players, it is not clear that playing such an equilibrium strategy would be wise. If each player in such a game independently computes and plays a Nash equilibrium strategy, the resulting strategy profile may not be a Nash equilibrium. One example of this is the Lemonade Stand Game [164], illustrated in Figure 2.1, in which each player simultaneously picks a point on a ring and the winner is whoever is farthest from any other player. In every Nash equilibrium with more than two players, players are spaced uniformly along the ring, but there are infinitely many ways this can be accomplished and therefore infinitely many Nash equilibria. If each player independently computes one of those equilibria, the joint strategy profile is unlikely to result in all players being spaced uniformly along the ring. Two-player zero-sum games are a special case where even if the players independently compute and play Nash equilibrium strategies, the resulting strategy profile is still a Nash equilibrium.

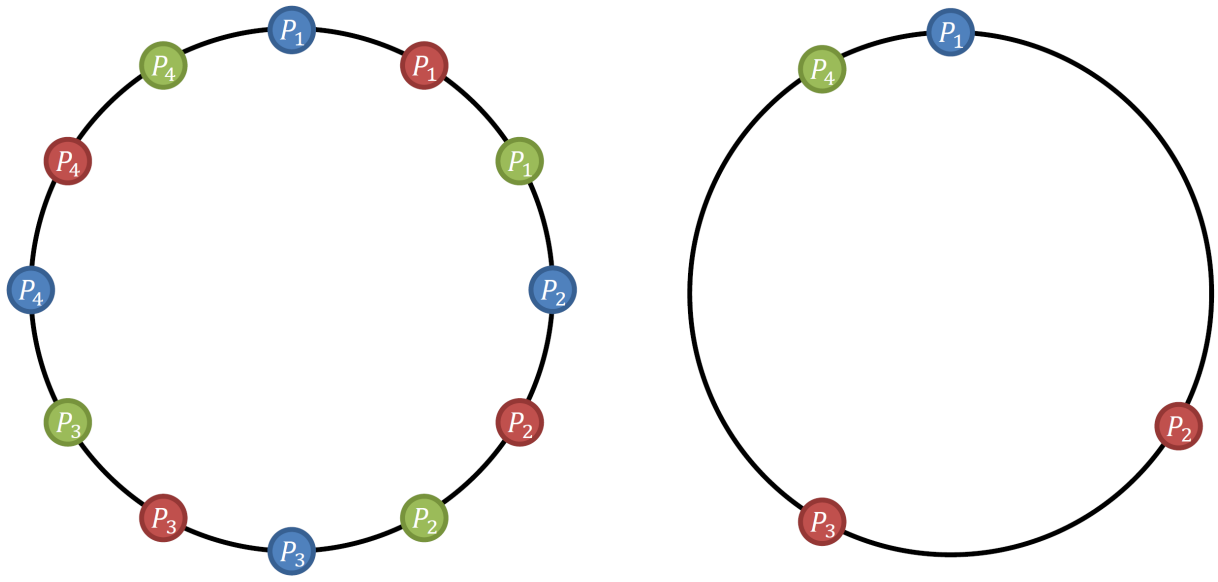


Figure 2.1: **An example of the equilibrium selection problem.** In the Lemonade Stand Game, players simultaneously choose a point on a ring and want to be as far away as possible from any other player. In every Nash equilibrium with more than two players, players are spaced uniformly around the ring. There are infinitely many such Nash equilibria. However, if each player independently chooses one Nash equilibrium to play, their joint strategy profile is unlikely to be a Nash equilibrium. **Left:** An illustration of three different Nash equilibria in this game, distinguished by three different colors. **Right:** Each player independently chooses one Nash equilibrium. Their joint strategy is not a Nash equilibrium.

The algorithms discussed throughout this thesis typically are proven to converge to a Nash equilibrium in two-player zero-sum games, which empirically leads to strong performance in those games. Outside of two-player zero-sum games, the algorithms are typically not guaranteed to converge to a Nash equilibrium (though they may still converge to weaker equilibria, such as coarse correlated equilibrium [62]). Still, we observe that even in multiplayer games they produce strong performance in the real-world domain of no-limit Texas hold'em poker, which suggests they may be useful in a wider variety of settings than just two-player zero-sum games.

2.3 Regret Minimization

Regret minimization is a concept commonly used in online learning but which has also become a key approach for computing equilibria in large imperfect-information games. We will start by considering the single-agent setting, then move to the normal-form two-player zero-sum game setting, and finally in Section 2.3.5 we will describe counterfactual regret minimization, which describes an efficient way of extending regret minimization to extensive-form games.

Consider an agent in a repeated setting in which the agent must choose between $|A|$ actions each episode. Specifically, on each episode t, nature first assigns a reward $v^t(a)$ for each action a (with the range of rewards being bounded by L, that is $\max_{b \in A} v^t(b) - \min_{a \in A} v^t(a) \leq L$). These reward assignments are not observed by the agent. The rewards may change arbitrarily between episodes (and may even be adversarial against the agent). After nature assigns rewards to the actions, the agent chooses a probability distribution σ^t over actions, and receives a reward equal to the expected value over actions: $v^t = \sum_{a \in A} \sigma^t(a) v^t(a)$.

Since nature can choose the rewards in an arbitrary manner, one could equivalently describe nature as choosing the rewards *after* the agent chooses its probability distribution over actions.

Let S be a set of sequences of strategies. **Regret** measures how much better some sequence of strategies $s \in S$ would have done compared to the sequence of strategies $(\sigma^1, \sigma^2, ..., \sigma^T)$ that the agent chose, where T is the number of iterations. For example, the regret for a sequence of strategies s_a consisting of always choosing action a would be

$$R^T(s_a) = \sum_{t=1}^{T} \left(v^t(a) - v^t \right) \tag{2.1}$$

For an algorithm to be no-regret, it must choose strategies on each iteration in a way that guarantees that $R^T(s)$ grows sublinearly for every $s \in S$ (i.e., $R^T(s) \in o(T)$).

Whether or not an algorithm is no-regret ultimately depends on the set of strategy sequences S being considered. An algorithm that has no **external regret** guarantees that regret grows sublinearly for S consisting of sequences of strategies in which the same single action is chosen on each iteration. As discussed in Section 2.3.4, a no-external-regret algorithm can be used to approximate a Nash equilibrium in two-player zero-sum games. For this reason, this thesis focuses on no-external-regret guarantees, and we use "regret" to refer to external regret unless otherwise specified.

Other notions of regret exist as well, and can be used to compute other forms of equilibria such as correlated equilibria [11].

2.3.1 Regret Matching

Regret matching (RM) [65] is a simple no-regret learning algorithm that is widely used for imperfect-information game solving. In RM, the probability of an action on an iteration is proportional to the positive regret on that action. Formally, on each iteration $t + 1$, action $a \in A$ is selected according to probabilities

$$\sigma^{t+1}(a) = \frac{R^t_+(a)}{\sum_{a' \in A} R^t_+(a')} \tag{2.2}$$

where $R^t_+(a) = \max\{0, R^t(a)\}$. If $\sum_{a' \in A} R^t_+(a') = 0$ then any arbitrary strategy may be chosen. Typically in this situation, each action is assigned equal probability, but this is not required and in some cases (such as Deep CFR, discussed in Section 4.1 and SAEF in Section 4.3) other options are used.

Our analysis of RM makes frequent use of a **potential function** of a regret vector. For a vector of regret values $\vec{R}^T$, the potential function is defined to be

$$\Phi(\vec{R}^T) = \sum_{a \in A} \left(R^T_+(a)\right)^2 \tag{2.3}$$

RM guarantees [36] that after T iterations,

$$\Phi(\vec{R}^T) \leq \sum_{a \in A} \left(\left(R^{T-1}_+(a)\right)^2 + \left(r^T(a)\right)^2 \right) \tag{2.4}$$

This means that after T iterations of regret matching are played,

$$\Phi(\vec{R}^T) \leq L^2 |A| T \tag{2.5}$$

and therefore RM guarantees that regret is bounded by

$$R^T(a) \leq L \sqrt{|A|} \sqrt{T} \tag{2.6}$$

where L is the range of payoffs, $|A|$ is the number of actions, and T is the number of iterations [36].

RM is one of many no-regret learning algorithms. and there exists many no-regret learning algorithms with better bounds on regret than RM. For example, Hedge [36] has a convergence bound that is $\mathcal{O}(\ln |A|)$ rather than $\mathcal{O}(\sqrt{|A|})$. Nevertheless, RM and its variants (which include RM+ and more general variants that are introduced in Section 3.1) are by far the most popular no-regret learning algorithms for solving sequential imperfect-information games. There are several reasons for this. First, RM is simple to implement and does not require any parameter tuning. Second, it does not require any expensive operations like exponentiation (which is used in Hedge) or line search (which is used in NormalHedge [38]). Finally, and perhaps most importantly, RM and its variants empirically perform extremely well in large-scale imperfect-information sequential games.

2.3.2 Regret Matching+

Regret Matching+ (RM+) [149, 150] is a simple modification of RM in which regret for every action is floored at zero. That is, RM+ chooses action probabilities based on the following formula

$$\sigma^{t+1}(a) = \frac{Q^t_+(a)}{\sum_{a' \in A} Q^t_+(a')} \tag{2.7}$$

where $Q^t(a) = \max\{0, Q^{t-1}(a) + r^t(a)\}$. If $\sum_{a' \in A} Q^t_+(a') \leq 0$, then actions can be chosen arbitrarily.

RM+ is not known to have a superior convergence bound in theory compared to RM. Nevertheless, it empirically converges faster.

2.3.3 Hedge

In Hedge, a player picks a distribution over actions according to

$$\sigma^{T+1}(a) = \frac{e^{\eta_T R^T(a)}}{\sum_{a' \in A} e^{\eta_T R^T(a')}} \tag{2.8}$$

where η_T is a tuning parameter. There is a substantial literature on how to set η_T for best performance [36, 37]. If a player plays according to Hedge on every iteration t and uses $\eta_t = \sqrt{\frac{2\ln(|A|)}{T}}$ then on iteration T, $R^T \leq L\sqrt{2\ln(|A|)T}$ [36].

2.3.4 Equilibrium convergence of no-regret learning algorithms in games

When both players in a two-player zero-sum game use a no-regret learning algorithm, the average of the strategies played over all iterations converges to a Nash equilibrium. We use $\bar{\sigma}_i$ to represent player i's average strategy over all iterations. Since we will reference the details of the following known result later, we reproduce the proof here.

Theorem 1. *In a two-player zero-sum game, if $\frac{R_i^T}{T} \leq \epsilon_i$ for both players $i \in \mathcal{P}$, then $\bar{\sigma}^T$ is a $\frac{(\epsilon_1+\epsilon_2)}{2}$-equilibrium.*

Proof. We follow the proof approach of Waugh et al. [160]. We use Σ_i to represent the set of strategies for player i. From the definition of regret, we have that

$$\max_{\sigma_i' \in \Sigma_i} \frac{1}{T}\Big(\sum_{t=1}^{T} u_i(\sigma_i', \sigma_{-i}^t) - u_i(\sigma_i^t, \sigma_{-i}^t)\Big) \leq \epsilon_i \tag{2.9}$$

Since σ_i' is the same on every iteration, this becomes

$$\max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \bar{\sigma}_{-i}^T) - \frac{1}{T}\sum_{t=1}^{T} u_i(\sigma_i^t, \sigma_{-i}^t) \leq \epsilon_i \tag{2.10}$$

Since $u_1(\sigma) = -u_2(\sigma)$, if we sum (2.10) for both players

$$\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2^T) + \max_{\sigma_2' \in \Sigma_2} u_2(\bar{\sigma}_1^T, \sigma_2') \leq \epsilon_1 + \epsilon_2 \tag{2.11}$$

$$\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2^T) - \min_{\sigma_2' \in \Sigma_2} u_1(\bar{\sigma}_1^T, \sigma_2') \leq \epsilon_1 + \epsilon_2 \tag{2.12}$$

Since $u_1(\bar{\sigma}_1^T, \bar{\sigma}_2^T) \geq \min_{\sigma_2' \in \Sigma_2} u_1(\bar{\sigma}_1^T, \sigma_2')$ so we have $\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_{-2}^T) - u_1(\bar{\sigma}_1^T, \bar{\sigma}_2^T) \leq \epsilon_1 + \epsilon_2$. By symmetry, this is also true for Player 2. Therefore, $\langle \bar{\sigma}_1^T, \bar{\sigma}_2^T \rangle$ is a $\frac{(\epsilon_1+\epsilon_2)}{2}$-equilibrium. □

Thus, no-regret learning algorithms like RM constitute anytime algorithms for approximating Nash equilibria.

2.3.5 Counterfactual Regret Minimization (CFR)

The no-regret learning algorithms described so far are intended for normal-form games (i.e., matrix games). While they could in theory be used for sequential games, doing so would require representing the sequential game in normal form. In other words, the number of actions would equal the number of pure strategies in the game. This quickly becomes intractable for large sequential games.

Counterfactual regret minimization (CFR) is an algorithm for extensive-form games that independently minimizes regret in each information set [163]. While any regret-minimizing algorithm can be used in the information sets, RM and its variants are the most popular option [65].

Our analysis of CFR makes frequent use of **counterfactual value**. Informally, the counterfactual value of an infoset I where $P(I) = i$ is the expected utility to player i given that I has been reached, weighed by the external reach of I for player i. Formally,

$$v^{\sigma}(I) = \sum_{h \in I} \left(\pi^{\sigma}_{-i}(h) \sum_{z \in \mathcal{Z}} \left(\pi^{\sigma}(h, z) u_i(z) \right) \right) \tag{2.13}$$

and the counterfactual value of an action a is

$$v^{\sigma}(I, a) = \sum_{h \in I} \left(\pi^{\sigma}_{-i}(h) \sum_{z \in \mathcal{Z}} \left(\pi^{\sigma}(h \cdot a, z) u_i(z) \right) \right) \tag{2.14}$$

Let σ^t be the strategy profile used on iteration t. The **instantaneous regret** on iteration t for action a in information set I is

$$r^t(I, a) = v^{\sigma^t}(I, a) - v^{\sigma^t}(I) \tag{2.15}$$

The **counterfactual regret** for action a in I on iteration T is

$$R^T(I, a) = \sum_{t=1}^{T} r^t(I, a) \tag{2.16}$$

Additionally, $R^T_+(I, a) = \max\{R^T(I, a), 0\}$ and $R^T(I) = \max_a\{R^T_+(I, a)\}$. Regret for player i in the entire game is

$$R^T_i = \max_{\sigma'_i \in \Sigma_i} \sum_{t=1}^{T} \left(u_i(\sigma'_i, \sigma^t_{-i}) - u_i(\sigma^t_i, \sigma^t_{-i}) \right) \tag{2.17}$$

where Σ_i is the set of pure strategies for player i.

If player i plays according to RM in information set I on iteration T then, just as in Equation 2.4,

$$\sum_{a \in A(I)} \left(R^T_+(I, a) \right)^2 \leq \sum_{a \in A(I)} \left(\left(R^{T-1}_+(I, a) \right)^2 + \left(r^T(I, a) \right)^2 \right) \tag{2.18}$$

Since counterfactual regret is weighed by external reach, we obtain the following lemma.

Lemma 1. *After T iterations of regret matching are played in an information set I,*

$$\sum_{a \in A(I)} \left(R^T_+(I, a) \right)^2 \leq \left(L(I) \right)^2 |A(I)| \sum_{t=1}^{T} \left(\pi^{\sigma^t}_{-i}(I) \right)^2 \tag{2.19}$$

In turn, this leads to a bound on regret of

$$R^T(I) \leq L(I)\sqrt{|A(I)|}\sqrt{\sum_{t=1}^{T}(\pi_{-i}^{\sigma^t}(I))^2} \tag{2.20}$$

The key result of CFR is that total regret in the full game is bounded by the sum of counterfactual regrets:

$$R_i^T \leq \sum_{I\in\mathcal{I}_i} R^T(I) \leq \sum_{I\in\mathcal{I}_i} L(I)\sqrt{|A(I)|}\sqrt{\sum_{t=1}^{T}(\pi_{-i}^{\sigma^t}(I))^2} \leq \sum_{I\in\mathcal{I}_i} L(I)\sqrt{|A(I)|}\sqrt{T} \tag{2.21}$$

Therefore, by choosing a series of strategies that produce sublinear counterfactual regret in each infoset, one can achieve sublinear total regret. CFR accomplishes this by applying any no-regret learning algorithm, such as RM or RM+, locally at each infoset, operating on counterfactual regret at the infoset. Thus, CFR constitutes an anytime algorithm for finding an ϵ-Nash equilibrium in two-player zero-sum games.

The average strategy $\bar{\sigma}_i^T(I)$ for an infoset I on iteration T is

$$\bar{\sigma}_i^T(I) = \frac{\sum_{t=1}^{T}\left(\pi_i^{\sigma^t}(I)\sigma_i^t(I)\right)}{\sum_{t=1}^{T}\pi_i^{\sigma^t}(I)} \tag{2.22}$$

In practice, faster convergence is achieved by alternating which player updates their regrets on each iteration rather than updating the regrets of both players simultaneously each iteration, though this complicates the theory [33, 45]. We assume the alternating-updates form of CFR is used unless otherwise specified.

While any regret minimization algorithm can be used at infosets to minimize counterfactual regret, typically RM is used in "vanilla" CFR.

2.3.6 Monte Carlo Counterfactual Regret Minimization (MCCFR)

Vanilla CFR requires full traversals of the game tree, which is infeasible in large games. One method to combat this is **Monte Carlo CFR** (MCCFR), in which only a portion of the game tree is traversed on each iteration [96].

In MCCFR, a subset of nodes Q^t in the game tree is traversed at each iteration, where Q^t is sampled from some distribution $\mathcal{Q}$. Sampled regrets $\tilde{r}^t$ are tracked rather than exact regrets. For infosets that are sampled at iteration t, $\tilde{r}^t(I,a)$ is equal to $r^t(I,a)$ divided by the probability of having sampled I; for unsampled infosets $\tilde{r}^t(I,a) = 0$.

There exist a number of MCCFR variants [55, 76, 81], but for this paper we focus specifically on the **external sampling** variant due to its simplicity and strong performance. In external-sampling MCCFR the game tree is traversed for one player at a time, alternating back and forth. We refer to the player who is traversing the game tree on the iteration as the **traverser**. Regrets are updated only for the traverser on an iteration. At infosets where the traverser acts, all actions are explored. At other infosets and chance nodes, only a single action is explored (sampled according to the strategy on that iteration of the player that acts at the infoset).

We begin by reviewing the derivation of convergence bounds for external sampling MCCFR from Lanctot et al. [96].

An MCCFR scheme is completely specified by a set of *blocks* $\mathcal{Q} = \{Q_i\}$ which each comprise a subset of all terminal histories Z. On each iteration MCCFR samples one of these blocks, and only considers terminal histories within that block. Let $q_j > 0$ be the probability of considering block Q_j in an iteration.

Let Z_I be the set of terminal nodes that contain a prefix in I, and let $z[I]$ be that prefix. Define $\pi^\sigma(h \to z)$ as the probability of playing to z given that player p is at node h with both players playing σ.

$$\pi^\sigma(h \to z) = \sum_{z \in Z_I} \frac{\pi^\sigma(z[I])}{\pi^\sigma(I)} \pi^\sigma(z).$$

$\pi^\sigma(I \to z)$ is undefined when $\pi(I) = 0$.

Let $q(z) = \sum_{j:z \in Q_j} q_j$ be the probability that terminal history z is sampled in an iteration of MCCFR. For external sampling MCCFR, $q(z) = \pi^\sigma_{-i}(z)$.

The **sampled value** $\tilde{v}^\sigma_i(I|j)$ when sampling block j is

$$\tilde{v}^\sigma_i(I|j) = \sum_{z \in Q_j \cap Z_I} \frac{1}{q(z)} u_i(z) \pi^\sigma_{-i}(z[I]) \pi^\sigma(z[I] \to z) \tag{2.23}$$

For external sampling, the sampled value reduces to

$$\tilde{v}^\sigma_i(I|j) = \sum_{z \in Q_j \cap Z_I} u_i(z) \pi^\sigma_i(z[I] \to z) \tag{2.24}$$

The sampled value is an unbiased estimator of the true value $v_i(I)$. Therefore the **sampled instantaneous regret** $\tilde{r}^t(I,a) = \tilde{v}^{\sigma^t}_i(I,a) - \tilde{v}^{\sigma^t}_i(I)$ is an unbiased estimator of $r^t(I,a)$.

The *sampled regret* is calculated as $\tilde{R}^T(I,a) = \sum_{t=1}^T \tilde{r}^t(I,a)$.

We first state the general bound shown in [95], Theorem 3.

Lanctot 95 defines $\mathcal{B}_i$ to be a set with one element per distinct action sequence $\vec{a}$ played by player i, containing all infosets that may arise when player i plays $\vec{a}$. M_i is then defined by $\sum_{B \in \mathcal{B}_i} |B|$. Let L be the difference between the maximum and minimum payoffs in the game.

Theorem 2. *(Lanctot 95, Theorem 3) For any $p \in (0,1]$, when using any algorithm in the MCCFR family such that for all $Q \in \mathcal{Q}$ and $B \in \mathcal{B}_i$,*

$$\sum_{I \in B} \left(\sum_{z \in Q \cap Z_I} \frac{\pi^\sigma(z[I] \to z) \pi^\sigma_{-i}(z[I])}{q(z)} \right)^2 \leq \frac{1}{L^2} \tag{2.25}$$

where $L \leq 1$, then with probability at least $1 - p$, total regret is bounded by

$$R^T_i \leq \left(M_i + \frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}} \right) \left(\frac{1}{L} \right) L \sqrt{|A|T} \tag{2.26}$$

For the case of external sampling MCCFR, $q(z) = \pi^\sigma_{-i}(z)$. Lanctot et al. 96, Theorem 9 shows that for external sampling, for which $q(z) = \pi^\sigma_{-i}(z)$, the inequality in (2.25) holds for $L = 1$, and thus the bound implied by (2.26) is

$$\bar{R}_i^T \leq \left(M_i + \frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i|}}{\sqrt{p}} \right) L \frac{\sqrt{|A|}}{\sqrt{T}} \tag{2.27}$$

$$\leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}} \right) L|\mathcal{I}_i| \frac{\sqrt{|A|}}{\sqrt{T}} \qquad \text{because } |\mathcal{B}_i| \leq M_i \leq |\mathcal{I}_i| \tag{2.28}$$

2.3.7 Counterfactual Regret Minimization+ (CFR+)

CFR+ is like CFR but with the following two changes. First, after each iteration any action with negative regret is set to zero regret. Formally, CFR+ chooses its strategy on iteration $T+1$ according to RM+ (discussed in Section 2.3.2). Second, CFR+ uses a weighted average strategy where iteration T is weighted by T rather than using a uniformly weighted average strategy as in CFR. Specifically, the average strategy for CFR+ is

$$\bar{\sigma}_i^T(I) = \frac{\sum_{t=1}^{T} \left(t\pi_i^{\sigma^t}(I)\sigma_i^t(I) \right)}{\sum_{t=1}^{T} t\pi_i^{\sigma^t}(I)} \tag{2.29}$$

The best known convergence bound for CFR+ is higher (that is, worse in exploitability) than CFR by a constant factor of 2 [150]. Despite that, CFR+ typically converges much faster than CFR and in some games even converges faster than $O(\frac{1}{\epsilon})$.

However, in some games CFR+ converges slower than $\frac{1}{T}$. We now provide a two-player zero-sum game with this property. Consider the payoff matrix $\left[\begin{smallmatrix} 1 & 0.9 \\ -0.7 & 1 \end{smallmatrix}\right]$ (where P_1 chooses a row and P_2 simultaneously chooses a column; the chosen entry in the matrix is the payoff for P_1 while P_2 receives the opposite).

2.3.8 Comparison to other equilibrium-finding algorithms

Regret minimization is far from the only method for computing Nash equilibria in two-player zero-sum games.

Small games can be converted to normal form and solved using linear programming. However, the memory requirements of this approach grow exponentially with the size of the game. Sequence-form linear programming can be used instead to find a Nash equilibrium in a matrix of size $|I_1||A_1| + |I_2|$ by $|I_2||A_2| + |I_1|$ with $O(|Z|)$ non-zero entries [30, 85, 86, 123, 157]. Sequence-form linear programming was used to solve Rhode Island hold'em, a small synthetic form of poker, after lossless abstraction was applied to reduce the size of the game to approximately 10^8 nodes [56]. However, sequence-form linear programming does not scale well to larger games. Instead, iterative algorithms that compute a coarse approximation of a Nash equilibrium are preferred.

First-order methods such as the **Excessive Gap Technique** [72, 90, 92, 93] converge to a Nash equilibrium in $O(1/\epsilon)$, which is asymptotically much better than CFR's proven convergence rate. However, the fastest first-order methods in practice converge slower than the fastest CFR variants in large games, require careful tuning of the parameters, do not handle sampling

and approximation error as well as CFR, and are more difficult to implement. For all these reasons, the leading approach for solving large imperfect-information games over the past decade has almost always been a variant of CFR.

2.3.9 Proofs of Theoretical Results

Proof of Lemma 1

Proof. From (2.18) we see that

$$\sum_{a \in A(I)} \left(R_+^T(I,a)\right)^2 \leq \sum_{a \in A(I)} \sum_{t=1}^{T} \left(r^t(I,a)\right)^2$$

From (2.13) and (2.14), we see that $r^t(I,a) \leq \pi_{-i}^{\sigma^t}(I)L(I)$, so

$$\sum_{a \in A(I)} \left(R_+^T(I,a)\right)^2 \leq |A(I)|\left(L(I)\right)^2 \sum_{t=1}^{T} \left(\pi_{-i}^{\sigma^t}(I)\right)^2$$

We know $0 \leq \pi_{-i}^{\sigma^t}(I) \leq 1$. Therefore, $\sum_{t=1}^{T} \left(\pi_{-i}^{\sigma^t}(I)\right)^2 \leq \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I) \leq T\pi_{-i}^{\bar{\sigma}^T}(I)$. Thus, we have

$$\sum_{a \in A(I)} \left(R_+^T(I,a)\right)^2 \leq \pi_{-i}^{\bar{\sigma}^T}(I)\left(L(I)\right)^2 |A(I)|T$$

□

2.4 Benchmark Imperfect-Information Games

There are a number of imperfect-information game benchmarks that are used for evaluating equilibrium-finding techniques. The most prominent game for this setting is poker.

Poker is a family of games that involves hidden information, deception, and bluffing. It has been used for nearly a century as a domain for the expression, development, and testing of game-theoretic techniques related to imperfect-information games. This section describes the variants of poker that are used in experiments throughout this paper.

We also describe Goofspiel, a non-poker card game, and Liar's Dice, a game with many of the elements of poker but using dice rather than a deck of cards.

2.4.1 Leduc Hold'em Poker

Leduc hold'em [143] is a popular benchmark problem for imperfect-information game solving due to its size (large enough to be highly nontrivial but small enough to be solvable) and strategic complexity.

In Leduc hold'em, there is a deck consisting of six cards: two each of Jack, Queen, and King. There are two rounds. In the first round, each player places an ante of 1 chip in the pot and

receives a single private card. A round of betting then takes place with a two-bet maximum, with Player 1 going first. A public shared card is then dealt face up and another round of betting takes place. Again, Player 1 goes first, and there is a two-bet maximum. If one of the players has a pair with the public card, that players wins. Otherwise, the player with the higher card wins.

In standard Leduc hold'em, the bet size in the first round is 2 chips, and 4 chips in the second round. We also conduct experiments in a scaled-up variant, which we call Leduc-5, that has 5 bet sizes to choose from: in the first round a player may bet 0.5, 1, 2, 4, or 8 chips, while in the second round a player may bet 1, 2, 4, 8, or 16 chips.

Leduc hold'em contains 288 information sets. Leduc-5 contains 34,224 information sets.

2.4.2 Limit Texas Hold'em Poker

Limit Texas hold'em (LTH) is a form of poker that was, and to some extent still is, played competitively by humans. Two-player LTH, which we refer to as **heads-up LTH (HULH)** was approximately solved (to within a precision that a human could not distinguish it from a perfect solution within a lifetime of play) in 2015 [12].

In HULH the **blinds** (the amount of money the players must contribute to the pot before play begins) are $2 for Player 1 and $1 for Player 2. On the first betting round, Player 2 acts first. On subsequent betting rounds, Player 1 acts first.

HULH consists of four rounds of betting. On a round of betting, each player can choose to either fold, call, or raise. If a player folds, they are considered to no longer be part of the hand. That is, the player cannot win any money in the pot and takes no further actions. If a player calls, that players places a number of chips in the pot equal to the most that any other player has contributed to the pot. If a player raises, that player adds more chips to the pot than any other player so far. At most three raises are allowed on the first betting round and at most four are allowed on subsequent betting rounds. A round ends when each player still in the hand has acted and has the same amount of money in the pot as every other player still in the hand.

Each raise on the earlier two betting rounds adds an extra $2 to the pot beyond what it would cost to call. On the latter two betting rounds, each raise adds an extra $4 to the pot beyond what it would cost to call.

At the start of the first round, every player receive two private cards from a standard 52-card deck. At the start of the second round, three **community** cards are dealt face up for all players to observe. At the start of the third betting round, an additional community card is dealt face up. At the start of the fourth betting round, a final fifth community card is dealt face up. If at any point only one player remains in the hand, that player collects all the money that all players have contributed to the pot. Otherwise, the player with the best five-card poker hand, constructed from the player's two private cards and the five face-up community cards, wins the pot. In the case of a tie, the pot is split equally among the winning players.

For the next hand—i.e., the next repetition of the game—the positions of the players are reversed.

2.4.3 No-Limit Texas Hold'em Poker

No-limit Texas hold'em (NLTH) has been the most common and popular form of poker for more than a decade. It is used, for example, to determine the winner of the World Series of Poker Main Event. The rules are somewhat similar to LTH but allow for a wider range of raising options. We discuss the difference in the rules, as well as the rules as they apply to the six-player version of the game, in this section.

In two-player NLTH, which is referred to as **heads-up NLTH (HUNL)**, both players start each hand with $20,000. The blinds (the amount of money the players must contribute to the pot before play begins) are $100 for Player 1 and $50 for Player 2. By having each hand start with the same number of chips, we are able to treat each hand as a separate sample when measuring win rate. On the first betting round, Player 2 acts first. On subsequent betting rounds, Player 1 acts first.

In multi-player (i.e., more than two players) NLTH, all players start each hand with $10,000. The blinds are $50 for Player 1 and $100 for Player 2. This is the standard buy-in amount for both live and online play. On the first betting round, Player 3 acts first. On subsequent betting rounds, Player 1 acts first if still in the hand. Play proceeds through each consecutive player still in the hand, with Player 1 following Player 6 if the round has not yet ended.

NLTH consists of four rounds of betting. Unlike LTH, there is no limit to the number of times a player can raise, and players can choose how much to raise. The initial raise on each round must be at least $100. Any subsequent raise on the round must be at least as large as the previous raise (i.e., at least as large as the amount of money beyond a call that the previously-raising player contributed). Raises can be in any whole-dollar amount. No player can raise more than their remaining amount of money.

2.4.4 Flop Texas Hold'em

Flop Texas hold'em (FTH) is identical to Texas hold'em, except the game ends after the second betting round with only three community cards ever revealed. Both limit Texas hold'em and no-limit Texas hold'em can be modified in this way.

FTH is a synthetic game that retains much of the complexity of HUNL and HULH, while being much more manageable in size.

2.4.5 Goofspiel

In Goofspiel, each player has N hidden cards in their hand (1, 2, ..., N), A deck of N cards (1, 2, ..., N), is placed between the two players. In the variant we consider, both players know the order of revealed cards in the center will be 1, 2, ..., N. On each round, the top card of the deck is flipped and is considered the prize card. Each player then simultaneously plays a card from their hand. The player who played the higher-ranked card wins the prize card. If the players played the same rank, then they split the prize's value. The cards that were bid are discarded. At the end of the game, players add up the ranks of their prize cards. A player's payoff is the difference between his total value and the total value of his opponent.

2.4.6 Liar's Dice

Liar's Dice is a two-player zero-sum game in our experiments, though in general it can be played with more than two players. At the beginning of a game each player privately rolls d dice with f faces each. After that a betting stage starts where players take turns trying to predict how many dice of a specific kind there are among all the players, e.g., 4 dice with face 5. A player's bid must either be for more dice than the previous player's bid, or the same number of dice but a higher face. The round ends when a player challenges the previous bid (a call of *liar*). If all players together have at least as many dice of the specified face as was predicted by the last bid, then the player who made the bid wins. Otherwise the player who challenged the bid wins. We use the highest face as a *wild* face, i.e., dice with this face count towards a bid for any face.

Chapter 3

Equilibrium Finding via Counterfactual Regret Minimization

CFR is the leading equilibrium-finding algorithmic framework for imperfect-information games. However, the original CFR algorithm is relatively slow and today is not competitive with more modern variants of CFR or other equilibrium-finding algorithms. This chapter introduces several improvements to CFR that dramatically improve its performance.

Section 3.1 introduces Discounted CFR (DCFR), a new class of CFR algorithms that discounts regrets and leads to much faster empirical convergence while maintaining CFR's bound on regret. We show that one form of DCFR consistently matches or exceeds CFR+, the prior state-of-the-art equilibrium-finding algorithm. This form of DCFR is today the state-of-the-art equilibrium-finding algorithm and is widely used in commercial equilibrium-finding software. CFR+ and most forms of DCFR result in poor performance when combined with Monte Carlo CFR. Linear CFR (LCFR) is another form of DCFR that is robus to variance from sampling (unlike CFR+ and most forms of DCFR). LCFR can therefore be combined with Monte Carlo methods and is now the state-of-the-art equilibrium-finding algorithm for Monte Carlo variants of CFR.

Section 3.2 introduces the first general technique for warm starting CFR from an arbitrary starting strategy. The algorithm is agnostic to the origins of the strategy: it may come from imitating human strategies, from a previous approximate equilibrium, or even from the output of a different equilibrium-finding algorithm. We prove that this warm start technique does not hurt the convergence bound of CFR and prove that under certain assumptions the technique is optimal.

Sections 3.3, 3.4, and 3.5 introduce pruning techniques that allow CFR to avoid traversing the entire game tree. This results in each CFR iteration being conducted much more quickly while retaining the convergence bound of CFR. In particular, best response pruning, discussed in Section 3.5, provably improves the asymptotic complexity of conducting an iteration of CFR and the asymptotic memory usage of CFR to be dependent on the number of actions in the game that are a best response to an equilibrium, rather than simply the number of actions in the game.

Together, these improvements can result in orders of magnitude faster convergence and orders of magnitude reduction in asymptotic memory usage for large imperfect-information games.

3.1 Faster Convergence with Discounted CFR (DCFR)

The development of CFR+, a variant of CFR that uses RM+ at each infoset, was a key breakthrough that in many cases is at least an order of magnitude faster than vanilla CFR [149, 150]. CFR+ was used to essentially solve heads-up limit Texas hold'em poker [12] and was used to approximately solve heads-up no-limit Texas hold'em (HUNL) endgames in *Libratus*, which defeated HUNL top professionals [20, 21]. A blend of CFR and CFR+ was used by *DeepStack* to defeat poker professionals in HUNL [110].

Nevertheless, we describe in this section variants of CFR that significantly outperform CFR+, and in particular one variant, **Discounted CFR (DCFR)**, that matches or exceeds the performance of CFR+ in every game tested. We show that CFR+ does relatively poorly in games where some actions are very costly mistakes (that is, they cause high regret in that iteration) and provide an intuitive example and explanation for this. To address this weakness, we introduce variants of CFR that do not assign uniform weight to each iteration. Instead, earlier iterations are discounted. As we show, this high-level idea can be instantiated in many different ways.

DCFR is today the leading equilibrium-finding algorithm for large imperfect-information games. Another variant that we introduce in this section, **Linear CFR (LCFR)** outperforms DCFR in certain games with wide ranges in payoffs, and also is robust to variance from sampling (unlike CFR+ and DCFR) and can therefore be combined with Monte Carlo methods.

3.1.1 Weighted Averaging Schemes for CFR+

As described in Section 2.3.7, CFR+ traditionally uses "linear" averaging, in which iteration t's contribution to the average strategy is proportional to t. In this subsection we generalize this result and show that any sequence of non-decreasing weights may be used when calculating the average strategy for CFR+. However, the bound on convergence is never lower than that of vanilla CFR (that is, uniformly equal weight on the iterations).

Theorem 3. *Suppose T iterations of RM+ are played in a two-player zero-sum game. Then the weighted average strategy profile, where iteration t is weighed proportional to $w_t > 0$ and $w_i \leq w_j$ for all $i < j$, is a $\frac{w_T}{\sum_{t=1}^{T} w_t} L|\mathcal{I}|\sqrt{|A|}\sqrt{T}$-Nash equilibrium.*

Empirically we find that CFR+ converges faster when iteration t is assigned weight t^2 rather than t when calculating the average strategy. We refer to this as quadratic-weighting CFR+ and use it in the experiments in this section.

3.1.2 Regret Discounting for CFR and Its Variants

Although CFR+ assigns non-uniform weight to iterations when computing the average strategy, in all past variants of CFR, including CFR+, each iteration's contribution to the *regrets* is assigned equal weight. In this subsection we discuss discounting iterations in CFR when determining regrets—in particular, assigning less weight to earlier iterations. This is very different from, and orthogonal to, the idea of discounting iterations when computing the average strategy, described in the previous subsection.

To motivate discounting, consider the simple case of an agent deciding between three actions. The payoffs for the actions are 0, 1, and -1,000,000, respectively. From Equation 2.2 we see that CFR and CFR+ assign equal probability to each action on the first iteration. This results in regrets of 333,333, 333,334, and 0, respectively, when using CFR+. If we continue to run CFR+ or CFR, the next iteration will choose the first and second action with roughly 50% probability each, and the regrets will be updated to be roughly 333,332.5 and 333,334.5, respectively. It will take 471,407 iterations for the agent to choose the second action—that is, the best action—with 100% probability. Discounting the first iteration over time would dramatically speed convergence in this case. While this might seem like a contrived example, many games include highly suboptimal actions. In this simple example the bad action was chosen on the first iteration, but in general bad actions may be chosen throughout a run, and discounting may be useful far beyond the first few iterations.

Discounting prior iterations has received relatively little attention in the equilibrium-finding community. "Optimistic" regret minimizing variants exist that assign a higher weight to recent iterations, but this extra weight is temporary and typically only applies to a short window of recent iterations; for example, counting the most recent iterate twice [148]. CFR+ discounts prior iterations' contribution to the *average strategy*, but not the *regrets*. Discounting prior iterations has also been used in CFR for situations where the game structure changes, for example due to interleaved abstraction and equilibrium finding [14, 16]. There has also been some work on applying discounting to perfect-information game solving in Monte Carlo Tree Search [66].

Outside of equilibrium finding, prior research has analyzed the theory for discounted regret minimization [36]. That work investigates applying RM (and other regret minimizers) to a sequence of iterations in which iteration t has weight w_t (assuming $w_t \leq 1$ and the final iteration has weight 1). For RM, it proves that if $\sum_{t=1}^{\infty} w_t = \infty$ then weighted average regret, defined as $R_i^{w,T} = \max_{a \in A} \frac{\sum_{t=1}^{T}(w_t r^t(a))}{\sum_{t=1}^{T} w^t}$ is bounded by

$$R_i^{w,T} \leq \frac{L\sqrt{|A|}\sqrt{\sum_{t=1}^{T} w_t^2}}{\sum_{t=1}^{T} w_t} \tag{3.1}$$

Prior work has shown that, in two-player zero-sum games, if weighted average regret is ϵ, then the weighted average strategy, defined as $\sigma_i^{w,T}(I) = \frac{\sum_{t \in T}\left(w_t \pi_i^{\sigma^t}(I)\sigma_i^t(I)\right)}{\sum_{t \in T}(w_t \pi_i^{\sigma^t}(I))}$ for infoset I, is a 2ϵ-Nash equilibrium [14].

While there are a limitless number of discounting schemes that converge in theory, not all of them perform well in practice. We introduce a number of variants that perform particularly well also in practice. The first algorithm, which we refer to as linear CFR (LCFR), is identical to CFR, except on iteration t the updates to the regrets and average strategies are given weight t. That is, the iterates are weighed linearly. (Equivalently, one could multiply the accumulated regret by $\frac{t}{t+1}$ on each iteration. We do this in our experiments to reduce the risk of numerical instability.) This means that after T iterations of LCFR, the first iteration only has a weight of $\frac{2}{T^2+T}$ on the regrets rather than a weight of $\frac{1}{T}$, which would be the case in CFR and CFR+. In the motivating example introduced at the beginning of this subsection, LCFR chooses the second action with 100% probability after only 970 iterations while CFR+ requires 471,407 iterations.

Furthermore, from (3.1), the theoretical bound on the convergence of regret is only greater than vanilla CFR by a factor of $\frac{2}{\sqrt{3}}$.

Since the changes from CFR that lead to LCFR and CFR+ do not conflict, it is natural to attempt to combine them into a single algorithm that weighs each iteration t proportional to t and also has a floor on regret at zero like CFR+. However, we empirically observe that this algorithm, which we refer to as **LCFR+**, actually leads to performance that is *worse* than LCFR and CFR+ in the games we tested, even though its theoretical bound on convergence is the same as for LCFR.

Nevertheless, we find that using a less-aggressive discounting scheme leads to consistently strong performance. We can consider a family of algorithms called Discounted CFR with parameters α β, and γ (DCFR$_{\alpha,\beta,\gamma}$), defined by multiplying accumulated positive regrets by $\frac{t^\alpha}{t^\alpha+1}$, negative regrets by $\frac{t^\beta}{t^\beta+1}$, and contributions to the average strategy by $(\frac{t}{t+1})^\gamma$ on each iteration t. In this case, LCFR is equivalent to DCFR$_{1,1,1}$, because multiplying iteration t's regret and contribution to the average strategy by $\frac{t'}{t'+1}$ on every iteration $t \leq t' < T$ is equivalent to weighing iteration t by $\frac{t}{T}$. CFR+ (where iteration t's contribution to the average strategy is proportional to t^2) is equivalent to DCFR$_{\infty,-\infty,2}$.

In preliminary experiments we found the optimal choice of α, β, and γ varied depending on the specific game. However, we found that setting $\alpha = 3/2$, $\beta = 0$, and $\gamma = 2$ led to performance that was consistently stronger than CFR+. Thus, when we refer to DCFR with no parameters listed, we assume this set of parameters are used.

Theorem 4 shows that DCFR has a convergence bound that differs from CFR+ only by a factor of at most 3.

Theorem 4. *Assume that T iterations of DCFR are conducted in a two-player zero-sum game. Then the weighted average strategy profile is a $6L|\mathcal{I}|(|\sqrt{|A|} + \frac{1}{\sqrt{T}})/\sqrt{T}$-Nash equilibrium.*

One of the drawbacks of setting $\beta \leq 0$ is that suboptimal actions (that is, actions that have an expected value lower than some other action in every equilibrium) no longer have regrets that approach $-\infty$ over time. Instead, for $\beta = 0$ they will approach some constant value and for $\beta < 0$ they will approach 0. This makes the algorithm less compatible with improvements that prune negative-regret actions [15, 19], discussed in Section 3.3 and Section 3.5. Such pruning algorithms can lead to more than an order of magnitude reduction in computational and space requirements for some games. Setting $\beta > 0$ better facilitates this pruning. For this reason in our experiments we also show results for $\beta = 0.5$.

3.1.3 Experimental setup

We conduct experiments on subgames of HUNL poker. Although the HUNL game tree is too large to traverse completely without sampling, state-of-the-art agents for HUNL solve subgames of the full game in real time during play [20, 21, 27, 110] using a small number of the available bet sizes. For example, *Libratus* solved in real time the remainder of HUNL starting on the third betting round. We conduct our HUNL experiments on four subgames generated by *Libratus* [1]. The subgames were selected prior to testing. We use a simplified betting structure that has bet

[1] https://github.com/CMU-EM/LibratusEndgames

sizes of 0.5x and 1x the size of the pot, as well as an all-in bet (betting all remaining chips) for the first bet of each round. For subsequent bets in a round, we consider 1x the pot and all-in.

Subgame 1 begins at the start of the third betting round and continues to the end of the game. There are $500 in the pot at the start of the round. This is the most common situation to be in upon reaching the third betting round, and is also the hardest for AIs to solve because the remaining game tree is the largest. Since there is only $500 in the pot but up to $20,000 could be lost, this subgames contains a number of high-penalty mistake actions. Subgame 2 begins at the start of the third betting round and has $4,780 in the pot at the start of the round. Subgame 3 begins at the start of the fourth (and final) betting round with $500 in the pot, which is a common situation. Subgame 4 begins at the start of the fourth betting round with $3,750 in the pot. Exploitability is measured in terms of milli big blinds per game (mbb/g), a standard measurement in the field, which represents the number of big blinds (P_1's original contribution to the pot) lost per hand of poker multiplied by 1,000.

In addition to HUNL subgames, we also consider a version of the game of Goofspiel (limited to just five cards per player). In the variant we consider, both players know the order of revealed cards in the center will be 1, 2, 3, 4, 5. A player's payoff is the difference between his total value and the total value of his opponent.

3.1.4 Experiments on Regret Discounting and Weighted Averaging

Our experiments are run for 32,768 iterations for HUNL subgames and 8,192 iterations for Goofspiel. Since all the algorithms tested only converge to an ϵ-equilibrium rather than calculating an exact equilibrium, it is up to the user to decide when a solution is sufficiently converged to terminate a run. In practice, this is usually after 100 - 1,000 iterations [21, 110]. For example, an exploitability of 1 mbb/g is considered sufficiently converged so as to be essentially solved [12]. Thus, the performance of the presented algorithms between 100 and 1,000 iterations is arguably more important than the performance beyond 10,000 iterations. Nevertheless, we show performance over a long time horizon to display the long-term behavior of the algorithms. All our experiments use the alternating-updates form of CFR. We measure the average exploitability of the two players.

In all of the experiments in this section, we use quadratic weighting for CFR+.

Our experiments show that LCFR can dramatically improve performance over CFR+ over reasonable time horizons in certain games. However, asymptotically, LCFR appears to do worse in practice than CFR+. LCFR does particularly well in subgame 1 and 3, which (due to the small size of the pot relative to the amount of money each player can bet) have more severe mistake actions compared to subgames 2 and 4. It also does poorly in Goofspiel, which also likely does not have severely suboptimal actions. This suggests that LCFR is particularly well suited for games with the potential for large mistakes.

Our experiments also show that DCFR$_{\frac{3}{2},0,2}$ matches or outperforms CFR+ across the board. The improvement is usually a factor of 2 or 3. In Goofspiel, DCFR$_{\frac{3}{2},0,2}$ results in essentially identical performance as CFR+.

DCFR$_{\frac{3}{2},-\infty,2}$, which sets negative regrets to zero rather than multiplying them by $\frac{1}{2}$ each iteration, generally also leads to equally strong performance, but in rare cases (such as in Figure 3.2), can produce a spike in exploitability that takes many iterations to recover from. Thus,

we generally recommend using DCFR$_{\frac{3}{2},0,2}$ over DCFR$_{\frac{3}{2},-\infty,2}$.

DCFR$_{\frac{3}{2},\frac{1}{2},2}$ multiplies negative regrets by $\frac{\sqrt{t}}{\sqrt{t}+1}$ on iteration t, which allows suboptimal actions to decrease in regret to $-\infty$ and thereby facilitates algorithms that temporarily prune negative-regret sequences. In the HUNL subgames, DCFR$_{\frac{3}{2},\frac{1}{2},2}$ performed very similarly to DCFR$_{\frac{3}{2},0,2}$. However, in Goofspiel it does noticeably worse. This suggests that DCFR$_{\frac{3}{2},\frac{1}{2},2}$ may be preferable to DCFR$_{\frac{3}{2},0,2}$ in games with large mistakes when a pruning algorithm may be used, but that DCFR$_{\frac{3}{2},0,2}$ should be used otherwise.

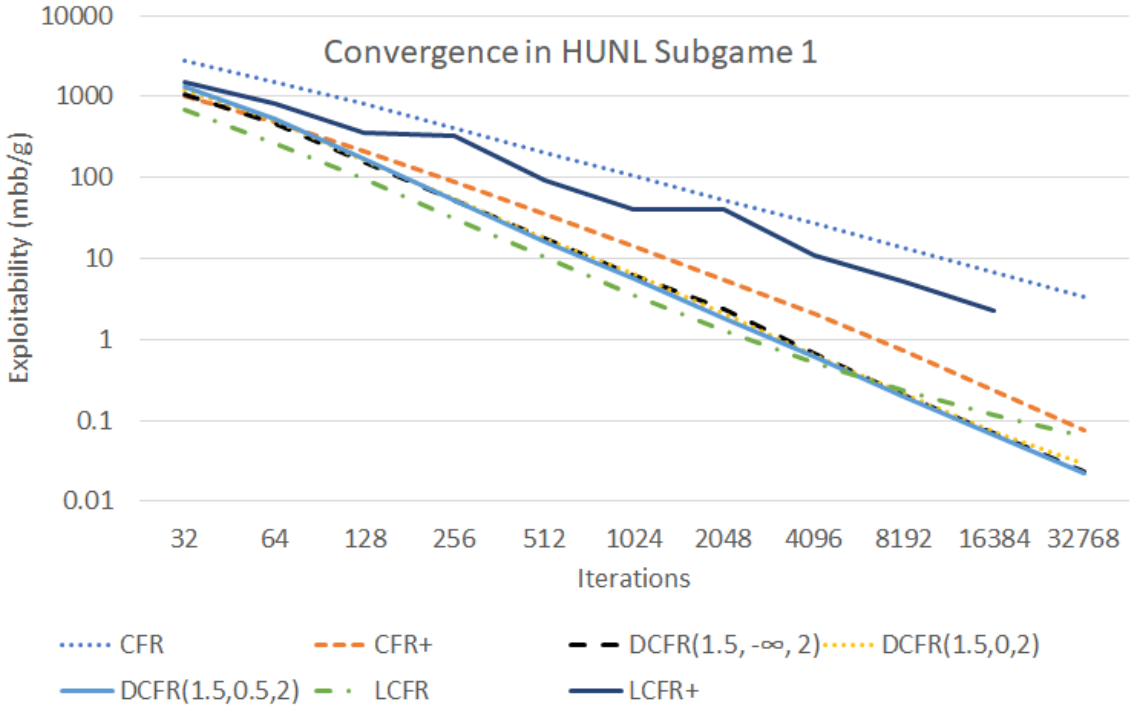


Figure 3.1: Convergence in HUNL Subgame1.

3.1.5 Discounted Monte Carlo CFR

Monte Carlo CFR (MCCFR) is a variant of CFR in which certain player actions or chance outcomes are sampled [55, 96]. MCCFR combined with abstraction has produced state-of-the-art HUNL poker AIs [21]. It is also particularly useful in games that do not have a special structure that can be exploited to implement a fast vector-based implementation of CFR [79, 96]. There are many forms of MCCFR with different sampling schemes. The most popular is external-sampling MCCFR, in which opponent and chance actions are sampled according to their probabilities, but all actions belonging to the player updating his regret are traversed. Other MCCFR variants exist that achieve superior performance [76], but external-sampling MCCFR is simple and widely used, which makes it useful as a benchmark for our experiments.

Although CFR+ provides a massive improvement over CFR in the unsampled case, the changes present in CFR+ (a floor on regret at zero and linear averaging), do not lead to superior performance when applied to MCCFR [30]. In contrast, in this subsection we show

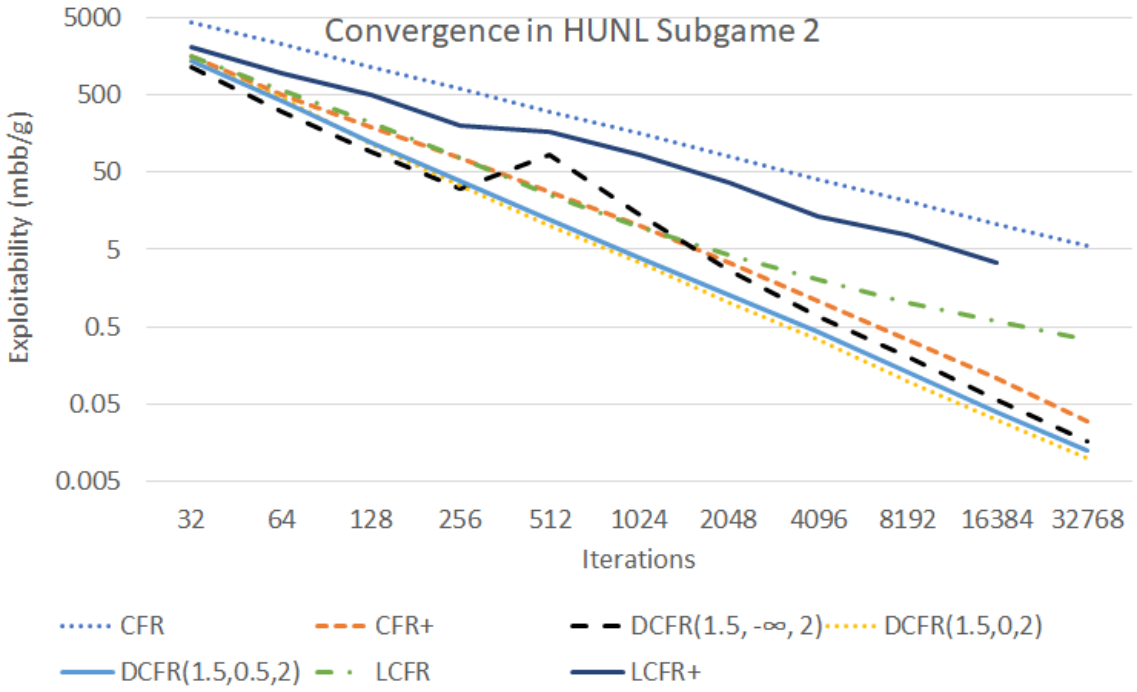


Figure 3.2: Convergence in HUNL Subgame2.

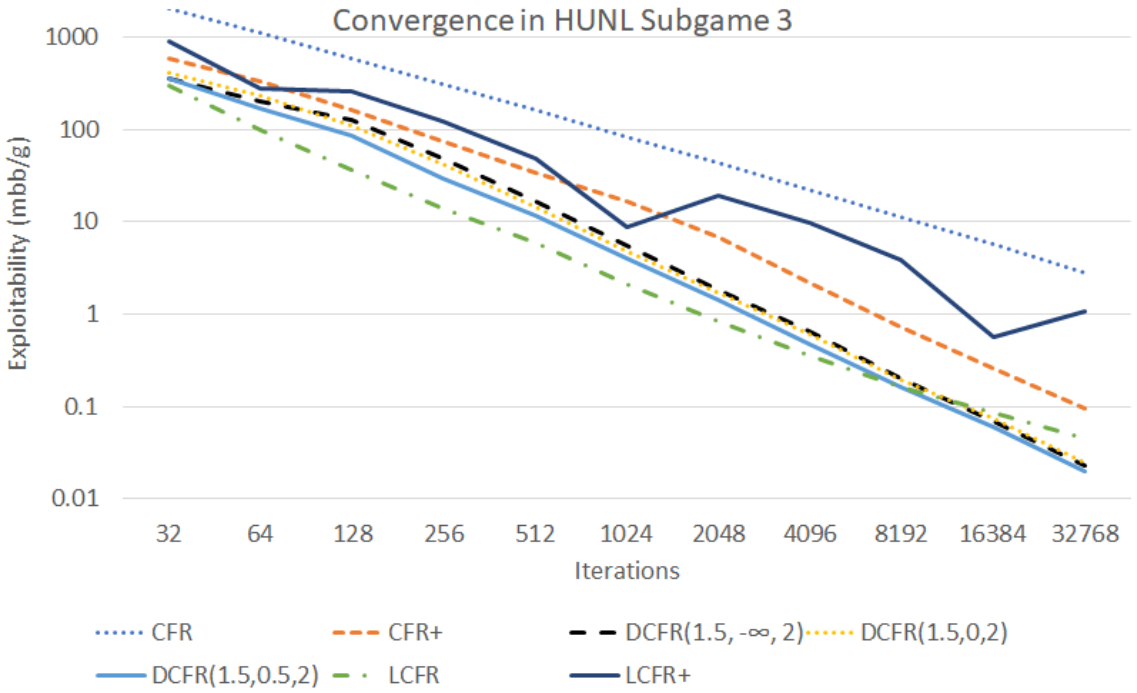


Figure 3.3: Convergence in HUNL Subgame 3.

that the changes present in LCFR do lead to superior performance when applied to MCCFR. Specifically, we divide the MCCFR run into periods of 10^7 nodes touched. Nodes touched is an

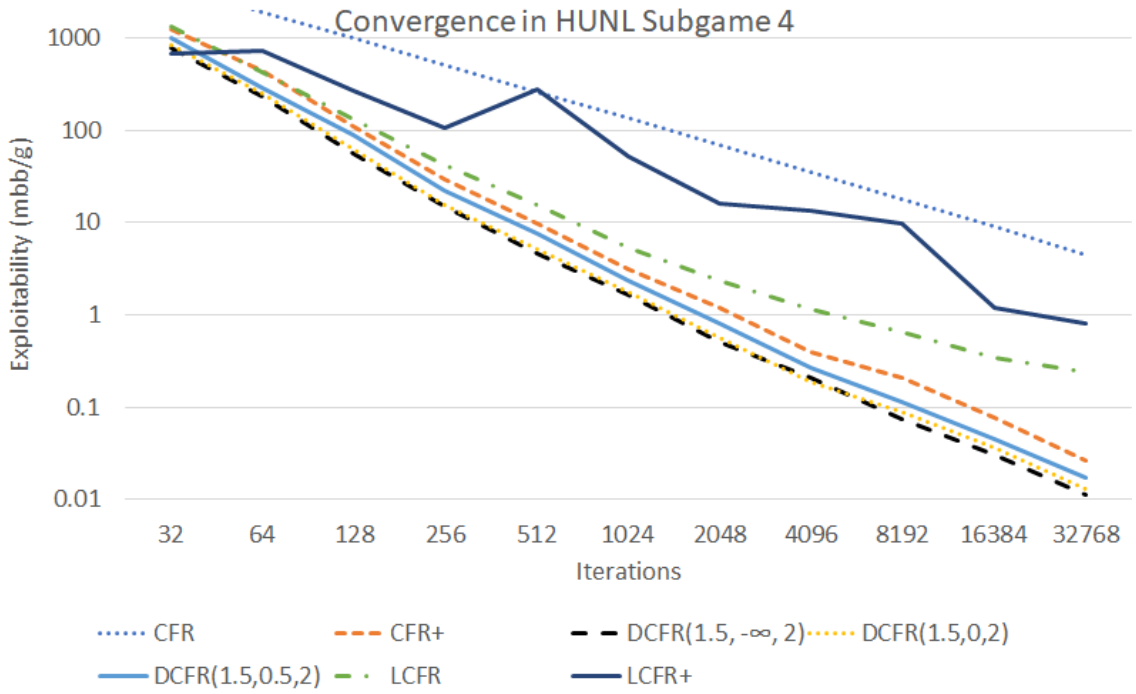


Figure 3.4: Convergence in HUNL Subgame 4.

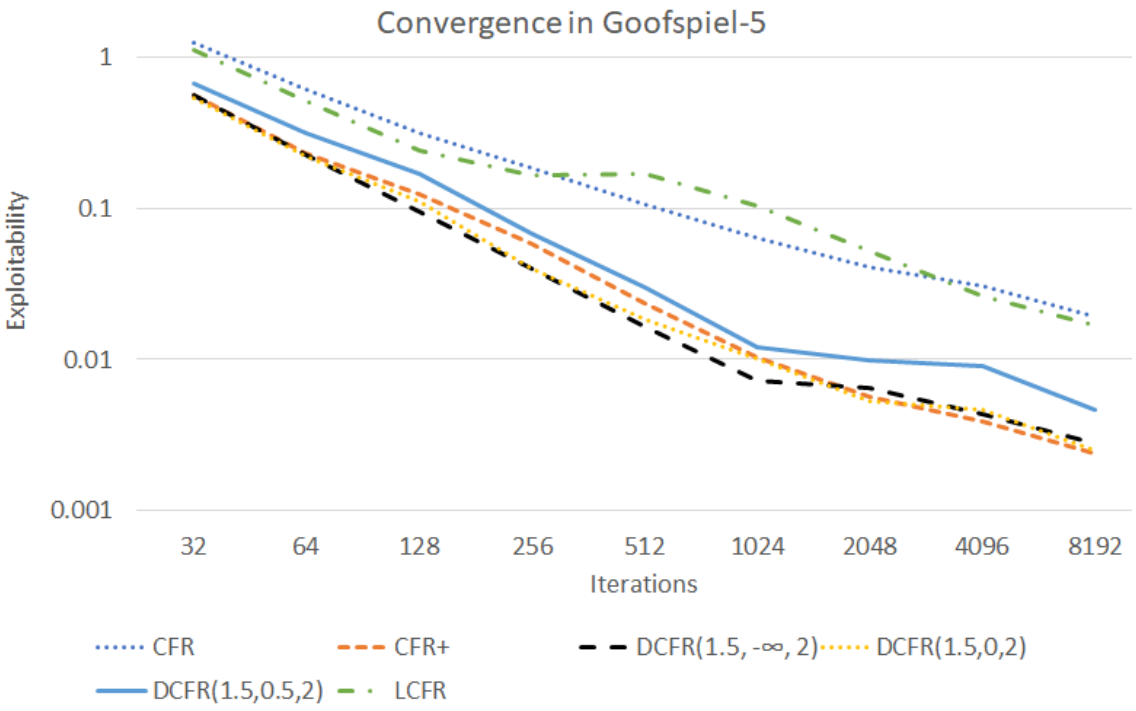


Figure 3.5: Convergence in 5-card Goofspiel variant.

implementation-independent and hardware-independent proxy for time that counts the number

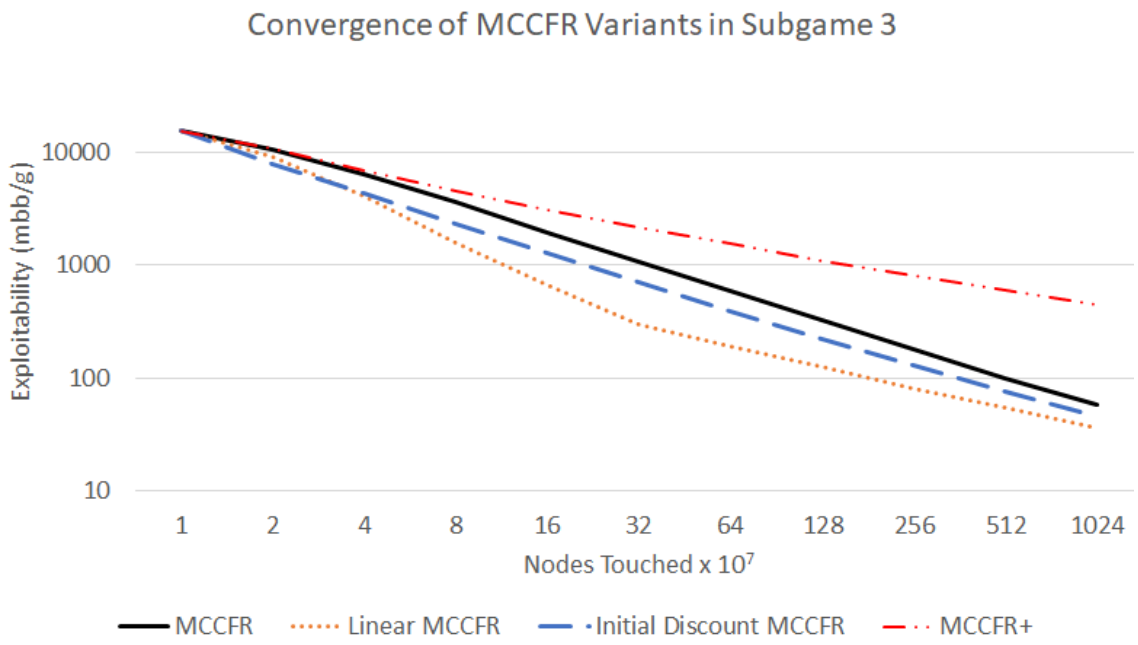


Figure 3.6: Convergence of MCCFR in HUNL Subgame 3.

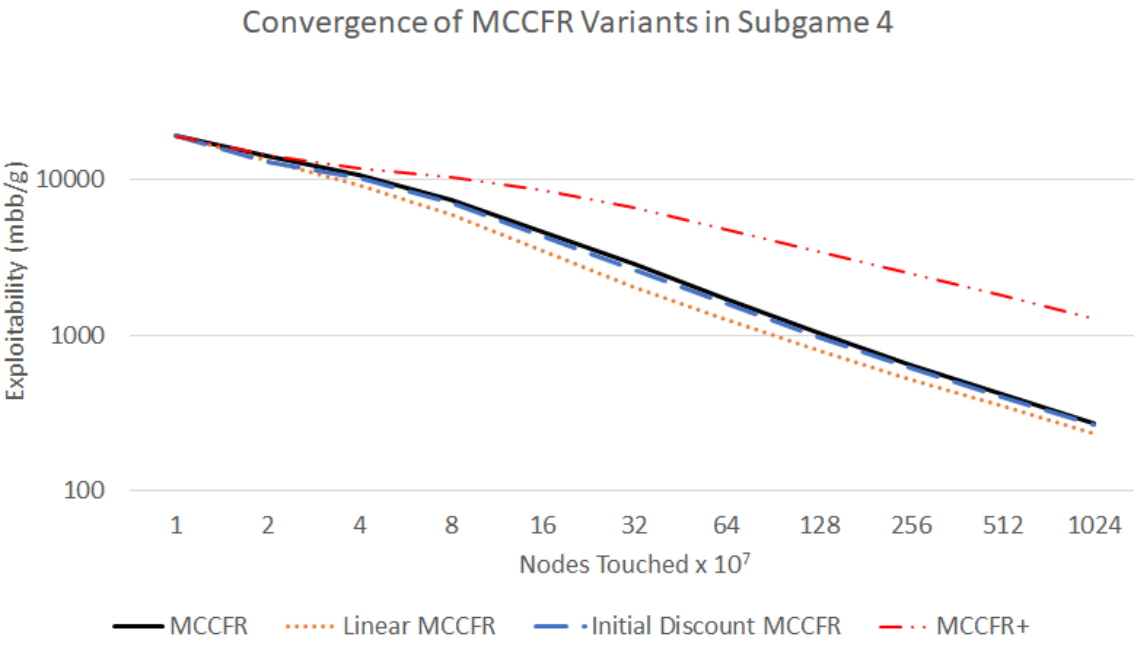


Figure 3.7: Convergence of MCCFR in HUNL Subgame 4.

of nodes traversed (including terminal nodes). After each period n ends, we multiply all accumulated regrets and contributions to the average strategies by $\frac{n}{n+1}$. Figure 3.6 and Figure 3.7 demonstrate that this leads to superior performance in HUNL compared to vanilla MCCFR. The improvement is particularly noticeable in subgame 3, which features the largest mistake actions.

We also show performance if one simply multiplies the accumulated regrets and contributions to the average strategy by $\frac{1}{10}$ after the first period ends, and thereafter runs vanilla MCCFR (the "Initial Discount MCCFR" variant). The displayed results are the average of 100 different runs.

3.1.6 Conclusions

We introduced variants of CFR that discount prior iterations, leading to stronger performance than the prior state-of-the-art CFR+, particularly in settings that involve large mistakes. In particular, the DCFR$_{\frac{3}{2},0,2}$ variant matched or outperformed CFR+ in all settings, and linear MCCFR leads to clearly better performance over vanilla MCCFR.

While DCFR clearly outperforms CFR+, linear CFR does even better than DCFR in settings with very large mistake actions. Unfortunately, naïvely combining the two into LCFR+ results in poor performance. Developing a single algorithm that achieves matches or exceeds both LCFR and DCFR remains an interesting and important direction for future work.

3.1.7 Proofs of Theoretical Results

Proof of Theorem 3

Consider the weighted sequence of iterates $\sigma'^1, ..., \sigma'^T$ in which σ'^t is identical to σ^t, but weighed by w_t. The regret of action a in infoset I on iteration t of this new sequence is $R'^t(I, a)$.

From Lemma 3 we know that $R^t(I, a) \leq L\sqrt{|A|}\sqrt{T}$ for player i for action a in infoset I. Since $w_{a,t}$ is a non-decreasing sequence, so we can apply Lemma 2 using weight w_t for iteration t with $B = L\sqrt{|A|}\sqrt{T}$ and $C = 0$. From Lemma 2, this means that $R'^t(I, a) \leq w_T L\sqrt{|A|}\sqrt{T}$. Applying (3.1), we get weighted regret is at most $w_T L|\mathcal{I}_i|\sqrt{|A|}\sqrt{T}$ for player i. Thus, weighted average regret is at most $\frac{w_T L|\mathcal{I}_i|\sqrt{|A|}\sqrt{T}}{\sum_{t=1}^{T} w_t}$. Since $|\mathcal{I}_1| + |\mathcal{I}_2| = |\mathcal{I}|$, so the weighted average strategies form a $\frac{w_T L|\mathcal{I}|\sqrt{|A|}\sqrt{T}}{\sum_{t=1}^{T} w_t}$-Nash equilibrium.

Proof Theorem 4

Proof. Since the lowest amount of instantaneous regret on any iteration is $-L$ and DCFR multiplies negative regrets by $\frac{1}{2}$ each iteration, so regret for any action at any point is greater than $-2L$.

Consider the weighted sequence of iterates $\sigma'^1, ..., \sigma'^T$ in which σ'^t is identical to σ^t, but weighed by $w_{a,t} = \Pi_{i=t}^{T-1} \frac{i^2}{(i+1)^2} = \frac{6t^2}{T(T+1)(2T+1)}$ rather than $w_t = \Pi_{i=t}^{T-1} \frac{i^{3/2}}{i^{3/2}+1}$. The regret of action a in infoset I on iteration t of this new sequence is $R'^t(I, a)$.

From Lemma 5 we know that $R^t(I, a) \leq 2L\sqrt{|A|}\sqrt{T}$ for player i for action a in infoset I. Since $w_{a,t}$ is an increasing sequence, so we can apply Lemma 2 using weight $w_{a,t}$ for iteration t with $B = 2L\sqrt{|A|}\sqrt{T}$ and $C = -2L$. From Lemma 2, this means that $R'^t(I, a) \leq \frac{6T^2(2L\sqrt{|A|}\sqrt{T}+2L)}{\left(T(T+1)(2T+1)\right)} \leq 6L(|\sqrt{|A|} + \frac{1}{\sqrt{T}})/\sqrt{T}$. Applying (3.1), we get weighted regret is at most $6L|\mathcal{I}_i|(|\sqrt{|A|} + \frac{1}{\sqrt{T}})/\sqrt{T}$. Since the weights sum to one, this is also weighted average regret.

Since $|\mathcal{I}_1|+|\mathcal{I}_2| = |\mathcal{I}|$, so the weighted average strategies form a $6L(|\mathcal{I}|(\sqrt{|A|}+\frac{1}{\sqrt{T}})/\sqrt{T}$-Nash equilibrium. □

Lemma 2. *Call a sequence $x_1, ..., x_T$ of bounded real values BC-plausible if $B > 0$, $C \leq 0$, $\sum_{t=1}^{i} x_t \geq C$ for all i, and $\sum_{t=1}^{T} x_t \leq B$. For any BC-plausible sequence and any sequence of non-decreasing weights $w_t \geq 0$, $\sum_{t=1}^{T}(w_t x_t) \leq w_T(B-C)$.*

Proof. The lemma closely resembles Lemma 3 from [150] and the proof shares some elements.

We construct a BC-plausible sequence $x_1^*, ..., x_T^*$ that maximizes the weighted sum. That is, $\sum_{t=1}^{T} w_t x_t' = \max_{x_1', ..., x_T'} \sum_{t=1}^{T} w_t x_t'$. We show that $x_1^* = C$, $x_t^* = 0$ for $1 < t < T$, and $x_T^* = (B-C)$.

Consider x_T^*. Clearly in order to maximize the weighted sum, $x_T^* = B - \sum_{t=1}^{T-1}(w_t x_t^*)$. Next, consider x_t^* for $t < T$ and assume $x_{t'}^* = C - \sum_{t=1}^{t'}(w_t x_t^*)$ for $t < t' < T$ and assume $x_T^* = B - \sum_{t=1}^{T-1}(w_t x_t^*)$. Since $w_t \leq w_T$ and $w_t \leq w_{t'}$, so $\sum_{i=t}^{T}(w_i x_i^*)$ would be maximized if $x_t^* = C - \sum_{i=1}^{t-1}(w_i x_i^*)$. By induction, this means $x_1^* = C$, $x_t^* = 0$ for $1 < t < T$, and $x_T^* = B - C$. In this case $\sum_{t=1}^{T}(w_t x_t^*) \leq w_T(B-C) + w_1 C \leq w_T(B-C)$. Since x^* is a maximizing sequence, so for any sequence x we have that $\sum_{t=1}^{T}(w_t x_t) \leq w_T(B-C)$. □

Lemma 3. *Given a sequence of strategies $\sigma^1, ..., \sigma^T$, each defining a probability distribution over a set of actions A, consider any definition for $Q^t(a)$ satisfying the following conditions:*

1. $Q^0(a) = 0$
2. $Q^t(a) = Q^{t-1}(a) + r^t(a)$ *if* $Q^{t-1}(a) + r^t(a) > 0$
3. $0 \geq Q^t(a) \geq Q^{t-1}(a) + r^t(a)$ *if* $Q^{t-1}(a) + r^t(a) \leq 0$

The regret-like value $Q^t(a)$ is then an upper bound on the regret $R^t(a)$ and $Q^t(a) - Q^{t-1}(a) \geq r^t(a) = R^t(a) - R^{t-1}(a)$.

Proof. The lemma and proof closely resemble Lemma 1 in [150]. For any $t \geq 1$ we have $Q^{t+1}(a) - Q^t(a) \geq Q^t(a) + r^{t+1}(a) - Q^t(a) = R^{t+1}(a) - R^t(a)$. Since $Q^0(a) = 0$ and $R^0(a) = 0$, so $Q^t(a) \geq R^t(a)$. □

Lemma 4. *Given a set of actions A and any sequence of rewards v^t such that $|v^t(a) - v^t(b)| \leq L$ for all t and all $a, b \in A$, after playing a sequence of strategies determined by regret matching but using the regret-like value $Q^t(a)$ in place of $R^t(a)$, $Q^T(a) \leq L\sqrt{|A|T}$ for all $a \in A$.*

Proof. The proof is identical to that of Lemma 2 in [150]. □

Lemma 5. *Assume that player i conducts T iterations of DCFR. Then weighted regret for the player is at most $L|\mathcal{I}_i|\sqrt{|A|}\sqrt{T}$ and weighted average regret for the player is at most $2L|\mathcal{I}_i|\sqrt{|A|}/\sqrt{T}$.*

Proof. The weight of iteration $t < T$ is $w_t = \Pi_{i=t}^{T-1} \frac{i^{3/2}}{i^{3/2}+1}$ and $w_T = 1$. Thus, $w_t \leq 1$ for all t and therefore $\sum_{t=1}^{T} w_t^2 \leq T$.

Additionally, $w_t \geq \Pi_{i=t}^{T-1} \frac{i}{i+1} = \frac{t}{T}$ for $t < T$ and $w_T = 1$. Thus, $\sum_{t=1}^{T} w_t \geq T(T+1)/2 > T^2/2$.

Applying (3.1) and Lemma 4, we see that $Q_i^{w,T}(I,a) \leq \frac{L\sqrt{|A|}\sqrt{\sum_{t=1}^{T} w_t^2}}{\sum_{t=1}^{T} w_t} \leq \frac{2L\sqrt{|A|}\sqrt{T}}{T^2}$. From (3.1) we see that $Q_i^{w,T} \leq \frac{2L|\mathcal{I}_i|\sqrt{|A|}\sqrt{T}}{T^2}$. □

Correctness of DCFR(3/2, 1/2, 2)

Theorem 5. *Assume that T iterations of DCFR$_{\frac{3}{2},\frac{1}{2},2}$ are conducted in a two-player zero-sum game. Then the weighted average strategy profile is a $9L|\mathcal{I}||\sqrt{|A|}/\sqrt{T}$-Nash equilibrium.*

Proof. From Lemma 6, we know that regret in DCFR$_{\frac{3}{2},\frac{1}{2},2}$ for any infoset I and action a cannot be below $-L\sqrt{T}$.

Consider the weighted sequence of iterates $\sigma'^1, ..., \sigma'^T$ in which σ'^t is identical to σ^t, but weighed by $w_{a,t} = \Pi_{i=t}^{T-1} \frac{i^2}{(i+1)^2} = \frac{6t^2}{T(T+1)(2T+1)}$ rather than $w_t = \Pi_{i=t}^{T-1} \frac{i^{3/2}}{i^{3/2}+1}$. The regret of action a in infoset I on iteration t of this new sequence is $R'^t(I,a)$.

From Lemma 5 we know that $R^t(I,a) \leq 2L\sqrt{|A|}\sqrt{T}$ for player i for action a in infoset I. Since $w_{a,t}$ is an increasing sequence, so we can apply Lemma 2 using weight $w_{a,t}$ for iteration t with $B = 2L\sqrt{|A|}\sqrt{T}$ and $C = -L\sqrt{|A|}\sqrt{T}$. From Lemma 2, this means that $R'^t(I,a) \leq \frac{6T^2(3L\sqrt{|A|}\sqrt{T})}{\left(T(T+1)(2T+1)\right)} \leq 9L(|\sqrt{|A|})/\sqrt{T}$. Applying (3.1), we get weighted regret is at most $9L|\mathcal{I}_i|(|\sqrt{|A|})/\sqrt{T}$. Since the weights sum to one, this is also weighted average regret. Since $|\mathcal{I}_1| + |\mathcal{I}_2| = |\mathcal{I}|$, so the weighted average strategies form a $9L(|\mathcal{I}|(\sqrt{|A|})/\sqrt{T}$-Nash equilibrium. □

Lemma 6. *Suppose after each of T iterations of CFR, regret is multiplied by $\frac{\sqrt{t}}{\sqrt{t}+1}$ on iteration t. Then $R^T(I,a) \geq -L\sqrt{T}$ for any infoset I and action a.*

Proof. We prove this inductively. On the first iteration, the lowest regret could be after multiplying by $\frac{\sqrt{1}}{\sqrt{1}+1}$ is $-\frac{L}{2}$. Now assume that after T iterations of CFR in which regret is multiplied by $\frac{\sqrt{t}}{\sqrt{t}+1}$ on each iteration, $R^T(I,a) \geq -L\sqrt{T}$ for infoset I action a. After conducting an additional iteration of CFR and multiplying by $\frac{\sqrt{T+1}}{\sqrt{T+1}+1}$, $R^{T+1}(I,a) \leq -L(\sqrt{T}+1)\frac{\sqrt{T+1}}{\sqrt{T+1}+1}$. Since $\sqrt{T}+1 \leq \sqrt{T+1}+1$, so $-\frac{\sqrt{T}+1}{\sqrt{T+1}+1}L\sqrt{T+1} = -L(\sqrt{T}+1)\frac{\sqrt{T+1}}{\sqrt{T+1}+1} \geq -L\sqrt{T+1}$. Thus, $R^{T+1}(I,a) \geq -L\sqrt{T+1}$. □

3.2 Strategy-Based Warm Starting of CFR

One of the main constraints in solving large games is the time taken to arrive at a solution. For example, essentially solving Limit Texas Hold'em required running CFR on 4,800 cores for 68 days [150]. Even though Limit Texas Hold'em is a popular human game with many domain experts, and even though several near-Nash equilibrium strategies had previously been computed for the game [79, 80], there was no known way to leverage that prior strategic knowledge to speed up CFR. In this section we introduce such a method, enabling CFR to be warm started in a theoretically sound way from arbitrary strategies.

Aside from simply speeding up convergence to a Nash equilibrium, warm starting has important synergies with techniques discussed in later sections. For example, warm starting magnifies the benefits of the pruning techniques discussed in Section 3.3 and Section 3.5, in which some parts of the game tree need not be traversed during an iteration of CFR. This results in faster iterations and therefore faster convergence to a Nash equilibrium. The frequency of pruning opportunities increases as equilibrium finding progresses. This may result in later iterations being completed multiple orders of magnitude faster than early iterations. Our warm starting algorithm can "skip" these early expensive iterations that might otherwise account for the bulk of the time spent on equilibrium finding.

Warm starting can also be used to improve the performance of ReBeL, discussed in Section 5.3. Without warm starting, ReBeL has to solve every subgame from scratch starting from a uniform random strategy. This means its value network must be accurate even when players play uniform random strategies and similarly unrealistic strategies. However, if ReBeL trains a policy network and warm starts CFR in subgames using this policy network then it can focus its value network on more important situations and also solve subgames faster.

3.2.1 Further Details on CFR

In Section 2.3.5, Equation 2.19 showed that when using CFR, counterfactual regret of an infoset is bounded by

$$\sum_{a \in A(I)} \big(R_+^T(I,a)\big)^2 \leq \big(L(I)\big)^2 |A(I)| \big(\sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)\big)^2$$

Since $\sum_{t=1}^{T} (\pi_{-i}^{\sigma^t}(I))^2 \leq \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I) = T\pi_{-i}^{\bar{\sigma}^T}(I)$ we can also use the following looser bound that only depends on $\bar{\sigma}^T$ rather than each σ^t for each iteration t. This will be useful for the theory of our warm start technique.

$$\sum_{a \in A(I)} \big(R_+^T(I,a)\big)^2 \leq \big(L(I)\big)^2 |A(I)| T\pi_{-i}^{\bar{\sigma}^T}(I) \tag{3.2}$$

In turn, this leads to a bound on regret of

$$R^T(I) \leq L(I)\sqrt{|A(I)|}\sqrt{T}\sqrt{\pi_{-i}^{\bar{\sigma}^T}(I)} \tag{3.3}$$

3.2.2 Warm-Starting Algorithm

In this section we explain the theory of how to warm start CFR and prove the method's correctness. By warm starting, we mean we wish to effectively "skip" the first T iterations of CFR (defined more precisely later in this section). When discussing intuition, we use normal-form games due to their simplicity. Normal-form games are a special case of games in which each player only has one information set. They can be represented as a matrix of payoffs where Player 1 picks a row and Player 2 simultaneously picks a column.

The key to warm starting CFR is to correctly initialize the regrets. To demonstrate the necessity of this, we first consider an ineffective approach in which we set only the starting strategy,

but not the regrets. Consider the two-player zero-sum normal-form game defined by the payoff matrix $\left[\begin{smallmatrix}1 & 0\\ 0 & 2\end{smallmatrix}\right]$ with payoffs shown for Player 1 (the row player). The Nash equilibrium for this game requires Player 1 to play $\langle \frac{2}{3}, \frac{1}{3}\rangle$ and Player 2 to play $\langle \frac{2}{3}, \frac{1}{3}\rangle$. Suppose we wish to warm start regret matching with the strategy profile σ^* in which both players play $\langle 0.67, 0.33\rangle$ (which is very close to the Nash equilibrium). A naïve way to do this would be to set the strategy on the first iteration to $\langle 0.67, 0.33\rangle$ for both players, rather than the default of $\langle 0.5, 0.5\rangle$. This would result in regret of $\langle 0.0023, -0.0067\rangle$ for Player 1 and $\langle -0.0023, 0.0067\rangle$ for Player 2. Since regret matching picks actions in proportion to positive regret (see Equation 2.2), on the second iteration Player 1 would play $\langle 1, 0\rangle$ and Player 2 would play $\langle 0, 1\rangle$, resulting in regret of $\langle 0.0023, 1.9933\rangle$ for Player 1. That is a huge amount of regret, and makes this warm start no better than starting from scratch. Intuitively, this naïve approach is comparable to warm starting gradient descent by setting the initial point close to the optimum, but not reducing the step size. The result is that we overshoot the optimal strategy significantly. In order to add some "inertia" to the starting strategy so that CFR does not overshoot, we need a method for setting the regrets as well in CFR.

Fortunately, it is possible to efficiently calculate how far a strategy profile is from the optimum (that is, from a Nash equilibrium). This knowledge can be leveraged to initialize the regrets appropriately. To provide intuition for this warm starting method, we consider warm starting CFR to T iterations in a normal-form game based on an arbitrary strategy σ. Later, we discuss how to determine T based on σ.

First, the average strategy profile is set to $\bar{\sigma}^T = \sigma$. We now consider the regrets. Regret for action a after T iterations of CFR would normally be $R_i^T(a) = \sum_{t=1}^{T} \left(u_i(a, \sigma_{-i}^t) - u_i(\sigma^t) \right)$ (see Equation 2.16). Since $\sum_{t=1}^{T} u_i(a, \sigma_{-i}^t)$ is the value of having played action a on every iteration, it is the same as $T u_i(a, \bar{\sigma}_{-i}^T)$. When warm starting, we can calculate this value because we set $\bar{\sigma}^T = \sigma$. However, we cannot calculate $\sum_{t=1}^{T} u_i(\sigma^t)$ because we did not define individual strategies played on each iteration. Fortunately, it turns out we can substitute another value we refer to as $T v_i'^{\bar{\sigma}^T}$, chosen from a range of acceptable options. To see this, we first observe that the value of $\sum_{t=1}^{T} u_i(\sigma^t)$ is not relevant to the proof that two no-regret learning algorithms converge to a Nash equilibrium (see Theorem 1 presented in Section 2.3.4). Specifically, in Equation 2.11 in the proof, we see that $\sum_{t=1}^{T} u_i(\sigma^t)$ cancels out. Thus, if we choose $v_i'^{\bar{\sigma}^T}$ such that $v_1'^{\bar{\sigma}^T} + v_2'^{\bar{\sigma}^T} \leq 0$, Theorem 1 still holds. This is our first constraint.

There is an additional constraint on our warm start technique. We must ensure that no information set violates the bound on regret guaranteed in Equation 3.2. If regret exceeds this bound, then convergence to a Nash equilibrium may be slower than CFR guarantees. Thus, our second constraint is that when warm starting to T iterations, the initialized regret in every information set must satisfy Equation 3.2. If these conditions hold and CFR is played after the warm start, then the bound on regret will be the same as if we had played T iterations from scratch instead of warm starting. When using our warm start method in extensive-form games, we do not directly choose $v_i'^{\bar{\sigma}^T}$ but instead choose a value $u'^{\bar{\sigma}^T}(I)$ for every information set (and we will soon see that these choices determine $v_i'^{\bar{\sigma}^T}$).

We now proceed to formally presenting our warm-start method and proving its effectiveness. Theorem 6 shows that we can warm start based on an arbitrary strategy σ by replacing $\sum_{t=1}^{T} v^{\sigma^t}(I)$ for each I with some value $T v'^{\sigma}(I)$ (where $v'^{\sigma}(I)$ satisfies the constraints mentioned above). Then, Corollary 1 shows that this method of warm starting is lossless: if T iterations of

CFR were played and we then warm start using $\bar{\sigma}^T$, we can warm start to T iterations.

We now define some terms that will be used in the theorem. When warm starting, a **substitute information set value** $u'^{\sigma}(I)$ is chosen for every information set I (we will soon describe how). Define $v'^{\sigma}(I) = \pi^{\sigma}_{-P(I)}(I)u'^{\sigma}(I)$ and define $v'^{\sigma}_i(h)$ for $h \in I$ as $\pi^{\sigma}_{-i}(h)u'^{\sigma}(I)$. Define $v'^{\sigma}_i(z)$ for $z \in Z$ as $\pi^{\sigma}_{-i}u_i(z)$.

As explained earlier in this section, in normal-form games $\sum_{t=1}^{T} u_i(a, \sigma^t_{-i}) = Tu_i(a, \bar{\sigma}^T_{-i})$. This is still true in extensive-form games for information sets where a leads to a terminal payoff. However, it is not necessarily true when a leads to another information set, because then the value of action a depends on how the player plays in the next information set. Following this intuition, we will define substitute counterfactual value for an action. First, define $\mathit{Succ}^{\sigma}_i(h)$ as the set consisting of histories h' that are the earliest reachable histories from h such that $P(h') = i$ or $h' \in Z$. By "earliest reachable" we mean $h \sqsubseteq h'$ and there is no h'' in $\mathit{Succ}^{\sigma}(h)$ such that $h'' \sqsubset h'$. Then the **substitute counterfactual value** of action a, where $i = P(I)$, is

$$v'^{\sigma}(I, a) = \sum_{h \in I} \Big(\sum_{h' \in \mathit{Succ}^{\sigma}_i(h \cdot a)} v'^{\sigma}_i(h') \Big) \tag{3.4}$$

and **substitute value** for player i is defined as

$$v'^{\sigma}_i = \sum_{h' \in \mathit{Succ}^{\sigma}_i(\emptyset)} v'^{\sigma}_i(h') \tag{3.5}$$

We define **substitute regret** as

$$R'^{T}(I, a) = T\big(v'^{\sigma}(I, a) - v'^{\sigma}(I)\big)$$

and

$$R'^{T,T'}(I, a) = R'^{T}(I, a) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}}(I, a) - v^{\sigma^{t'}}(I)\big)$$

Also, $R'^{T,T'}(I) = \max_{a \in A(I)} R'^{T,T'}(I, a)$. We also define the **combined strategy profile**

$$\sigma'^{T,T'} = \frac{T\sigma + T'\bar{\sigma}^{T'}}{T + T'}$$

Using these definitions, we wish to choose $u'^{\sigma}(I)$ such that

$$\sum_{a \in A(I)} \big(v'^{\sigma}(I, a) - v'^{\sigma}(I)\big)^2_+ \leq \frac{\pi^{\sigma}_{-i}(I)\big(L(I)\big)^2|A(I)|}{T} \tag{3.6}$$

We now proceed to our main result on warm starting.

Theorem 6. *Let σ be an arbitrary strategy profile for a two-player zero-sum game. Choose any T and choose $u'^{\sigma}(I)$ in every information set I such that $v'^{\sigma}_1 + v'^{\sigma}_2 \leq 0$ and Equation 3.6 is satisfied for every information set I. If we play T' iterations according to CFR, where on iteration T^*, $\forall I\ \forall a$ we use substitute regret $R'^{T,T^*}(I, a)$, then $\sigma'^{T,T'}$ forms a $(\epsilon_1 + \epsilon_2)$-equilibrium where $\epsilon_i = \frac{\sum_{I \in \mathcal{I}_i} \sqrt{\pi^{\sigma'^{T,T'}}_{-i}(I)} L(I) \sqrt{|A(I)|}}{\sqrt{T + T'}}$.*

Theorem 6 allows us to choose from a range of valid values for T and $u'^{\sigma}(I)$. Although it may seem optimal to choose the values that result in the largest T allowed, this is typically not the case in practice. This is because in practice CFR converges significantly faster than the theoretical bound. In the next two sections we cover how to choose $u'^{\sigma}(I)$ and T within the theoretically sound range so as to converge even faster in practice.

The following corollary shows that warm starting using Equation 3.6 is lossless: if we play CFR from scratch for T iterations and then warm start using $\bar{\sigma}^T$ by setting $u'^{\bar{\sigma}^T}(I)$ to even the lowest value allowed by Equation 3.6, we can warm start to T.

Corollary 1. *Assume T iterations of CFR were played and let $\sigma = \bar{\sigma}^T$ be the average strategy profile. If we choose $u'^{\sigma}(I)$ for every information set I such that $\sum_{a \in A(I)} \left(v'^{\sigma}(I,a) - v'^{\sigma}(I)\right)_+^2 = \frac{\pi^{\sigma}_{-i}(I)\left(L(I)\right)^2 |A(I)|}{T}$, and then play T' additional iterations of CFR where on iteration T^*, $\forall I\ \forall a$ we use $R_i'^{T,T^*}(I,a)$, then the average strategy profile over the $T+T'$ iterations forms a $(\epsilon_1+\epsilon_2)$-equilibrium where $\epsilon_i = \frac{\sum_{I \in \mathcal{I}_i} \sqrt{\pi^{\sigma'^{T,T'}}_{-i}(I)} L(I) \sqrt{|A(I)|}}{\sqrt{T+T'}}$.*

3.2.3 Choosing the Number of Warm-Start Iterations

In this section we explain how to determine the number of iterations T to warm start to, given only a strategy profile σ. We give a method for determining a theoretically acceptable range for T. We then present a heuristic for choosing T within that range that delivers strong practical performance.

In order to apply Theorem 1, we must ensure $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$. Thus, a theoretically acceptable upper bound for T would satisfy $v_1'^{\sigma} + v_2'^{\sigma} = 0$ when $u'^{\sigma}(I)$ in every information set I is set as low as possible while still satisfying (3.6).

In practice, setting T to this theoretical upper bound would perform very poorly because CFR tends to converge much faster than its theoretical bound. Fortunately, CFR also tends to converge at a fairly consistent rate within a game. Rather than choose a T that is as large as the theory allows, we can instead choose T based on how CFR performs over a short run in the particular game we are warm starting.

Specifically, we generate a function $f(T)$ that maps an iteration T to an estimate of how close $\bar{\sigma}^T$ would be to a Nash equilibrium after T iterations of CFR starting from scratch. This function can be generated by fitting a curve to the first few iterations of CFR in a game. $f(T)$ defines another function, $g(\sigma)$, which estimates how many iterations of CFR it would take to reach a strategy profile as close to a Nash equilibrium as σ. Thus, in practice, given a strategy profile σ we warm start to $T = g(\sigma)$ iterations. In those experiments that required guessing an appropriate T (namely Figures 3.9 and 3.10) we based $g(\sigma)$ on a short extra run (10 iterations of CFR) starting from scratch. The experiments show that this simple method is sufficient to obtain near-perfect performance.

3.2.4 Choosing Substitute Counterfactual Values

Theorem 6 allows for a range of possible values for $u'^{\sigma}(I)$. In this section we discuss how to choose a particular value for $u'^{\sigma}(I)$, assuming we wish to warm start to T iterations.

From (3.4), we see that $v'^{\sigma}(I,a)$ depends on the choice of $u'^{\sigma}(I')$ for information sets I' that follow I. Therefore, we set $u'^{\sigma}(I)$ in a bottom-up manner, setting it for information sets at the bottom of the game tree first. This method resembles a best-response calculation. When calculating a best response for a player, we fix the opponent's strategy and traverse the game tree in a depth-first manner until a terminal node is reached. This payoff is then passed up the game tree. When all actions in an information set have been explored, we pass up the value of the highest-utility action.

Using a best response would likely violate the constraint $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$. Therefore, we compute the following response instead. After every action in information set I has been explored, we set $u'^{\sigma}(I)$ so that (3.6) is satisfied. We then pass $v'^{\sigma}(I)$ up the game tree.

From (3.6) we see there are a range of possible options for $u'^{\sigma}(I)$. In general, lower regret (that is, playing closer to a best response) is preferable, so long as $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$ still holds. In this section we choose an information set-independent parameter $0 \leq \lambda_i \leq 1$ for each player and set $u'^{\sigma}(I)$ such that

$$\sum_{a \in A(I)} \big(v'^{\sigma}(I) - v'^{\sigma}(I,a)\big)_+^2 = \frac{\lambda_i \pi_{-i}^{\sigma}(I)\big(L(I)\big)^2 |A(I)|}{T}$$

Finding λ_i such that $v_1'^{\sigma} + v_2'^{\sigma} = 0$ is difficult. Fortunately, performance is not very sensitive to the choice of λ_i . Therefore, when we warm start, we do a binary search for λ_i so that $v_1'^{\sigma} + v_2'^{\sigma}$ is close to zero (and not positive).

Using λ_i is one valid method for choosing $u'^{\sigma}(I)$ from the range of options that (3.6) allows. However, there may be heuristics that perform even better in practice. In particular, $\pi_{-i}^{\sigma}\big(L(I)\big)^2$ in (3.6) acts as a bound on $\big(r^t(I,a)\big)^2$. If a better bound, or estimation, for $\big(r^t(I,a)\big)^2$ exists, then substituting that in (3.6) may lead to even better performance.

3.2.5 Experiments

We now present experimental results for our warm-starting algorithm. We begin by demonstrating an interesting consequence of Corollary 1. It turns out that in two-player zero-sum games, we need not store regrets at all. Instead, we can keep track of only the average strategy played. On every iteration, we can "warm start" using the average strategy to directly determine the probabilities for the next iteration. We tested this algorithm on random 100x100 normal-form games, where the entries of the payoff matrix are chosen uniformly at random from $[-1, 1]$. On every iteration $T > 0$, we set $v_1'^{\bar{\sigma}^T} = v_2'^{\bar{\sigma}^T}$ such that

$$\frac{|L_1|^2|A_1|}{\sum_{a_1}\big(u_1(a_1, \bar{\sigma}_2^T) - v_1'^{\bar{\sigma}^T}\big)_+^2} = \frac{|L_2|^2|A_2|}{\sum_{a_2}\big(u_2(a_2, \bar{\sigma}_1^T) - v_2'^{\bar{\sigma}^T}\big)_+^2}$$

Figure 3.8 shows that warm starting every iteration in this way results in performance that is virtually identical to CFR.

The remainder of our experiments are conducted on Limit Flop Texas Hold'em (FTH) (described in Section 2.4.4).

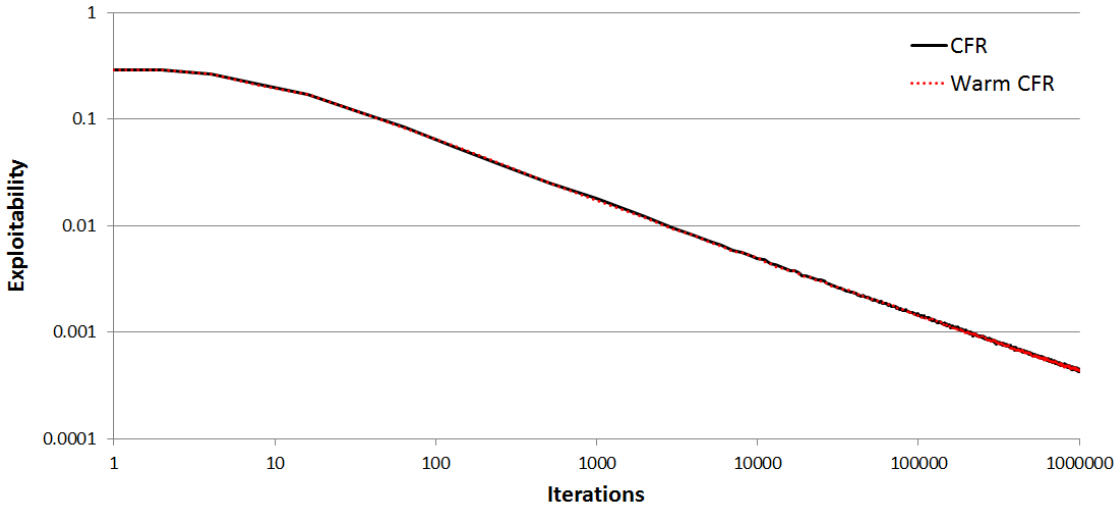


Figure 3.8: Comparison of CFR vs warm starting every iteration. The results shown are the average over 64 different 100x100 normal-form games.

The second experiment compares our warm starting to CFR in FTH. We run CFR for some number of iterations before resetting the regrets according to our warm start algorithm, and then continuing CFR. We compare this to just running CFR without resetting. When resetting, we determine the number of iterations to warm start to based on an estimated function of the convergence rate of CFR in FTH, which is determined by the first 10 iterations of CFR. Our projection method estimated that after T iterations of CFR, $\bar{\sigma}^T$ is a $\frac{10.82}{T}$-equilibrium. Thus, when warm starting based on a strategy profile with exploitability x, we warm start to $T = \frac{10.82}{x}$. Figure 3.9 shows performance when warm starting at 100, 500, and 2500 iterations. These are three separate runs, where we warm start once on each run. We compare them to a run of CFR with no warm starting. Based on the average strategies when warm starting occurred, the runs were warm started to 97, 490, and 2310 iterations, respectively. The figure shows there is almost no performance difference between warm starting and not warm starting.[2]

The third experiment demonstrates one of the main benefits of warm starting: being able to use a small coarse abstraction and/or quick-but-rough equilibrium-finding technique first, and starting CFR from that solution, thereby obtaining convergence faster. In all of our experiments, we leverage a number of implementation tricks that allow us to complete a full iteration of CFR in FTH in about three core minutes [79]. This is about four orders of magnitude faster than vanilla CFR. Nevertheless, there are ways to obtain good strategies even faster. To do so, we use two approaches. The first is a variant of CFR called External-Sampling Monte Carlo CFR (MCCFR) [96], in which chance nodes and opponent actions are sampled, resulting in much faster (though less accurate) iterations. The second is abstraction, in which several similar information sets are bucketed together into a single information set (where “similar" is defined by some heuristic). This constrains the final strategy, potentially leading to worse long-term perfor-

[2]Although performance between the runs is very similar, it is not identical, and in general there may be differences in the convergence rate of CFR due to seemingly inconsequential differences that may change to which equilibrium CFR converges, or from which direction it converges.

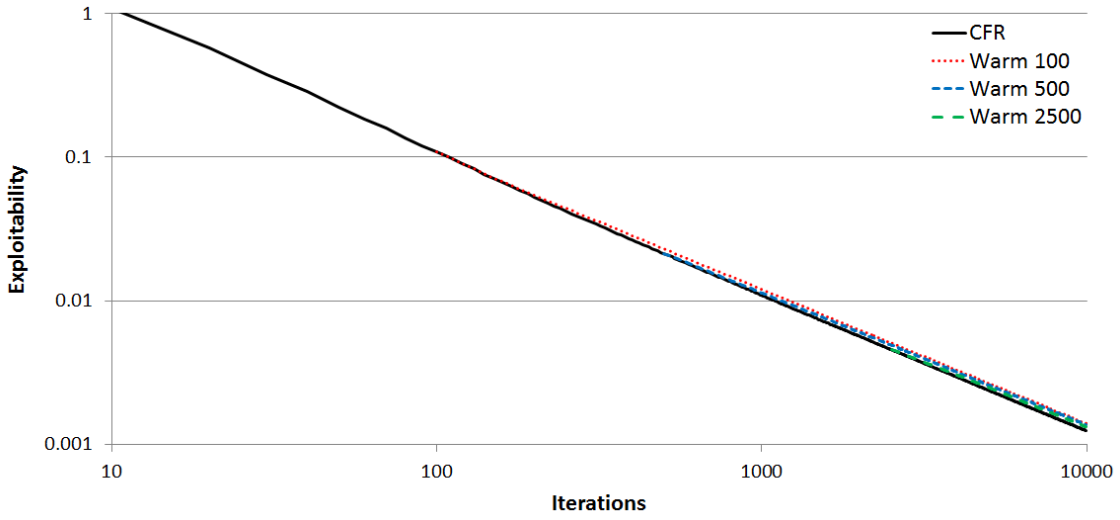


Figure 3.9: Comparison of CFR vs warm starting after 100, 500, or 2500 iterations. We warm started to 97, 490, and 2310 iterations, respectively. We used $\lambda = 0.08, 0.05, 0.02$ respectively (using the same λ for both players).

mance. However, it can lead to faster convergence early on due to all information sets in a bucket sharing their acquired regrets and due to the abstracted game tree being smaller. Abstraction is particularly useful when paired with MCCFR, since MCCFR can update the strategy of an entire bucket by sampling only one information set.

In our experiment, we compare three runs: CFR, MCCFR in which the 1,286,792 flop poker hands have been abstracted into just 5,000 buckets, and CFR that was warm started with six core minutes of the MCCFR run. As seen in Figure 3.10, the MCCFR run improves quickly but then levels off, while CFR takes a relatively long time to converge, but eventually overtakes the MCCFR run. The warm start run combines the benefit of both, quickly reaching a good strategy while converging as fast as CFR in the long run.

In many extensive-form games, later iterations are cheaper than earlier iterations due to the increasing prevalence of pruning, in which sections of the game tree need not be traversed. In this experiment, the first 10 iterations took 50% longer than the last 10, which is a relatively modest difference due to the particular implementation of CFR we used and the relatively small number of player actions in FTH. In other games and implementations, later iterations can be orders of magnitude cheaper than early ones, resulting in a much larger advantage to warm starting.

In this next experiment, we demonstrate the accuracy of our projection method for determining the number of iterations to warm start to. We plot the convergence rate of CFR in FTH, and also plot what the convergence rate is projected to be based on data from the first 10 iterations of CFR in FTH. Specifically, it predicts the rate of convergence as $\frac{10.82}{T}$, where T is the number of iterations. Although it bases this projection on just 10 iterations, the results show it to be very accurate even up to 10,000 iterations.

Next we demonstrate that performance is not particularly sensitive to the choice of λ. Figure 3.12 shows the results of warm starting after 500 iterations of CFR (warm starting to $T = 500$) when using various choices of λ (same λ for both players). Performance is virtually identi-

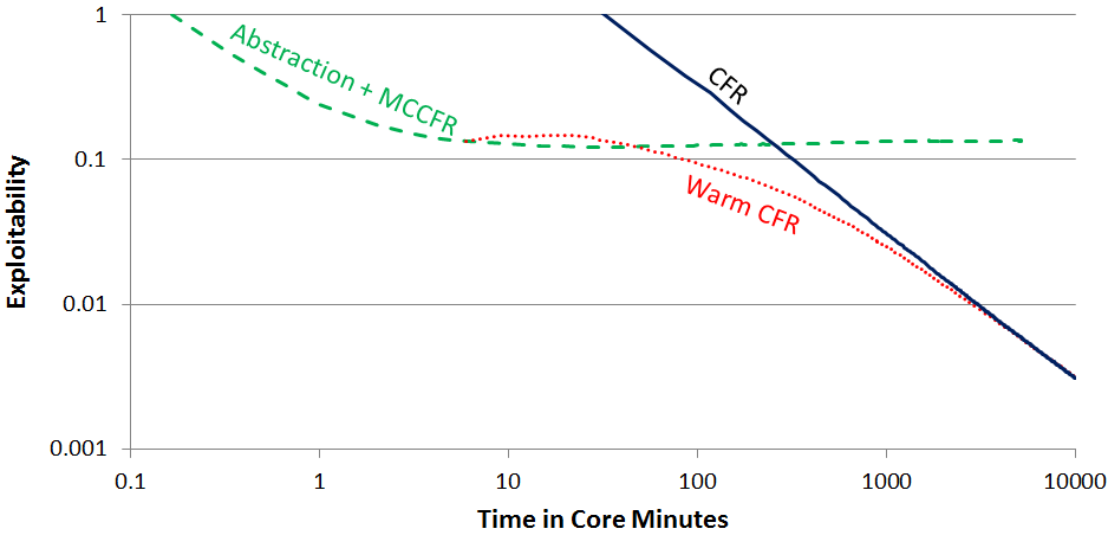


Figure 3.10: Performance of full-game CFR when warm started. The MCCFR run uses an abstraction with 5,000 buckets on the flop. After six core minutes of the MCCFR run, its average strategy was used to warm start CFR in the full to $T = 70$ using $\lambda = 0.08$.

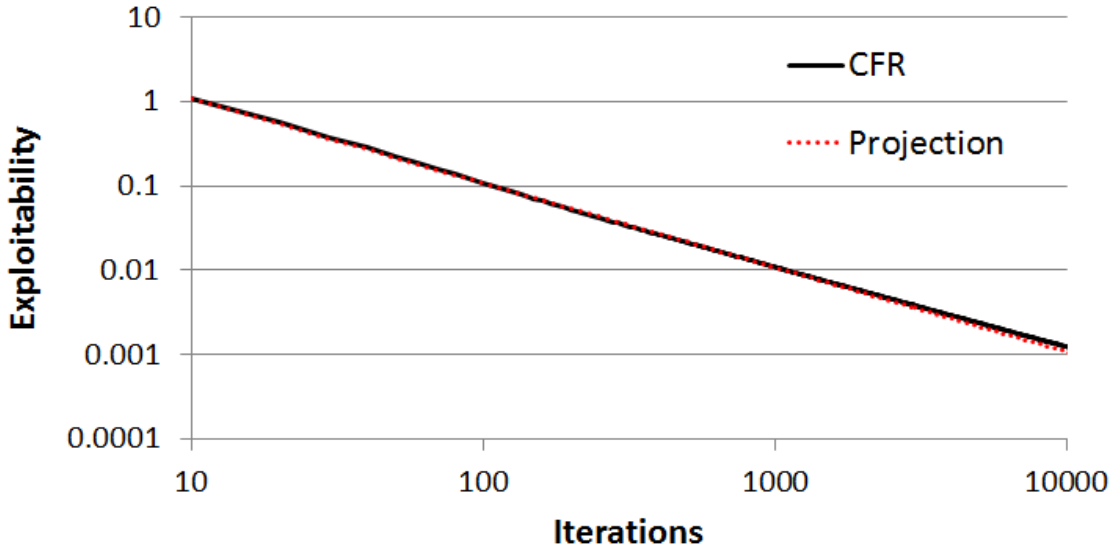


Figure 3.11: Actual convergence of CFR compared to a projection of convergence based on the first 10 iterations of CFR.

cal for $\lambda = 0, 0.05$, and 0.1, though $\lambda = 0.05$ performs the best by a small margin. Nevertheless, performance degrades dramatically when choosing a value such as $\lambda = 0.5$.

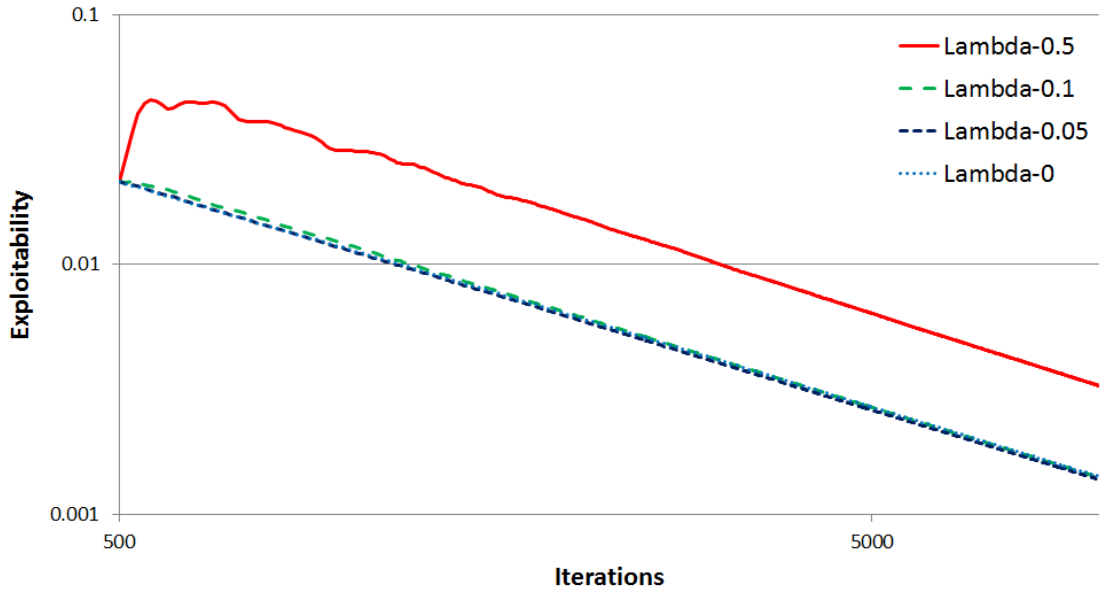


Figure 3.12: Comparison of different choices for λ when warm starting (using the same λ_i for both players).

3.2.6 Conclusions

We introduced a general method for warm starting RM and CFR in zero-sum games. We proved that after warm starting to T iterations, CFR converges just as quickly as if it had played T iterations of CFR from scratch. Moreover, we proved that this warm start method is "lossless." That is, when warm starting with the average strategy of T iterations of CFR, we can warm start to T iterations.

While other warm start methods exist, they can only be applied in special cases. A benefit of ours is that it is agnostic to the origins of the input strategies. We demonstrated that this can be leveraged by first solving a coarse abstraction and then using its solution to warm start CFR in the full game.

Recent research that finds close connections between CFR and other iterative equilibrium-finding algorithms [158] suggests that our techniques may extend beyond CFR as well. There are a number of equilibrium-finding algorithms with better long-term convergence bounds than CFR, but which are not used in practice due to their slow initial convergence [42, 72, 90, 114]. Our work suggests that a similar method of warm starting in these algorithms could allow their faster asymptotic convergence to be leveraged later in the run while CFR is used earlier on.

3.2.7 Proofs of Theoretical Results

Lemma 7

Lemma 7 proves that the growth of substitute regret has the same bound as the growth rate of normal regret. This lemma is used in the proof of Theorem 6.

Lemma 7. *If*

$$\sum_{a \in A} \left(R'^{T}(I,a)\right)_+^2 \leq \left(\pi_{-i}^{\sigma}(I)\right)\left(L(I)\right)^2 |A(I)|T$$

and strategies are chosen in I according to CFR using $R'^{T,T^*}(I,a)$ for all a on every iteration T^*, then $R'^{T,T'}(I) \leq \sqrt{\pi_{-i}^{\sigma'^{T,T'}}(I)} L(I) \sqrt{|A(I)|}\sqrt{T+T'}$.

Proof. After T' iterations of CFR, we are guaranteed that $\left(R'^{T,T'}(I,a)\right)_+^2 \leq \left(R'^{T}(I,a)\right)_+^2 + \sum_{t'=1}^{T'} \left(r^{t'}(I,a)\right)^2$. On iteration $t' \leq T'$, from (2.13) and (2.14), we know that $\left(r^{t'}(I,a)\right)^2 \leq \left(\pi_{-i}^{\sigma^{t'}}(I) L(I)\right)^2$. Thus,

$$\left(R'^{T,T'}(I,a)\right)_+^2 \leq \left(R'^{T}(I,a)\right)_+^2 + \sum_{t'=1}^{T'} \left(\pi_{-i}^{\sigma^{t'}}(I) L(I)\right)^2$$

$$\left(R'^{T,T'}(I,a)\right)_+^2 \leq \left(T\pi_{-i}^{\sigma}(I) + \sum_{t'=1}^{T'} \left(\pi_{-i}^{\sigma^{t'}}(I)\right)^2\right)\left(L(I)\right)^2$$

Since $0 \leq \pi_{-i}^{\sigma^{t'}}(I) \leq 1$, we know that $\sum_{t'=1}^{T'} \left(\pi_{-i}^{\sigma^{t'}}(I)\right)^2 \leq \sum_{t'=1}^{T'} \pi_{-i}^{\sigma^{t'}}(I)$. Also, $\sum_{t'=1}^{T'} \pi_{-i}^{\sigma^{t'}}(I) = T'\pi_{-i}^{\bar{\sigma}^{T'}}(I)$. So

$$\left(R'^{T,T'}(I,a)\right)_+^2 \leq \left(T\pi_{-i}^{\sigma}(I) + T'\pi_{-i}^{\bar{\sigma}^{T'}}(I)\right)\left(L(I)\right)^2$$

$$\left(R'^{T,T'}(I,a)\right)_+^2 \leq \left(\pi_{-i}^{\sigma'^{T,T'}}(I)\right)\left(L(I)\right)^2 (T+T')$$

Since $R'^{T,T'}(I) \leq \sqrt{\sum_{a \in A(I)} \left(R'^{T,T'}(I,a)\right)_+^2}$ so we get

$$R'^{T,T'}(I) \leq \sqrt{\pi_{-i}^{\sigma'^{T,T'}}(I)} L(I) \sqrt{|A(I)|}\sqrt{T+T'}$$

□

Lemma 8

Lemma 8 proves that we can use substitute regret to prove convergence to a Nash equilibrium just as we could use normal regret. Specifically, it proves that if average substitute regret is bounded for both players in the whole game, then the strategy profile is as close to a Nash equilibrium as if average normal regret were similarly bounded. This lemma is used in the proof of Theorem 6.

Lemma 8. *Define $R_i'^{T,T'}$ as*

$$\max_{\sigma_i^* \in \Sigma_i} \left(T\left(u_i(\sigma_i^*, \sigma_{-i}) - v_i'^{\sigma}\right) + \sum_{t'=1}^{T'} \left(u_i(\sigma_i^*, \sigma^{t'}) - u_i(\sigma^{t'})\right)\right) \tag{3.7}$$

In a two-player zero-sum game, if $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$ and $\frac{R'^{T,T'}}{T+T'} \leq \epsilon_i$, then $\sigma'^{T,T'}$ is a $(\epsilon_1 + \epsilon_2)$-equilibrium.

Proof. Since σ_i^* is the same on every iteration,

$$R_i'^{T,T'} = (T+T') \max_{\sigma_i^* \in \Sigma_i} u_i(\sigma_i^*, \sigma_{-i}'^{T,T'}) - T v_i'^{\sigma} - \sum_{t'=1}^{T'} u_i(\sigma^{t'})$$

Since $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$ and $u_1(\sigma^{t'}) = -u_2(\sigma^{t'})$, so

$$\max_{\sigma_1^* \in \Sigma_1} u_1(\sigma_1^*, \sigma_2'^{T,T'}) + \max_{\sigma_2^* \in \Sigma_2} u_2(\sigma_1'^{T,T'}, \sigma_2^*) \leq \epsilon_1 + \epsilon_2$$

$$\max_{\sigma_1^* \in \Sigma_1} u_1(\sigma_1^*, \sigma_2'^{T,T'}) - \min_{\sigma_2^* \in \Sigma_2} u_1(\sigma_1'^{T,T'}, \sigma_2^*) \leq \epsilon_1 + \epsilon_2$$

Since $u_1(\sigma_1'^{T,T'}, \sigma_2'^{T,T'}) \geq \min_{\sigma_2' \in \Sigma_2} u_1(\sigma_1'^{T,T'}, \sigma_2^*)$ so we have

$$\max_{\sigma_1^* \in \Sigma_1} u_1(\sigma_1^*, \sigma_2'^{T,T'}) - u_1(\sigma_1'^{T,T'}, \sigma_2'^{T,T'}) \leq \epsilon_1 + \epsilon_2$$

By symmetry, this is also true for Player 2. Therefore, $\sigma'^{T,T'}$ is a $(\epsilon_1 + \epsilon_2)$-equilibrium. □

Proof of Theorem 6

Proof. After setting T and $v'^{\sigma}(I)$ for every information set I, assume T' iterations were played according to CFR, where on iteration T^*, $\forall I \; \forall a$ we used substitute regret $R'^{T,T^*}(I,a)$.

We begin with some definitions. Define $v_i^{\sigma}(h) = \pi_{-i}^{\sigma}(h) \sum_{z \in Z} \big(\pi^{\sigma}(h,z) u_i(z)\big)$. Define $D(I)$ to be the information sets of player i reachable from I (including I). Define $\sigma|_{D(I) \to \sigma'}$ to be a strategy profile equal to σ except in the information sets in $D(I)$ where it is equal to σ'. Define $\mathit{succ}_i^{\sigma}(I'|I,a)$ to be the probability that I' is the next information set of player i visited given that the action a was just selected in information set I, and σ is the current strategy. Define $\mathit{Succ}(I,a)$ to be the set of all possible next information sets of player $P(I)$ visited given that action $a \in A(I)$ was just selected in information set I. Define $\mathit{Succ}(I) = \cup_{a \in A(I)} \mathit{Succ}(I,a)$. The *substitute full counterfactual regret* when warm starting from strategy σ and where $i = P(I)$ is

$$R_{\mathit{full}}'^{T,T'}(I) = \max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I) \to \sigma'}}(I) - v_i'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I) \to \sigma'}}(I) - v^{\sigma^{t'}}(I)\big)\Big) \quad (3.8)$$

We now prove recursively that

$$R_{\mathit{full}}'^{T,T'}(I) \leq \sum_{I' \in D(I)} \sqrt{\pi_{-i}^{\sigma' T,T'}(I')} L(I') \sqrt{|A(I')|} \sqrt{T+T'} \quad (3.9)$$

We define the *level* of an information set as follows. Any information set I such that $\mathit{Succ}(I) = \emptyset$ is level 1. Let ℓ be the maximum level of any $I' \in \mathit{Succ}(I)$. The level of I is $\ell + 1$.

First, consider an information set I of level 1. Then there are no Player i information sets following I, so

$$R'^{T,T'}_{i,full}(I) = \max_{a \in A(I)} \Big(T\big(v^{\sigma}(I,a) - v'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}}(I,a) - v^{\sigma^{t'}}(I)\big)\Big)$$

Since there are no further player i actions in this case, so $v^{\sigma}(I,a) = v'^{\sigma}(I,a)$. Therefore, $R'^{T,T'}_{full}(I) = R'^{T,T'}(I)$. By Lemma 7, (3.9) holds.

Now assume that (3.9) holds for all I' where the level of I' is at most ℓ. We prove (3.9) holds for all I with level $\ell + 1$ where $i = P(I)$.

From Lemma 7, we know that

$$Tv'^{\sigma}(I) + \sum_{t'=1}^{T'} v^{\sigma^t}(I) \geq \max_{a \in A(I)} \Big(Tv'^{\sigma}(I,a) + \sum_{t'=1}^{T'} v^{\sigma^t}(I,a) - \sqrt{\pi^{\sigma' T,T'}_{-i}(I)} L(I) \sqrt{|A(I)|} \sqrt{T+T'} \Big) \quad (3.10)$$

$$Tv'^{\sigma}(I) + \sum_{t'=1}^{T'} v^{\sigma^t}(I) \geq \max_{a \in A(I)} \Big(T \sum_{h \in I} \sum_{h' \in Succ_i(h \cdot a)} v_i'^{\sigma}(h') + \sum_{t'=1}^{T'} v^{\sigma^t}(I,a) - \sqrt{\pi^{\sigma' T,T'}_{-i}(I)} L(I) \sqrt{|A(I)|} \sqrt{T+T'} \Big) \quad (3.11)$$

We also know that for any σ,

$$\max_{\sigma' \in \Sigma_i} v^{\sigma|_{D(I) \to \sigma'}}(I) = \max_{a \in A(I)} \max_{\sigma'_i \in \Sigma_i} \sum_{h \in I} \Big(\sum_{z \in Z : z \in Succ_i(h \cdot a)} v_i^{\sigma}(z) + \sum_{h' \notin Z : h' \in Succ_i(h \cdot a)} v^{\sigma|_{D(I(h')) \to \sigma'}}(h') \Big) \quad (3.12)$$

Since for any $z \in Z$, $v_i^{\sigma}(z) = v_i'^{\sigma}(z)$, so combining (3.11) and (3.12) we get

$$\max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I)\to\sigma'}}(I) - v_i'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I)\to\sigma'}}(I) - v^{\sigma^{t'}}(I)\big)\Big) \leq$$
$$\max_{\sigma_i' \in \Sigma_i} \Big(T \sum_{h \in I} \sum_{h' \notin Z : h' \in \mathit{Succ}_i(h \cdot a)} \big(v^{\sigma|_{D(I(h'))\to\sigma'}}(h') - v_i'^{\sigma}(h')\big) +$$
$$\sum_{t'=1}^{T'} \sum_{h \in I} \sum_{h' \notin Z : h' \in \mathit{Succ}_i(h \cdot a)} \big(v^{\sigma^{t'}|_{D(I(h'))\to\sigma'}}(h') - v_i^{\sigma^{t'}}(h')\big) +$$
$$\sqrt{\pi_{-i}^{\sigma' T, T'}(I)} L(I) \sqrt{|A(I)|} \sqrt{T + T'} \quad (3.13)$$

Since we sum over only $h' \in \mathit{Succ}_i(h \cdot a)$ where $h' \notin Z$, this becomes

$$\max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I)\to\sigma'}}(I) - v_i'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I)\to\sigma'}}(I) - v^{\sigma^{t'}}(I)\big)\Big) \leq$$
$$\max_{\sigma_i' \in \Sigma_i} \Big(T \sum_{I' \in \mathit{Succ}(I,a)} \big(v^{\sigma|_{D(I')\to\sigma'}}(I') - v_i'^{\sigma}(I')\big) +$$
$$\sum_{t'=1}^{T'} \sum_{I' \in \mathit{Succ}(I,a)} \big(v^{\sigma^{t'}|_{D(I')\to\sigma'}}(I') - v_i^{\sigma^{t'}}(I')\big) +$$
$$\sqrt{\pi_{-i}^{\sigma' T, T'}(I)} L(I) \sqrt{|A(I)|} \sqrt{T + T'} \quad (3.14)$$

From the recursion assumption, for any $I' \in \mathit{Succ}(I, a)$,

$$\max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I)\to\sigma'}}(I') - v_i'^{\sigma}(I')\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I')\to\sigma'}}(I') - v^{\sigma^{t'}}(I')\big)\Big) \leq$$
$$\sum_{I'' \in D(I')} \sqrt{\pi_{-i}^{\sigma' T, T'}(I'')} L(I'') \sqrt{|A(I'')|} \sqrt{T + T'} \quad (3.15)$$

Therefore,

$$\max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I)\to\sigma'}}(I) - v_i'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I)\to\sigma'}}(I) - v^{\sigma^{t'}}(I)\big)\Big) \leq \sum_{I' \in \mathit{Succ}(I,a)} \sum_{I'' \in D(I')} \sqrt{\pi_{-i}^{\sigma' T,T'}(I'')} L(I'') \sqrt{|A(I'')|} \sqrt{T+T'} + \sqrt{\pi_{-i}^{\sigma' T,T'}(I)} L(I) \sqrt{|A(I)|} \sqrt{T+T'} \quad (3.16)$$

Since $\mathit{Succ}(I,a) \subseteq \mathit{Succ}(I)$ and since $D(I) = \cup_{I' \in \mathit{Succ}(I)} D(I') \cup \{I\}$, so

$$\max_{\sigma' \in \Sigma_i} \Big(T\big(v^{\sigma|_{D(I)\to\sigma'}}(I) - v_i'^{\sigma}(I)\big) + \sum_{t'=1}^{T'} \big(v^{\sigma^{t'}|_{D(I)\to\sigma'}}(I) - v^{\sigma^{t'}}(I)\big)\Big) \leq \sum_{I' \in D(I)} \sqrt{\pi_{-i}^{\sigma' T,T'}(I')} L(I') \sqrt{|A(I')|} \sqrt{T+T'} \quad (3.17)$$

Therefore, (3.9) holds by recursion.

Define $R_i'^{T,T'}$ according to (3.7). If $P(\emptyset) = i$, then (3.9) implies

$$R_i'^{T,T'} \leq \sum_{I \in \mathcal{I}_i} \sqrt{\pi_{-i}^{\sigma' T,T'}(I)} L(I) \sqrt{|A(I)|} \sqrt{T+T'} \quad (3.18)$$

If $P(\emptyset) \neq i$, then we could simply add a Player i information set at the beginning of the game with a single action. Therefore, (3.18) holds for every player i. Since $v_1'^{\sigma} + v_2'^{\sigma} \leq 0$ by construction, so we can applying Lemma 8 using (3.18), and thereby see that Theorem 6 holds. □

Proof of Corollary 1

Proof. After T iterations of CFR, for every information set I we could clearly assign $v'^{\bar{\sigma}^T}(I) = \frac{1}{T}\sum_{t=1}^{T} v^{\sigma^t}(I)$ in order to satisfy Theorem 6, since this would set regrets to exactly what they were before. From (3.2) we see this choice of $v'^{\bar{\sigma}^T}(I)$ satisfies (3.6). We instead choose $v'^{\bar{\sigma}^T}(I) \leq \frac{1}{T}\sum_{t=1}^{T} v^{\sigma^t}(I)$, where $v'^{\bar{\sigma}^T}(I)$ still satisfies (3.6). Since $v'^{\bar{\sigma}^T}(I) \leq \frac{1}{T}\sum_{t=1}^{T} v^{\sigma^t}(I)$ for every information set I, so from (3.5) we know $v_i'^{\bar{\sigma}^T} \leq \frac{1}{T}\sum_{t=1}^{T} u_i(\sigma^t)$. Therefore, $v_1'^{\bar{\sigma}^T} + v_2'^{\bar{\sigma}^T} \leq 0$ and we can apply Theorem 6 to warm start to T iterations. □

3.3 Regret-Based Pruning (RBP)

A key improvement that makes CFR practical in large games is *pruning*. At a high level, pruning allows the algorithm to avoid traversing the entire game tree while still maintaining the same convergence guarantees. The classic version of pruning, which we will refer to as **partial pruning**,

allows the algorithm to skip updates for a player in a sequence if the other player's current strategy does not reach the sequence with positive probability. This dramatically reduces the cost of each iteration. The magnitude of this reduction varies considerably depending on the game, but can easily be higher than 90% [96], which improves the convergence speed of the algorithm by a factor of 10. Moreover, the benefit of partial pruning empirically seems to be more significant as the size of the game increases.

While partial pruning leads to a large gain in speed, we observe that there is still room for much larger speed improvement. Partial pruning only skips updates for a player if an *opponent's* action in the path leading to that point has zero probability. This can fail to prune paths that are actually prunable. Consider a game where the first player to act (Player 1) has hundreds of actions to choose from, and where, over several iterations, the reward received from many of them is extremely poor. Intuitively, we should be able to spend less time updating the strategy for Player 1 following these poor actions, and more time on the actions that proved worthwhile so far. However, here, partial pruning will continue to update Player 1's strategy following each action in every iteration.

In this section we introduce a better version of pruning, **regret-based pruning (RBP)**, in which CFR can avoid traversing a path in the game tree if *either* player takes actions leading to that path with zero probability. This pruning needs to be temporary, because the reach probabilities may become positive in a later CFR iteration. However, we can continue to prune for as long as we know with certainty that the reach probability of a sequence is zero. Since reach probability is determined by regret, and since regret cannot increase by more than the maximum range of payoffs in the game on each iteration, so the number of CFR iterations that an action can be pruned is proportional to how negative the regret is for that action.

RBP can lead to a dramatic improvement depending on the game. As a rough example, consider a game in which each player has very negative regret for actions leading to 90% of nodes. Partial pruning, which skips updates for a player when the opponent does not reach the node, would traverse 10% of the game tree per iteration. In contrast, regret-based pruning, which skips updates when either player does not reach the node, would traverse only $0.1 \cdot 0.1 = 1\%$ of the game tree. In general, RBP asymptotically roughly squares the performance gain of partial pruning.

We test RBP with CFR and CFR+. Experiments show that it leads to more than an order of magnitude speed improvement over partial pruning. The benefit increases with the size of the game.

3.3.1 Applying Best Response to Zero-Reach Sequences

In Section 2.3.4 it was explained that if both players' average regret approaches zero, then their average strategies approach a Nash equilibrium. CFR provides one way to compute strategies that have bounded regret, but it is not the only way. CFR-BR [80] is a variant of CFR in which one player plays CFR and the other player plays a best response to the opponent's strategy in every iteration. Calculating a best response to a fixed strategy is computationally cheap (in games of perfect recall), costing only a single traversal of the game tree. By playing a best response in every iteration, the best-responder is guaranteed to have at most zero regret. Moreover, the CFR player's regret is still bounded sublinearly because the CFR regret bounds hold regardless of the

other player's policy. However, in practice the CFR player's regret in CFR-BR tends to be higher than when both players play vanilla CFR (since the opponent is clairvoyantly maximizing the CFR player's regret). For this reason, empirical results show that CFR-BR converges slower than CFR, even though the best-responder's regret is always at most zero.

We now discuss a modification of CFR that will motivate regret-based pruning. The idea is that *by applying a best response only in certain situations* (and CFR in others), we can lower regret for one player without increasing it for the opponent. Without loss of generality, we discuss how to reduce regret for Player 1. Specifically, consider an information set $I \in \mathcal{I}_1$ and action a where $\sigma^t(I, a) = 0$ and any history $h \in I$. Then for any ancestor history h' such that $h' \sqsubset h{\cdot}a$, we know $\pi_1^{\sigma^t}(h', h{\cdot}a) = 0$. Likewise, for any descendant history h' such that $h{\cdot}a \sqsubseteq h'$, we know $\pi_1^{\sigma^t}(h') = 0$. Thus, from Equation 2.13 we see that Player 1's strategy on iteration t in any information set following action a has no effect on Player 2's regret for that iteration. Moreover, it also has no effect on Player 1's regret for any information set except $R(I, a)$ and information sets that follow action a. Therefore, by playing a best response *only* in information sets following action a (and playing vanilla CFR elsewhere), Player 1 guarantees zero regret for himself in all information sets following action a, without the practical cost of increasing his regret in information sets before I or of increasing Player 2's regret. This may increase regret for action a itself, but if we only do this when $R(I, a) \leq -L(I)$, we can guarantee $R(I, a) \leq 0$ even after the iteration. Similarly, Player 2 can simultaneously play a best response in information sets following an action a' where $\sigma_2^t(I', a') = 0$ for $I' \in \mathcal{I}_2$. This approach leads to lower regret for both players.

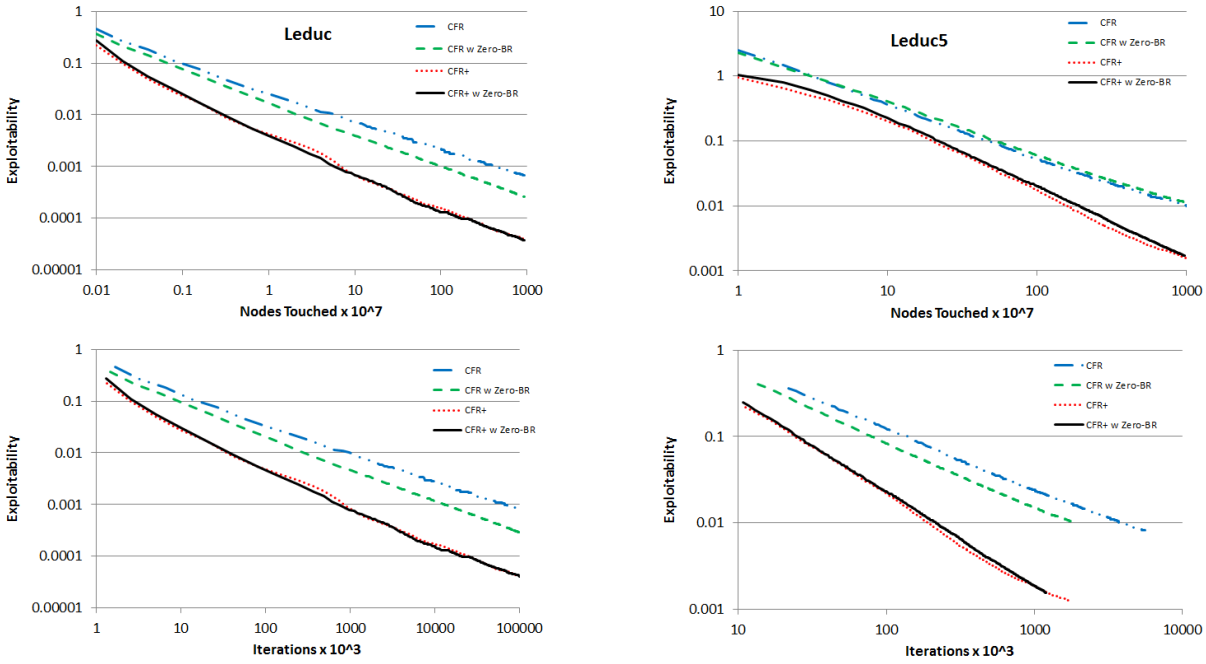
(In situations where *both* players' sequences of reaching an information set have zero probability ($\pi_1(h) = \pi_2(h) = 0$) the strategies chosen have no impact on the regret or average strategy for either player, so there is no need to compute what strategies should be played from then on.)

Figure 3.13 shows that this technique (of playing a best response only in the special case where an action is taken with zero probability) leads to a dramatic improvement over CFR in terms of the number of iterations needed—though the theoretical convergence bound remains the same. However, each iteration touches more nodes—because negative-regret actions more quickly become positive and are not skipped with partial pruning—and thus takes longer. Thus, in Leduc-5, the number of nodes touched (which is a better measure of time required), is actually slightly higher for a given level of convergence. In both games, playing a best response in zero-reach sequences led to no substantial difference when combined with CFR+. It depends on the game whether CFR or this technique is faster overall.

Regret-based pruning, introduced in the next section, outperforms both of these approaches significantly.

3.3.2 Description of Regret-Based Pruning

In this section we present **regret-based pruning (RBP)**, a technique for soundly pruning—on a temporary basis—negative-regret actions from the tree traversal in order to speed it up significantly. In Section 3.3.1 we proposed a variant of CFR where a player plays a best response in information sets that the player reaches with zero probability. In this section, we show that these information sets and their descendants need not be traversed in every iteration. Rather, the frequency that they must be traversed is proportional to how negative the regret is for the action



3.13a Leduc Hold'em

3.13b Leduc-5 Hold'em

Figure 3.13: Top: Exploitability vs Nodes Touched. Bottom: Exploitability vs Iterations.

leading to them. This less-frequent traversal does not hurt the regret bound (2.20). Consider an information set $I \in \mathcal{I}_1$ and action a where $R^t(I,a) = -1000$ and regret for at least one other action in I is positive, and assume $L(I) = 1$. From (2.2), we see that $\sigma_1^{t+1}(I,a) = 0$. As described in Section 3.3.1, the strategy played by Player 1 on iteration $t+1$ in any information set following action a has no effect on Player 2. Moreover, it has no immediate effect on what Player 1 will do in the next iteration (other than in information sets following action a), because we know regret for action a will still be at most -999 on iteration $t+2$ (since $L(I) = 1$) and will continue to not be played. So rather than traverse the game tree following action a, we could "procrastinate" in deciding what Player 1 did on iteration $t+1, t+2, ..., t+1000$ in that branch until after iteration $t+1000$ (at which point regret for that action may be positive). That is, we could (in principle) store Player 2's strategy for each iteration between $t+1$ and $t+1000$, and on iteration $t+1000$ calculate a best response to each of them and announce that Player 1 played those best responses following action a on iterations $t+1$ to $t+1000$ (and update the regrets to match this). Obviously this itself would not be an improvement, but performance would be identical to the algorithm described in Section 3.3.1.

However, rather than have Player 1 calculate and play a best response for each iteration between $t+1$ and $t+1000$ separately, we could simply calculate a best response against the average strategy that Player 2 played in those iterations. This can be accomplished in a single traversal of the game tree. We can then announce that Player 1 played this best response on each iteration between $t+1$ and $t+1000$. This provides benefits similar to the algorithm described in Section 3.3.1, but allows us to do the work of 1000 iterations in a single traversal!

Let $D(I,a)$ be the set of infosets following $P(I)$ taking action a in infoset I. We now present a theorem that guarantees that when $R(I,a) \leq 0$, we can prune $D(I,a)$ through regret-based pruning for $\lfloor \frac{|R(I,a)|}{L(I)} \rfloor$ iterations.

Theorem 7. *Consider a two-player zero-sum game. Let $a \in A(I)$ be an action such that on iteration T_0, $R^{T_0}(I,a) \leq 0$. Let I' be an information set for any player such that $I' \notin D(I,a)$ and let $a' \in A(I')$. Let $m = \lfloor \frac{|R(I,a)|}{L(I)} \rfloor$. If $\sigma(I,a) = 0$ when $R(I,a) \leq 0$, then regardless of what is played in $D(I,a)$ during $\{T_0, ..., T_0 + m\}$, $R_+^T(I',a')$ is identical for $T \leq T_0 + m$.*

We can improve this approach significantly by not requiring knowledge *beforehand* of exactly how many iterations can be skipped. Rather, we will *decide in light of what happens during the intervening CFR iterations when an action needs to be revisited.* From (2.15) we know that $r^T(I,a) \propto \pi_{-i}^{\sigma^T}(I)$. Moreover, $v_{P(I)}^{\sigma^T}(I)$ does not depend on $D(I,a)$. Thus, we can prune $D(I,a)$ from iteration T_0 until iteration T_1 so long as

$$\sum_{t=1}^{T_0} v_{P(I)}^{\sigma^t}(I,a) + \sum_{t=T_0+1}^{T_1} \pi_{-i}^{\sigma^t}(I) U(I,a) \leq \sum_{t=1}^{T_1} v_{P(I)}^{\sigma^t}(I) \tag{3.19}$$

where $U(I,a)$ is the maximum payoff achievable for taking action a in infoset I. In the worst case, this allows us to skip only $\lfloor \frac{R(I,a)}{L(I)} \rfloor$ iterations. However, in practice it performs significantly better, though we cannot know on iteration T_0 how many iterations it will skip because it depends on what is played in $T_0 \leq t \leq T_1$. Our exploratory experiments showed that in practice performance also improves by replacing $U(I,a)$ with a more accurate upper bound on reward in (3.19). CFR will still converge if $D(I,a)$ is pruned for too many iterations; however, that hurts convergence speed. In the experiments included in this section, we conservatively use $U(I,a)$ as the upper bound.

3.3.3 Best Response Calculation for Regret-Based Pruning

In this section we discuss how one can efficiently compute the best responses as called for in regret-based pruning. The advantage of Theorem 7 is that we can wait until after pruning has finished—that is, until we revisit an action—to decide what strategies were played in $D(I,a)$ during the intervening iterations. We can then calculate a single best response to the average strategy that the opponent played, and say that that best response was played in $D(I,a)$ in each of the intervening iterations. This results in zero regret over those iterations for information sets in $D(I,a)$. We now describe how this best response can be calculated efficiently.

Typically, when playing CFR one stores $\sum_{t=1}^{T} \pi_i^t(I)\sigma^t(I)$ for each information set I, where i is the player that acts at I. This allows one to immediately calculate the average strategy defined in (2.22) in any particular iteration. If we start pruning on iteration T_0 and revisit on iteration T_1, we wish to calculate a best response to $\bar{\sigma}_i^{T_1 - T_0}$ where $\bar{\sigma}^{T_1-T_0}(I) = \frac{\sum_{t=T_0}^{T_1} \pi_i^t(I)\sigma^t(I)}{\sum_{t=T_0}^{T_1} \pi_i^t(I)}$. An easy approach would be to store the opponent's cumulative strategy before pruning begins and subtract it from the current cumulative strategy when pruning ends. In fact, we only need to store the opponent's strategy in information sets that follow action a. However, this could potentially use $O(H)$ memory because the same information set I belonging to Player 2 may be

reached from multiple information sets belonging to Player 1. In contrast, CFR only requires $O(|\mathcal{I}||A|)$ memory, and we want to maintain this desirable property. We accomplish that as follows.

To calculate a best response against $\bar{\sigma}_2^T$, we traverse the game tree and calculate the counterfactual value, defined in (2.14), for every action for every information set belonging to Player 1 that does not lead to any further Player 1 information sets. Specifically, we calculate $v_1^{\bar{\sigma}^{T_0-1}}(I,a)$ for every action a in I such that $D(I,a) = \emptyset$. Since we calculate this only for actions where $D(I,a) = \emptyset$, so $v_1^{\bar{\sigma}^{T_0-1}}(I,a)$ does not depend on $\bar{\sigma}_1$. Then, starting from the bottom information sets, we set the best-response strategy $\sigma_1^{BR}(I)$ to always play the action with the highest counterfactual value (ties can be broken arbitrarily), and pass this value up as the payoff for reaching I, repeating the process up the tree. In order to calculate a best response to $\bar{\sigma}_2^{T_1-T_0}$, we first store, before pruning begins, the counterfactual values for Player 1 against Player 2's average strategy for every action a in each information set I where $D(I,a) = \emptyset$. When we revisit the action on iteration T_1, we calculate a best response to $\bar{\sigma}_2^{T_1}$ except that we set the counterfactual value for every action a in information set I where $D(I,a) = \emptyset$ to be $T_1 v_1^{\bar{\sigma}^{T_1}}(I,a) - (T_0-1)v_1^{\bar{\sigma}^{T_0-1}}(I,a)$. The latter term was stored, and the former term can be calculated from the current average strategy profile. As before, we set $\sigma_1^{BR}(I)$ to always play whichever action has the highest counterfactual value, and pass this term up.

A slight complication arises when we are pruning an action a in information set I and wish to start pruning an earlier action a' from information set I' such that $I \in D(I',a')$. In this case, it is necessary to explore action a in order to calculate the best response in $D(I',a')$. However, if such traversals happen frequently, then this would defeat the purpose of pruning action a. One way to address this is to only prune an action a' when the number of iterations guaranteed (or estimated) to be skipped exceeds some threshold. This ensures that the overhead is worthwhile, and that we are not frequently traversing an action a farther down the tree that is already being pruned. Another option is to add some upper bound to how long we will prune an action. If the lower bound for how long we will prune a exceeds the upper bound for how long we will prune a', then we need not traverse a in the best response calculation for a' because a will still be pruned when we are finished with pruning a'. In our experiments, we use the former approach. Experiments to determine a good parameter for this are presented in Appendix 3.3.6.

3.3.4 Regret-Based Pruning with DCFR and CFR+

CFR+ [149] is a variant of CFR where the regret is never allowed to go below 0. At first glance, it would seem that CFR+ and RBP are incompatible. RBP allows actions to be traversed with decreasing frequency as regret decreases below zero. However, CFR+ sets a floor for regret at zero. Nevertheless, it is possible to combine the two, as we now show. We modify the definition of regret in CFR+ so that it can drop below zero, but *immediately returns to being positive as soon as regret begins increasing*. Formally, we modify the definition of regret in CFR+ for $T > 0$ to be as follows: $R^T(I,a) = r^T(I,a)$ if $r^T(I,a) > 0$ and $R^{T-1}(I,a) \leq 0$, and $R^T(I,a) = R^{T-1}(I,a) + r^T(I,a)$ otherwise. This leads to identical behavior in CFR+, and also allows regret to drop below zero so actions can be pruned.

When using RBP with CFR+, regret does not strictly follow the rules for CFR+. CFR+ calls for an action to be played with positive probability whenever instantaneous regret for it is positive

in the previous iteration. Since RBP only checks the regret for an action after potentially several iterations have been skipped, there may be a delay between the iteration when an action would return to play in CFR+ and the iteration when it returns to play in RBP. This does not pose a theoretical problem: CFR's convergence rate still applies.

However, this difference is noticeable when combined with *linear averaging* of the average strategy, which is shown in Equation 2.29. Empirically, linear averaging causes CFR+ to converge to a Nash equilibrium much faster. However, in vanilla CFR it results in worse performance and there is no proof guaranteeing convergence. Since RBP with CFR+ results in behavior that does not strictly conform to CFR+, linear averaging results in somewhat noisier convergence. This can be mitigated by reporting the strategy profile found so far that is closest to a Nash equilibrium rather than the current average strategy profile, and we do this in the experiments.

Rather than use CFR+, a better option is to use DCFR, discussed in Section 3.1, with a set of parameters that allows negative regrets to decrease to $-\infty$. Specifically, using DCFR with parameters $\alpha = 1.5$, $\beta = 0.5$, and $\gamma = 2$ results in performance that outperforms CFR+ in most settings and is usually comparable to DCFR with $\beta = 0$, while still allowing negative regret to decrease to $-\infty$.

3.3.5 Experiments

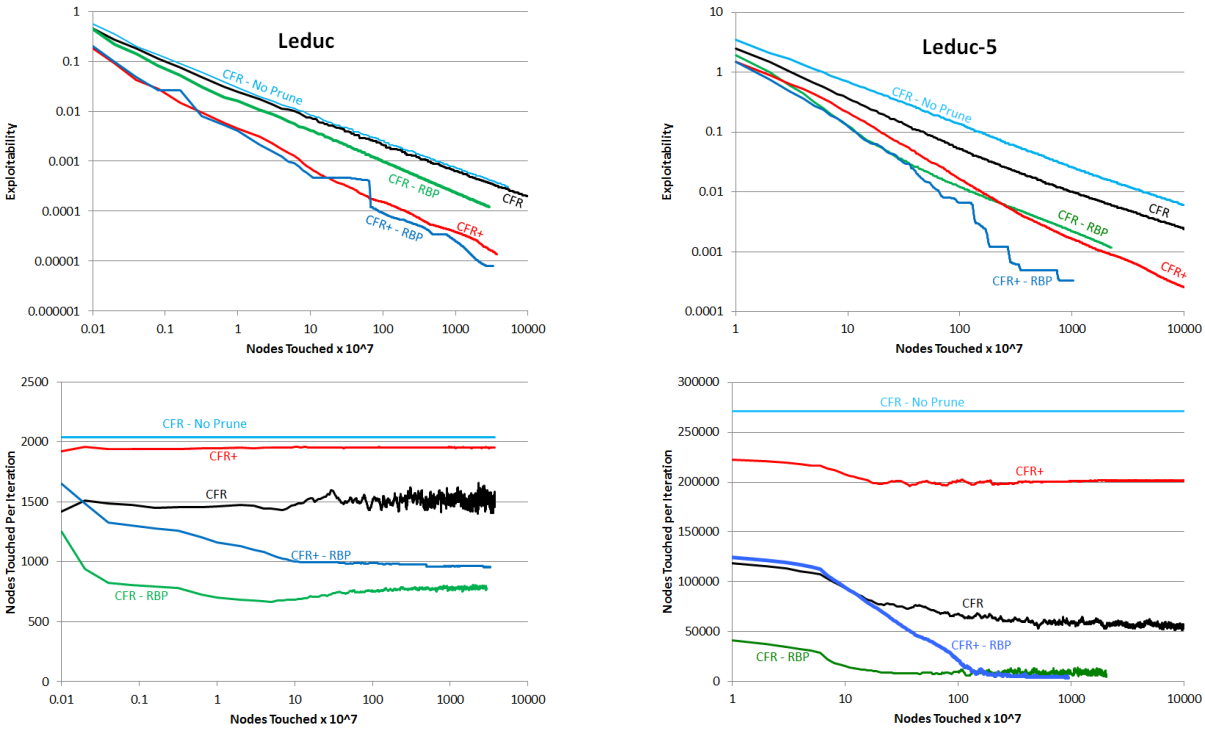
We tested regret-based pruning in both CFR and CFR+ against partial pruning, as well as against CFR with no pruning. The implementation in these experiments traverses the game tree once each iteration.[3] We tested our algorithm on standard Leduc Hold'em [143], described in Section 2.4.1, and a scaled-up variant we call **Leduc-5** in which there are 5 bet sizes to choose from: in the first round a player may bet 0.5, 1, 2, 4, or 8 chips, while in the second round a player may bet 1, 2, 4, 8, or 16 chips.

We measure the total exploitability of the strategy profiles. As shown in Figure 1, RBP leads to a substantial improvement over vanilla CFR with partial pruning in Leduc Hold'em, increasing the speed of convergence by more than a factor of 8. This is partially due to the game tree being traversed twice as fast, and partially due to the use of a best response in sequences that are pruned (the benefit of which was described in Section 3.3.1). The improvement when added on top of CFR+ is smaller, increasing the speed of convergence by about a factor of 2. This matches the reduction in game tree traversal size.

The benefit from RBP is more substantial in the larger benchmark game, Leduc-5. RBP increases convergence speed of CFR by a factor of 12, and reduces the per-iteration game tree traversal cost by about a factor of 7. In CFR+, RBP improves the rate of convergence by about an order of magnitude. RBP also decreases the number of nodes touched per iteration in CFR+ by about a factor of 40.

The results imply that larger games benefit more from RBP than smaller games. This is not universally true, since it is possible to have a large game where every action is part of the Nash equilibrium. Nevertheless, there are many games with very large action spaces where the vast majority of those actions are suboptimal, but players do not know beforehand which are

[3]Canonical CFR+ traverses the game tree twice each iteration, updating the regrets for each player in separate traversals [149]. This difference does not, however, affect the error measure (y-axis) in the experiments.



3.14a Leduc Hold'em

3.14b Leduc-5 Hold'em

Figure 3.14: Top: Exploitability. Bottom: Nodes touched per iteration.

suboptimal. In such games, RBP would improve convergence tremendously.

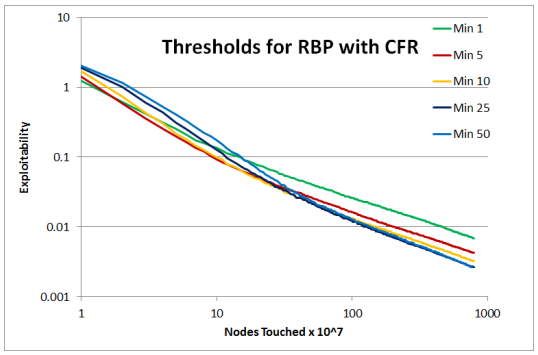
3.3.6 Comparison of Minimum Skip Thresholds

As discussed in Section 3.3.3, it may be worthwhile to establish some minimum threshold for the anticipated number of iterations to be skipped in order to prune an action. This ensures that the overhead of pruning is worth the gain, and also prevents descendant actions that are already pruned from being repeatedly traversed. Setting such a threshold does not affect the theoretical guarantees of the algorithm, and may lead to better empirical performance.

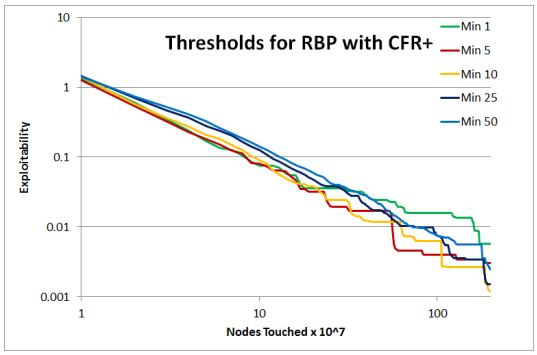
We now describe how to set such a threshold. We estimate the number of iterations that will be skipped if we prune an action by modifying (3.19) to assume the information set will continue to be reached by the opponent as often as it has, on average, in the past. We also modify the equation to assume that $v^{\sigma^t}_{P(I)}(I)$ will be the average that has been received in the past. This allows us to solve the equation for the number of iterations we estimate will be skipped. If this estimated number of iterations exceeds some minimum threshold, then we proceed with pruning the action. Formally, we calculate the estimate as

$$\hat{T}_1 - T_0 = \frac{R^{T_0}(I,a)}{\frac{\sum_{t=1}^{T_0} v^{\sigma^t}(I)}{T_0} - \frac{\sum_{t=1}^{T_0} \pi^{\sigma^t}_{-i} U(I,a)}{T_0}} \tag{3.20}$$

In Figure 3.15, we compare 1, 5, 10, 25, and 50 as possible thresholds in Leduc-5 for regret-based pruning in CFR and CFR+. All of the options performed similarly, suggesting that the algorithm is not very sensitive to the parameter chosen. The experiments show that a low threshold is preferable early on, while a higher threshold leads to better performance later. The long-term gain of increasing the threshold appears to quickly diminish however, as there is only a slight long-term difference between a threshold of 10 iterations and a threshold of 50 iterations. In the experiments presented in Section 3.3.5, we use a threshold of 25 iterations for RBP in all cases.



3.15a Leduc Hold'em



3.15b Leduc-5 Hold'em

Figure 3.15: A comparison of minimum thresholds for estimated number of iterations pruned for RBP in CFR (left) and CFR+ (right)

3.3.7 Conclusions

RBP is a new method of pruning that allows CFR to avoid traversing high-regret actions in every iteration. RBP temporarily ceases their traversal in a sound way without compromising the overall convergence rate. Experiments show an order of magnitude speed improvement over partial pruning, and suggest that the benefit of RBP increases with game size. Thus RBP is particularly useful in large games where many actions are suboptimal, but where it is not known beforehand which actions those are.

One limitation of RBP is that its benefits are reduced when using vector-form implementations of CFR introduced by Johanson et al. [79], in which the regrets for all infosets sharing a public state are updated simultaneously. While the regrets for individual infosets can be skipped, it is still necessary to explore the descendant public nodes and evaluate all terminal nodes unless all infosets in the public state can be pruned.

3.3.8 Proofs of Theoretical Results

Proof of Theorem 7

Proof. By definition of $L(I)$, we know that $r^t(I,a) \leq L(I)$. Thus, for iteration $T_0 \leq T \leq T_0 + m$, $R^T(I,a) \leq 0$. Clearly the theorem is true for $T < T_0$. We prove the theorem continues to hold inductively for $T \leq T_0 + m$. Assume the theorem holds for iteration T and consider

iteration $T+1$. Suppose $I' \in \mathcal{I}_{P(I)}$ and either $I' \neq I$ or $a' \neq a$. Then for any $h' \in I'$, there is no ancestor of h' in an information set in $D(I,a)$. Thus, $\pi_{-i}^{\sigma^{T+1}}(h')$ does not depend on the strategy in $D(I,a)$. Moreover, for any $z \in Z$, if $h' \sqsubset h \sqsubset z$ for some $h \in I^* \in D(I,a)$, then $\pi^{\sigma^{T+1}}(h',z) = 0$ because $\sigma^{T+1}(I,a) = 0$. Since $I' \neq I$ or $a' \neq a$, it similarly holds that $\pi^{\sigma^{T+1}}(h' \cdot a', z) = 0$. Then from Equation 2.13 and Equation 2.14, $r^{T+1}(I,a)$ does not depend on the strategy in $D(I,a)$.

Now suppose $I' \in \mathcal{I}_i$ for $i \neq P(I)$. Consider some $h' \in I'$ and some $h \in I$. First suppose that $h \cdot a \sqsubseteq h'$. Since $\pi_i^{\sigma^{T+1}}(h \cdot a) = 0$, so $\pi_i^{\sigma^{T+1}}(h') = 0$ and h' contributes nothing to the regret of I'. Now suppose $h' \sqsubset h$. Then for any $z \in Z$, if $h' \sqsubset h \sqsubset z$ then $\pi^{\sigma^{T+1}}(h',z) = 0$ and does not depend on the strategy in $D(I,a)$. Finally, suppose $h' \not\sqsubset h$ and $h \cdot a \not\sqsubseteq h'$. Then for any $z \in Z$ such that $h' \sqsubset z$, we know $h \not\sqsubset z$ and therefore $\pi^{\sigma^{T+1}}(h',z) = 0$ does not depend on the strategy in $D(I,a)$.

Now suppose $I' = I$ and $a' = a$. We proved $R^T(I,a) \leq 0$ for $T_0 \leq T \leq T_0 + m$, so $R_+^T(I,a) = 0$. Thus, for all $T \leq T_0 + m$, $R^T(I',a')$ is identical regardless of what is played in $D(I,a)$. □

3.4 Dynamic Thresholding

While regret-minimizing algorithms other than RM can be used within CFR, and iterative algorithms other than CFR exist with better convergence bounds in terms of the number of iterations needed [61, 72, 121], CFR with RM exhibits superior empirical performance in large games [90]. A primary reason for this is that CFR with RM is able to put zero probability on some actions, and therefore prune large sections of the game tree, particularly in large games. That is, it need not traverse the entire game tree on each iteration. This behavior is shared by some other regret minimizing algorithms, but is relatively uncommon and is considered a desirable property [105].

In this section we describe **dynamic thresholding**, a method that allows pruning to be applied in a wider range of algorithms, and applied more frequently in settings that already support pruning. We focus on Hedge [47, 104], also known as the exponentially-weighted forecaster, which is the most popular regret-minimizing algorithm in domains other than extensive-form game solving, on RM, and on the Excessive Gap Technique (EGT) [61, 114], which converges to an ϵ-Nash equilibrium in two-player zero-sum games in $\mathcal{O}(\frac{1}{\epsilon})$, that is, in significantly fewer iterations (in theory) than CFR which converges in $\mathcal{O}(\frac{1}{\epsilon^2})$.

3.4.1 Dynamic Thresholding

As described in Section 3.3, regret-based pruning and partial pruning can lead to much faster convergence when running CFR. However, these pruning techniques can only be applied when an action is played with zero probability, so pruning is incompatible with regret minimization algorithms like Hedge that assign positive probability to all actions on every iteration. This motivates our introduction of **dynamic thresholding**, in which low-probability actions are set to zero probability.

In dynamic thresholding for Hedge, when each successive iteration t is computed we set any

action with probability less than $\frac{(C-1)\sqrt{\ln(|A(I)|)}}{\sqrt{2}|A(I)|^2\sqrt{t}}$ (where $C \geq 1$) to zero probability and normalize the remaining action probabilities accordingly so they sum to 1. We then use this new probability vector to determine the regrets of iteration $t+1$. If an action is thresholded, this deviation from what Hedge calls for may lead to worse performance and therefore higher regret. In particular, since the altered probability vectors determine the regrets for future iterations, there is a risk that this error could snowball. However, using the threshold that we just specified above, we ensure that the new regret is within a constant factor C of the traditional regret bound.

Theorem 8. *If player $P(I)$ plays according to Hedge in an information set I for T iterations using threshold $\frac{(C-1)\sqrt{\ln(|A(I)|)}}{\sqrt{2}|A(I)|^2\sqrt{t}}$ with $C \geq 1$ on every iteration t, then*

$$R^T(I) \leq C\sqrt{2}L(I)\sqrt{\ln(|A(I)|)}\sqrt{T}$$

To apply the above theorem within CFR, we get from Equation 2.21 that one can then just sum the regrets of all information sets to bound the total regret for this player.

Dynamic thresholding can in general be applied to any regret minimization algorithm. We present Theorem 8 specifically for Hedge in order to tailor the threshold for that algorithm, which provides a tighter theoretical bound. In Theorem 9, we also show that dynamic thresholding can be applied to RM. However, it results in very little, if any, additional pruning. This is because RM is very unlikely in practice to put extremely small probabilities on actions. Nevertheless, we prove that dynamic thresholding applies to RM for the sake of completeness and for its potential theoretical applications. Note that the formula for the threshold is now different.

Theorem 9. *If player $P(I)$ plays according to regret matching in an information set I for T iterations using threshold $\frac{C^2-1}{2C|A(I)|^2\sqrt{t}}$ with $C \geq 1$ on every iteration t, then*

$$R^T(I) \leq CL(I)\sqrt{|A(I)|}\sqrt{T}$$

Again, to apply the above theorem within CFR, we get from Equation 2.21 that one can then just sum the regrets of all information sets to bound the total regret for this player.

3.4.2 Regret-Based Pruning for Hedge

In this section we describe how dynamic thresholding enables regret-based pruning when using Hedge. To use RBP, it is necessary to determine a lower bound on the number of iterations for which an action will have zero probability. In RM without dynamic thresholding this is simply the minimum number of iterations it would take an action to achieve positive regret, as shown in (3.19). In Hedge with dynamic thresholding, we instead must determine the minimum number of iterations it would take for an action to reach probability above the dynamic threshold.

Let $R^{T_0}(I,a)$ be the regret for an action a in information set I on iteration T_0. If $\sigma^{T_0}(I,a) < \frac{(C-1)\sqrt{\ln(|A(I)|)}}{\sqrt{2}|A(I)|^2\sqrt{T_0}}$, where $\sigma^{T_0}(I,a)$ is determined according to the Hedge algorithm shown in (2.8), then pruning can begin on iteration T_0. By Theorem 8, we can prune the game tree following action a on any consecutive iteration T after that if

$$\frac{e^{\eta_T\left(R^{T_0}(I,a)+U(I,a)(T-T_0)\right)}}{\sum_{a'\in A(I)} e^{\eta_T\left(R^T(I,a')+\sum_{T'=T_0+1}^{T} v^{T'}(I,a')\right)}} < \frac{(C-1)\sqrt{\ln(|A(I)|)}}{\sqrt{2}|A(I)|^2\sqrt{t}} \tag{3.21}$$

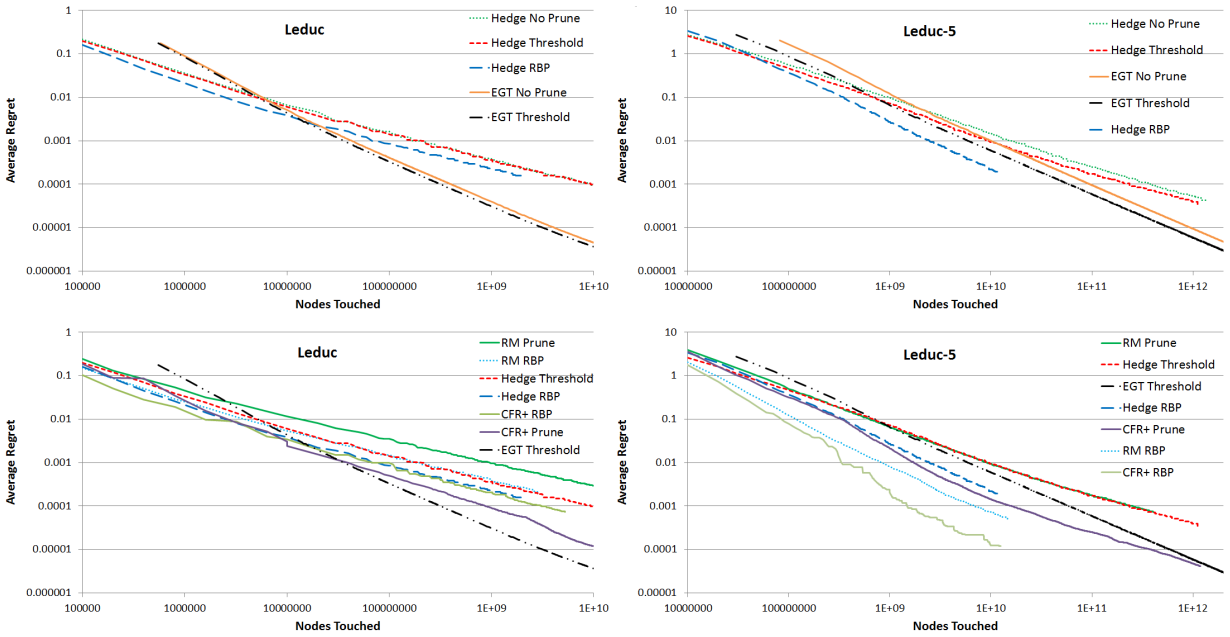


Figure 3.16: Performance of EGT, CFR with Hedge, and CFR with RM on Leduc and Leduc-5. CFR with Hedge is shown without any pruning (vanilla Hedge), with dynamic thresholding, and with RBP. EGT is shown without any pruning (vanilla EGT) and with dynamic thresholding. CFR with RM is shown with partial pruning (vanilla RM) and with RBP. Dynamic thresholding on RM resulted in identical performance to vanilla RM, and is therefore not shown separately.

Once this no longer holds, skipping ceases. If we later find another T_0 that satisfies the condition above, we do another sequence of iterations where we skip traversing after a, etc.

3.4.3 Experiments

We show results for dynamic thresholding with and without RBP on Leduc Hold'em and Leduc-5.

Hedge requires the user to set the tuning parameter η_t. When proving worst-case regret bounds, the parameter is usually defined as a function of $L(I)$ for an information set I (for example, $\eta_t = \frac{\sqrt{8\ln(|A(I)|)}}{L(I)\sqrt{t}}$) [36]. However, this is overly pessimistic in practice, and better performance can be achieved with heuristics while still guaranteeing convergence, albeit at a weaker convergence bound.[4] In our experiments, we set $\eta_t = \frac{\sqrt{\ln(|A(I)|)}}{3\sqrt{\text{VAR}(I)_t}\sqrt{t}}$, where $\text{VAR}(I)_t$ is the observed variance of $v(I)$ up to iteration t, based on a heuristic by Chaudhuri et al. [38].

[4]Convergence is still guaranteed so long as $L(I)$ is replaced with a value that has a constant lower and upper bound, though the worst-case bound may be worse.

In addition to the regret-minimization algorithms, for comparison my collaborator Christian Kroer also experimented with dynamic thresholding in the leading gradient-based algorithm for finding ϵ-equilibrium in zero-sum games, the *excessive gap technique (EGT)* [72, 114], coupled with the distance-generating function from Kroer et al. [93]. It converges to an ϵ-equilibrium in two-player zero-sum games in $\mathcal{O}(\frac{1}{\epsilon})$ iterations, that is, in significantly fewer iterations than CFR which converges in $\mathcal{O}(\frac{1}{\epsilon^2})$. In this EGT variant the gradient is computed by traversing the game tree. This enables pruning and dynamic thresholding to be implemented in EGT as well. In our experiments with EGT, we stop traversing a branch in the game tree when the probability (over nature and the opposing player) of the branch falls below $\frac{c}{T}$ for various values of c. We leave the theoretical verification of this approach as future work.

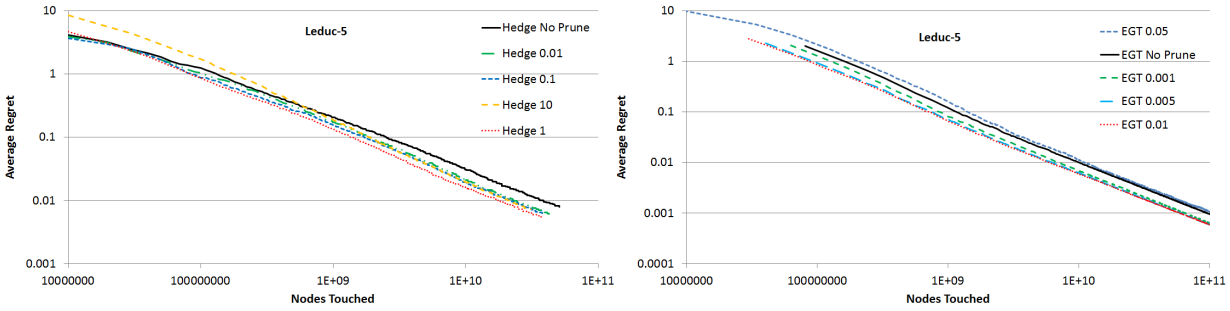


Figure 3.17: Varying the aggressiveness of dynamic thresholding.

Figure 3.16 shows the performance of dynamic thresholding on Hedge and EGT against the vanilla versions of the algorithm as well as against the benchmark algorithms CFR+ and CFR with RM. We present our results with the number of nodes touched on the x axis. Nodes touched is a hardware- and implementation-independent proxy for time. Hedge involves exponentiation when determining strategies, which takes longer than the simple floating point operations of RM. In our implementation, regret matching traverses 36% more nodes per second than Hedge. However, in large-scale multi-core implementations of CFR, memory access is the bottleneck on performance and therefore the penalty for using Hedge should not be as significant.

The two figures on the left show that dynamic thresholding benefits EGT and Hedge, and the relative benefit increases with game size. In Leduc-5, dynamic thresholding improves the performance of EGT by a factor of 2, and dynamic thresholding combined with RBP improves the performance of CFR with Hedge by a factor of 7. The graphs on the right show that, when using thresholding and RBP, Hedge outperforms RM in Leduc, but RM outperforms Hedge in Leduc-5. RM's better performance in Leduc-5 is due to more widespread pruning than Hedge.

Nevertheless, CFR+ exceeds the performance of CFR with Hedge and CFR with RM in Leduc-5. Thus, while dynamic thresholding enables widespread pruning in CFR with Hedge, more must be done to make Hedge competitive with CFR+ and DCFR.

Figure 3.17 shows the performance of EGT and Hedge with different aggressiveness of dynamic thresholding. For EGT, we threshold by $\frac{c}{T}$, where the number shown in the legend is c. For Hedge, we threshold by $\frac{d\sqrt{\ln(|A|)}}{\sqrt{2}|A|^2\sqrt{T}}$, where d is shown in the legend. The results show that for Hedge the performance is not sensitive to the parameter, and threshold only helps. For EGT, the results using $c = \{0.001, 0.005, 0.01\}$ are all similar and beneficial, while using a value of 0.05

is too aggressive, and hurts performance slightly.

3.4.4 Conclusions

Dynamic thresholding sets a threshold at every iteration such that any action with probability below the threshold is set to zero probability. This enables pruning for the first time in a wide range of algorithms. We showed that it can be applied to both Regret Matching and Hedge—regardless of whether they are used in isolation for any problem or as subroutines at each information set within CFR. We proved that the regret bound increases by only a small constant factor, and each iteration becomes faster due to enhanced pruning. Our experiments demonstrated substantial speed improvements in Hedge; the relative speedup increases with problem size. We also showed that dynamic thresholding leads to a large improvement in convergence speed when applied to a version of EGT [93], despite lacking known theoretical guarantees.

Future work could examine whether the idea of dynamic thresholding could be applied to other iterative algorithms that place at least some small positive probability on all actions [42, 121].

3.4.5 Proofs of Theoretical Results

Proof of Theorem 8

Proof. Without loss of generality, assume the minimum payoff in infoset I is zero. We use $\eta = \frac{\sqrt{2\ln(|A(I)|)}}{L(I)\sqrt{T}}$ and define $\Phi(R^t(I))$ as

$$\Phi(R^t(I)) = \frac{1}{\eta} \ln\Big(\sum_{a \in A(I)} \exp(\eta R^t(I,a)) \Big) \tag{3.22}$$

Since for all $a \in A(I)$ we know $\exp(\eta R^t(I,a)) > 0$, so

$$\max_{a \in A(I)} R^T(I,a) \leq \Phi(R^T(I)) \tag{3.23}$$

We prove inductively on t that

$$\Phi(R^t(I)) \leq \frac{\ln(|A(I)|)}{\eta} + C(L(I))^2 \eta t \tag{3.24}$$

If (3.24) holds for all t, then from (3.23) the lemma is satisfied.

For $t = 1$, dynamic thresholding produces the same strategy as vanilla Hedge, so (3.24) is trivially true. We now assume that (3.24) is true for $t-1$ and consider iteration $t > 1$. Vanilla Hedge calls for a probability vector $\sigma^t(I)$ that, if played on every iteration t, would result in (3.24) holding for T. Dynamic thresholding creates a new strategy vector $\hat{\sigma}^t(I)$. Let $\delta^t(a) = \hat{\sigma}^t(I,a) - \sigma^t(I,a)$ and $\delta^t = \max_{a \in A(I)} \delta^t(a)$.

In the worst case, all but one action is reduced to zero and the probability mass is added to the single remaining action. Thus, $|\delta^t(a)| \leq \frac{(C-1)\sqrt{\ln(|A(I)|)}}{\sqrt{2}|A(I)|\sqrt{t}}$. After playing $\hat{\sigma}^t(I,a)$ on iteration

t, we have

$$\Phi(R^t(I)) \leq \frac{1}{\eta} \ln \Big(\sum_{a \in A(I)} \exp \Big(\eta \big(R^{t-1} + r^t(I,a) \big) \Big) \Big)$$

$$\Phi(R^t(I)) \leq \frac{1}{\eta} \ln \Big(\sum_{a \in A(I)} \exp \Big(\eta \big(R^{t-1} + v^t(I,a) - v^t(I) \big) \Big) \Big)$$

$$\Phi(R^t(I)) \leq \frac{1}{\eta} \ln \Big(\sum_{a \in A(I)} \exp \Big(\eta \Big(R^{t-1} + v^t(I,a) - \sum_{a' \in A(I)} \big(\hat{\sigma}^t(I,a') v^t(I,a') \big) \Big) \Big) \Big)$$

Since $\hat{\sigma}^t(I,a') = \sigma^t(I,a) + \delta^t(a)$, we get

$$\Phi(R^t(I)) \leq \frac{1}{\eta} \ln \Big(\sum_{a \in A(I)} \exp \Big(\eta \Big(R^{t-1} + v^t(I,a) - \sum_{a' \in A(I)} \big(\sigma^t(I,a') v^t(I,a') + \delta(a') v^t(I,a') \big) \Big) \Big) \Big)$$

Since $v^t(I,a') \leq L(I)$ and $\delta^t(a') \leq \delta^t$, this becomes

$$\Phi(R^t(I)) \leq \frac{1}{\eta} \ln \Big(\exp \big(\eta \delta^t L(I) |A(I)| \big) \sum_{a \in A(I)} \exp \Big(\eta \Big(R^{t-1} + v^t(I,a) - \sum_{a' \in A(I)} \big(\sigma^t(I,a') v^t(I,a') \big) \Big) \Big) \Big)$$

$$\Phi(R^t(I)) \leq \delta^t L(I) |A(I)| + \frac{1}{\eta} \ln \Big(\sum_{a \in A(I)} \exp \Big(\eta \Big(R^{t-1} + v^t(I,a) - \sum_{a' \in A(I)} \big(\sigma^t(I,a') v^t(I,a') \big) \Big) \Big) \Big)$$

Since $v^t(I,a) - \sum_{a' \in A(I)} \big(\sigma^t(I,a') v^t(I,a')$ is the original update Hedge intended, we apply Theorem 2.1 from Cesa-Bianchi and Lugosi [36] and Lemma 1 from Brown and Sandholm [14] to get

$$\Phi(R^t(I)) \leq \delta^t L(I) |A(I)| + \Phi(R^{t-1}(I)) + \frac{(L(I))^2 \eta}{2}$$

Since $\delta^t < \frac{(C-1)L(I)\eta}{2|A(I)|}$, we get

$$\Phi(R^t(I)) \leq \Phi(R^{t-1}(I)) + C(L(I))^2 \eta$$

Substituting the bound on $\Phi(R^{t-1}(I))$ we arrive at

$$\Phi(R^t(I)) \leq \frac{\ln(|A(I)|)}{\eta} + C(L(I))^2 \eta t$$

This satisfies the inductive step. □

Proof of Theorem 9

Proof. We find it useful to define

$$\Phi(R^T(I)) = \sum_{a \in A(I)} \left(R^T(I,a)_+^2\right) \tag{3.25}$$

We prove inductively on T that

$$\Phi(R^T(I)) \leq C^2\big(L(I)\big)^2 A(I) T \tag{3.26}$$

If (3.26) holds, then $R(I) \leq CL(I)\sqrt{|A(I)|}\sqrt{T}$. On iteration 1, regret matching calls for probability $\frac{1}{|A(I)|}$ on each action, which is above the threshold. Thus, dynamic thresholding produces identical strategies as vanilla regret matching, so from Theorem 2.1 in Cesa-Bianchi and Lugosi [36], (3.26) holds.

We now assume (3.26) holds for iteration $T-1$ and consider iteration $T > 1$. Vanilla regret matching calls for a probability vector $\sigma^T(I)$ that, if played, would result in (3.26) holding for T. Dynamic thresholding creates a new strategy vector $\hat{\sigma}^T(I)$ in which $\hat{\sigma}^T(I,a) = 0$ if $\sigma^T(I,a) \leq \frac{C^2-1}{2C|A(I)|^2\sqrt{T}}$. After reducing actions to zero probability, the strategy vector is renormalized. Let $\delta(a) = \hat{\sigma}^T(I,a) - \sigma^T(I,a)$ and $\delta = \max_{a \in A(I)} \delta(a)$. In the worst case, all but one action is reduced to zero and the probability mass is added to the single remaining action. Thus, $|\delta(a)| \leq \frac{C^2-1}{2C|A(I)|\sqrt{T}}$.

After playing $\hat{\sigma}^T(I,a)$ on iteration T, we have

$$\Phi(R^T(I)) \leq \sum_{a \in A(I)} \left(R^{T-1}(I,a) + r^T(I,a)\right)_+^2$$

From Lemma 7 in Lanctot et al. [96], we get

$$\Phi(R^T(I)) \leq \left(\Phi(R^{T-1}(I)) + 2\sum_a (R_+^{T-1}(I,a) r^T(I,a)) + r^T(I,a)^2\right)$$

$$\Phi(R^T(I)) \leq \left(\Phi(R^{T-1}(I)) + 2\sum_a (R_+^{T-1}(I,a) r^T(I,a)) + (L(I))^2\right)$$

From (2.15) and (2.2),

$$r^T(I,a) = v^T(I,a) - \sum_{a' \in A(I)} \left(\hat{\sigma}^T(I,a') v^T(I,a')\right)$$

Since $\hat{\sigma}^T(I,a) = \sigma(I,a)^T + \delta(a)$, we get

$$r^T(I,a) = v^T(I,a) - \sum_{a' \in A(I)} \left(\left(\sigma^T(I,a') + \delta(a)\right) v^T(I,a')\right)$$

Regret matching satisfies the Blackwell condition [10] which, as shown in Lemma 2.1 in Cesa-Bianchi and Lugosi [36], means

$$\sum_{a \in A(I)} \left((R_+^{T-1}(I,a)\big(v^T(I,a) - \sum_{a' \in A(I)} \left(\sigma^T(I,a') v^T(I,a')\right)\right)\right) \leq 0$$

Thus, we are left with

$$\sum_{a \in A(I)} (R_+^{T-1}(I,a) r^T(I,a)) \leq |\delta| \sum_{a \in A(I)} \Big(R_+^{T-1}(I,a) \sum_{a' \in A(I)} \big(v^T(I,a') \big) \Big)$$

Since $v^T(I,a') \leq L(I)$, this leads to

$$\Phi(R^T(I)) \leq \Big(\Phi(R^{T-1}(I)) + 2|\delta| \sum_{a \in A(I)} \big(R_+^{T-1}(I,a) L(I) |A(I)| \big) + (L(I))^2 |A(I)| \Big)$$

By the induction assumption,

$$\sum_{a \in A(I)} \big(R_+^{T-1}(I,a) \big)^2 \leq C^2 (L(I))^2 |A(I)| (T-1)$$

so by Lemma 5 in Lanctot et al. [96]

$$\sum_{a \in A(I)} R_+^{T-1}(I,a) \leq C L(I) |A(I)| \sqrt{T-1}$$

This gives us

$$\Phi(R^T(I)) \leq \Big(\Phi(R^{T-1}(I)) + (L(I))^2 |A(I)| \big(2C|\delta||A(I)|\sqrt{T-1} + 1 \big) \Big)$$

Since $|\delta| < \frac{C^2-1}{2C|A(I)|\sqrt{T}}$ this becomes

$$\Phi(R^T(I)) \leq \Big(\Phi(R^{T-1}(I)) + C^2 (L(I))^2 |A(I)| \Big)$$

Substituting the bound of $\Phi(R^T(I))$ we get

$$\Phi(R^T(I)) \leq C^2 (L(I))^2 |A(I)| T$$

This satisfies the inductive step. □

3.5 Best-Response Pruning (BRP)

Both computation time and storage space are difficult challenges when solving large imperfect-information games. For example, solving limit Texas hold'em [12] required nearly 8 million core hours and a complex, domain-specific streaming compression algorithm to store the 262 TiB of uncompressed data in only 10.9 TiB. This data had to be repeatedly decompressed from disk into memory and then compressed back to disk in order to run CFR+ [150].

As discussed in Section 3.3, regret-based pruning can be applied to speed up the traversal of the game tree in CFR. However, regret-based pruning does not reduce the space needed to solve a game.

In this section we discuss **Best-Response Pruning** (BRP), a form of pruning for iterative algorithms such as CFR in large imperfect-information games. BRP leverages the fact that in iterative algorithms we are typically interested in performance against the opponent's *average* strategy over all iterations, and that the opponent's average strategy cannot change faster than a rate of $\frac{1}{t}$, where t is the number of iterations conducted so far. Thus, if part-way through a run one of our actions has done extremely poorly relative to other available actions against the opponent's average strategy, then after just a few more iterations the opponent's average strategy cannot change sufficiently for the poorly-performing action to now be doing well against the opponent's updated average strategy. In fact, we can bound how much an action's performance can improve over any number of iterations against the opponent's average strategy. So long as the upper bound on that performance is still not competitive with the other actions, then we can safely ignore the poorly-performing action.

BRP provably reduces the computation time needed to solve imperfect-information games. Additionally, a primary advantage of BRP is that in addition to faster convergence, it also reduces the *space* needed over time. Specifically, once pruning begins on a branch, BRP discards the memory allocated on that branch and does not reallocate the memory until pruning ends and the branch cannot immediately be pruned again. In Section 3.5.2, we prove that after enough iterations of CFR are completed, space for certain pruned branches will *never* need to be allocated again. Specifically, we prove that when using BRP it is asymptotically only necessary to store data for parts of the game that are reached with positive probability in a best response to a Nash equilibrium. This is extremely advantageous when solving large imperfect-information games, which are often constrained by space and in which the set of best response actions may be orders of magnitude smaller than the size of the game [132].

While BRP still requires enough memory to store the entire game in the early iterations, Section 3.2 describes a warm starting technique that can skip the early iterations of CFR, and possibly other iterative algorithms, by first solving a low-memory abstraction of the game and then using its solution to warm start CFR in the full game [18]. BRP's reduction in space is also helpful to the Simultaneous Abstraction and Equilibrium Finding (SAEF) algorithm [16], described in Section 4.3, which starts CFR with a small abstraction of the game and progressively expands the abstraction while also solving the game. SAEF's space requirements increase the longer the algorithm runs, and may eventually exceed the constraints of a system. BRP can counter this increase in space by eliminating the need to store suboptimal paths of the game tree.

BRP provably results in CFR converging faster because suboptimal paths in the game tree will only need to be traversed $\mathcal{O}\big(\ln(T)\big)$ times over T iterations. We also prove that BRP uses asymptotically less space, which is a major advantage over regret-based pruning. Additionally, BRP easily generalizes to iterative algorithms beyond CFR such as Fictitious Play [70].

3.5.1 Description of Best-Response Pruning

This section describes the behavior of BRP. We focus on the case where BRP is applied to CFR.

We begin by defining a **counterfactual best response (CBR)**, which is stronger than a best response. Given a partial strategy profile σ_{-i}, a counterfactual best response to it, $\phi(\sigma_{-i})$, is a best response to σ_{-i} with the additional condition that for every player i infoset I, $v_i^{\langle\phi(\sigma_{-i}),\sigma_{-i}\rangle}(I) =$

$\max_{a \in A(I)} v_i^{\langle \phi(\sigma_{-i}), \sigma_{-i} \rangle}(I, a)$. In other words, a CBR must always choose an action resulting in maximum expected value in every infoset, regardless of whether that infoset is actually reached as part of a best response. In a regular best response, an infoset need only choose a maximum-EV action if that infoset is reached with positive probability (i.e., if all earlier actions lead to it with positive probability).

BRP begins pruning an action a in an infoset I whenever playing perfectly beyond that action against the opponent's average strategy (that is, playing a CBR) still does worse than what has been achieved in the iterations played so far. That is, the condition for pruning to begin is that $T v_i^{\langle \phi(\bar{\sigma}^T_{-i}), \bar{\sigma}^T_{-i} \rangle}(I, a) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I)$. Pruning continues for the minimum number of iterations it could take for the opponent's average strategy to change sufficiently such that the pruning starting condition no longer holds. Since $v_i^{\sigma^t}(I, a) - v_i^{\sigma^t}(I) \leq L(I)$ for any iteration t, pruning continues for at least $\frac{\sum_{t=1}^{T} v^{\sigma^t}(I) - T v_i^{\langle \phi(\bar{\sigma}^T_{-i}), \bar{\sigma}^T_{-i} \rangle}(I,a)}{L(I)}$ iterations. When pruning ends, BRP calculates a CBR in the pruned branch against the opponent's average strategy over all iterations played so far, and sets regret in the pruned branch as if that CBR strategy were played on *every* iteration played in the game so far—even those that were played before pruning began.

While using a CBR works correctly when applying BRP to CFR, it is also sound to choose a strategy that is *almost* a CBR, as long as that strategy ensures

$$\sum_{a \in A(I)} \left(R^T_+(I, a)\right)^2 \leq \left(L(I)\right)^2 |A(I)| T$$

for every infoset I. In practice, this means that the strategy is close to a CBR, and approaches a CBR as $T \to \infty$. However, in practice CFR converges much faster than the theoretical bound, so while a near-CBR rather than an exact CBR may allow for slightly longer pruning in theory, it may actually result in worse performance in practice. For this reason, in this section we discuss the theory and algorithms in terms of exact CBRs. Nevertheless, future research on deciding on a near-CBR may in the future lead to near-CBRs resulting in better performance in practice. For a description of how BRP can be conducted with near-CBRs, see Brown and Sandholm [19].

We define the **counterfactual best response value** as $\psi^{\sigma_{-i}}(I, a) = \max_{\sigma'_i} v_i^{\langle \sigma'_i, \sigma_{-i} \rangle}(I, a)$ and $\psi^{\sigma_{-i}}(I) = \max_{\sigma'_i} v_i^{\langle \sigma'_i, \sigma_{-i} \rangle}(I)$ where player i acts at I.

When applying BRP to CFR, an action is pruned only if it would still have negative regret had a CBR against the opponent's average strategy been played on every iteration. Specifically, on iteration T of CFR with RM, if

$$T \psi^{\bar{\sigma}^T_{-i}}(I, a) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I) \tag{3.27}$$

then $D(I, a)$ can be pruned for

$$T' = \frac{\sum_{t=1}^{T} v_i^{\sigma^t}(I) - T \psi^{\bar{\sigma}^T_{-i}}(I, a)}{L(I)} \tag{3.28}$$

iterations. After those T' iterations are over, we calculate a CBR in $D(I, a)$ to the opponent's average strategy and set regret as if that CBR had been played on every iteration. Specifically,

for each $t \leq T + T'$ we set[5] $v_i^{\sigma^t}(I,a) = \psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)$ so that

$$R^{T+T'}(I,a) = \big(T+T'\big)\big(\psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)\big) - \sum_{t=1}^{T+T'} v_i^{\sigma^t}(I) \tag{3.29}$$

and for every information set $I' \in D(I,a)$ we set $v_i^{\sigma^t}(I',a') = \psi^{\bar{\sigma}_{-i}^{T+T'}}(I',a')$ and $v_i^{\sigma^t}(I') = \psi^{\bar{\sigma}_{-i}^{T+T'}}(I')$ so that

$$R^{T+T'}(I',a') = \big(T+T'\big)\big(\psi^{\bar{\sigma}_{-i}^{T+T'}}(I',a') - \psi^{\bar{\sigma}_{-i}^{T+T'}}(I')\big) \tag{3.30}$$

Theorem 10 proves that if (3.27) holds for some action, then the action can be pruned for T' iterations, where T' is defined in (3.28). The proof for Theorem 10 draws from the theory for warm starting CFR using only a strategy profile [18], which was described in Section 3.2. Essentially, we warm start regrets in the pruned branch using only the average strategy of the opponent and knowledge of T.

Theorem 10. *Assume T iterations of CFR with RM have been played in a two-player zero-sum game and assume $T\psi^{\bar{\sigma}_{-i}^{T}}(I,a) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I)$ where $P(I) = i$. Let*

$$T' = \Big\lfloor \frac{\sum_{t=1}^{T} v_i^{\sigma^t}(I) - T\psi^{\bar{\sigma}_{-i}^{T}}(I,a)}{L(I)} \Big\rfloor$$

If both players play according to CFR with RM for the next T' iterations in all information sets $I'' \notin D(I,a)$ except that $\sigma(I,a)$ is set to zero and $\sigma(I)$ is renormalized, then

$$(T+T')\big(\psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)\big) \leq \sum_{t=1}^{T+T'} v_i^{\sigma^t}(I)$$

Moreover, if one then sets $v_i^{\sigma^t}(I,a) = \psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)$ for each $t \leq T+T'$ and $v_i^{\sigma^t}(I',a') = \psi^{\bar{\sigma}_{-i}^{T+T'}}(I',a')$ for each $I' \in D(I,a)$, then after T'' additional iterations of CFR with RM, the bound on exploitability of $\bar{\sigma}^{T+T'+T''}$ is no worse than having played $T+T'+T''$ iterations of CFR with RM without BRP.

In practice, rather than check whether (3.27) is met for an action on every iteration, one could only check actions that have very negative regret, and do a check only once every several iterations. This would still be safe and would save some computational cost of the checks, but would lead to less pruning.

Similar to regret-based pruning, the duration of pruning in BRP can be increased by giving up knowledge beforehand of exactly how many iterations can be skipped. From (2.14) and (2.13) we see that if $\pi_{-i}^{\sigma^t}(I)$ is very low, then (3.27) would continue to hold for more iterations than (3.28) guarantees. Specifically, we can prune $D(I,a)$ from iteration t_0 until iteration t_1 as long as

$$t_0\big(\psi^{\bar{\sigma}_{-i}^{t_0},t_0}(I,a)\big) + \sum_{t=t_0+1}^{t_1} \pi_{-i}^{\sigma^t}(I)U(I,a) \leq \sum_{t=1}^{t_1} v_i^{\sigma^t}(I) \tag{3.31}$$

where $U(I,a)$ is the maximum payoff achievable for taking action a in infoset I.

[5] In practice, only the sums $\sum_{t=1}^{T} v_i^{\sigma^t}(I)$ and either $\sum_{t=1}^{T} v_i^{\sigma^t}(I,a)$ or $R^T(I,a)$ are stored.

3.5.2 Best-Response Pruning Requires Less Space

A key advantage of BRP over regret-based pruning is that setting the new regrets according to (3.29) and (3.30) requires no knowledge of what the regrets were before pruning began. Thus, once pruning begins, all the regrets in $D(I, a)$ can be discarded and the space that was allocated to storing the regret can be freed. That space need only be re-allocated once (3.31) ceases to hold *and* we cannot immediately begin pruning again (that is, (3.27) does not hold). Theorem 11 proves that for any information set I and action $a \in A(I)$ that is not part of a best response to a Nash equilibrium, there is an iteration $T_{I,a}$ such that for all $T \geq T_{I,a}$, action a in information set I (and its descendants) can be pruned.[6] Thus, once this $T_{I,a}$ is reached, it will never be necessary to allocate space for regret in $D(I, a)$ again.

Theorem 11. *In a two-player zero-sum game, if for every Nash equilibrium σ^*, $\psi^{\sigma^*_{-i}}(I, a) < \psi^{\sigma^*_{-i}}(I)$, then there exists a $T_{I,a}$ and $\delta_{I,a} > 0$ such that after $T \geq T_{I,a}$ iterations of CFR, $\psi^{\bar{\sigma}^T_{-i}}(I, a) - \frac{\sum_{t=1}^{T} v_i^{\sigma^t}(I)}{T} \leq -\delta_{I,a}$.*

While such a constant $T_{I,a}$ exists for any suboptimal action, BRP cannot determine whether or when $T_{I,a}$ is reached. Thus, it is still necessary to check whether (3.27) is satisfied whenever (3.31) no longer holds, and to recalculate how much longer $D(I, a)$ can safely be pruned. This requires the algorithm to periodically calculate a best response in $D(I, a)$. However, this best response calculation does not require knowledge of regret in $D(I, a)$, so it is still never necessary to store regret after iteration $T_{I,a}$ is reached.

While it is possible to discard regrets in $D(I, a)$ without penalty once pruning begins, regret is only half the space requirement of CFR. Every information set I also stores a sum of the strategies played $\sum_{t=1}^{T} \left(\pi_i^{\sigma^t}(I)\sigma^t(I)\right)$ which is normalized once CFR ends in order to calculate $\bar{\sigma}^T(I)$. Fortunately, if action a in information set I is pruned for long enough, then the stored cumulative strategy in $D(I, a)$ can also be discarded at the cost of a small increase in the distance of the final average strategy from a Nash equilibrium. This is similar to dynamic thresholding, discussed in Section 3.4, except we need only apply it to the stored *average* strategy rather than the strategy played on every iteration. (Of course, one could also simply use dynamic thresholding to set small probabilities to zero on every individual iteration.)

Specifically, if $\pi_i^{\bar{\sigma}^T}(I, a) \leq \frac{C}{\sqrt{T}}$, where C is some constant, then setting $\bar{\sigma}^T(I, a) = 0$ and renormalizing $\bar{\sigma}^T(I)$, and setting $\bar{\sigma}^T(I', a') = 0$ for $I' \in D(I, a)$, can result in at most $\frac{C|I|L}{\sqrt{T}}$ higher exploitability for the whole strategy $\bar{\sigma}^T$. Since CFR only guarantees that $\bar{\sigma}^T$ is a $\frac{2|\mathcal{I}|L\sqrt{|A|}}{\sqrt{T}}$-Nash equilibrium anyway, $\frac{C|I|L}{\sqrt{T}}$ is only a constant factor of the bound. If an action is pruned from T' to T, then $\sum_{t=1}^{T} \left(\pi_i^{\sigma^t}(I)\sigma^t(I, a)\right) \leq \frac{T'}{T}$. Thus, if an action is pruned for long enough, then eventually $\sum_{t=1}^{T} \left(\pi_i^{\sigma^t}(I)\sigma^t(I, a)\right) \leq \frac{C}{\sqrt{T}}$ for any C, so $\sum_{t=1}^{T} \left(\pi_i^{\sigma^t}(I)\sigma^t(I, a)\right)$ could be set to zero (as well as all descendants of $I \cdot a$), while suffering at most a constant factor increase in exploitability. As more iterations are played, this penalty will continue to decrease and eventually be negligible. The constant C can be set by the user: a higher C allows the average strategy to be discarded sooner, while a lower C reduces the potential penalty in exploitability.

[6] If CFR converges to a *particular* Nash equilibrium, then this condition could be broadened to any information set I and action $a \in A(I)$ that is not a best response to *that particular* Nash equilibrium. While empirically CFR does appear to always converge to a particular Nash equilibrium, there is no known proof that it always does so.

We define $\mathcal{I}_S$ as the set of information sets that are not guaranteed to be asymptotically pruned by Theorem 11. Specifically, $I \in \mathcal{I}_S$ if $I \notin D(I', a')$ for some I' and $a' \in A(I')$ such that for every opponent Nash equilibrium strategy σ^*_{-i}, $\psi^{\sigma^*_{-i}}(I', a') < \psi^{\sigma^*_{-i}}(I')$. Theorem 11 implies the following.

Corollary 2. *In a two-player zero-sum game with some threshold on the average strategy $\frac{C}{\sqrt{T}}$ for $C > 0$, after a finite number of iterations CFR with BRP requires only $\mathcal{O}\big(|\mathcal{I}_S||A|\big)$ space.*

Using a threshold of $\frac{C}{T}$ rather than $\frac{C}{\sqrt{T}}$ does not change the theoretical properties of the corollary, and is likely more appropriate when using variants of CFR that empirically converge faster than $\mathcal{O}(\frac{1}{\epsilon^2})$ such as DCFR or Linear CFR (both of which are discussed in Section 3.1), or variants of CFR that provably converge faster than $\mathcal{O}(\frac{1}{\epsilon^2})$ [44]. Additionally, if BRP can be extended to first-order methods that converge to an ϵ-Nash equilibrium in $\mathcal{O}(\frac{1}{\epsilon})$ iterations rather than $\mathcal{O}(\frac{1}{\epsilon^2})$ iterations, such as the Excessive Gap Technique [72, 93], then a threshold of $\frac{C}{T}$ may be more appropriate when those algorithms are used.

3.5.3 Best-Response Pruning Converges Faster

We now prove that BRP in CFR speeds up convergence to an ϵ-Nash equilibrium. Section 3.5.1 proved that CFR with BRP converges in the same number of iterations as CFR alone. In this section, we prove that BRP allows each iteration to be traversed more quickly. Specifically, if an action $a \in A(I)$ is not a CBR to a Nash equilibrium, then $D(I, a)$ need only be traversed $\mathcal{O}(\ln(T))$ times over T iterations. Intuitively, as both players converge to a Nash equilibrium, actions that are not a counterfactual best response will eventually do worse than actions that are, so those suboptimal actions will accumulate increasing amounts of negative regret. This negative regret allows the action to be safely pruned for increasingly longer periods of time.

Specifically, let $S \subseteq H$ be the set of histories where $h \cdot a \in S$ if $h \in S$ and action a is part of some CBR to some Nash equilibrium. Formally, S contains $\emptyset$ and every history $h \cdot a$ such that $h \in S$ and $\psi^{\sigma^*_{-i}}(I, a) = \psi^{\sigma^*_{-i}}(I)$ for some Nash equilibrium σ^*.

Theorem 12. *In a two-player zero-sum game, if both players choose strategies according to CFR with BRP, then conducting T iterations requires only $\mathcal{O}\big(|S|T + |H| \ln(T)\big)$ nodes to be traversed.*

The definition of S uses properties of the Nash equilibria of the game, and an action $a \in A(I)$ not in S is only guaranteed to be pruned by BRP after some $T_{I,a}$ is reached, which also depends on the Nash equilibria of the game. Since CFR converges to only an ϵ-Nash equilibrium, CFR cannot determine with certainty which nodes are in S or when $T_{I,a}$ is reached. Nevertheless, both S and $T_{I,a}$ are fixed properties of the game.

3.5.4 Experiments

We compare the convergence speed of BRP to regret-based pruning, to only partial pruning, and to no pruning at all. We also show that BRP uses less space as as more iterations are conducted, unlike regret-based pruning and partial pruning. The experiments are conducted on Leduc hold'em and Leduc-5 (described in Section 2.4.1).

Nodes touched is a hardware and implementation-independent proxy for time which we use to measure performance of the various algorithms. Overhead costs are counted in nodes touched. As described in Section 3.3, since regret-based pruning can only prune negative-regret actions, regret-based pruning modifies the definition of CFR+ so that regret can be negative but immediately jumps up to zero as soon as regret increases. BRP does not require this modification. Still, BRP also modifies the behavior of CFR+ because without pruning, CFR+ would put positive probability on an action as soon as its regret increases, while BRP waits until pruning is over. This is not, by itself, a problem. However, CFR+'s linear weighting of the average strategy is only guaranteed to converge to a Nash equilibrium if pruning does not occur. While both regret-based pruning and BRP do well empirically with CFR+, the convergence is noisy. This noise can be reduced by using the lowest-exploitability average strategy profile found so far, which we do in the experiments.[7] BRP does not do as well empirically with the linear-averaging component of CFR+. Thus, for BRP we only measure performance using RM+ with CFR, which is the same as CFR+ but without linear averaging. CFR+ with and without linear averaging has the same theoretical performance as CFR, but CFR+ does better empirically (particularly with linear averaging).

Figure 3.18 and Figure 3.19 show the reduction in space needed to store the average strategy and regrets for BRP—for various values of the constant threshold C, where an action's probability is set to zero if it is reached with probability less than $\frac{C}{\sqrt{T}}$ in the average strategy, as we explained in Section 3.5.2. In both games, a threshold between 0.01 and 0.1 performed well in both space and number of iterations, with the lower thresholds converging somewhat faster and the higher thresholds reducing space faster. We also tested thresholds below 0.01, but the speed of convergence was essentially the same as when using 0.01. In Leduc, all variants resulted in a quick drop-off in space to about half the initial amount. In Leduc-5, a threshold of 0.1 resulted in about a factor of 7 reduction for both CFR with RM and CFR with RM+. This space reduction factor appears to continue to increase.

Figure 3.20 and Figure 3.21 compare the convergence rates of BRP, regret-based pruning, and only partial pruning for CFR with RM, CFR with RM+, and CFR+. In Leduc, BRP and regret-based pruning perform comparably when added to CFR. regret-based pruning with CFR+ does significantly better, while BRP with CFR using RM+ sees no improvement over BRP with CFR. In Leduc-5, which is a far larger game, BRP outperforms regret-based pruning by a factor of 2 when added to CFR. BRP with CFR using RM+ also performs comparably to regret-based pruning with CFR+, while retaining theoretical guarantees and not suffering from noisy convergence.

3.5.5 Conclusions

BRP is a form of pruning that provably reduces both the space needed to solve an imperfect-information game and the time needed to reach an ϵ-Nash equilibrium. This addresses both of the major bottlenecks in solving large imperfect-information games: time and space. Experimentally, BRP reduced the space needed to solve a game by a factor of 7, with the reduction factor

[7]Exploitability is no harder to compute than one iteration of CFR or CFR+. Snapshots are not plotted at every iteration but only after every 10,000,000 nodes touched—except for the first few snapshots.

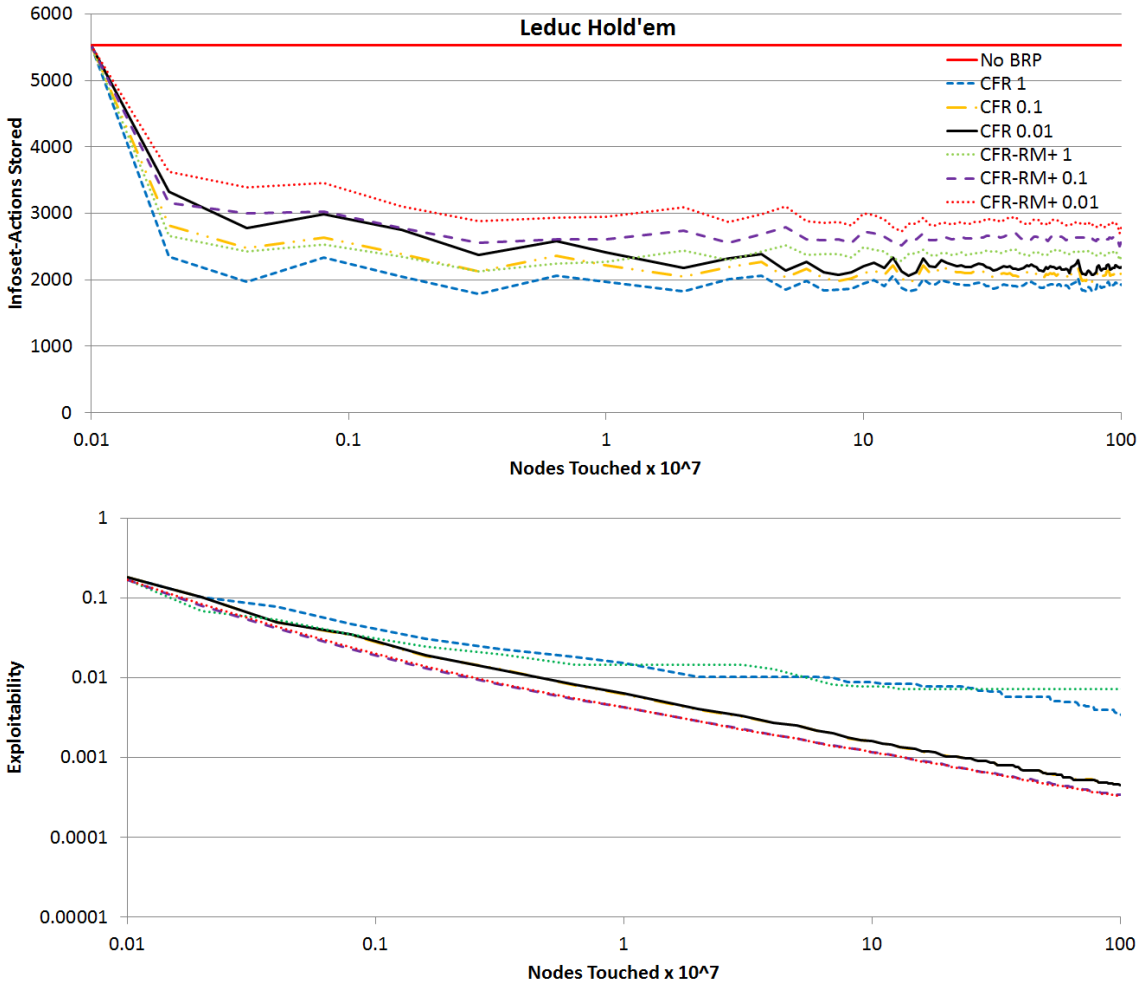


Figure 3.18: Convergence and space required for CFR using RM and RM+ with best-response pruning in Leduc hold'em. The y-axis on the top graph is linear scale.

increasing with game size. While the early iterations may still be slow and require the same amount of space as CFR without BRP, these early iterations can be skipped by warm starting CFR with a rough approximation of an equilibrium, as described in Section 3.2.

As is the case with regret-based pruning, the faster convergence of BRP is reduced when using a vector-based implementation of CFR [79]. However, the reduction in space requirements is retained even if a vector-based implementation of CFR is used.

This section focused on the theory of BRP when applied to CFR, which is the most popular algorithm for solving imperfect-information games, but BRP can also be applied to Fictitious Play [70] and likely extends to other iterative algorithms such as the Excessive Gap Technique [72].

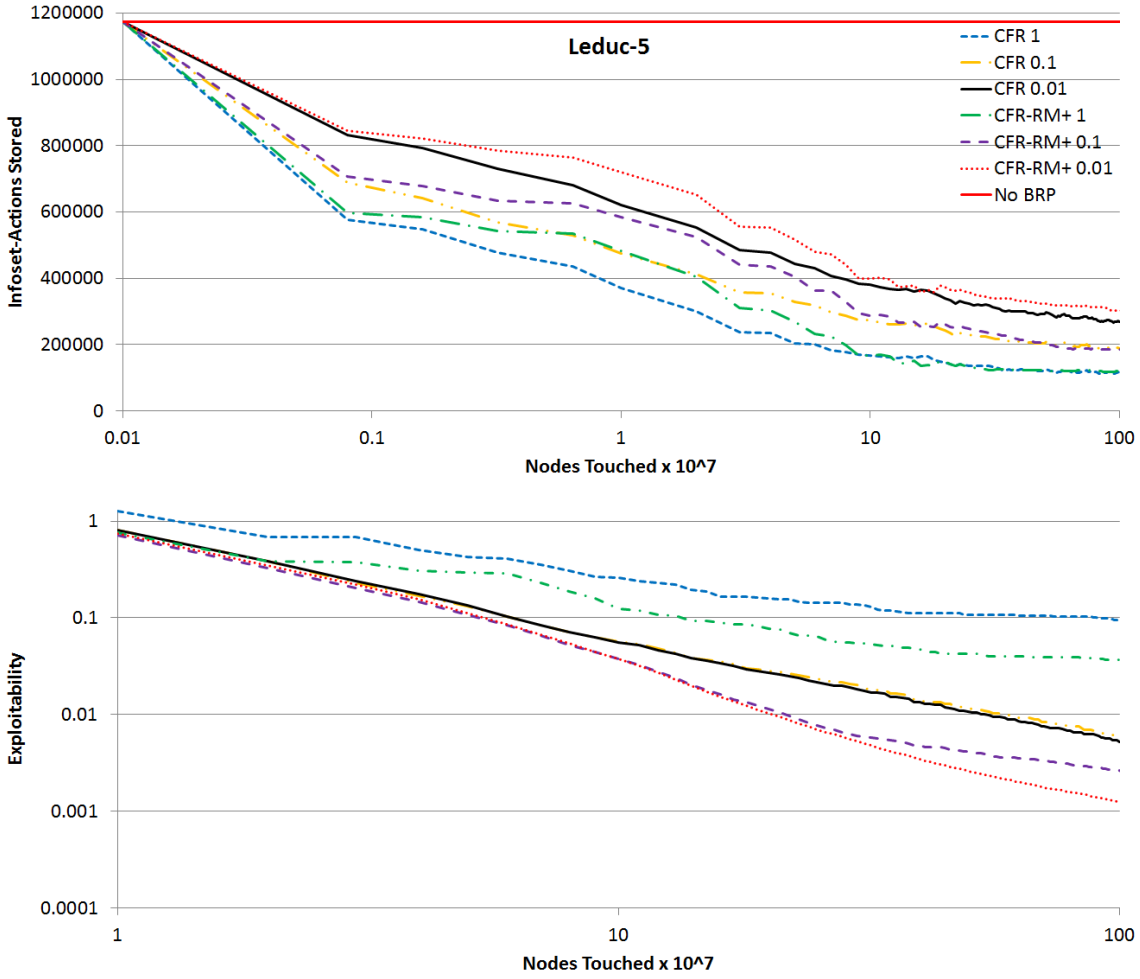


Figure 3.19: Convergence and space required for CFR using RM and RM+ with best-response pruning in Leduc-5. The y-axis on the top graph is linear scale.

3.5.6 Proofs of Theoretical Results

Lemma 9

Lemma 9 proves that if (3.27) is satisfied for some action $a \in A(I)$ on iteration T, then the value of action a and all its descendants on every iteration played so far can be set to the counterfactual best response value. The proof for Lemma 9 draws from the theory for warm starting CFR using only a strategy profile, discussed in Section 3.2.

Lemma 9. *Assume T iterations of CFR with RM have been played in a two-player zero-sum game. If $T\left(\psi^{\bar{\sigma}^T_{-i}}(I,a)\right) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I)$ and one sets $v_i^{\sigma^t}(I,a) = \psi^{\bar{\sigma}^T_{-i}}(I,a)$ for each $t \leq T$ and for each $I' \in D(I,a)$ sets $v_i^{\sigma^t}(I',a') = \psi^{\bar{\sigma}^T_{-i}}(I',a')$ and $v_i^{\sigma^t}(I') = \psi^{\bar{\sigma}^T_{-i}}(I')$ then after T' additional iterations of CFR with RM, the bound on exploitability of $\bar{\sigma}^{T+T'}$ is no worse than having played $T+T'$ iterations of CFR with RM unaltered.*

Proof. The proof builds upon Theorem 6 in Section 3.2. Assume $T\left(\psi^{\bar{\sigma}^T_{-i}}(I,a)\right) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I)$.

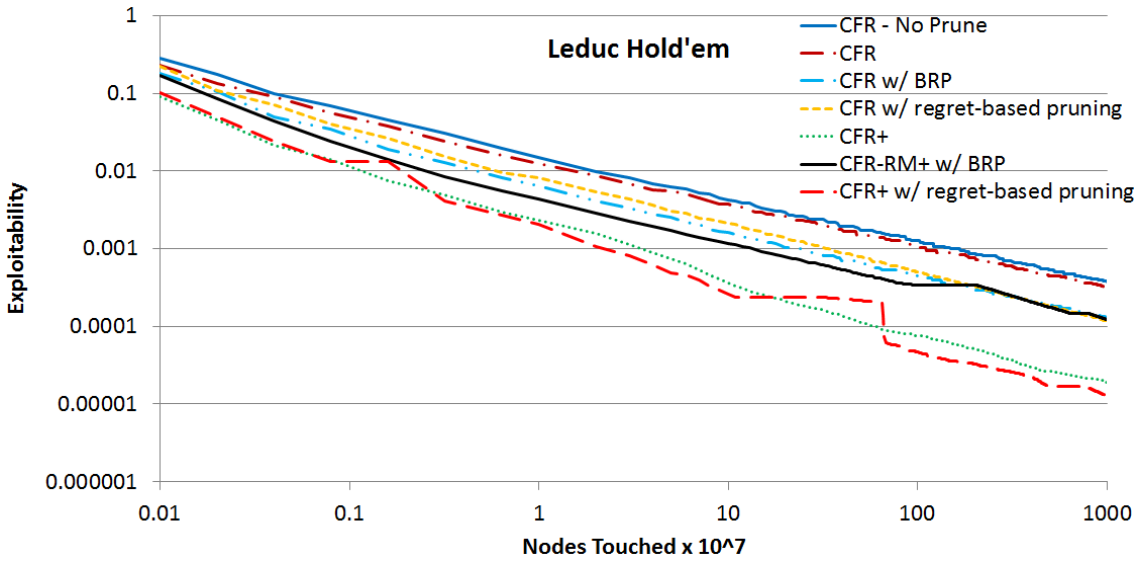


Figure 3.20: Convergence for partial pruning, regret-based pruning, and best-response pruning in Leduc. "CFR - No Prune" is CFR without any pruning.

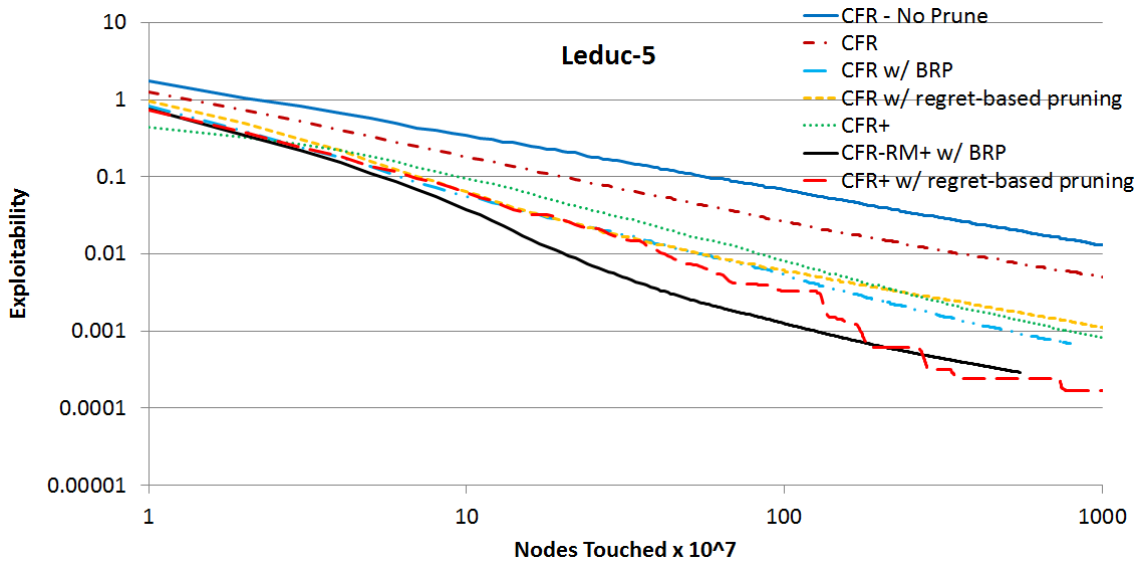


Figure 3.21: Convergence for partial pruning, regret-based pruning, and best-response pruning in Leduc-5. "CFR - No Prune" is CFR without any pruning.

We wish to warm start to T iterations. For each $I' \in D(I,a)$ set $v_i^{\sigma^t}(I',a') = \psi^{\bar{\sigma}^T_{-i}}(I',a')$ and $v_i^{\sigma^t}(I') = \psi^{\bar{\sigma}^T_{-i}}(I')$ and set $v_i^{\sigma^t}(I,a) = \psi^{\bar{\sigma}^T_{-i}}(I,a)$ for all $t \leq T$. For every other action, leave regret unchanged. For each $I' \in D(I,a)$ we know by construction that $\Phi(R^T(I')) \leq 0$. By assumption, $T\left(\psi^{\bar{\sigma}^T_{-i}}(I,a)\right) \leq \sum_{t=1}^{T} v_i^{\sigma^t}(I)$, so $R^T(I,a) \leq 0$ and therefore $\Phi(R^T(I))$ is unchanged. Finally, since the T iterations were played according to CFR with RM and regret is unchanged for every other information set I'', so the conditions for Theorem 6 in Section 3.2 hold for every information set, and therefore we can warm start to T iterations of CFR with RM

with no penalty to the convergence bound. □

Proof of Theorem 10

Proof. From Lemma 9 we can immediately set regret for $a \in A(I)$ to $v_i^{\sigma^t}(I,a) = \psi^{\bar{\sigma}_{-i}^T}(I,a)$. By construction of T', $R^t(I,a)$ is guaranteed to be nonpositive for $T \le t \le T+T'$ and therefore $\sigma^t(I,a) = 0$. Thus, $\bar{\sigma}_i^{T+T'}(I')$ for $I' \in D(I,a)$ is identical regardless of what is played in $D(I,a)$ during $T \le t \le T+T'$.

Since $(T+T')\big(\psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)\big) \le T\big(\psi^{\bar{\sigma}_{-i}^T}(I,a)\big) + T'\big(U(I,a)\big)$ and since $\sum_{t=1}^{T+T'} v_i^{\sigma^t}(I) \ge \sum_{t=1}^{T} v_i^{\sigma^t}(I) + T'\big(L(I)\big)$, so by the definition of T', $(T+T')\big(\psi^{\bar{\sigma}_{-i}^{T+T'}}(I,a)\big) \le \sum_{t=1}^{T+T'} v_i^{\sigma^t}(I)$. So if regrets in $D(I,a)$ and $R^{T+T'}(I,a)$ are set according to Lemma 9, then after T'' additional iterations of CFR with RM, the bound on exploitability of $\bar{\sigma}^{T+T'+T''}$ is no worse than having played $T+T'+T''$ iterations of CFR with RM from scratch. □

Proof of Theorem 11

Proof. Consider an information set I and action $a \in A(I)$ where for every opponent Nash equilibrium strategy σ_{-i}^*, $\psi^{\sigma_{-i}^*}(I,a) < \psi^{\sigma_{-i}^*}(I)$ where $i = P(I)$. Let $\delta = \min_{\sigma_{-i} \in \Sigma^*} \big(\psi^{\sigma_{-i}}(I) - \psi^{\sigma_{-i}}(I,a)\big)$ where Σ^* is the set of Nash equilibria. Let

$$\sigma'_{-i} = \operatorname*{argmax}_{\sigma_{-i} \in \Sigma_{-i} | \psi^{\sigma_{-i}}(I) - \psi^{\sigma_{-i}}(I,a) \le \frac{3\delta}{4}} u_{-i}(\sigma_{-i}, BR(\sigma_{-i}))$$

Since σ'_{-i} is not a Nash equilibrium strategy and CFR converges to a Nash equilibrium strategy for both players, so there exists a T_δ such that for all $T \ge T_\delta$, $\psi^{\bar{\sigma}_{-i}^T}(I) - \psi^{\bar{\sigma}_{-i}^T}(I,a) > \frac{3\delta}{4}$. Let $T'_{I,a} = \frac{4|\mathcal{I}|^2 L^2 |A|}{\delta^2}$. For $T \ge T'_{I,a}$ since $R_i^T \le \sum_{I \in \mathcal{I}_i} R^T(I)$, so $\psi^{\bar{\sigma}_{-i}^T}(I) - \sum_{t=1}^T v_i^{\sigma^t}(I) \le \frac{\delta}{2}$. Let $T_{I,a} = \max(T'_{I,a}, T_\delta)$ and $\delta_{I,a} = \frac{\delta}{4}$. Then for $T \ge T_{I,a}$, $\psi^{\bar{\sigma}_{-i}^T}(I,a) - \frac{\sum_{t=1}^T v_i^{\sigma^t}(I)}{T} \le -\delta_{I,a}$. □

Proof of Corollary 2

Proof. Let $I \notin \mathcal{I}_S$. Then $I \in D(I',a')$ for some I' and $a' \in A(I')$ such that for every opponent Nash equilibrium strategy σ_{-i}^*, $\psi^{\sigma_{-i}^*}(I',a') < \psi^{\sigma_{-i}^*}(I')$ where $P(I') = i$. Applying Theorem 11, this means there exists a $T_{I',a'}$ and $\delta_{I',a'} > 0$ such that for $T \ge T_{I',a'}$, $\psi^{\bar{\sigma}_{-i}^T}(I',a') - \frac{\sum_{t=1}^T v_i^{\sigma^t}(I')}{T} \le -\delta_{I',a'}$. So (3.27) always applies for $T \ge T_{I',a'}$ for I' and a' and I will always be pruned. Since (3.30) does not require knowledge of regret, it need not be stored for I.

Since $D(I',a')$ will always be pruned for $T \ge T_{I',a'}$, so for any $T \ge \frac{(T_{I',a'})^2}{C^2}$ iterations for some constant $C > 0$, $\pi_i^{\bar{\sigma}^T}(I) \le \frac{C}{\sqrt{T}}$, which satisfies the threshold of the average strategy. Thus, the average strategy in $D(I,a)$ can be discarded. □

Lemma 10

Lemma 10. *If for all $T \ge T'$ iterations of CFR with RBP, $T\big(\psi^{\bar{\sigma}_{-i}^T}(I,a)\big) - \sum_{t=1}^T v_i^{\sigma^t}(I) \le -CT$ for some $C > 0$, then any history h' such that $h \cdot a \sqsubseteq h'$ for some $h \in I$ need only be traversed at most $\mathcal{O}\big(\ln(T)\big)$ times.*

Proof. Let $a \in A(I)$ be an action such that for all $T \geq T'$, $T\big(\psi^{\bar{\sigma}^T_{-i}}(I,a)\big) - \sum_{t=1}^{T} v_i^{\sigma^t}(I) \leq -CT$ for some $C > 0$. Then from Theorem 10, $D(I,a)$ can be pruned for $m \geq \lfloor \frac{CT}{L(I)} \rfloor$ iterations on iteration T. Thus, over iterations $T \leq t \leq T + m$, only a constant number of traversals must be done. So each iteration requires only $\frac{K}{m}$ work when amortized, where K is a constant. Since C and $L(I)$ are constants, so on each iteration $t \geq T'$, only an average of $\frac{K}{t}$ traversals of $D(I,a)$ is required. Summing over all $t \leq T$ for $T \geq T'$, and recognizing that T' is a constant, we get that action a is only taken $\mathcal{O}\big(\ln(T)\big)$ over T iterations. Thus, any history h' such that $h \cdot a \sqsubseteq h'$ for some $h \in I$ need only be traversed at most $\mathcal{O}\big(\ln(T)\big)$ times. □

Proof of Theorem 12

Proof. Consider an $h^* \notin S$. Then there exists some $h \cdot a \sqsubseteq h^*$ such that $h \in S$ but $h \cdot a \notin S$. Let $I = I(h)$ and $i = P(I)$. Since $h \cdot a \notin S$ but $h \in S$, so for every Nash equilibrium σ^*, $\psi^{\sigma^*_{-i}}(I,a) < \psi^{\sigma^*_{-i}}(I)$. From Theorem 11, there exists a $T_{I,a}$ and $\delta_{I,a} > 0$ such that after $T \geq T_{I,a}$ iterations of CFR, $\psi^{\bar{\sigma}^T_{-i}}(I,a) - \frac{\sum_{t=1}^{T} v_i^{\sigma^t(I)}}{T} \leq -\delta_{I,a}$. Thus from Lemma 10, h^* need only be traversed at most $\mathcal{O}\big(\ln(T)\big)$ times. □

Chapter 4

Automated Abstraction for Imperfect-Information Games

Iterative algorithms such as CFR scale to much larger games than previous techniques. Linear programming was used to solve Rhode Island hold'em [56], a game of about 10^8 infosets, but has not been successful for games much larger than that. In contrast, CFR is able to scale to games such as HULH (described in Section 2.4.2), which has about 10^{13} infosets (after removing symmetries).

However, CFR alone cannot scale to games much larger than HULH because CFR stores regrets for each infoset it encounters. Even scaling to HULH required a complex streaming compression algorithm to store as many of the regrets as possible on disk rather than in memory [150]. Best response pruning, discussed in Section 3.5, helps reduce this storage, but even pruning would not make a game like HUNL (described in Section 2.4.3), which has about 10^{161} infosets, tractable for CFR.

In order to make such large games tractable, one can use **abstraction** to bucket similar infosets together. Most research on imperfect-information games consider two forms of abstraction separately.

The first is **information abstraction**, which buckets together "similar" infosets that may differ only in the outcomes of chance nodes (e.g., different poker hands). Information abstraction has been very effective in poker AI's and was used in Libratus and Pluribus. However, information abstraction tends to be very domain-specific, and several papers have been written on how to do information abstraction just in the domain of poker. Section 4.1 describes Deep CFR, an alternative to information abstraction that uses deep neural network function approximation to generalize between similar infosets rather than simply bucketing them together. Deep CFR requires far less domain knowledge than past information abstraction techniques, which makes it easier to deploy in imperfect-information games other than poker. Deep CFR was the first scalable non-tabular CFR variant to be successful in large games.

The other form of abstraction is **action abstraction**, in which the action space for players is reduced. Section 4.2 and Section 4.3 describe techniques for automated action abstraction. These were the first automated action abstraction techniques for imperfect-information games with theoretical guarantees of convergence to a locally optimal set of actions.

4.1 Deep Counterfactual Regret Minimization

Prior techniques for scaling CFR to extremely large imperfect-information games have used abstraction, which buckets similar states together and treats them identically. The simplified (abstracted) game is then approximately solved via tabular CFR. However, constructing an effective abstraction requires extensive domain knowledge and the abstract solution may only be a coarse approximation of a true equilibrium.

In contrast, reinforcement learning has been successfully extended to large state spaces by using function approximation with deep neural networks rather than a tabular representation of the policy (deep RL). This approach has led to a number of breakthroughs in constructing strategies in large MDPs [107] as well as in zero-sum perfect-information games such as Go [141, 142].[1] Importantly, deep RL can learn good strategies with relatively little domain knowledge for the specific game [141]. However, most popular RL algorithms do not converge to good policies (equilibria) in imperfect-information games in theory or in practice.

Rather than use tabular CFR with abstraction, this section introduces a form of CFR, which we refer to as **Deep Counterfactual Regret Minimization**, that uses function approximation with deep neural networks to approximate the behavior of tabular CFR on the full, unabstracted game. We prove that Deep CFR converges to an ϵ-Nash equilibrium in two-player zero-sum games and empirically evaluate performance in poker variants, including heads-up limit Texas hold'em. We show that even though Deep CFR uses relatively little domain knowledge, it is competitive with domain-specific tabular abstraction techniques.

Neural Fictitious Self Play (NFSP) [69] previously combined deep learning function approximation with Fictitious Play [13] to produce an AI for heads-up limit Texas hold'em. NFSP was the prior leading function approximation algorithm for imperfect-information games. However, Fictitious Play has weaker theoretical convergence guarantees than CFR, and in practice converges slower. We show in our experiments Deep CFR outperforms NFSP. Model-free policy gradient algorithms have been shown to minimize regret when parameters are tuned appropriately [144] and achieve performance comparable to NFSP.

Past work has investigated using deep learning to estimate values at the depth limit of a subgame in imperfect-information games [27, 110]. However, tabular CFR was used within the subgames themselves. Large-scale function approximated CFR has also been developed for single-agent settings [77]. Our algorithm is intended for the multi-agent setting and is very different from the one proposed for the single-agent setting.

Prior work has combined regression tree function approximation with CFR [162] in an algorithm called **Regression CFR (RCFR)**. This algorithm defines a number of features of the infosets in a game and calculates weights to approximate the regrets that a tabular CFR implementation would produce. Regression CFR is algorithmically similar to Deep CFR, but uses hand-crafted features similar to those used in abstraction, rather than learning the features. RCFR also uses full traversals of the game tree (which is infeasible in large games) and has only been evaluated on toy games. It is therefore best viewed as the first proof of concept that function approximation can be applied to CFR.

[1] Deep RL has also been applied successfully to some partially observed games such as Doom [94], as long as the hidden information is not too strategically important.

Since Deep CFR was announced, several related algorithms have also been developed. Double Neural CFR [101] was developed independently and in parallel with Deep CFR and investigates a similar combination of deep learning with CFR. DREAM [146] extends Deep CFR to the model-free setting.

4.1.1 Description of Deep Counterfactual Regret Minimization

In this section we describe Deep CFR. The goal of Deep CFR is to approximate the behavior of CFR without calculating and accumulating regrets at each infoset, by generalizing across similar infosets using function approximation via deep neural networks.

On each iteration t, Deep CFR conducts a constant number K of partial traversals of the game tree, with the path of the traversal determined according to external sampling MCCFR. At each infoset I it encounters, it plays a strategy $\sigma^t(I)$ determined by regret matching on the output of a neural network $V : I \rightarrow \mathbf{R}^{|A|}$ defined by parameters θ_i^{t-1} that takes as input the infoset I and outputs values $V(I, a|\theta^{t-1})$. Our goal is for $V(I, a|\theta^{t-1})$ to be approximately proportional to the regret $R^{t-1}(I, a)$ that tabular CFR would have produced.

When a terminal node is reached, the value is passed back up. In chance and opponent infosets, the value of the sampled action is passed back up unaltered. In traverser infosets, the value passed back up is the weighted average of all action values, where action a's weight is $\sigma^t(I, a)$. This produces samples of this iteration's instantaneous regrets for various actions. Samples are added to a memory $\mathcal{M}_{v,i}$, where i is the traverser, using reservoir sampling [155] if capacity is exceeded.

Consider a nice property of the sampled instantaneous regrets induced by external sampling:

Lemma 11. *For external sampling MCCFR, the sampled instantaneous regrets are an unbiased estimator of the* ***advantage****, i.e. the difference in expected payoff for playing a vs $\sigma_i^t(I)$ at I, assuming both players play σ^t everywhere else.*

$$\mathbb{E}_{Q \in \mathcal{Q}_t}\left[\tilde{r}_i^{\sigma^t}(I, a)\middle| Z_I \cap Q \neq \emptyset\right] = \frac{v^{\sigma^t}(I, a) - v^{\sigma^t}(I)}{\pi_{-i}^{\sigma^t}(I)}.$$

Recent work in deep reinforcement learning has shown that neural networks can effectively predict and generalize advantages in challenging environments with large state spaces, and use that to learn good policies [108].

Once a player's K traversals are completed, a new network is trained *from scratch* to determine parameters θ_i^t by minimizing MSE between predicted advantage $V_i(I, a|\theta^t)$ and samples of instantaneous regrets from prior iterations $t' \leq t$ $\tilde{r}^{t'}(I, a)$ drawn from the memory. The average over all sampled instantaneous advantages $\tilde{r}^{t'}(I, a)$ is proportional to the total sampled regret $\tilde{R}^t(I, a)$ (across actions in an infoset), so once a sample is added to the memory it is never removed except through reservoir sampling, even when the next CFR iteration begins.

One can use any loss function for the value and average strategy model that satisfies Bregman divergence [4], such as mean squared error loss.

While almost any sampling scheme is acceptable so long as the samples are weighed properly, external sampling has the convenient property that it achieves both of our desired goals by assigning all samples in an iteration equal weight. Additionally, exploring all of a traverser's

actions helps reduce variance. However, external sampling may be impractical in games with extremely large branching factors, so a different sampling scheme, such as outcome sampling [96], may be desired in those cases.

In addition to the value network, a separate policy network $\Pi : I \rightarrow \mathbf{R}^{|A|}$ approximates the average strategy at the end of the run, because it is the *average strategy played over all iterations* that converges to a Nash equilibrium. To do this, we maintain a separate memory $\mathcal{M}_\Pi$ of sampled infoset probability vectors for both players. Whenever an infoset I belonging to player i is traversed during the opposing player's traversal of the game tree via external sampling, the infoset probability vector $\sigma^t(I)$ is added to $\mathcal{M}_\Pi$ and assigned weight t.

If the number of Deep CFR iterations and the size of each value network model is small, then one can avoid training the final policy network by instead storing each iteration's value network [145]. During actual play, a value network is sampled randomly and the player plays the CFR strategy resulting from the predicted advantages of that network. This eliminates the function approximation error of the final average policy network, but requires storing all prior value networks. Nevertheless, strong performance and low exploitability may still be achieved by storing only a subset of the prior value networks [75].

Theorem 13 states that if the memory buffer is sufficiently large, then with high probability Deep CFR will result in average regret being bounded by a constant proportional to the square root of the function approximation error.

Theorem 13. *Let T denote the number of Deep CFR iterations, $|A|$ the maximum number of actions at any infoset, and K the number of traversals per iteration. Let $\mathcal{L}_V^t$ be the average MSE loss for $V_i(I, a|\theta^t)$ on a sample in $\mathcal{M}_{V,i}$ at iteration t , and let $\mathcal{L}_{V^*}^t$ be the minimum loss achievable for any function V. Let $\mathcal{L}_V^t - \mathcal{L}_{V^*}^t \leq \epsilon_{\mathcal{L}}$.*

If the value memories are sufficiently large, then with probability $1 - p$ total regret at time T is bounded by

$$R_p^T \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i|\sqrt{|A|}\sqrt{T} + 4T|\mathcal{I}_i|\sqrt{|A|L\epsilon_{\mathcal{L}}} \tag{4.1}$$

with probability $1 - p$.

Corollary 3. *As $T \rightarrow \infty$, average regret $\frac{R_i^T}{T}$ is bounded by*

$$4|\mathcal{I}_i|\sqrt{|A|L\epsilon_{\mathcal{L}}}$$

with high probability.

We do not provide a convergence bound for Deep CFR when using linear weighting, since the convergence rate of Linear CFR has not been shown in the Monte Carlo case. However, Figure 4.4 shows moderately faster convergence in practice.

4.1.2 Experimental Setup

We measure the performance of Deep CFR (Algorithm 1) in approximating an equilibrium in heads-up limit flop hold'em poker (FHP) (described in Section 2.4.4). FHP is a large game with over 10^{12} nodes and over 10^9 infosets. In contrast, the network we use has 98,948 parameters. We

Algorithm 1 Deep Counterfactual Regret Minimization

function DEEPCFR

Initialize each player's advantage network $V(I, a|\theta_i)$ with parameters θ_i so that it returns 0 for all inputs.

Initialize reservoir-sampled advantage memories $\mathcal{M}_{V,1}, \mathcal{M}_{V,2}$ and strategy memory $\mathcal{M}_\Pi$.

for CFR iteration $t = 1$ to T **do**

for each player i **do**

for traversal $k = 1$ to K **do**

TRAVERSE($\emptyset, i, \theta_1, \theta_2, \mathcal{M}_{V,i}, \mathcal{M}_\Pi$) ▷ Collect data from a game traversal with external sampling

Train θ_i from scratch on loss $\mathcal{L}(\theta_i) = \mathbb{E}_{(I,t',\tilde{r}^{t'})\sim\mathcal{M}_{V,i}}\left[t' \sum_a \left(\tilde{r}^{t'}(a) - V(I, a|\theta_i)\right)^2\right]$

Train θ_Π on loss $\mathcal{L}(\theta_\Pi) = \mathbb{E}_{(I,t',\sigma^{t'})\sim\mathcal{M}_\Pi}\left[t' \sum_a \left(\sigma^{t'}(a) - \Pi(I, a|\theta_\Pi)\right)^2\right]$

return θ_Π

Algorithm 2 CFR Traversal with External Sampling

function TRAVERSE($h, i, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi$, t)

Input: History h, traverser player i, regret network parameters θ for each player, advantage memory $\mathcal{M}_V$ for player i, strategy memory $\mathcal{M}_\Pi$, CFR iteration t.

if h is terminal **then**

return the payoff to player i

else if h is a chance node **then**

$a \sim \sigma(h)$

return TRAVERSE($h \cdot a, i, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi$, t)

else if $P(h) = i$ **then** ▷ If it's the traverser's turn to act

Compute strategy $\sigma^t(I)$ from predicted advantages $V(I(h), a|\theta_i)$ using regret matching.

for $a \in A(h)$ **do**

$v(a) \leftarrow$ TRAVERSE($h \cdot a, i, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi$, t) ▷ Traverse each action

for $a \in A(h)$ **do**

$\tilde{r}(I, a) \leftarrow v(a) - \sum_{a' \in A(h)} \sigma(I, a') \cdot v(a')$ ▷ Compute advantages

Insert the infoset and its action advantages $(I, t, \tilde{r}^t(I))$ into the advantage memory $\mathcal{M}_V$

else ▷ If it's the opponent's turn to act

Compute strategy $\sigma^t(I)$ from predicted advantages $V(I(h), a|\theta_{3-i})$ using regret matching.

Insert the infoset and its action probabilities $(I, t, \sigma^t(I))$ into the strategy memory $\mathcal{M}_\Pi$

Sample an action a from the probability distribution $\sigma^t(I)$.

return TRAVERSE($h \cdot a, i, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi$, t)

also measure performance relative to domain-specific abstraction techniques in the benchmark domain of HULH poker (described in Section 2.4.2), which has over 10^{17} nodes and over 10^{14}

infosets.

In both games, we compare performance to NFSP, which was the previous leading algorithm for imperfect-information game solving using domain-independent function approximation, as well as state-of-the-art abstraction techniques designed for the domain of poker [24, 50, 82].

Network Architecture

We use the neural network architecture shown in Figure 4.1 for both the value network V that computes advantages for each player and the network Π that approximates the final average strategy. This network has a depth of 7 layers and 98,948 parameters. Infosets consist of sets of cards and bet history. The cards are represented as the sum of three embeddings: a rank embedding (1-13), a suit embedding (1-4), and a card embedding (1-52). These embeddings are summed for each set of permutation invariant cards (hole, flop, turn, river), and these are concatenated. In each of the N_{rounds} rounds of betting there can be at most 6 sequential actions, leading to $6N_{rounds}$ total unique betting positions. Each betting position is encoded by a binary value specifying whether a bet has occurred, and a float value specifying the bet size.

The neural network model begins with separate branches for the cards and bets, with three and two layers respectively. Features from the two branches are combined and three additional fully connected layers are applied. Each fully-connected layer consists of $x_{i+1} = \text{ReLU}(Ax[+x])$. The optional skip connection $[+x]$ is applied only on layers that have equal input and output dimension. Normalization (to zero mean and unit variance) is applied to the last-layer features. The network architecture was not highly tuned, but normalization and skip connections were used because they were found to be important to encourage fast convergence when running preliminary experiments on pre-computed equilibrium strategies in FHP.

In the value network, the vector of outputs represented predicted advantages for each action at the input infoset. In the average strategy network, outputs are interpreted as logits of the probability distribution over actions.

Model training

We allocate a maximum size of 40 million infosets to each player's advantage memory $\mathcal{M}_{V,p}$ and the strategy memory $\mathcal{M}_{\Pi}$. The value model is trained from scratch each CFR iteration, starting from a random initialization. We perform 4,000 mini-batch stochastic gradient descent (SGD) iterations using a batch size of 10,000 and perform parameter updates using the Adam optimizer [84] with a learning rate of 0.001, with gradient norm clipping to 1. For HULH we use 32,000 SGD iterations and a batch size of 20,000. Figure 4.4 shows that training the model from scratch at each iteration, rather than using the weights from the previous iteration, leads to better convergence.

Linear CFR

In our experiments we use Linear CFR (described in Section 3.1, a variant of CFR that is faster than CFR and in certain settings is the fastest-known variant of CFR (particularly in settings with wide distributions in payoffs), and which tolerates approximation error well. LCFR is not

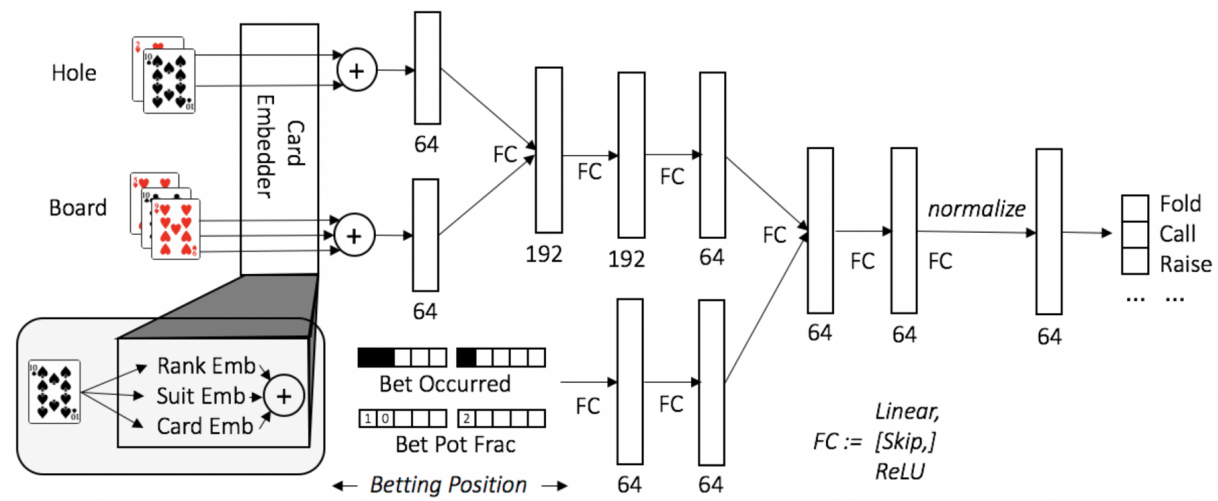


Figure 4.1: The neural network architecture used for Deep CFR. The network takes an infoset (observed cards and bet history) as input and outputs values (advantages or probability logits) for each possible action.

essential and does not appear to lead to better performance asymptotically, but does result in faster convergence in our experiments.

LCFR is like CFR except iteration t is weighed by t. Specifically, we maintain a *weight* on each entry stored in the advantage memory and the strategy memory, equal to t when this entry was added. When training θ_p each iteration T, we rescale all the batch weights by $\frac{2}{T}$ and minimize weighted error.

4.1.3 Experimental Results

Figure 4.2 compares the performance of Deep CFR to different-sized domain-specific abstractions in FHP. The abstractions are solved using external-sampling Linear Monte Carlo CFR [22, 96], which is the leading algorithm in this setting. The 40,000 cluster abstraction means that the more than 10^9 different decisions in the game were clustered into 40,000 abstract decisions, where situations in the same bucket are treated identically. This bucketing is done using K-means clustering on domain-specific features. The *lossless abstraction* only clusters together situations that are strategically isomorphic (e.g., flushes that differ only by suit), so a solution to this abstraction maps to a solution in the full game without error.

Performance and exploitability are measured in terms of milli big blinds per game (mbb/g), which is a standard measure of win rate in poker.

The figure shows that Deep CFR asymptotically reaches a similar level of exploitability as the abstraction that uses 3.6 million clusters, but converges substantially faster. Although Deep CFR is more efficient in terms of nodes touched, neural network inference and training requires considerable overhead that tabular CFR avoids. However, Deep CFR does not require advanced domain knowledge. We show Deep CFR performance for 10,000 CFR traversals per step. Using more traversals per step is less sample efficient and requires greater neural network training time

but requires fewer CFR steps.

Figure 4.2 also compares the performance of Deep CFR to NFSP, an existing method for learning approximate Nash equilibria in imperfect-information games. NFSP approximates fictitious self-play, which is proven to converge to a Nash equilibrium but in practice does so far slower than CFR. We observe that Deep CFR reaches an exploitability of 37 mbb/g while NFSP converges to 47 mbb/g.[2] We also observe that Deep CFR is more sample efficient than NFSP. However, these methods spend most of their wallclock time performing SGD steps, so in our implementation we see a less dramatic improvement over NFSP in wallclock time than sample efficiency.

Figure 4.3 shows the performance of Deep CFR using different numbers of game traversals, network SGD steps, and model size. As the number of CFR traversals per iteration is reduced, convergence becomes slower but the model converges to the same final exploitability. This is presumably because it takes more iterations to collect enough data to reduce the variance sufficiently. On the other hand, reducing the number of SGD steps does not change the rate of convergence but affects the asymptotic exploitability of the model. This is presumably because the model loss decreases as the number of training steps is increased per iteration (see Theorem 13). Increasing the model size also decreases final exploitability up to a certain model size in FHP.

In Figure 4.4 we consider ablations of certain components of Deep CFR. Retraining the regret model from scratch at each CFR iteration converges to a substantially lower exploitability than fine-tuning a single model across all iterations. We suspect that this is because a single model gets stuck in bad local minima as the objective is changed from iteration to iteration. The choice of reservoir sampling to update the memories is shown to be crucial; if a sliding window memory is used, the exploitability begins to increase once the memory is filled up, even if the memory is large enough to hold the samples from many CFR iterations.

Finally, we measure head-to-head performance in HULH. We compare Deep CFR and NFSP to the approximate solutions (solved via Linear Monte Carlo CFR) of three different-sized abstractions: one in which the more than 10^{14} decisions are clustered into $3.3 \cdot 10^6$ buckets, one in which there are $3.3 \cdot 10^7$ buckets and one in which there are $3.3 \cdot 10^8$ buckets. The results are presented in Table 4.1. For comparison, the largest abstractions used by the poker AI Polaris in its 2007 HULH man-machine competition against human professionals contained roughly $3 \cdot 10^8$ buckets. When variance-reduction techniques were applied, the results showed that the professional human competitors lost to the 2007 Polaris AI by about 52 ± 10 mbb/g [83]. In contrast, our Deep CFR agent loses to a $3.3 \cdot 10^8$ bucket abstraction by only -11 ± 2 mbb/g and beats NFSP by 43 ± 2 mbb/g.

4.1.4 Conclusions

We described a method to find approximate equilibria in large imperfect-information games by combining the CFR algorithm with deep neural network function approximation. This method is

[2]We run NFSP with the same model architecture as we use for Deep CFR. In the benchmark game of Leduc hold'em, our implementation of NFSP achieves an average exploitability (total exploitability divided by two) of 37 mbb/g in the benchmark game of Leduc hold'em, which is substantially lower than originally reported in Heinrich and Silver [69]. We report NFSP's best performance in FHP across a sweep of hyperparameters.

Model	Opponent Model				
			Abstraction Size		
	NFSP	Deep CFR	$3.3 \cdot 10^6$	$3.3 \cdot 10^7$	$3.3 \cdot 10^8$
NFSP	-	-43 ± 2 mbb/g	-40 ± 2 mbb/g	-49 ± 2 mbb/g	-55 ± 2 mbb/g
Deep CFR	$+43 \pm 2$ mbb/g	-	$+6 \pm 2$ mbb/g	-6 ± 2 mbb/g	-11 ± 2 mbb/g

Table 4.1: Head-to-head expected value of NFSP and Deep CFR in HULH against converged CFR equilibria with varying abstraction sizes. For comparison, in 2007 an AI using abstractions of roughly $3 \cdot 10^8$ buckets defeated human professionals by about 52 mbb/g (after variance reduction techniques were applied).

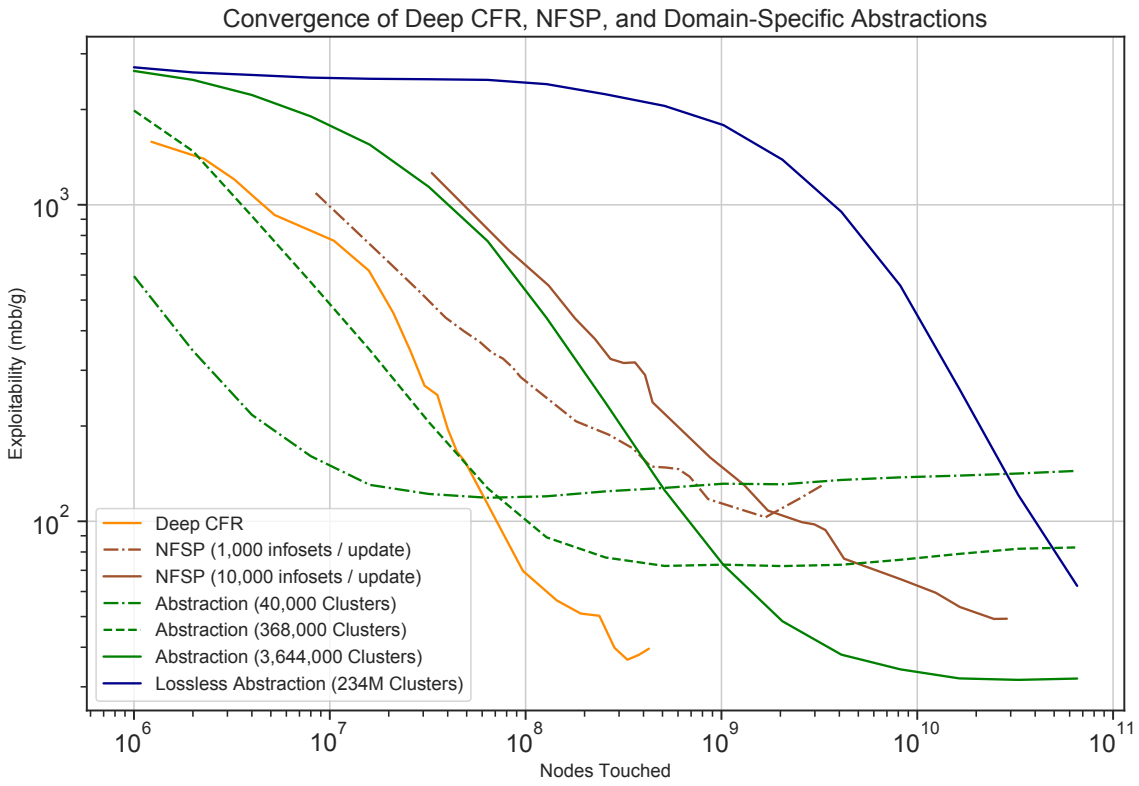


Figure 4.2: Comparison of Deep CFR with domain-specific tabular abstractions and NFSP in FHP. Coarser abstractions converge faster but are more exploitable. Deep CFR converges with 2-3 orders of magnitude fewer samples than a lossless abstraction, and performs competitively with a 3.6 million cluster abstraction. Deep CFR achieves lower exploitability than NFSP, while traversing fewer infosets.

theoretically principled and achieves strong performance in large poker games relative to domain-specific abstraction techniques without relying on advanced domain knowledge. This was the first non-tabular variant of CFR to be successful in large games.

Deep CFR and other neural methods for imperfect-information games provide a promising direction for tackling large games whose state or action spaces are too large for tabular methods and where abstraction is not straightforward.

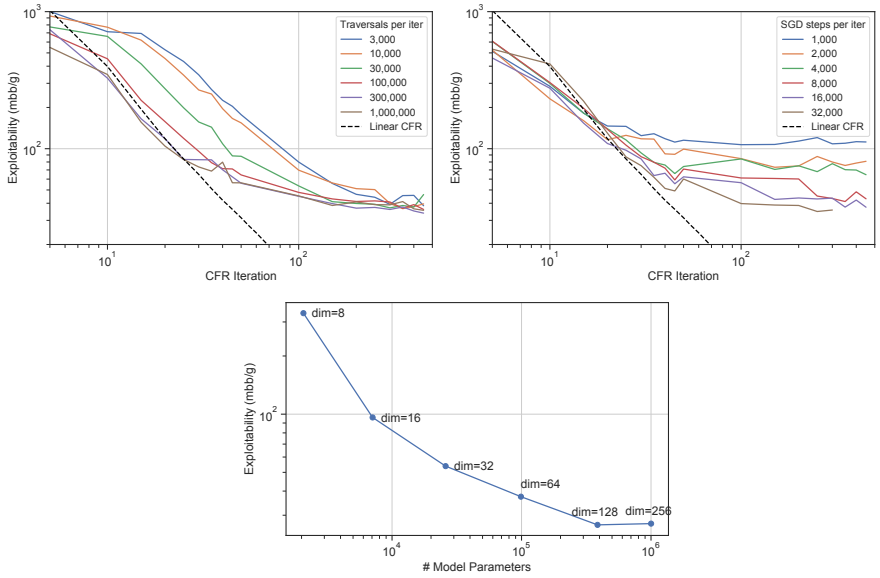


Figure 4.3: **Left:** FHP convergence for different numbers of training data collection traversals per simulated LCFR iteration. The dotted line shows the performance of vanilla tabular Linear CFR without abstraction or sampling. **Middle:** FHP convergence using different numbers of minibatch SGD updates to train the advantage model at each LCFR iteration. **Right:** Exploitability of Deep CFR in FHP for different model sizes. Label indicates the dimension (number of features) in each hidden layer of the model.

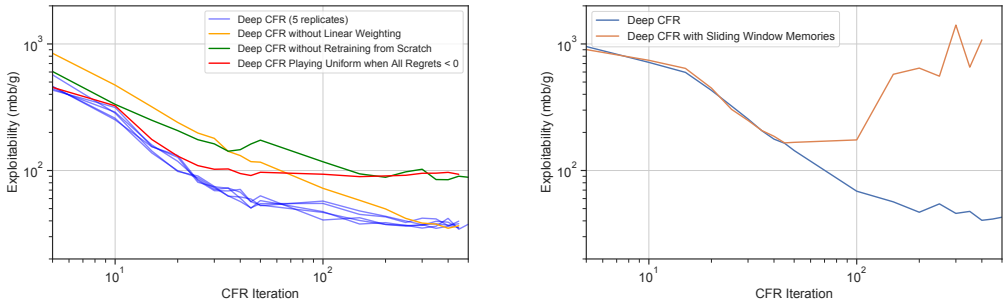


Figure 4.4: Ablations of Deep CFR components in FHP. **Left:** As a baseline, we plot 5 replicates of Deep CFR, which show consistent exploitability curves (standard deviation at $t = 450$ is 2.25 mbb/g). Deep CFR without linear weighting converges to a similar exploitability, but more slowly. If the same network is fine-tuned at each CFR iteration rather than training from scratch, the final exploitability is about 50% higher. Also, if the algorithm plays a uniform strategy when all regrets are negative (i.e. standard regret matching), rather than the highest-regret action, the final exploitability is also 50% higher. **Right:** If Deep CFR is performed using sliding-window memories, exploitability stops converging once the buffer becomes full. However, with reservoir sampling, convergence continues after the memories are full.

4.1.5 Proofs of Theoretical Results

Proof of Lemma 11

We show

$$\mathbb{E}_{Q_j \sim \mathcal{Q}} \left[\tilde{v}_i^{\sigma^t}(I) \middle| Z_I \cap Q_j \neq \emptyset \right] = v^{\sigma^t}(I) / \pi_{-i}^{\sigma^t}(I).$$

Let $q_j = P(Q_j)$.

$$\begin{aligned}
\mathbb{E}_{Q_j \sim \mathcal{Q}} \left[\tilde{v}_i^{\sigma^t}(I) \middle| Z_I \cap Q \neq \emptyset \right] &= \frac{\mathbb{E}_{Q_j \sim \mathcal{Q}} \left[\tilde{v}_i^{\sigma^t}(I) \right]}{P_{Q_j \sim \mathcal{Q}}(Z_I \cap Q_j \neq \emptyset)} \\
&= \frac{\sum_{Q_j \in \mathcal{Q}} q_j \sum_{z \in Z_I \cap Q_j} u_i(z) \pi_{-i}^{\sigma^t}(z[I]) \pi^{\sigma^t}(z[I] \to z) / q(z)}{\pi_{-i}^{\sigma^t}(I)} \\
&= \frac{\sum_{z \in Z_I \cap Q_j} \left(\sum_{Q_j : z \in Q_j} q_j \right) u_i(z) \pi_{-i}^{\sigma^t}(z[I]) \pi^{\sigma^t}(z[I] \to z) / q(z)}{\pi_{-i}^{\sigma^t}(I)} \\
&= \frac{\sum_{z \in Z_I} q(z) u_i(z) \pi_{-i}^{\sigma^t}(z[I]) \pi^{\sigma^t}(z[I] \to z) / q(z)}{\pi_{-i}^{\sigma^t}(I)} \\
&= \frac{v^{\sigma^t}(I)}{\pi_{-i}^{\sigma^t}(I)}
\end{aligned}$$

The result now follows directly.

K-external sampling

We first show that performing MCCFR with K external sampling traversals per iteration (K-ES) shares a similar convergence bound with standard external sampling (i.e. 1-ES). We will refer to this result in the next section when we consider the full Deep CFR algorithm. This convergence bound is rather obvious and the derivation pedantic, so the reader is welcome to skip this section.

We model T rounds of K-external sampling as $T \times K$ rounds of external sampling, where at each round $t \cdot K + d$ (for integer $t \geq 0$ and integer $0 \leq d < K$) we play

$$\sigma_{tK+d}(a) = \begin{cases} \frac{R_{tK}^+(a)}{R_{\Sigma,tK}^+} \text{ if } R_{\Sigma,tK}^+ > 0 \\ \text{arbitrary, otherwise} \end{cases} \tag{4.2}$$

In prior work, σ is typically defined to play $\frac{1}{|A|}$ when $R_{\Sigma,T}^+(a) \leq 0$, but in fact the convergence bounds do not constraint σ's play in these situations, which we will demonstrate explicitly here. We need this fact because minimizing the loss $\mathcal{L}(V)$ is defined only over the samples of (visited) infosets and thus does not constrain the strategy in unvisited infosets.

Lemma 12. *If regret matching is used in K-ES, then for $0 \leq d < K$*

$$\sum_{a \in A} R_{tK}^+(a) r_{tK+d}(a) \leq 0 \tag{4.3}$$

Proof. If $R^+_{\Sigma,tK} \leq 0$, then $R^+_{tK}(a) = 0$ for all a and the result follows directly. For $R^+_{\Sigma,tK} > 0$,

$$\sum_{a\in A} R^+_{tK}(a) r_{tK+d}(a) = \sum_{a\in A} R^+_T(a)(u_{tK+d}(a) - u_{tK+d}(\sigma_{tK})) \tag{4.4}$$

$$= \left(\sum_{a\in A} R^+_{tK}(a) u_{tK+d}(a)\right) - \left(u_{tK+d}(\sigma_{tK}) \sum_{a\in A} R^+_{tK}(a)\right) \tag{4.5}$$

$$= \left(\sum_{a\in A} R^+_{tK}(a) u_{tK+d}(a)\right) - \left(\sum_{a\in A} \sigma_{tK+d}(a) u_{tK+d}(a)\right) R^+_{\Sigma,tK}(a) \tag{4.6}$$

$$= \left(\sum_{a\in A} R^+_{tK}(a) u_{tK+d}(a)\right) - \left(\sum_{a\in A} \frac{R^+_{tK}(a)}{R^+_{\Sigma,tK}(a)} u_{tK+d}(a)\right) R^+_{\Sigma,tK}(a) \tag{4.7}$$

$$= \left(\sum_{a\in A} R^+_{tK}(a) u_{tK+d}(a)\right) - \left(\sum_{a\in A} R^+_{tK}(a)(a) u_{tK+d}(a)\right) \tag{4.8}$$

$$= 0 \tag{4.9}$$

□

Theorem 14. *Playing according to Equation 4.2 guarantees the following bound on total regret*

$$\sum_{a\in A} (R^+_{TK}(a))^2 \leq |A| L^2 K^2 T \tag{4.10}$$

Proof. We prove by recursion on T.

$$\sum_{a\in A} (R^+_{TK}(a))^2 \leq \sum_{a\in A} \left(R^+_{(T-1)K}(a) + \sum_{d=0}^{K-1} r_{tK-d}(a)\right)^2 \tag{4.11}$$

$$= \sum_{a\in A} \left(R^+_{(T-1)K}(a)^2 + 2\sum_{d=0}^{K-1} r_d(a) R^+_{(T-1)K}(a) + \sum_{d=0}^{K-1}\sum_{d'=0}^{K-1} r_{TK-d}(a) r_{TK-d'}(a)\right) \tag{4.12}$$

By Lemma 12,

$$\sum_{a\in A} (R^+_{TK}(a))^2 \leq \sum_{a\in A} (R^+_{(T-1)K}(a))^2 + \sum_{a\in A}\sum_{d=0}^{K-1}\sum_{d'=0}^{K-1} r_{TK-d}(a) r_{TK-d'}(a) \tag{4.13}$$

By induction,

$$\sum_{a\in A} (R^+_{(T-1)K}(a))^2 \leq |A| L^2 (T-1) \tag{4.14}$$

From the definition, $|r_{TK-d}(a)| \leq L$

$$\sum_{a\in A} (R^+_{TK}(a))^2 \leq |A| L^2 (T-1) + K^2 |A| L^2 = |A| L^2 K^2 T \tag{4.15}$$

□

Theorem 15. *(Lanctot [95], Theorem 3 & Theorem 5) After T iterations of K-ES, average regret is bounded by*

$$\bar{R}_i^{TK} \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) |\mathcal{I}_i| L \frac{\sqrt{|A|}}{\sqrt{T}} \tag{4.16}$$

with probability $1 - p$.

Proof. The proof follows Lanctot [95], Theorem 3. Note that K-ES is only different from ES in terms of the choice of σ_T, and the proof in Lanctot [95] only makes use of σ_T via the bound on $(\sum_a R_+^T(a))^2$ that we showed in Theorem 14. Therefore, we can apply the same reasoning to arrive at

$$\tilde{R}_i^{TK} \leq \frac{LMi\sqrt{|A|TK}}{L} \tag{4.17}$$

(Lanctot [95], Eq. (4.30)).

Lanctot et al. [96] then shows that $\tilde{R}_i^{TK}$ and R_i^{TK} are similar with high probability, leading to

$$\mathbb{E}\left[\left(\sum_{I \in \mathcal{I}_i} (R_i^{TK}(I) - \tilde{R}_i^{TK}(I))\right)^2\right] \leq \frac{2|\mathcal{I}_i||\mathcal{B}_i||A|TKL^2}{L^2} \tag{4.18}$$

(Lanctot [95], Eq. (4.33), substituting $T \to TK$).

Therefore, by Markov's inequality, with probability at least $1 - p$,

$$R_i^{TK} \leq \frac{\sqrt{2|\mathcal{I}_i||\mathcal{B}_i||A|TK}L}{L\sqrt{p}} + \frac{LM\sqrt{|A|T}K}{L} \tag{4.19}$$

where external sampling permits $L = 1$ [95].

Using the fact that $M \leq |\mathcal{I}_i|$ and $|\mathcal{B}_i| < |\mathcal{I}_i|$ and dividing through by KT leads to the simplified form

$$\bar{R}_i^{TK} \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i| \frac{\sqrt{|A|}}{\sqrt{T}} \tag{4.20}$$

with probability $1 - p$.

□

We point out that the convergence of K-ES is faster as K increases (up to a point), but it still requires the same order of iterations as ES.

Proof of Theorem 13

Proof. Assume that an online learning scheme plays

$$\sigma^t(I,a) = \begin{cases} \frac{y_+^t(I,a)}{\sum_a y_+^t(I,a)} \text{ if } \sum_a y_+^t(I,a) > 0 \\ \text{arbitrary, otherwise} \end{cases} \tag{4.21}$$

Morrill [111], Corollary 3.0.6 provides the following bound on the total regret as a function of the L2 distance between y_t^+ and $R^{T,+}$ at each infoset.

$$\max_{a \in A}(R^T(I,a))^2 \leq |A|L^2T + 4L|A| \sum_{t=1}^{T} \sum_{a \in A} \sqrt{(R_+^t(I,a) - y_+^t(I,a))^2} \tag{4.22}$$

$$\leq |A|L^2T + 4L|A| \sum_{t=1}^{T} \sum_{a \in A} \sqrt{(R^t(I,a) - y^t(I,a))^2} \tag{4.23}$$

Since $\sigma^t(I,a)$ from Eq. 4.21 is invariant to rescaling across all actions at an infoset, it's also the case that for any $C(I) > 0$

$$\max_{a \in A}(R^T(I,a))^2 \leq |A|L^2T + 4L|A| \sum_{t=1}^{T} \sum_{a \in A} \sqrt{(R^t(I,a) - C(I)y^t(I,a))^2} \tag{4.24}$$

Let $x^t(I)$ be an indicator variable that is 1 if I was traversed on iteration t. If I was traversed then $\tilde{r}^t(I)$ was stored in $M_{V,i}$, otherwise $\tilde{r}^t(I) = 0$. Assume for now that $\mathcal{M}_{V,i}$ is not full, so all sampled regrets are stored in the memory.

Let $\Pi^t(I)$ be the fraction of iterations on which $x^t(I) = 1$, and let

$$\epsilon^t(I) = \left\| \mathbb{E}_t \left[\tilde{r}^t(I) | x^t(I) = 1 \right] - V(I, a|\theta^t) \right\|_2$$

Inserting canceling factors of $\sum_{t'=1}^{t} x^{t'}(I)$ and setting $C(I) = \sum_{t'=1}^{t} x^{t'}(I)$,[3]

$$\max_{a \in A} (\tilde{R}^T(I,a))^2 \leq |A|L^2T + 4L|A| \sum_{t=1}^{T} \left(\sum_{t'=1}^{t} x^{t'}(I) \right) \sum_{a \in A} \sqrt{\left(\frac{\tilde{R}^t(I,a)}{\sum_{t'=1}^{t} x^{t'}(I)} - y^t(I,a) \right)^2} \tag{4.25}$$

$$= |A|L^2T + 4L|A| \sum_{t=1}^{T} \left(\sum_{t'=1}^{t} x^{t'}(I) \right) \left\| \mathbb{E}_t \left[\tilde{r}^t(I) | x^t(I) = 1 \right] - V(I,a|\theta^t) \right\|_2 \tag{4.26}$$

$$= |A|L^2T + 4L|A| \sum_{t=1}^{T} t\Pi^t(I)\epsilon^t(I) \quad \text{by definition} \tag{4.27}$$

$$\leq |A|L^2T + 4L|A|T \sum_{t=1}^{T} \Pi^t(I)\epsilon^t(I) \tag{4.28}$$

$$\tag{4.29}$$

The first term of this expression is the same as Theorem 14, while the second term accounts for the approximation error.

In the case of K-external sampling, the same derivation as shown in Theorem 14 leads to

$$\max_{a \in A} (\tilde{R}^T(I,a))^2 \leq |A|L^2TK^2 + 4L\sqrt{|A|}TK^2 \sum_{t=1}^{T} \Pi^t(I)\epsilon^t(I) \tag{4.30}$$

in this case. We elide the proof.

The new regret bound in Eq. (4.30) can be plugged into Lanctot [95], Theorem 3 as we do for Theorem 15, leading to

$$\bar{R}_i^T \leq \sum_{I \in \mathcal{I}_i} \left(\left(1 + \frac{\sqrt{2}}{\sqrt{pK}} \right) L \frac{\sqrt{|A|}}{\sqrt{T}} + \frac{4}{\sqrt{T}} \sqrt{|A|L \sum_{t=1}^{T} \Pi^t(I)\epsilon^t(I)} \right) \tag{4.31}$$

Simplifying the first term and rearranging,

[3]The careful reader may note that $C(I) = 0$ for unvisited infosets, but $\sigma^t(I,a)$ can play an arbitrary strategy at these infosets so it's okay.

$$\bar{R}_i^T \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i|\frac{\sqrt{|A|}}{\sqrt{T}} + \frac{4\sqrt{|A|L}}{\sqrt{T}} \sum_{I\in\mathcal{I}_i} \sqrt{\sum_{t=1}^{T} \Pi^t(I)\epsilon^t(I)} \tag{4.32}$$

$$\bar{R}_i^T \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i|\frac{\sqrt{|A|}}{\sqrt{T}} + \frac{4\sqrt{|A|L}}{\sqrt{T}} |\mathcal{I}_i| \frac{\sum_{I\in\mathcal{I}_i}}{|\mathcal{I}_i|} \sqrt{\sum_{t=1}^{T} \Pi^t(I)\epsilon^t(I)} \tag{4.33}$$

$$\leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i|\frac{\sqrt{|A|}}{\sqrt{T}} + \frac{4\sqrt{|A|L|\mathcal{I}_i|}}{\sqrt{T}} \sqrt{\sum_{t=1}^{T} \sum_{I\in\mathcal{I}_i} \Pi^t(I)\epsilon^t(I)} \qquad \text{by Jensen's inequality} \tag{4.34}$$

Now, lets consider the average MSE loss $\mathcal{L}_V^T(\mathcal{M}^T)$ at time T over the samples in memory $\mathcal{M}^{\mathcal{T}}$.

We start by stating two well-known lemmas:

Lemma 13. *The MSE can be decomposed into bias and variance components*

$$\mathbb{E}_x[(x-\theta)^2] = (\theta - \mathbb{E}[x])^2 + \mathrm{Var}(\theta) \tag{4.35}$$

Lemma 14. *The mean of a random variable minimizes the MSE loss*

$$\underset{\theta}{\mathrm{argmin}}\, \mathbb{E}_x[(x-\theta)^2] = \mathbb{E}[x] \tag{4.36}$$

and the value of the loss at when $\theta = \mathbb{E}[x]$ is $\mathrm{Var}(x)$.

$$\mathcal{L}_V^T = \frac{1}{\sum_{I\in\mathcal{I}_i}\sum_{t=1}^{T} x^t(I)} \sum_{I\in\mathcal{I}_i}\sum_{t=1}^{T} x^t(I) \left\|\tilde{r}^t(I) - V(I|\theta^T)\right\|_2^2 \tag{4.37}$$

$$\geq \frac{1}{|\mathcal{I}_i|T} \sum_{I\in\mathcal{I}_i}\sum_{t=1}^{T} x^t(I) \left\|\tilde{r}^t(I) - V(I|\theta^T)\right\|_2^2 \tag{4.38}$$

$$= \frac{1}{|\mathcal{I}_i|} \sum_{I\in\mathcal{I}_i} \Pi^T(I)\, \mathbb{E}_t\left[\left\|\tilde{r}^t(I) - V(I|\theta^T)\right\|_2^2 \middle| x^t(I) = 1\right] \tag{4.39}$$

Let V^* be the model that minimizes $\mathcal{L}^T$ on $\mathcal{M}_T$. Using Lemmas 13 and 14,

$$\mathcal{L}_V^T \geq \frac{1}{|\mathcal{I}_i|T} \sum_{I\in\mathcal{I}_i} \Pi^T(I)\ \left(\left\|V(I|\theta^T) - \mathbb{E}_t\left[\tilde{r}^t(I)\middle|x^t(I)=1\right]\right\|_2^2 + \mathcal{L}_{V^*}^T\right) \tag{4.40}$$

So,

$$\mathcal{L}_V^T - \mathcal{L}_{V^*}^T \geq \frac{1}{|\mathcal{I}_i|} \sum_{I \in \mathcal{I}_i} \Pi^T(I)\, \epsilon^T(I) \tag{4.41}$$

$$\sum_{I \in \mathcal{I}_i} \Pi^T(I)\, \epsilon^T(I) \leq |\mathcal{I}_i|(\mathcal{L}_V^T - \mathcal{L}_{V^*}^T) \tag{4.42}$$

Plugging this into Eq. 4.32, we arrive at

$$\bar{R}_i^T \leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i| \frac{\sqrt{|A|}}{\sqrt{T}} + \frac{4\sqrt{|A|L|\mathcal{I}_i|}}{\sqrt{T}} \sqrt{|\mathcal{I}_i| \sum_{t=1}^{T} (\mathcal{L}_V^t - \mathcal{L}_{V^*}^t)} \tag{4.43}$$

$$\leq \left(1 + \frac{\sqrt{2}}{\sqrt{pK}}\right) L|\mathcal{I}_i| \frac{\sqrt{|A|}}{\sqrt{T}} + 4|\mathcal{I}_i| \sqrt{|A|L\epsilon_{\mathcal{L}}} \tag{4.44}$$

So far we have assumed that $\mathcal{M}_V$ contains all sampled regrets. The number of samples in the memory at iteration t is bounded by $K \cdot |\mathcal{I}_i| \cdot t$. Therefore, if $K \cdot |\mathcal{I}_i| \cdot T < |\mathcal{M}_V|$ then the memory will never be full, and we can make this assumption.[4] □

Proof of Corollary 3

Proof. Let $p = T^{-1/4}$.

$$P\left(\bar{R}_i^T > \left(1 + \frac{\sqrt{2}}{\sqrt{K}}\right) L|\mathcal{I}_i| \frac{\sqrt{|A|}}{T^{-1/4}} + 4|\mathcal{I}_i| \sqrt{|A|L\epsilon_{\mathcal{L}}}\right) < T^{-1/4} \tag{4.45}$$

Therefore, for any $\epsilon > 0$,

$$\lim_{T \to \infty} P\left(\bar{R}_i^T - 4|\mathcal{I}_i| \sqrt{|A|L\epsilon_{\mathcal{L}}} > \epsilon\right) = 0. \tag{4.46}$$

□

4.2 Regret Transfer and Parameter Optimization

CFR can only be run in games with discrete action spaces. In order to use CFR in a game with a continuous action space, such as no-limit poker, the action space must be discretized. This process is referred to as action abstraction, and has been used in every competitive HUNL poker agent. In almost all cases, these action abstractions were determined by hand based on domain knowledge. Some prior work investigated automated algorithms for action abstraction, but either

[4] We do not formally handle the case where the memories become full in this work. Intuitively, reservoir sampling should work well because it keeps an 'unbiased' sample of previous iterations' regrets. We observe empirically in Figure 4.4 that reservoir sampling performs well while using a sliding window does not.

had no convergence guarantees [67, 68] or had been defined only for a much narrower game class, stochastic games [129].

In this section we describe the first action abstraction algorithm for continuous action space extensive-form games that has locally optimal convergence guarantees. These techniques were ultimately used in Libratus to choose the bet sizes that were considered in the first two actions of the game.

We begin by proving that regrets on actions in one setting (game) can be transferred to warm start the regrets for solving a different setting with same structure but different payoffs that can be written as a function of parameters. We prove how this can be done by carefully discounting the prior regrets. We then study optimizing a parameter vector for a player in a two-player zero-sum game (e.g., optimizing bet sizes to use in poker). We propose a custom gradient descent algorithm that provably finds a locally optimal parameter vector while significantly saving regret-matching iterations at each step. It optimizes the parameter vector while simultaneously finding an equilibrium. We present experiments in no-limit Leduc hold'em and no-limit Texas hold'em to optimize bet sizing.

4.2.1 Regret Transfer: Initializing Regrets of Actions Based on Regrets Computed for Related Settings

We consider a space of games with identical structure but potentially differing rewards. Suppose that we ran T iterations of regret matching on a game Γ_1 and then made a very small change to the rewards so that we are now in a game Γ_2. Intuitively, if the change was small, then most of what we learned from Γ_1 should still apply to Γ_2, though perhaps it will not be quite as relevant as before. Indeed, in this section we build on that intuition and prove that we can transfer most of what we learned, and therefore avoid restarting regret matching from scratch. We will show that this can be done by appropriately discounting the former regrets.

The fundamental idea is that since we make only a small change to the payoffs, if the same exact sequence of strategies that were played in the T iterations of regret matching in game Γ_1 were repeated in Γ_2 (ignoring the regrets when choosing actions in each iteration), then we can still bound the regret in Γ_2. If we then carefully scale those iterations down so that they "weigh less", we can fit them into a bound that is equivalent to having played $T' \leq T$ iterations of regret matching from scratch in Γ_2. We then prove that we can continue regret matching for some number of iterations T_2 in Γ_2, and our resulting bound is equivalent to having played $T' + T_2$ iterations of regret matching from scratch in Γ_2. We call this algorithm **regret transfer**.

Section 3.2 presented a warm starting algorithm for CFR that can also transfer what has been learned in Γ_1 to Γ_2. However, the regret transfer algorithm we describe in this section can do this transfer instantaneously even without a full traversal of the game tree, and requires far less parameter tuning. However, as we will see, the regret transfer algorithm is more limited.

Key to our regret transfer algorithm is the assumption that we can replay the exact sequence of strategies from the T iterations of regret matching from Γ_1 in Γ_2. In general, this is not worthwhile to implement, as it would be better to simply play T iterations of regret matching in Γ_2 from scratch. However, we can "replay" the T iterations in $O(1)$ in the case we mentioned previously, where payoffs are all functions of a parameter vector $\vec{\theta}$, and we change only the value

of this vector. In this case, we can store the regret we accumulate in Γ_1 as a function of this vector. For example, consider the case of $\vec{\theta}$ being just a scalar θ. If all payoffs are of the form $u_{i,\langle a_i,a_{-i}\rangle} = \alpha_{i,\langle a_i,a_{-i}\rangle}\theta + \beta_{i,\langle a_i,a_{-i}\rangle}$, then we can store regret for every action with separate values $R_\alpha(a)$ and $R_\beta(a)$. When we access the regret to determine the action for the next iteration, we can simply calculate $R(a) = R_\alpha(a)\theta_1 + R_\beta(a)$, where θ_1 is the value of θ in Γ_1. This way, if we move to Γ_2 where θ is some different value θ_2, then we can simply evaluate $R_\alpha(a)\theta_2 + R_\beta(a)$ to get regret.

We will now proceed to proving that regrets can be transferred across settings by appropriately discounting the iterations conducted at prior settings, starting with the case of normal form games. First, we will present a result that will be useful in our proof.

Let $\vec{R}_i^T$ denote the vector of regrets for the set of actions for player i on iteration T. As a reminder, the potential function of a regret vector for a player i is $\Phi(\vec{R}_i^T) = \sum_{a\in A}\left(R_{i,+}^T(a)\right)^2$. As shown in (2.5), the bound on the potential function when playing according to regret matching every iteration is $\Phi(\vec{R}_i^T) \leq L^2|A|T$. Rather than playing according to RM, suppose instead we played a sequence of arbitrary strategies first and only then start playing according to regret matching. We will now show that we can still bound the potential function.

Lemma 15. *Suppose player i is given an initial potential $\Phi(\vec{R}_i^{T_0}) \leq wT_0|A_i|L_i^2$ for a game where T_0 iterations have been played and w is an arbitrary scalar. If regret matching is used in all future iterations, then at iteration $T_0 + T$, we can bound the potential:*

$$\Phi(\vec{R}_i^{T_0} + \vec{R}_i^T) \leq (wT_0 + T)|A_i|L_i^2 \tag{4.47}$$

We now show specifically how to weigh the previous iterations. Suppose after T iterations we have some regret R_T, and then we modify the game by changing the payoffs slightly. We will analyze how one can initialize regret matching in this new game.

We will find it useful to define **weighted average regret**:

$$\frac{R_{w,i}^{T,T_2}(a)}{wT + T_2} = \frac{w\sum_{t=1}^T r_i^t(a) + \sum_{t=1}^{T_2} r_i^t(a)}{wT + T_2} \tag{4.48}$$

Theorem 16. *Say we have played T iterations of regret matching in a game. Assume all players play the same sequence of strategies over T iterations in some other game, where the structure of the game is identical but the payoffs may differ. We denote regret in this new game by R'. Consider any weight w_i for agent i such that*

$$0 \leq w_i \leq \frac{L_i^2|A_i|T}{\Phi(\vec{R}_i'^T)} \tag{4.49}$$

If we scale the payoffs of the first T iterations by w_i and then play according to regret matching for T_2 iterations, then weighted average regret in the new game is

$$\frac{R_{w,i}'^{T,T_2}}{w_iT + T_2} \leq \frac{L_i\sqrt{|A_i|}}{\sqrt{(w_iT + T_2)}} \tag{4.50}$$

This is the same bound achieved from the player playing $w_i T + T_2$ iterations of regret-matching from scratch.

Later in this paper we will find it useful to define w_i in terms of the maximum change in regret. We therefore present the following proposition.

Proposition 1. *Let*

$$L_i = \frac{\max\left\{ \max_{a \in A}[R_i'^T(a) - R_i^T(a)],\ 0 \right\}}{T} \tag{4.51}$$

If we choose

$$0 \leq w_i \leq \frac{1}{1 + \frac{2L_i\sqrt{|A_i|}\sqrt{T}}{L_i} + \frac{L_i^2 T}{L_i^2}} \tag{4.52}$$

then Theorem 16 still holds.

4.2.2 Warm Start Toward Nash Equilibrium in Zero-Sum Games

We now strengthen Theorem 16 to show that we get a warm start toward an approximate Nash equilibrium in two-player zero-sum games.

We first define a **weighted average strategy** as the average of a series of strategies where the first T iterations are weighted by w and the latter T_2 iterations by 1. Formally, we define $\sigma_{w,i}^{T,T_2}$ such that the probability of action a is

$$p_{\sigma_{w,i}^{T,T_2}}(a) = \frac{w\sum_{t=1}^{T} p_{\sigma_i^t}(a) + \sum_{t=1}^{T_2} p_{\sigma_i^t}(a)}{wT + T_2} \tag{4.53}$$

Corollary 4. *Say both players have played T iterations of regret matching in a two-player game Γ. Let us transfer regret for both players to a new two-player zero-sum game with identical structure Γ' according to Theorem 16, and let $w = \min_i\{w_i\}$. If both players play an additional T_2 iterations of regret matching, then their weighted average strategies constitute a 2ϵ-Nash equilibrium where*

$$\epsilon = \max_i \left\{ \frac{L_i\sqrt{|A_i|}}{\sqrt{(wT + T_2)}} \right\}$$

From (4.49), we see that the algorithm allows a range of valid values for the weight w_i. At first glance it may seem always better to use the largest valid w_i so as to get the most aggressive warm start via discounting the prior iterations by as little as possible. However, this is usually not the case in practice. Because regret matching in practice converges significantly faster than $\frac{L_i\sqrt{|A_i|}}{\sqrt{T}}$, it may be possible to get a faster practical convergence by choosing a smaller w_i, even if this results in a theoretically worse convergence rate. One option—consistent with our theory—is to use $w_i = \frac{\Phi(\vec{R}_i^T)}{\Phi(\vec{R}_i'^T)}$; this performed well in our preliminary experiments, but has the slight downside of requiring repeated calculations of the potentials. Another option is to calculate w_i by replacing L_i with average payoff in (4.52). This performed well in practice and maintains the theoretical guarantees because w_i is guaranteed to be within the correct range. An additional benefit is that this way we express w_i as a function of the largest change in regret, which is typically easy to

bound—an aspect we will leverage in the next section. Therefore, in the experiments in the rest of this paper we calculate w_i according to (4.52) with estimated average payoff instead of L_i.[5]

4.2.3 Generalization to Extensive-Form Games

We now present a corollary that the same algorithm can be applied to extensive-form games when solved using CFR.

Corollary 5. *Let Γ be an extensive-form game. Suppose player i has played T CFR iterations in Γ. Assume that all players play the exact same sequence of strategies in some other game Γ' with identical structure but potentially different payoffs. We define for each information set $I_i \in \mathcal{I}_i$*

$$L_{I_i} = \frac{\max\left\{\max_a[R^T(I_i,a) - R'^T(I_i,a)],\ 0\right\}}{T}$$

We also define w_{I_i} using L_{I_i} according to Theorem 16. Let $w_i = min_{I_i}\{w_{I_i}\}$. If we scale the payoffs of the T iterations by w_i and then play according to CFR for T_2 iterations, then weighted average regret for player i is bounded by

$$\frac{R_{w,i}^{T,T_2}}{w_iT + T_2} \leq \frac{L_i|\mathcal{I}_i|\sqrt{|A_i|}}{\sqrt{w_iT + T_2}} \tag{4.54}$$

If Γ' is a two-player zero-sum game, then the weighted average strategies form an ϵ-Nash equilibrium, with $\epsilon = \frac{R_{w,1}^{T,T_2}}{w_1T+T_2} + \frac{R_{w,2}^{T,T_2}}{w_2T+T_2}$.

Recall that in regret transfer we can replay the iterations in the new game in $O(1)$ by storing regrets as a function of $\vec{\theta}$. For example, in the case where $\vec{\theta}$ is one-dimensional, we would need to store two values for regret instead of one, and therefore require twice as much memory. However, in extensive-form games, not every information set may be affected by such a change in $\vec{\theta}$. If an information set's possible payoffs are all constant (independent of $\vec{\theta}$), even though there is a variable payoff somewhere else in the game, then there is no need to use extra memory to store the coefficients on $\vec{\theta}$ at that information set. The exact amount of memory used thus depends on the structure of the game.

4.2.4 Regret Transfer Experiments

We now show experimental results on regret transfer. We use Leduc hold'em poker (described in Section 2.4.1 as the test problem here. These experiments will show average exploitability as a function of the number of iterations run.

In the experiments in this section, we consider warm starting after the allowed bet sizes in our game model have changed. We first estimated a solution to Leduc with 1 million CFR iterations. We then considered a modified form of Leduc where the bet sizes for the first and second round were slightly different. Specifically, we changed the first-round bet from 2 to 2.1 and the second-round bet from 4 to 4.1. In other words, $\vec{\theta}$ changed from $\langle 2, 4\rangle$ to $\langle 2.1, 4.1\rangle$. We tested three

[5] As a computational detail, we scale future iterations by $\frac{1}{w}$ rather than scaling previous iterations by w. Both yield identical results, but the former seems easier to implement.

approaches to solving this new game. In the first, we simply solved the game from scratch using CFR. In the second, we transferred the regret and average strategy from the original Leduc game, but did not de-weight them. Finally, in the third, we transferred regret and average strategy by de-weighting by $w = 0.125$, a value chosen by the method described in the previous section. Figure 4.5 shows the results.

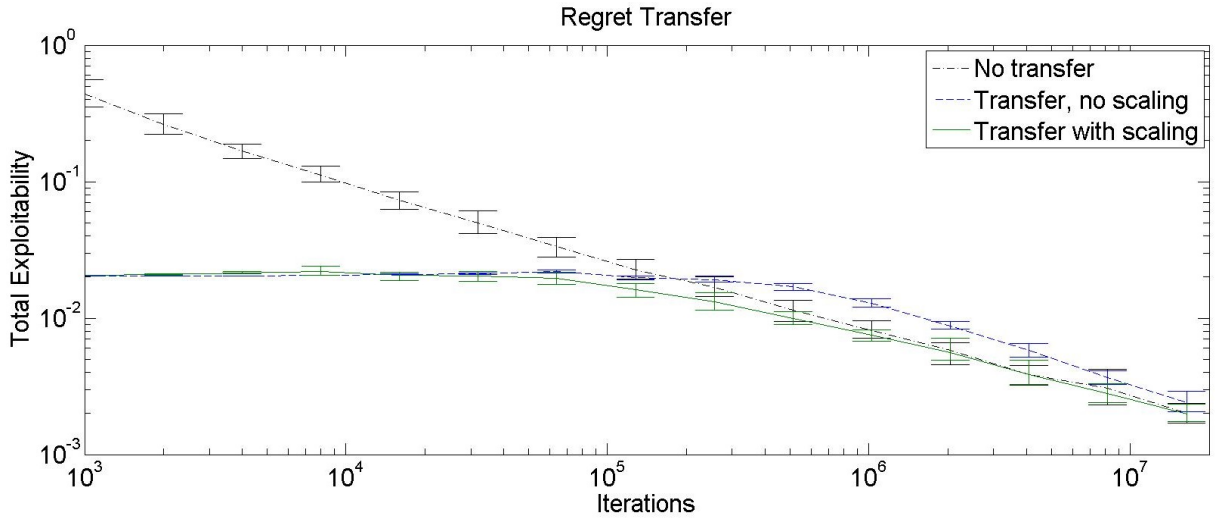


Figure 4.5: Regret transfer after increasing the bet size in both rounds of Leduc hold'em by 0.1. The average over 20 runs is shown with 95% confidence intervals. The warm start provides a benefit that is equivalent to about 125,000 iterations. In the long run, that benefit becomes visually almost imperceptible on the log scale. Unlike transferring regret without scaling, our method does not cause long-term harm.

It is clear that while transferring without scaling provides a short-term improvement, in the long-run it is actually detrimental (worse than starting from scratch). In contrast, when the transferred regret is properly weighed using our method, we see an equal improvement in the short-term without long-term detriment. Since the transfer only gives us a head-start of a fixed number of iterations, in the long run this gain, of course, becomes negligible (on a logarithmic plot). Nevertheless, if the change is small then this fixed number of iterations can be a substantial portion of the run.

4.2.5 Parameter Optimization

We now incorporate the result from the previous section into a custom gradient descent algorithm for two-player zero-sum games. Our objective is to find the value of a parameter vector $\vec{\theta}$ that results in a locally maximal value for a Nash equilibrium for a family of games with payoffs that are Lipschitz functions of $\vec{\theta}$. For example, we could be optimizing the bet sizes for a player in no-limit poker. Without loss of generality, in this section we present everything as if we are maximizing the parameter vector for Player 1.[6]

[6]The function used to maximize the parameter vector can be independent of the zero-sum game. It can be maximized for either player, or parts of it could be maximized for one player or the other. It could even be maximized

To do this, we will simultaneously solve the parameterized game using regret matching (CFR in the case of extensive-form games), storing regret in terms of $\vec{\theta}$, while running our gradient descent algorithm. Specifically, at each step s we will run some number of iterations, t_s, of regret matching. Then, for each $\theta_d \in \vec{\theta}$ we will calculate an estimated derivative $\hat{g}_{d,s}$, as we will detail later. We will then update θ_d as

$$\theta_{d,s+1} = \theta_{d,s} + \hat{g}_{d,s}\frac{\alpha}{\ell_s} \tag{4.55}$$

where ℓ_s is a learning rate (analyzed later) and α is any positive constant. We then multiply the weights of all prior steps $s' \leq s$ by w_s, where w_s is determined according to (4.49). So, the previous step ends up being discounted by w_s, the one before that by $w_s \cdot w_{s-1}$ (because it was multiplied by that previous w_s before), etc.

The number of regret matching iterations we conduct at step s is

$$t_s = \lceil Ks - w_s K(s-1) \rceil \tag{4.56}$$

for some constant K.[7] Thus, at the end of step s we will have run an equivalent of Ks regret matching iterations, and by Theorem 16 we have weighted average regret $\frac{R_{w_s,i}^{K(s-1),t_s}}{Ks} \leq \frac{L_i\sqrt{|A_i|}}{\sqrt{Ks}}$.

We compute the estimated derivative $\hat{g}_{d,s}$ (for each dimension d of $\vec{\theta}$ separately) as follows. Define $\vec{\xi}_d$ to be the unit vector in the vector space of $\vec{\theta}$ along dimension $\theta_d \in \vec{\theta}$. In concept, we will estimate the value of the game at two points $\vec{\theta}_s \pm s^{-\frac{1}{4}}\vec{\xi}_d$. We do this by running the equivalent of Ks iterations of regret matching at each of the two points. Then we use the slope between them as the estimated derivative. We later prove that this converges to the true derivative.

There is a problem, however, because $\vec{\theta}$ changes at each step and running Ks iterations from scratch becomes increasing expensive as s grows. To address this, we could transfer regret from a maintained strategy at $\vec{\theta}_s$ using Theorem 16, but even this would be too expensive because $s^{-\frac{1}{4}}$ shrinks very slowly. As a better solution to address the problem, we do the following. For each dimension d we maintain a strategy profile σ_{d-} for the game at $\vec{\theta}_s - s^{-\frac{1}{4}}\vec{\xi}_d$ and a strategy profile σ_{d+} for the game at $\vec{\theta}_s + s^{-\frac{1}{4}}\vec{\xi}_d$, and we estimate the derivative using those points, as well as transfer regret from them. At each step s, we run t_s iterations of regret matching at each of these $2N$ points. As $\vec{\theta}_s$ moves, these points move with it, so they are always at $\pm s^{-\frac{1}{4}}\vec{\xi}_d$ from it along each dimension d, respectively.

The pseudocode of the full algorithm is shown as Algorithm 3. The following theorem proves that Algorithm 3 is correct and shows its speed advantage.

Theorem 17. *Let $\Gamma_{\vec{\theta}}$ be a family of two-player zero-sum games with identical structure. Assume each payoff is, across games, an identical function of some N-dimensional parameter vector $\vec{\theta}$ that is bounded so that $\forall d$, $\theta_d \in [a_d, b_d]$. Assume also that $\forall d$, these functions are Lipschitz continuous in θ_d. Let the learning rate ℓ_s be such that $\ell_s = \Omega(\sqrt{s})$ and $\frac{1}{\ell_s}$ diverges as $s \to \infty$. Define $v_i^*(\vec{\theta})$ to be the Nash equilibrium value of $\Gamma_{\vec{\theta}}$. As $s \to \infty$, Algorithm 3 converges to a locally optimal $\vec{\theta}$ with respect to $v_i^*(\vec{\theta})$, and to a Nash equilibrium strategy profile at that $\vec{\theta}$.*

to the preferences of some third party.

[7] Theoretically, any positive constant K works, but in practice K could be chosen to ensure the overhead of stepping (conducting the re-weighting, calculating the gradient, etc.) is small compared to the cost of conducting the regret matching iterations.

Algorithm 3 Parameter optimization in two-player zero-sum games

Choose K, ℓ_s
Choose $U \leq L_i$ // In the experiments we used average payoff of the player.
Initialize $s \leftarrow 1$, $t \leftarrow 0$
Initialize N-dimensional parameter vector $\vec{\theta}$
Initialize $\Theta \leftarrow \{\vec{\theta}_{d-} = \vec{\theta} - \vec{\xi}_d, \vec{\theta}_{d+} = \vec{\theta} + \vec{\xi}_d : d \in \{1...N\}\}$
Initialize avg regret tables: $\forall p \in \Theta\ \forall I\ \forall a$, $r_p(I, a) \leftarrow 0$
Initialize avg strategy tables: $\forall p \in \Theta\ \forall I\ \forall a$, $\sigma_p(I, a) \leftarrow 0$
loop
  while $t < sK$ **do**
    for all $p \in \Theta$ **do**
      $r_p, \sigma_p \leftarrow$ *One-Iteration-of-Regret-Matching*(r_p)
    $t \leftarrow t + 1$
  for all d in 1 to N **do** // Loop over the parameters
    $\hat{g}_d \leftarrow \frac{v_i(\sigma_{\vec{\theta}_{d+}}) - v_i(\sigma_{\vec{\theta}_{d-}})}{2s^{-\frac{1}{4}}}$ // Estimate derivative wrt. θ_d
    $\theta'_d \leftarrow \theta_d + \hat{g}_d \frac{\alpha}{\ell(s)}$ // Revise value of parameter θ_d
  for all d in 1 to N **do** // Narrow the interval around each θ_d
    $\vec{\theta}_{d-} \leftarrow \vec{\theta'} - \vec{\xi}_d(s+1)^{-\frac{1}{4}}$
    $\vec{\theta}_{d+} \leftarrow \vec{\theta'} + \vec{\xi}_d(s+1)^{-\frac{1}{4}}$
  $L \leftarrow \max\left\{\max_{p\in\Theta, I\in\mathcal{I}, a\in A}\left\{r'_p(I,a) - r_p(I,a)\right\}, 0\right\}$
  $w \leftarrow \frac{1}{1+\frac{2L\sqrt{|A|t}}{U}+\frac{L^2 t}{U^2}}$ // Calculate weight based on Theorem 16
  for all $p \in \Theta, I \in \mathcal{I}, a \in A$ **do**
    $r_p(I, a) \leftarrow w r_p(I, a)$ // De-weight old regret
    $\sigma_p(I, a) \leftarrow w\sigma_p(I, a)$ // De-weight old average strategy
  $\vec{\theta} \leftarrow \vec{\theta'}, \quad t \leftarrow wt, \quad s \leftarrow s + 1$

Let

$$\hat{g}_{d,s} = \frac{v_i(\sigma_{d+}) - v_i(\sigma_{d-})}{2s^{-\frac{1}{4}}} \tag{4.57}$$

and let $\hat{g}_s = \max_d\ \hat{g}_{d,s}$. *At each step* s, *Algorithm 3 conducts* $O(s^{\frac{3}{2}}\hat{g}_s\frac{1}{\ell_s})$ *iterations of regret matching.*

This represents a substantial improvement over naïve gradient descent. If we were to do gradient descent without regret transfer, we would require $\Theta(s)$ iterations at each step, and thus take $\Theta(s^2)$ time. With regret transfer, however, if we use a learning rate $\ell_s = s$, and even if the gradient did not converge at any significant rate (although in reality it does), we only do $O(\sqrt{s})$ iterations at each step, thus taking $O(s\sqrt{s})$ time overall.

4.2.6 Parameter Optimization Experiments

Leduc Hold'em

As we mentioned in the regret transfer experiments, one can view Leduc as having two parameters: the first-round bet size θ_1 and the second-round bet size θ_2.

In the first experiment here, we held θ_1 fixed at 2.0 (the standard in Leduc), and ran Algorithm 3 to optimize θ_2. We used a learning rate $l_s = s^{\frac{3}{4}}$, $\alpha = 50$, and $K = 100$. We conducted three runs of the algorithm, starting from three different initial values for θ_2, respectively. Each run converged to $\theta_2 = 9.75 \pm 0.01$ within 10^8 iterations (Figure 4.6).

As a sanity check, we ran CFR on 41 models of the game with different values for θ_2. Specifically, we checked values for θ_2 in the set $\{3.0, 3.25, ..., 12.75, 13.0\}$. Indeed, $\theta_2 = 9.75$ maximized Player 1's payoff.

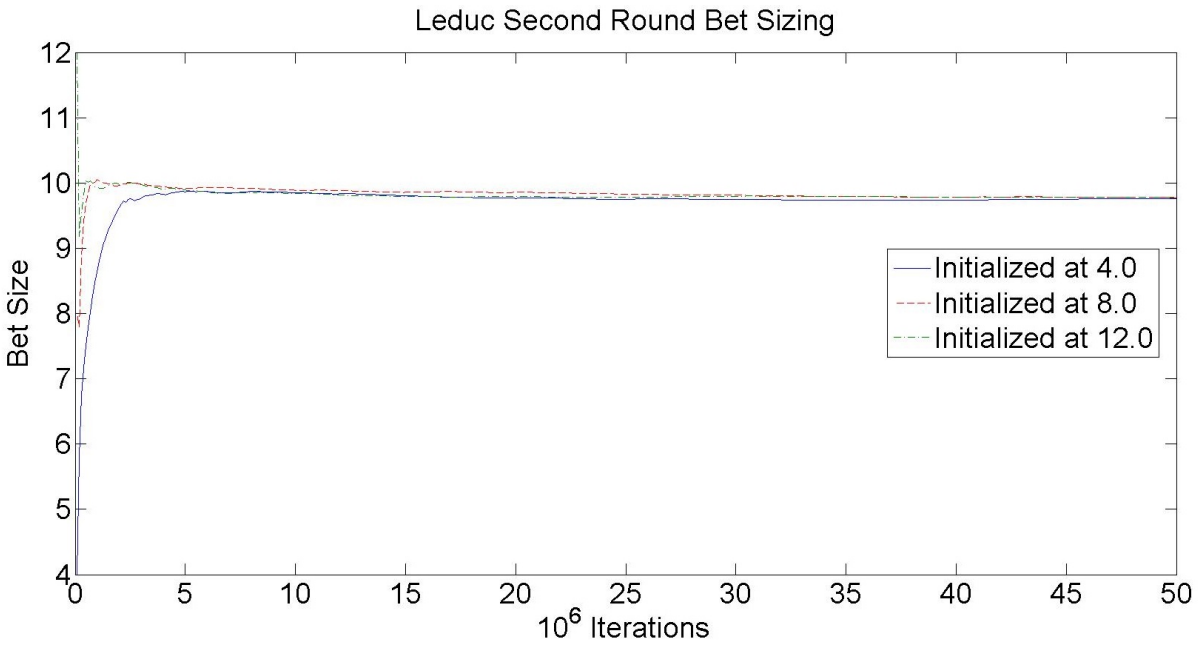


Figure 4.6: Parameter optimization where θ is the second-round bet size in Leduc hold'em.

In the next experiment, we ran Algorithm 3 to simultaneously optimize θ_1 and θ_2. The same learning rate, α, and K were used as in the previous experiment. Three initial points were chosen: $(\theta_1, \theta_2) = (2.0, 4.0)$, $(4.0, 2.0)$, and $(4.0, 8.0)$. Within $5 \cdot 10^8$ iterations, all runs converged to $\theta_1 = 1.69 \pm 0.01$ and $\theta_2 = 8.56 \pm 0.01$. The results of these experiments are shown in Figure 4.7, with θ_1 on the bottom and θ_2 on the top. The value of the game at the converged $(\theta_1, \theta_2) = (1.69, 8.56)$ was 0.1645 ± 0.002. This was an improvement over the 1-dimensional optimization in the previous experiment, which fixed $\theta_1 = 2.0$ and converged to $\theta_2 = 9.75$. The value of the game there was 0.1611 ± 0.002.

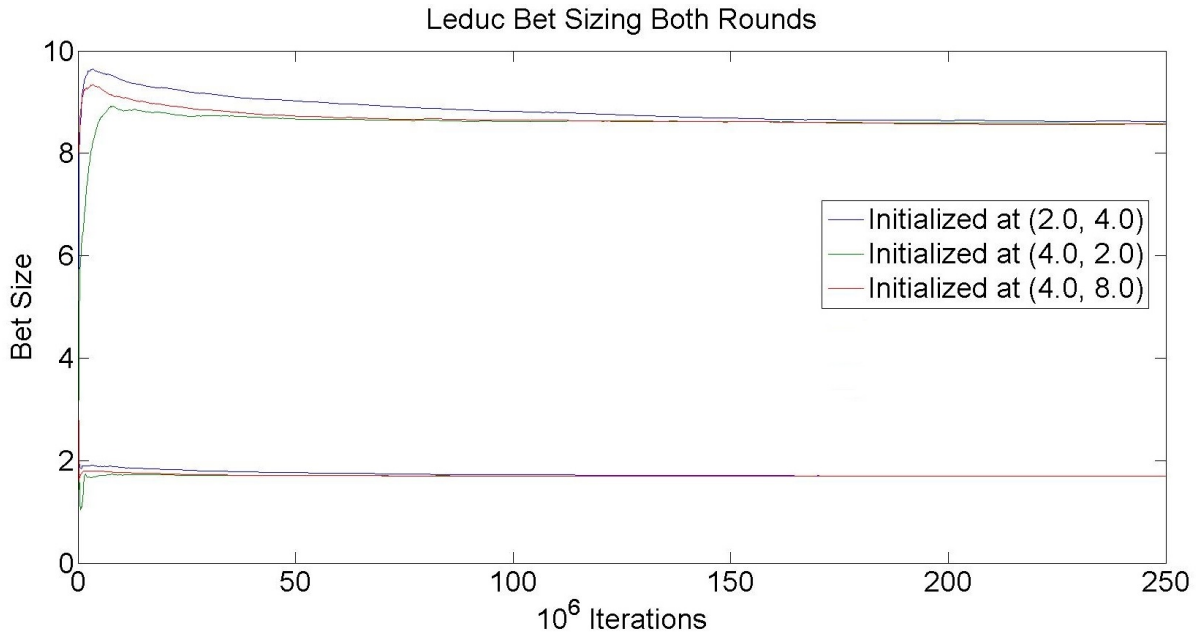


Figure 4.7: Parameter optimization where θ_1 is the first-round bet size in Leduc, and θ_2 is the second-round bet size.

No-Limit Texas Hold'em

We also provide results demonstrating that our algorithm even scales to two-player no-limit Texas hold'em poker (described in Section 2.4.3.

The following experiments were done with the betting abstraction used by Tartanian5 in the 2012 Annual Computer Poker Competition. For the card abstraction, we used the leading algorithm [82] for generating an imperfect-recall abstraction. We used no abstraction on the first round of the game (aka "preflop"), i.e., 169 buckets, and 200 buckets for each of the remaining three rounds. This resulted in an abstracted game with roughly 1.4 million information sets.

The 2012 betting abstraction allows the first player to act to either fold, call, raise, or go all-in. The raise amount is set to 1x the pot. This choice of bet size was based on human intuition, and may not be optimal. In these experiments we ran Algorithm 3 to find the raise amount that would be optimal for the first action of the game (in this abstraction).[8]

No-limit Texas hold'em with an imperfect-recall card abstraction posed a challenge in that evaluating $u_i(\sigma)$ is difficult in large imperfect-recall games. To get around this, we estimated

[8] As the initial bet size changed, it was necessary to adhere to the chip stack limitations in no-limit Texas hold'em. We dealt with this by maintaining the structure of the game, but limiting all payoffs to the size of the chip stack. When the initial bet size increased, all actions remained valid. However, some actions essentially had no consequence, as the chip stack limit had already been reached. When the initial bet size decreased, the all-in actions would nevertheless result in all chips being bet, up to the pre-determined chip stack limit. In this way we were able to maintain the structure of the game across parameter settings. However, when transferring regret, we did not explicitly consider the constraints of chip stacks. That is, if regret from an iteration with a parameter of 1.0 was transferred to a parameter of 1.1, then we would implicitly assume that the new chip stack was 1.1x the original. This was only the case for iterations that were transferred, and not for iterations being played at the current bet size. Moreover, as the parameter converged, this became an increasingly insignificant issue.

$u_i(\sigma)$ while running CFR at each step using the same public card samples. This resulted in some noise in the steps.[9] Nevertheless, after 10^9 iterations of chance-sampled CFR, all 4 runs converged to 0.77 ± 0.01 (Figure 4.8).

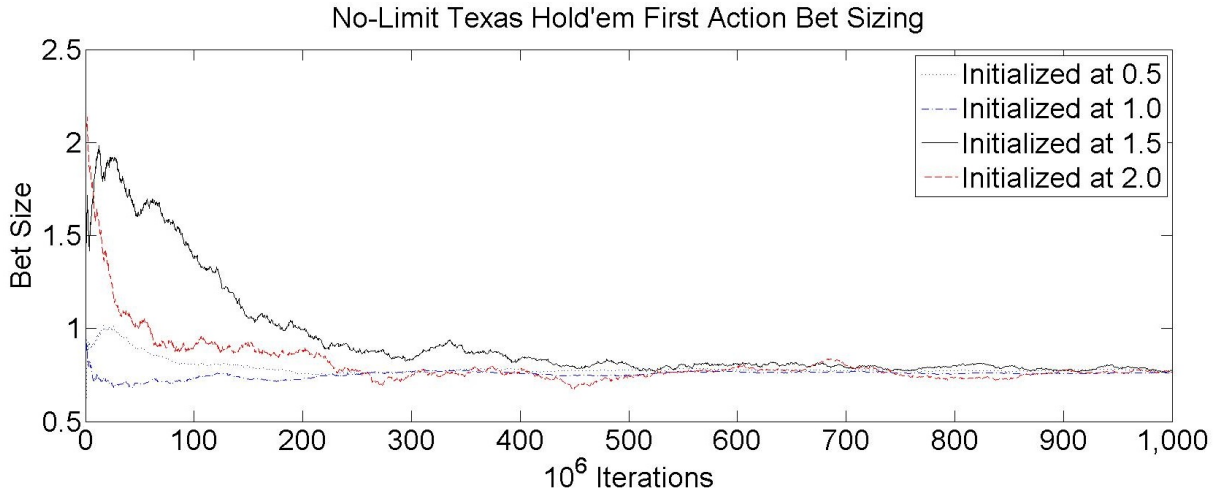


Figure 4.8: Parameter optimization where θ is the first action bet size in no-limit Texas hold'em. Runs with four different initializations are shown. The learning rate was $s^{\frac{3}{4}}$. For initializations at 0.5 and 1, $\alpha = 0.3$. For initializations at 1.5 and 2.0, $\alpha = 1.0$.

As a check, we ran CFR on models of the game with a fixed bet size in $\{0.5, 0.6, ..., 1.0\}$. Indeed, 0.8 resulted in the highest expected payoff for Player 1, followed by 0.7.

4.2.7 Conclusions

We began by proving that regrets on actions in one setting (game) can be transferred to warm start the regrets for solving a different setting with same structure but different payoffs that can be written as a function of parameters. Experiments on Leduc hold'em poker verified the benefits of the technique. It provided a significant head start over running CFR from scratch.

This is a more constrained form of warm starting than the method described in Section 3.2, but this form of warm starting is simpler, does not require a full traversal of the game tree to update regrets, and likely produces a better warm start in practice for small changes in parameters, though at the cost of increased memory usage and more constraints on when it can be used.

We then studied optimizing a parameter vector for a player in a two-player zero-sum game (e.g., optimizing bet sizes to use in poker). We proposed a custom gradient descent algorithm that provably finds a locally optimal parameter vector while leveraging our warm-start theory to significantly save regret-matching iterations at each step. It optimizes the parameter vector while simultaneously finding an equilibrium. To conduct s steps, it takes $O(s\sqrt{s})$ time (and

[9]Due to the challenge of evaluating $u_i(\sigma)$ in large imperfect-recall games, K was set to a minimum of 5,000 and a maximum of 200,000. We stepped when we had statistical significance that the derivative was of the right sign; otherwise we stepped at 200,000.

significantly less than that in practice) while straightforward gradient descent takes $\Theta(s^2)$ to conduct those same steps.

We ran experiments in no-limit Leduc hold'em and no-limit Texas hold'em to optimize bet sizing. This amounts to the first action abstraction algorithm (algorithm for selecting a small number of discrete actions to use from a continuum of actions—a key preprocessing step for solving large games using current equilibrium-finding algorithms) with convergence guarantees for extensive-form games. Very few other automated action abstraction algorithms have been presented, and the prior ones either have no convergence guarantees [67, 68] or are for a much narrower game class, stochastic games [129].

This automated action abstraction technique was ultimately used to determine the first two bet sizes for Libratus, the first AI bot to defeat top humans in no-limit Texas hold'em poker (described in Section 6.4). However, more work remains to be done in order to make the algorithm efficient enough in order to be used to determine action abstractions for an entire game. Due to repeated discounting, the algorithm converges more slowly than running CFR with fixed bet sizes. Another direction for future work is developing a way to converge to a globally optimal action abstraction, rather than just a locally optimal one.

4.2.8 Proofs of Theoretical Results

Proof of Lemma 15

Proof. From page 10 of Cesa-Bianchi and Lugosi [36], we know that for any t

$$\Phi(\vec{R}_i^t) - \Phi(\vec{R}_i^{t-1}) \leq \sum_{a \in A_i} r_i^t(a)^2$$

Summing over all iterations, we therefore have

$$\Phi(\vec{R}_i^{T_0} + \vec{R}_i^T) \leq \Phi(\vec{R}_i^{T_0}) + \sum_{t=T_0+1}^{T} \sum_{a \in A_i} r_i^t(a)^2$$

$$\Phi(\vec{R}_i^{T_0} + \vec{R}_i^T) \leq wT_0|A_i|L_i^2 + \sum_{t=T_0+1}^{T} \sum_{a \in A_i} r_i^t(a)^2$$

The maximum possible immediate regret for an action on a single iteration is L_i. Therefore,

$$\Phi(\vec{R}_i^{T_0} + \vec{R}_i^T) \leq wT_0|A_i|L_i^2 + T|A_i|L_i^2$$

□

Proof of Theorem 16

Proof. From (2.5), we have that

$$\Phi(\vec{R}_i^T) = \sum_{a \in A_i} R_i^T(a)_+^2 \leq L_i^2|A_i|T \tag{4.58}$$

After replaying the T iterations in the new game, we have

$$\Phi(\vec{R}'_{T,i}) = \sum_{a \in A_i} \left(R_i'^T(a)_+\right)^2 \tag{4.59}$$

We scale the payoffs from each of the T iterations by w_i.

$$\begin{aligned} \Phi(w_i \vec{R}_i'^T) &= \sum_{a \in A_i} \left(w_i R_i'^T(a)_+\right)^2 \\ \Phi(w_i \vec{R}_i'^T) &= w_i^2 \sum_{a \in A_i} \left(R_i'^T(a)_+\right)^2 \end{aligned} \tag{4.60}$$

Substituting the definition of w_i

$$\begin{aligned} \Phi(w_i \vec{R}_i'^T) &\leq w_i \frac{T L_i^2 |A_i|}{\Phi(\vec{R}_i'^T)} \sum_{a \in A_i} \left(R_i'^T(a)_+\right)^2 \\ \Phi(w_i \vec{R}_i'^T) &\leq w_i T L_i^2 |A_i| \frac{\sum_{a \in A_i} \left(R_i'^T(a)_+\right)^2}{\Phi(\vec{R}_i'^T)} \end{aligned}$$

Substituting (4.59), we simplify to

$$\Phi(w_i \vec{R}_i'^T) \leq w_i T L_i^2 |A_i| \tag{4.61}$$

If we play an additional T_2 iterations of regret-matching, then by Lemma 15 the bound will be

$$\Phi(w_i \vec{R}_i'^T + \vec{R}_i'^{T_2}) \leq (w_i T + T_2) L_i^2 |A_i| \tag{4.62}$$

From (2.4) this becomes

$$\begin{aligned} \sum_a \left(w_i \sum_{t=1}^{T} r_i'^t(a) + \sum_{T+1}^{T_2} r_i'^t(a)\right)^2 &\leq (w_i T + T_2) L_i^2 |A_i| \\ \max_a \left(w_i \sum_{t=1}^{T} r_i'^t(a) + \sum_{T+1}^{T_2} r_i'^t(a)\right)^2 &\leq (w_i T + T_2) L_i^2 |A_i| \\ \max_a \left(w_i \sum_{t=1}^{T} r_i'^t(a) + \sum_{T+1}^{T_2} r_i'^t(a)\right) &\leq \sqrt{w_i T + T_2} L_i \sqrt{|A_i|} \end{aligned}$$

Dividing by $w_i T + T_2$, we arrive at

$$\frac{R_{w,i}'^{T,T_2}}{w_i T + T_2} \leq \frac{L_i \sqrt{|A_i|}}{\sqrt{(w_i T + T_2)}}$$

□

Proof of Proposition 1

Proof. From (2.4) we have that

$$\Phi(\vec{R}_i^T) = \sum_{a \in A_i} R_i^T(a)_+^2 \leq L_i^2 |A_i| T \tag{4.63}$$

We then add at most L_i average regret to each action. So

$$\begin{aligned} \Phi(\vec{R}_i'^T) &\leq \sum_a (R_i^T(a)_+ + L_i)^2 \\ &\leq \sum_a [R_i^T(a)_+^2 + 2L_i R_i^T(a)_+ + L_i^2] \end{aligned}$$

Since $\sum_a R_i^T(a)_+^2$ is just our original potential function,

$$\Phi(\vec{R}_i'^T) \leq L_i^2 |A_i| T + 2L_i (\sum_a R_i^T(a)_+) + |A_i| L_i^2$$

From (2.6), we know that $R_i^T(a)_+ \leq L_i \sqrt{|A_i|} \sqrt{T}$, so therefore

$$\begin{aligned} \Phi(\vec{R}_i'^T) &\leq L_i^2 |A_i| T + 2L_i L_i |A_i|^{\frac{3}{2}} T^{\frac{3}{2}} + |A_i| L_i^2 \\ &\leq (T L_i^2 |A_i|)(1 + \frac{2L\sqrt{|A_i|}\sqrt{T}}{L_i} + \frac{L^2 T}{L_i^2}) \end{aligned}$$

and finally

$$\frac{1}{1 + \frac{2L\sqrt{|A_i|}\sqrt{T}}{L_i} + \frac{L^2 T}{L_i^2}} \leq \frac{T L_i^2 |A_i|}{\Phi(\vec{R}_i'^T)} \tag{4.64}$$

□

Proof of Corollary 1

Proof. Let σ_i^t be the strategy used by player i on time step $t \leq T$ and σ_i^{T+t} be the strategy used by player i on time step $T < t \leq T + T_2$. Since we have scaled down the payoffs of the first T iterations by $w \leq w_i$, we know from Theorem 16 and (4.48) that for each player i,

$$\frac{1}{wT + T_2} \max_{\sigma_i' \in \Sigma_i} \{ \sum_{t=1}^{T} [w(u_i(\sigma_i', \sigma_{-i}^t)) - w(u_i(\sigma_i^t, \sigma_{-i}^t))] + \sum_{t=1}^{T_2} [u_i(\sigma_i', \sigma_{-i}^{T+t}) - u_i(\sigma_i^{T+t}, \sigma_{-i}^{T+t})] \} \leq \frac{L_i |A_i|}{\sqrt{(wT + T_2)}}$$

Pulling out w and substituting ϵ_i:

$$\frac{1}{wT + T_2} \max_{\sigma_i' \in \Sigma_i} \{ w \sum_{t=1}^{T} [u_i(\sigma_i', \sigma_{-i}^t) - u_i(\sigma_i^t, \sigma_{-i}^t)] + \sum_{t=1}^{T_2} [u_i(\sigma_i', \sigma_{-i}^{T+t}) - u_i(\sigma_i^{T+t}, \sigma_{-i}^{T+t})] \} \leq \epsilon_i \tag{4.65}$$

Let $\bar{\sigma}_i$ be the *weighted average strategy* of player i, where iterations $t \leq T$ are weighted by w and the later iterations are weighted by 1. So,

$$\frac{1}{wT + T_2} \max_{\sigma'_i \in \Sigma_i} \{ w \sum_{t=1}^{T} [u_i(\sigma'_i, \sigma^t_{-i})] + \sum_{t=1}^{T_2} [u_i(\sigma'_i, \sigma^{T+t}_{-i})] \} = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \bar{\sigma}_{-i}) \quad (4.66)$$

and therefore we can rewrite Inequality (4.65) as

$$- \frac{w \sum_{t=1}^{T} u_i(\sigma^t_i, \sigma^t_{-i}) + \sum_{t=1}^{T_2} u_i(\sigma^{T+t}_i, \sigma^{T+t}_{-i})}{wT + T_2} + \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \bar{\sigma}_{-i}) \leq \epsilon_i \quad (4.67)$$

By summing the two player's versions of Inequality (4.67), the first terms cancel out (since $u_2(\cdot) = -u_1(\cdot)$), so we get

$$\max_{\sigma'_1 \in \Sigma_1} u_1(\sigma'_1, \bar{\sigma}_2) + \max_{\sigma'_2 \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma'_2) \leq \epsilon$$

where $\epsilon = \epsilon_1 + \epsilon_2$. This implies

$$u_1(\bar{\sigma}_1, \bar{\sigma}_2) + \max_{\sigma'_2 \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma'_2) \leq \epsilon$$

and so (since $u_2(\cdot) = -u_1(\cdot)$) we have

$$\max_{\sigma'_2 \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma'_2) - u_2(\bar{\sigma}_1, \bar{\sigma}_2) \leq \epsilon$$

and analogously for player 2. So, this is a ϵ-equilibrium. □

Proof of Corollary 2

Proof. Substituting counterfactual regret for regret in Theorem 16 and noting that $\forall I \; w \leq w_I$, we know that

$$\forall I \; \frac{wR'^T(I) + R'^{T_2}(I)}{wT + T_2} \leq \frac{L_i \sqrt{|A_i|}}{\sqrt{wT + T_2}}$$

From Theorem 3 in Zinkevich et al. [163], this means

$$\frac{wR'^T_i + R'^{T_2}_i}{wT + T_2} \leq \frac{L_i |\mathcal{I}| \sqrt{|A_i|}}{\sqrt{wT + T_2}}$$

$$R'^{T,T_2}_{w,i} \leq \frac{L_i |\mathcal{I}| \sqrt{|A_i|}}{\sqrt{wT + T_2}}$$

We further know that for all players

$$\max_{\sigma_i' \in \Sigma_i} \Big[\sum_{t=1}^{T} [w(u_i(\sigma_i', \sigma_{-i}^t)) - w(u_i(\sigma_i^t, \sigma_{-i}^t))] + \sum_{t=1}^{T_2} [u_i(\sigma_i', \sigma_{-i}^{T+t}) - u_i(\sigma_i^{T+t}, \sigma_{-i}^{T+t})] \Big] \leq \epsilon_i(wT + T_2)$$

Thus, the remainder of the proof to show that this results in a ϵ Nash equilibrium when Γ' is a two-player zero-sum game exactly follows Corollary 4. □

Proof of Theorem 17

Proof. First we prove that $\hat{g}_{d,s}$ converges to the true derivative $g_{d,s}$. From (4.56), we see that at each step s we have run the equivalent of sK iterations of regret matching. By (2.6) and Theorem 16, this means we can bound regret as $\frac{R_i^s}{sK} \leq \frac{L\sqrt{A}}{\sqrt{sK}}$. Additionally, it is well known that in two-player zero-sum games the strategy profile is an ϵ-equilibrium where $\epsilon = R_1^s + R_2^s$ (e.g., Zinkevich et al. [163, Thm. 2]). Since Algorithm 3 maintains strategy profiles σ_{d-} and σ_{d+} for each of the N dimensions d, and each of those strategy profiles have completed the equivalent of sK iterations of regret matching, so $|v(\sigma_{d+}) - v^*(\vec{\theta}_s + s^{-\frac{1}{4}}\vec{\xi}_d)| \leq \frac{C}{\sqrt{s}}$ and $|v(\sigma_{d-}) - v^*(\vec{\theta}_s - s^{-\frac{1}{4}}\vec{\xi}_d)| \leq \frac{C}{\sqrt{s}}$. So,

$$\begin{aligned} \hat{g}_{d,s} &= \frac{v(\sigma_{d+}) - v(\sigma_{d-})}{2s^{-\frac{1}{4}}} \\ &= \frac{v^*(\vec{\theta}_s + s^{-\frac{1}{4}}\vec{\xi}_d) - v^*(\vec{\theta}_s - s^{-\frac{1}{4}}\vec{\xi}_d) \pm O(\frac{1}{\sqrt{s}})}{2s^{-\frac{1}{4}}} \\ &= g_{d,s} \pm O(\frac{1}{s^{\frac{1}{4}}}) \end{aligned} \tag{4.68}$$

Since the gradient estimate converges to the true gradient, we have that under the standard assumptions on the learning rate and the functions presented in the theorem statement, the gradient descent will converge to a local optimum (e.g. [138]).

What remains to be proven is that at step s, Algorithm 3 conducts $O(s^{\frac{3}{2}}\hat{g}_s\frac{1}{\ell_s})$ iterations of regret matching. Since the payoffs of the games are Lipschitz functions, so $\forall\theta_d \in \vec{\theta}$, regret for each action is Lipschitz with respect to θ_d. Therefore, from (4.51),

$$L_s \leq \sum_d C_d \Big(|\theta_{d,s+1} - \theta_{d,s}| + s^{-\frac{1}{4}} - (s+1)^{-\frac{1}{4}} \Big) \tag{4.69}$$

From (4.55), $\theta_{d,s+1} - \theta_{d,s} = \hat{g}_{d,s}\frac{\alpha}{\ell_s}$. This is dependent only on $\hat{g}_{d,s}$ and ℓ_s. Since $\ell_s \in \Omega(\sqrt{s})$ and $\hat{g}_{d,s} \in O(1)$, we get the following by substituting L_s from (4.69) into (4.52):

$$w_s \in O\Big(\frac{1}{1 + \frac{\hat{g}_{d,s}\sqrt{s} + (s^{-\frac{1}{4}} - (s+1)^{-\frac{1}{4}})}{\ell_s}}\Big) \tag{4.70}$$

$$\in O\Big(\frac{\ell_s}{\ell_s + \hat{g}_{d,s}\sqrt{s} + (s^{-\frac{1}{4}} - (s+1)^{-\frac{1}{4}})}\Big) \tag{4.71}$$

Using (4.56) and that $O(s^{-\frac{1}{4}} - (s+1)^{-\frac{1}{4}}) \in O(\ell_s)$, we have

$$\begin{aligned} t_s &\in O\Big(s - \frac{s\ell_s}{\ell_s + \hat{g}_{d,s}\sqrt{s}}\Big) \\ &\in O\Big(\frac{\hat{g}_{d,s}s^{\frac{3}{2}}}{\ell_s + \hat{g}_{d,s}\sqrt{s}}\Big) \end{aligned} \tag{4.72}$$

Since $\ell_s \in \Omega(\sqrt{s})$ and $\hat{g}_{d,s} \in O(1)$, this gives us

$$t_s \in O\Big(\frac{\hat{g}_{d,s}s^{\frac{3}{2}}}{\ell_s}\Big) \tag{4.73}$$

□

4.3 Simultaneous Abstraction and Equilibrium Finding

A central challenge in solving imperfect-information games is that the game may be far too large to solve with an equilibrium-finding algorithm. For example, HUNL poker has more than 10^{165} nodes in the game tree and 10^{161} information sets [78]. For such games, *abstraction* has emerged as a key approach: a smaller, more tractable version of the game that maintains many of its strategic features is created [9, 24, 50, 57, 59, 82, 89, 97, 139]. In the abstract game, a player is constrained to behaving identically in a set of situations. The abstract game is then solved for (near-) equilibrium, and its solution (i.e., the strategies for all players) mapped back to the full game. Intuitively, an abstraction should retain "important" parts of the game as fine-grained as possible, while strategically similar or unimportant states could be grouped together. However, the key problem is that it is difficult to determine which states should be abstracted without knowing the equilibrium of the game. This begets a chicken-and-egg problem.

Another issue is that even if an equilibrium-finding algorithm in a given abstraction could determine what parts of the game are (un)important, the equilibrium-finding algorithm would have to be run from scratch on the new abstraction.

A related issue is that a given abstraction is good in the equilibrium-finding process only for some time. Coarse abstractions yield good strategies early in the run while larger, fine-grained abstractions take longer to reach reasonable strategies but yield better ones in the long run. Therefore, the abstraction size is typically hand crafted to the anticipated available run time of the equilibrium-finding algorithm using intuition and past experience with the approach.

There has been a great deal of research on generating good abstractions before equilibrium finding. Most of that work has focused on **information abstraction**, where a player is forced to ignore certain information in the game. Less research has gone into **action abstraction** (restricting the set of actions available to a player). Action abstraction has typically been done by hand using domain-specific knowledge (with some notable exceptions [89, 129]).

Beyond the manual strategy-based abstraction mentioned above, there has been some work on interleaving abstraction and equilibrium finding. Hawkin et al. [67, 68] proposed algorithms that adjust the sizes of some actions (some bet sizes in no-limit poker) in an abstraction during equilibrium finding, without convergence guarantees.

The Regret Transfer form of automated action abstraction, described in Section 4.2 presented an algorithm that adjusts action sizes during equilibrium finding in a way that guarantees convergence if the player's equilibrium value for the game is convex in the action-size vector. That approach works for adjusting a small number of action sizes but the memory usage scales linearly with the number of actions being set. The Regret Transfer approach, and no other prior approach, changes the number of actions in the abstraction and thus cannot be used for growing (refining) an abstraction.

In this section we describe an algorithm that intertwines action abstraction and equilibrium finding. It begins with a coarse abstraction and selectively adds actions that the equilibrium-finding algorithm deems important. It overcomes the chicken-and-egg problem mentioned above and does not require knowledge of how long the equilibrium-finding algorithm will be allowed to run. It can quickly—in constant time—add actions to the abstraction while provably not having to restart the equilibrium finding. Experiments show it outperforms fixed abstractions at every stage of the run: early on it improves as quickly as equilibrium finding in coarse abstractions, and later it converges to a better solution than does equilibrium finding in fine-grained abstractions.

4.3.1 Adding Actions to an Abstraction

A central challenge with abstraction is its inflexibility to change during equilibrium finding. After many iterations of equilibrium finding, one may determine that certain actions left out of the abstraction are actually important. Adding to an abstraction has been a recurring topic of research [54, 159]. Prior work by Burch et al. [31] examined how to reconstruct a Nash equilibrium strategy for a subgame after equilibrium finding completed by storing only the counterfactual values of the information sets in the roots of the subgame. Jackson [74] presented a related approach that selectively refines subgames already in the abstraction after equilibrium finding completed. However, no prior method guaranteed convergence to a Nash equilibrium if equilibrium finding continues after actions are added to an abstraction. The algorithm we present in this section builds upon these prior approaches, but provides this key theoretical guarantee.

In this section we introduce a general approach for adding actions to an abstraction at run time, and prove that we still converge to a Nash equilibrium. We discuss specifically adding subgames. That is, after some number T_0 of CFR iterations in a game Γ, we wish to add some actions leading to asubgame S, forming Γ'. A trivial way to do this is to simply restart equilibrium finding from scratch after the subgame is added. However, intuitively, if the added subgame is small, its addition should not significantly change the optimal strategy in the full game. Instead of restarting from scratch, we aim to preserve, as much as possible, what we have learned in Γ, without weakening CFR's long-term convergence guarantee.

We define the **head** of a subgame S as the union of infosets that have actions leading directly into S but are not in S. Formally, S_r is the set of histories h where $h \notin S$ and $\exists a \in A(h)$ such that either $h \cdot a \in S$, or $h \in I$ and for some history $h' \in I$, $h' \in S_r$. We also define the **greater subgame** of S as $S^* = S \cup S_r$.

The key idea to our approach is to act as if S had been in the abstraction the entire time, but that the actions in S_r leading to S were never played with positive probability. Since we never played the actions in S_r, we may have accumulated regret on those actions in excess of the bounds guaranteed by CFR. That would hurt the convergence rate. Later in this section we show

that we can overcome this excess regret by discounting the prior iterations played (and thereby their regrets as well). The factor we have to discount by increases the higher the regret is, and the more we discount, the slower the algorithm converges because it loses some of the work it did. Therefore, our goal is to ensure that the accumulated regret is as low as possible. The algorithm we present can add actions for multiple players. In our description, we assume without loss of generality that the added actions belong to Player 1 (P_1).

We now discuss how to "fill in" what happened in subgame S during those T_0 iterations. Since the actions leading to S were never played, we have that for any history $h \notin S$ and any leaf node $z \in S$, $\pi^{\sigma^t}(h, z) = 0$ for all $t \in T_0$. By the definition of counterfactual value (2.14) this means that regret for all actions in all information sets outside the greater subgame S^* remain *unchanged* after adding S. We therefore need only concern ourselves with S^*.

From (2.14) we see that $R^t(I, a) \propto \pi^{\sigma^t}_{-i}(h)$ for $h \in I$. Since S was never entered by P_1, for every history in S and every player other than P_1, $\pi^{\sigma^t}_{-i}(h) = 0$ and therefore $R^t(I, a) = 0$, regardless of what strategy they played. Normally, as discussed in Section 2.3.1 when an information set has zero regret on all its actions the information set's strategy is set to uniform random. This is obviously a poor strategy; had P_1's opponents played this way in S for all T_0 iterations, P_1 may have very high regret in S_r for not having entered S and taken advantage of this poor play. Fortunately, they need not have played randomly in S. In fact, they need not have played the same strategy on every iteration. We have complete flexibility in deciding what the other players played. Since our goal is to minimize overall regret, it makes sense to have them play strategies that would result in low regret for P_1 in S_r.

We construct an auxiliary game to determine regrets for one player and strategies for the others. Specifically, similar to the approach in Burch et al. [31], we define a **recovery game** $S_i^{\sigma_\Gamma^{1..T}}$ for a subgame S for player i of a game Γ and sequence of strategy profiles $\sigma^{1..T}$. The game consists of an initial chance node leading to a history $h^* \in S_r$ with probability[10]

$$\frac{\sum_{t \leq T} \pi^{\sigma^t}_{-i}(h^*)}{\sum_{h^* \in S_r} \sum_{t \leq T} \pi^{\sigma^t}_{-i}(h^*)} \tag{4.74}$$

If this is undefined, then all information sets in S^* have zero regret and we are done. In each $h \in S_r$, P_i has the choice of either taking the weighted average counterfactual value of the information set $I = I(h)$, $\frac{\sum_{t \leq T} v_i^{\sigma^t}(I)}{\sum_{t \leq T} \pi^{\sigma^t}_{-i}(I)}$, or of taking the action leading into S, where play continues in S until a leaf node is reached. We play $T_S = \sum_{h^* \in S_r} \sum_{t \leq T} \pi^{\sigma^t}_{-i}(h^*)$ iterations of CFR (or any other regret-minimization algorithm) in the recovery game.

We now define the **combined game** $\Gamma + S$ of a game Γ and a subgame S following play of a recovery game $S_i^{\sigma_\Gamma^{1..T}}$ as the union of Γ and S with regret and average strategy set as follows. For any action a belonging to I such that $I \cdot a \notin S$, $R_{\Gamma+S}(I, a) = R_\Gamma(I, a)$. Otherwise, if $P(I) \neq i$ then $R_{\Gamma+S}(I, a) = 0$. If $P(I) = i$ and $I \subseteq S$ then $R_{\Gamma+S}(I, a) = R_S(I, a)$. In the case that $I \cdot a \in S$ but $I \not\subseteq S$, then $R_{\Gamma+S}(I, a) = \sum_{t \leq T_S} v_i^{\sigma_S^t}(I, a) - \sum_{t \leq T} v_i^{\sigma_\Gamma^t}(I)$.

[10] In two-player games without sampling of chance nodes, $\sum_{t \leq T} \pi^{\sigma^t}_{-1}(h)$ is easily calculated as the product of $\sum_{t \leq T} \pi_2^{\sigma^t}(I)$ for the last information set I of P_2 before h, and multiplying it by $\sigma_c(a'|h')$ for all chance nodes $h' \sqsubset h$ where $h' \cdot a' \sqsubseteq h$.

If $I \in \Gamma$ then $\bar{\sigma}^T_{\Gamma+S}(I) = \bar{\sigma}^T_\Gamma(I)$. Otherwise, if $P(I) = i$ then $\bar{\sigma}^T_{\Gamma+S}(I) = \vec{0}$, and if $P(I) \neq i$ then $\bar{\sigma}^T_{\Gamma+S}(I) = \bar{\sigma}^T_S(I)$.

The theorem below proves that this is equivalent to having played a sequence of iterations in $\Gamma + S$ from the beginning. The power of this result is that if we play according to CFR separately in both the original game Γ and the recovery game $S_i^{\sigma_\Gamma^{1..T}}$, then the regret of every action in every information set of their union is bounded according to CFR (with the important exception of actions in S_r leading to S, which the algorithm handles separately as described after the theorem).

Theorem 18. *Assume T iterations were played in Γ and then T_S iterations were played in the recovery game $S_i^{\sigma_\Gamma^{1..T}}$ and these are used to initialize the combined game $\Gamma + S$. Now consider the uninitialized game Γ' identical to $\Gamma + S$. There exists a sequence of T' iterations (where $|T'| = |T_S||T|$) in Γ' such that, after weighing each iteration by $\frac{1}{|T_S|}$, for any action a in any information set $I \in \Gamma'$, $R^{T'}_{\Gamma'}(I, a) = R^T_{\Gamma+S}(I, a)$ and $\bar{\sigma}^{T'}_{\Gamma'}(I) = \bar{\sigma}^T_{\Gamma+S}(I)$.*

As mentioned earlier, if we played according to CFR in both the original abstraction and the recovery game, then we can ensure regret for every action in every information set in the expanded abstraction is under the bound for CFR, with the important exception of actions a in information sets $I \in S_r$ such that for $h \in I$, $h \cdot a \in S$. This excessive regret can hurt convergence in the entire game.

Fortunately, in Section 4.2 we showed that if an information set I exceeds a bound on regret, then one can discount all prior iterations in order to maintain CFR's guarantees on performance. However, the result from Section 4.2 requires all information sets to be scaled according to the highest-regret information set in the game. This is problematic in large games where a small rarely-reached information set may perform poorly and exceed its bound significantly. We improve upon this in the theorem below.

Recall the definitions of **weighted regret**

$$R^{T,T_2}_{w,i}(a) = w \sum_{t=1}^{T} r_i^t(a) + \sum_{t=1}^{T_2} r_i^t(a) \tag{4.75}$$

and **weighted average strategy**

$$p_{\sigma^{T,T_2}_{w,i}}(a) = \frac{w \sum_{t=1}^{T} p_{\sigma_i^t}(a) + \sum_{t=1}^{T_2} p_{\sigma_i^t}(a)}{wT + T_2} \tag{4.76}$$

Theorem 19. *Suppose T iterations were played in some game. Choose any weight w_i such that*

$$0 \leq w_i \leq \min\left\{1, \frac{|\mathcal{I}_i| L_i^2 |A_i| T}{\sum_{I \in \mathcal{I}_i} \sum_{a \in A(I)} \left(R^T_+(I, a)\right)^2}\right\} \tag{4.77}$$

If we weigh the T iterations by w_i, then after T' additional iterations of CFR,

$$R^{T,T'}_{w_i,i} \leq |\mathcal{I}_i| L_i \sqrt{|A_i|} \sqrt{w_i T + T'}$$

where $T' > \max\{T, \frac{\max_I \sum_{a \in A(I)} R^T_+(I,a)^2}{L^2 |A|}\}$.

Corollary 6. *In a two-player zero-sum game, if we weigh the T iterations by $w = \min_i\{w_i\}$, then after T' additional iterations, the players' weighted average strategies constitute a 2ϵ-equilibrium where*

$$\epsilon = \max_i \frac{|\mathcal{I}_i| L_i \sqrt{|A_i|}}{\sqrt{wT + T'}}$$

While using the largest w that our theory allows may seem optimal according to the theory, better performance is achieved in practice by using a lower w. This is because CFR tends to converge faster than its theoretical bound. Say we add subgame S to an abstraction Γ using a recovery game to form $\Gamma + S$. Let S_i, Γ_i, and $(\Gamma + S)_i$ represent the information sets in each game belonging to player i. Then, based on experiments, we recommend using

$$w_i = \frac{\sum_{I \in \Gamma_i} \sum_a \left(R_+^T(I,a)\right)^2 + \sum_{I \in S_i} \sum_a \left(R_+^T(I,a)\right)^2}{\sum_{I \in (\Gamma+S)_i} \sum_a \left(R_+^T(I,a)\right)^2} \tag{4.78}$$

where the numerator uses the regret of $I \subseteq S_r$ before adding the subgame, and the denominator uses its regret after adding the subgame. This value of w_i also satisfies our theory.

4.3.2 Adding Actions with Regret Transfer

In certain games, it is possible to bypass the recovery game and add subtrees in $O(1)$. We accomplish this with the regret transfer approach described in Section 4.2, which allows regret to be transferred from one game to another in $O(1)$ in special cases.

Suppose we have a subgame S_1 in Γ and now wish to add a new subgame S_2 that has identical structure as S_1. Instead of playing according to the recovery game, we could (hypothetically) record the strategies played in S_1 on every iteration, and repeat those strategies in S_2. Of course, this would require huge amounts of memory, and provide no benefit over simply playing new strategies in S_2 through the recovery game. However, it turns out that this repetition can be done in $O(1)$ time in certain games.

Suppose all payoffs in S_1 are functions of a vector $\vec{\theta}_1$. For example, in the case of $\vec{\theta}_1$ being a scalar, one payoff might be $u_i(z_1) = \alpha_{i,z}\theta_1 + \beta_{i,z}$. Now suppose the payoffs in S_2 are identical to their corresponding payoffs in S_1, but are functions of $\vec{\theta}_2$ instead of $\vec{\theta}_1$. The corresponding payoff in S_2 would be $u_i(z_2) = \alpha_{i,z}\theta_2 + \beta_{i,z}$. Suppose we play T iterations and store regret in S_1^* as a function of $\vec{\theta}_1$. Then we can immediately "repeat" in S_2 the T iterations that were done in S_1 by simply copying over the regrets in S_1 that were stored as a function of $\vec{\theta}_1$, and replacing $\vec{\theta}_1$ with $\vec{\theta}_2$.

The regret transfer method described in Section 4.2 stored regret for the entire game as a function of $\vec{\theta}_1$. For SAEF, we can just store the regret in S_1^* as a function of $\vec{\theta}_1$. This is because when we transfer to S_2, the "replaying" of iterations in S_2 has no effect on the rest of the game outside of S_2^*. Moreover, if $\vec{\theta}_1$ is entirely determined by one player, say P_1, then there is no need to store regret for P_2 as a function of $\vec{\theta}$, because P_2's regret will be set to 0 whenever we add a subgame for P_1.

Regret transfer can be extremely useful. For example, suppose whenever P_1 takes an action that sets $\vec{\theta}_1$, that in any subsequent information set belonging to P_1 all reachable payoffs are

multiplied by $\vec{\theta}_1$. Then subsequent regrets need not be stored as a function of $\vec{\theta}_1$, only the information sets that can choose $\vec{\theta}_1$. This is very useful in games like poker, where bets are viewed as multiplying the size of the pot and after P_1 bets, if P_2 does not immediately fold, then every payoff is multiplied by the size of the bet. In that case, regret for an action need only be stored as a function of that action's bet size.

It is also not strictly necessary for the structure of the subgames to be identical. If S_2 has additional actions that are not present in S_1, one could recursively add subgames by first adding the portion of S_2 that is identical to S_1, and then adding the additional actions either with regret transfer internally in S_2, or with a recovery game. However, if S_2 has fewer actions than S_1, then applying regret transfer would imply that illegal actions were taken, which would invalidate the theoretical guarantees.

Typically, slightly less discounting is required if one uses the recovery game. Moreover, regret transfer requires extra memory to store regret as a function of $\vec{\theta}$. However, being able to add a subgame in $O(1)$ is extremely beneficial, particularly for large subgames.

4.3.3 Computing Exploitability in Games with Continuous Action Spaces

In order to calculate the exploitability of an abstract strategy profile in the full game, it is necessary to define the player's strategy in situations that do not arise in the abstraction—because the opponent(s) (and perhaps also chance) may take actions that are not included in the abstraction. Typically, this is accomplished by mapping an action not in the abstraction to one that is. This is referred to as **action translation**. Empirical results have shown that the randomized pseudo-harmonic mapping [49] performs best among action translation techniques, though in Section 5.1 we discuss alternatives to action translation.

To calculate exploitability in a game, it is typically necessary to traverse the entire game. This is infeasible in large and infinite games. However, in situations where a player maps a range of actions to a single abstract action, it may be possible to express the exploitability of each action as a function whose maximum is easy to find. With that we can calculate exact exploitability in the original (unabstracted) game by traversing only the abstraction.

We now define one class of such games. Consider the case of an abstraction that maps a range of full-game actions $[L_I, U_I] \subset \mathbb{R}$ in I to a single abstract action a, and suppose an action $\theta \in [L_I, U_I]$ is taken. Suppose further that for every information set I' in the abstraction belonging to $P(I)$ and reachable from I following a, any payoff z that can be reached from I' has a payoff that is multiplied by θ. That is, for any history $h' \in I'$ and $z \in Z$ such that $\pi(h', z) > 0$, we have $u_{P(I)}(z) = \theta u'_{P(I)}(z)$. Since the abstraction maps all actions $\theta \in [L_I, U_I]$ to the same state, the abstraction will play identically regardless of which θ is chosen. With that in mind, since every reachable payoff is scaled identically, each choice of θ results in a strategically identical situation for $P(I)$. Thus, rather than choosing a specific θ, we can instead choose the entire range $[L_I, U_I]$. Our traversal will then return the entire expected payoff as a function of θ, and we can then choose the value that would maximize the function.

We use this approach in our full-game exploitability calculation, allowing us to calculate exploitability in a full game of infinite size.

4.3.4 Where and When to Add Actions?

In previous sections we covered how one can add actions to an abstraction during equilibrium finding. In this section, we discuss where in the game, and when, to add actions.

For this section we will assume that each iteration of CFR takes $\mathcal{O}(|\mathcal{H}|)$ time. Modern implementations of CFR can traverse game trees faster. For example, Monte Carlo CFR can, in certain "balanced" games, conduct an iteration in $\mathcal{O}(\sqrt{|\mathcal{H}|})$. Vector-based implementations of CFR (introduced in Johanson et al. [79]) have a more complex computational cost that is usually $\mathcal{O}(|\mathcal{I}| \ln |\mathcal{I}|$. The ideas in this section can be adapted depending on the computational complexity of the implementation of CFR.

Regardless of how CFR is implemented, if useless subgames are added then the amount of time each iteration will take will be longer. We therefore need some method of determining when it is worthwhile to add an action to an abstraction. Generally our goal in regret-minimization algorithms is to keep average overall regret low. Thus, the decision of where and when to add an infoset will depend on how best we can minimize overall regret. Regret in information sets where we play CFR is $\mathcal{O}(\sqrt{T})$, while regret in information sets not played according to CFR is $\mathcal{O}(T)$. So, intuitively, if an information set has low regret, we would do a better job of minimizing average regret by not including it in the abstraction and doing faster iterations. But as it accumulates regret in $\mathcal{O}(T)$, eventually we could better minimize average regret by including it in the abstraction.

Following this intuition, we propose the following formula for determining when to add an action. Essentially, it determines when the derivative of summed average regret, taken with respect to the number of nodes traversed, would be more negative with the subgame added. It assumes that regrets for actions not in the abstraction grow at rate $\sim T$ while all other regrets grow at rate $\sim \sqrt{T}$.

Proposition 2. *Consider a game $\Gamma + S$ consisting of a main game Γ and subgame S. Assume a player i begins by playing CFR only in Γ, so that each iteration takes $O(|\Gamma|)$, but at any time may choose to also play CFR in S (after which each iteration takes $O(|\Gamma| + |S|)$). Assume that when playing CFR on an information set I, squared regret for an action a where $\forall h \in I$, $h \cdot a \in \Gamma$ grows by a fixed amount every iteration: $\left(R^{T+1}(I,a)\right)^2 = \left(R^T(I,a)\right)^2 + C_I$ for some constant C_I. Assume that for others actions $\left(R^T(I,a)\right)^2 = C_I T^2$. Then the optimal point to begin playing CFR in S is on the earliest iteration T where*

$$\frac{\sum_{I \in \mathcal{I}_{\Gamma,i}} R^T(I)}{|\Gamma|} < \frac{\sum_{I \in \mathcal{I}_{\Gamma,i}} R^T(I) + \sum_{I' \in \mathcal{I}_{S,i}} \left(2R^T(I') - \frac{R^T(I')}{T}\right)}{|\Gamma| + |S|}$$

This proposition relies on two important assumptions: 1) we know how fast regret will grow (and that it grows at the rate specified in the proposition), and 2) we can calculate regret for information sets outside the abstraction. Generally, it is not possible to precisely know the growth rate of regret. In our experiments, we use the rate of growth in regret up to the current iteration as an estimate of future growth. It is possible that better heuristics can be constructed depending on the domain. For example, using the rate of growth over only the most recent iterations, or some weighted average of that form, may provide a more accurate measurement. In our experiments, we found that the speed of our algorithm can be enhanced by making the condition

in Proposition 2 slightly stronger by increasing the left hand side by a small amount (1% was a good value in our experiments, as we will discuss).

We can estimate regret for information sets outside the abstraction. Suppose we want to calculate $\sum_{t \leq T} v_i^{\sigma^t}(I, a)$ for some action a leading to a subgame not in our abstraction. The opponents must have some defined strategies following this action. We can calculate a best response against those strategies. Since we could have played that best response on each iteration, we can calculate an upper bound on regret for a by multiplying the counterfactual value from the best response by the number of iterations. This approach can be applied to any finite game, and can even be used to evaluate all actions in some infinite games, such as those defined in Section 4.3.3. In special cases, we can also use regret transfer to provide an instantaneous measure of regret using the approach described in Section 4.3.2.

In general, games involving abstraction exhibit **abstraction pathology**: a Nash equilibrium computed in a finer-grained abstraction can be further from the full-game Nash equilibrium than a Nash equilibrium computed in a coarser abstraction [160]. Since the method described in this section examines regret in the *full* game when considering adding actions, it ensures eventual *convergence to a Nash equilibrium in the full game*! Any full-game action experiencing linear growth in regret would, by design, eventually be added to the abstraction. Thus, any "weak points" of the abstraction in the full game are quickly addressed by including them in the abstraction.

4.3.5 Removing Actions from an Abstraction

One potential problem with adding many actions to the abstraction is that some may later turn out to not be as important as we thought. In that case, we may want to remove actions from the abstraction in order to traverse the game faster. There are a few ways to accomplish this.

First, even in vanilla CFR, there are situations where we can effectively remove subgames. If for every player i, the probability on iteration t of reaching history h, $\pi_i^{\sigma^t}(h)$, is zero, then regret and average strategy will not be updated for any player. In that case, there is no need to traverse the descendants of h. If the path leading to a given subgame has, for each player, an action belonging to that player with sufficiently negative regret, then it may make sense to "archive" the subgame by removing it from memory and storing it on disk. If CFR updates the regrets on that path so that at least one player has positive probability of reaching the subgame, then we can bring the subgame back into memory. In the experiments we use only this first action removal method (and we actually do not use disk but RAM).

A temporary method for removing subgames would be to apply regret-based pruning, which was described in Section 3.3. A more permanent method for removing subgames would be to apply a combination of dynamic thresholding (described in Section 3.4) and best-response pruning (described in Section 3.5). Specifically, if the average strategy reaches a subgame with probability less than $\frac{C}{\sqrt{T}}$ for some constant C and the condition for best response pruning is satisfied, then the subgame can be completely wiped from the game tree and its memory freed. If, in the future, the subgame ends up being reached with positive probability in the Nash equilibrium, then it would eventually be added again through the algorithm in Section 4.3.4.

4.3.6 Experiments

We tested our algorithm on a game we coin **continuous Leduc hold'em** (CLH), a modification of regular Leduc hold'em (described in Section 2.4.1). CLH is identical to traditional Leduc except a player may bet or raise any real amount between 1% of the pot and 100% of the pot. (There are no "chip stacks" in this game.)

We created three fixed abstractions of CLH. All bet sizes were viewed as fractions of the pot. All abstractions contained a fold and call action. Abstraction *Branch-2* included a min bet and max bet at every information set. *Branch-3* additionally contained a bet size of $\frac{1}{3}$, selected according to the pseudo-harmonic mapping. *Branch-5* further contained $\frac{1}{7}$ and $\frac{3}{5}$, again selected by the pseudo-harmonic mapping.

We initialized the abstraction that was used in automated action addition to contain only the minimum and maximum possible bet at each information set (in addition to fold and call). We ran vanilla CFR on each abstraction. The automated abstractions considered adding actions every 5 iterations according to the heuristic presented in Section 4.3.4 using regret transfer to estimate regret. As mentioned in that section, the heuristic cannot exactly predict how regret will grow. We therefore also tested automated abstraction refinement with slightly stronger conditions for adding actions to the abstraction: Recovery-1.01 and Transfer both multiply the left term in the condition by 1.01. Such changes to the heuristic, of course, retain our theoretical guarantees. Recovery-1.0 and Recovery-1.01 use a recovery game to add IISGs as described in Section 4.3.1, while Transfer uses regret transfer as described in Section 4.3.2.

We calculated exploitability in the full continuous game assuming the randomized pseudo-harmonic action translation is used. Figure 4.9 shows that Recovery-1.01 outperformed all the fixed abstractions at every point in the run. Moreover, while the fixed abstractions leveled off in performance, the automated abstractions continued to improve throughout the run. We also tested a threshold of 1.1, which performed comparably to 1.01, while a threshold of 2.0 performed worse.

Although regret transfer allows adding a subgame in $O(1)$ time, that method performed worse than using a recovery game. This is due to the regret from adding the subgame being higher, thereby requiring more discounting of prior iterations. The "bump" in the Transfer curve in Figure 4.9 is due to a particularly poor initialization of a subgame, which required heavy discounting. However, our heuristic tended to favor adding small subgames near the bottom of the game tree. It is possible that in situations where larger subgames are added, the benefit of adding subgames in $O(1)$ would give regret transfer an advantage.

4.3.7 Conclusions

We introduced a method for adding actions to an abstraction simultaneously with equilibrium finding, while maintaining convergence guarantees. We additionally presented a method for determining strategic locations to add actions to the abstraction based on the progress of the equilibrium-finding algorithm, as well as a method for determining when to add them. In experiments, the automated abstraction algorithm outperformed all fixed abstractions at every snapshot, and does not level off in performance.

The algorithm is game independent, and is particularly useful in games with large action

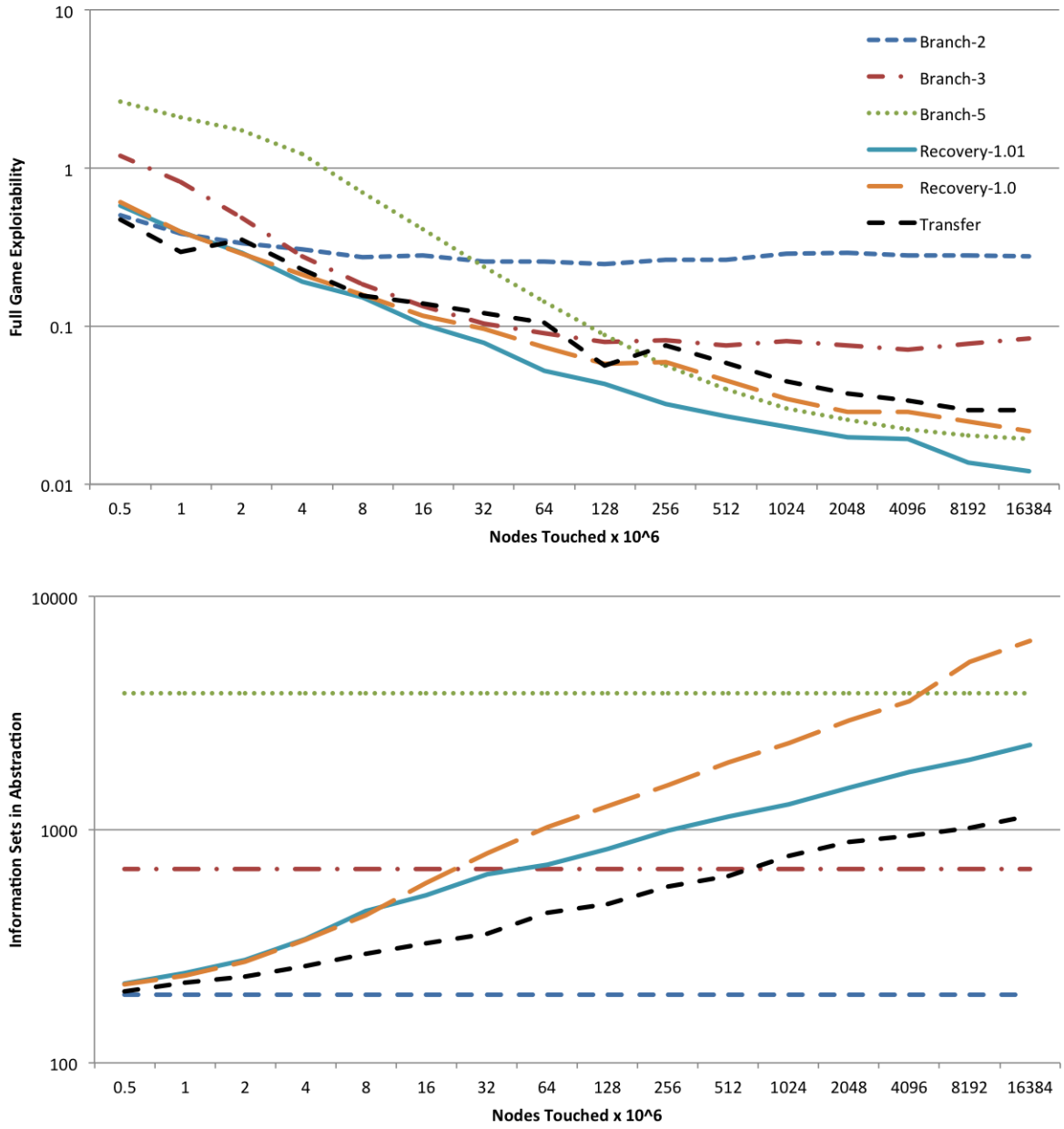


Figure 4.9: Top: Full game exploitability. Bottom: Abstraction size.

spaces. The results show that it can overcome the challenges posed by an extremely large branching factor in actions, or even an infinite one, in the search for a Nash equilibrium.

Many of these ideas were later used in Libratus's self-improvement component, in which subgames were added to its strategy offline. However, rather than compute a best response to Libratus's current strategy in order to choose actions to add, Libratus instead used its human opponents as a natural "best response". The actions that the humans most frequently selected were the ones selected to be added to the strategy.

Beyond what we have described in this section, there is still substantial room for further research. In particular, the heuristic for deciding where and when to add actions, described in Section 4.3.4, could likely be improved. Subsequent research has also expanded on this topic focusing especially on imperfect recall [35].

4.3.8 Proofs of Theoretical Results

Proof of Theorem 18

Proof. Consider the set of $|T_S T|$ ordered pairs (t_1, t_2), where $t_1 \leq T$ and $t_2 \leq T_S$. Each iteration $t' \leq T'$ will correspond to such a pair. On iteration $t' = (t_1, t_2)$ for information sets $I \in \mathcal{I}_i$ we play according to $\sigma_\Gamma^{t_1}(I)$ when $I \not\subseteq S$ and according to $\sigma_S^{t_2}(I)$ for $I \subseteq S$.

From the definition of an imperfect-information subgame, for any information set I and action a, either $\forall h \in I \; h \cdot a \in S$ or $\forall h \in I \; h \cdot a \notin S$. We first consider information sets I and actions a such that $\forall h \in I$, $h \cdot a \notin S$. Consider an arbitrary terminal node z such that $h \cdot a \sqsubseteq z$. If $z \in S$, then there is a unique history h' such that $h \cdot a \sqsubset h' \sqsubset z$ and $h' \notin S$ but there is an action a' such that $h' \cdot a' \in S$. Since t' plays according to $\sigma_\Gamma^{t_1}$ in this h', so $\pi^{\sigma_{\Gamma'}^{t'}}(h', z) = 0$ and therefore $\pi^{\sigma_{\Gamma'}^{t'}}(h \cdot a, z) = 0$ and $\pi^{\sigma_{\Gamma'}^{t'}}(h, z) = 0$.

Now suppose $z \notin S$. Then for every ancestor $h' \sqsubset z$, $h' \notin S$, so $\pi^{\sigma_{\Gamma'}^{t'}}(h, z) = \pi^{\sigma_\Gamma^{t_1}}(h, z)$. Moreover, from the definition of counterfactual regret, $R_{\Gamma'}^{T'}(I, a) = |T_S| R_\Gamma^T(I, a)$.

Now consider information sets I and actions a such that for $h \in I$, $h \cdot a \in S$. First suppose $P(I) \neq i$. Then $I \subseteq S$. But P_i plays according to $\sigma_\Gamma^{t_1}$ outside S and never enters S, so $\pi_{-i}^{\sigma_{\Gamma'}^{t'}}(I) = 0$, so $R_{\Gamma'}^{T'}(I, a) = 0$.

Now suppose $P(I) = i$. Either $I \subseteq S_r$ or $I \subseteq S$. First suppose $I \subseteq S$. From (2.16) we have that $R^{T'}(I, a) = \sum_{t' \leq T'} \left(v_i^{\sigma_{\Gamma'}^{t'}}(I, a) - v_i^{\sigma_{\Gamma'}^{t'}}(I) \right) =$

$$\sum_{t' \in T'} \sum_{h \in I} \left(\pi_{-i}^{\sigma_{\Gamma'}^{t'}}(h) \sum_{z \in Z} \left(\pi^{\sigma_{\Gamma'}^{t'}}(h, z) - \pi^{\sigma_{\Gamma'}^{t'}}(h \cdot a, z) \right) u_i(z) \right)$$

For each $h \in I$, there exists a unique history $h^* \sqsubset h$ such that $h^* \in S_r$. On iteration t', $\pi_{-i}^{\sigma_{\Gamma'}^{t'}}(h) = \pi_{-i}^{\sigma_\Gamma^{t_1}}(h^*) \pi_{-i}^{\sigma_S^{t_2}}(h^*, h)$. Moreover, since for every h' such that $h \sqsubset h'$ and $h' \sqsubset z$ we know that $h' \in S$, so $\pi^{\sigma_{\Gamma'}^{t'}}(h, z) = \pi^{\sigma_S^{t_2}}(h, z)$. So we now have that $\frac{\sum_{t' \in T'} v_i^{\sigma_{\Gamma'}^{t'}}(I)}{T_S} =$

$$\left(\sum_{t_1 \leq T} \pi_{-i}^{\sigma_\Gamma^{t_1}}(h^*) \right) \sum_{t_2 \leq T_S} \sum_{h \in I} \left(\pi_{-i}^{\sigma_S^{t_2}}(h^*, h) \left(\sum_{z \in Z} \pi^{\sigma_S^{t_2}}(h, z) \right) u_i(z) \right)$$

and we have a similar result for $\sum_{t' \leq T'} v_i^{\sigma_{\Gamma'}^{t'}}(I, a)$. In the recovery game, from (4.74) and the definition of T_S, we have that $\sum_{t_2 \leq T} \pi_{-i}^{\sigma_S^{t_2}}(h^*) = \sum_{t_1 \leq T} \pi_{-i}^{\sigma_\Gamma^{t_1}}(h^*)$. Since every other term depends on T_S and not T, so $R_{\Gamma'}^{T'}(I, a) = |T_S| R_S^{T_S}(I, a)$.

Finally, consider $I \not\subseteq S$ and $a \in A(I)$ such that for $h \in I$, $h \cdot a \in S$. Let $i = P(I)$. Since $\sigma_i^{t'}(I) = \sigma_i^{t_1}(I)$ and $\sigma_i^{t_1}(I)$ never enters S, so $\sum_{t' \leq T'} v_i^{\sigma_{\Gamma'}^{t'}}(I) = T_S \sum_{t_1 \leq T} v_i^{\sigma_\Gamma^{t_1}}(I)$. Since

$h \cdot a \in S$, so we can use $\sum_{t' \leq T} v_i^{\sigma_{\Gamma'}^{t'}}(I, a)$ from the previous case, and we get $R_{\Gamma'}^{T'}(I, a) = |T_S| R_{\Gamma+S}^{T}(I, a)$ □

Proof of Theorem 19

Proof. We begin by proving that the theorem is satisfied for $w_i = 1$ when $\sum_{I \in \mathcal{I}_i} \sum_{a \in A(I)} \left(R_+^T(I, a)\right)^2 \leq |\mathcal{I}_i| L_i^2 |A_i| T$. Since the T' iterations were played according to CFR, so from Lemma 15, we have

$$\sum_{I \in \mathcal{I}_i} \sum_{a \in A(I)} \left(R_+^{T+T'}(I, a)\right)^2 \leq |\mathcal{I}_i| L_i^2 |A_i| (T + T') \tag{4.79}$$

Using Lemma 4 from Lanctot et al. [96], we know

$$\sum_{I \in \mathcal{I}_i} \sqrt{\sum_{a \in A(I)} \left(R_+^{T+T'}(I, a)\right)^2} \leq |\mathcal{I}_i| \sqrt{L_i^2 |A_i| (T + T')} \tag{4.80}$$

Since

$$\sum_{I \in \mathcal{I}_i} \max_a R_+^{T+T'}(I, a) \leq \sum_{I \in \mathcal{I}_i} \sqrt{\sum_{a \in A(I)} \left(R_+^{T+T'}(I, a)\right)^2} \tag{4.81}$$

and since $w_i = 1$, we have $R_{+,T+T'}^{w_i}(I, a) = R_+^{T+T'}(I, a)$. Therefore, we arrive at

$$R_{w_i,i}^{T,T'} \leq |\mathcal{I}_i| L_i \sqrt{|A_i|} \sqrt{w_i T + T'} \tag{4.82}$$

We now prove that the theorem is satisfied for

$$w_i = \frac{|\mathcal{I}_i| L_i^2 |A_i| T}{\sum_{I \in \mathcal{I}_i} \sum_{a \in A(I)} \left(R_+^T(I, a)\right)^2} \tag{4.83}$$

when $\sum_{I \in \mathcal{I}_i} \sum_{a \in A(I)} \left(R_+^T(I, a)\right)^2 > |\mathcal{I}_i| L_i^2 |A_i| T$. (We will later prove that if the theorem is satisfied for some w_i, then it is satisfied for any w_i' where $0 \leq w_i' \leq w_i$). After playing the original T iterations and weighing them by w_i, we have for each $I \in \mathcal{I}_i$ that

$$w_i^2 \sum_{a \in A(I)} \left(R_+^T(I, a)\right)^2 = \sum_{a \in A(I)} \left(w_i \sum_{t=1}^{T} r_i^t(I, a)_+\right)^2$$

From Lemma 15 we have that for any information set I after an additional T' iterations,

$$\sum_{a \in A(I)} \left(R_{w_i,i,+}^{T,T'}(I, a)\right)^2 \leq \sum_{a \in A(I)} w_i^2 \left(R_+^T(I, a)\right)^2 + L_i^2 |A| T'$$

and therefore

$$R_{w_i,i}^{T,T'}(I) \leq \sqrt{w_i^2 \sum_{a \in A(I)} \left(R_+^T(I, a)\right)^2 + L_i^2 |A| T'} \tag{4.84}$$

We wish to prove that w_i satisfies

$$\sum_{I \in \mathcal{I}_i} R_{w_i,i}^{T,T'}(I) \leq |\mathcal{I}_i| L_i \sqrt{|A_i|} \sqrt{w_i T + T'} \tag{4.85}$$

Let $S_I = \frac{\sum_{a \in A(I)} R_+^T(I,a)^2}{L_i^2 |A| T}$. Combining (4.84) and (4.85), it is sufficient to prove that

$$\sum_{I \in \mathcal{I}_i} \sqrt{w_i^2 S_I T + T'} \leq |\mathcal{I}_i| \sqrt{w_i T + T'} \tag{4.86}$$

The Taylor expansion of $\sqrt{w_i T + T'}$ is

$$\sqrt{T'} + \frac{T w_i}{2\sqrt{T'}} - \frac{T^2 w_i^2}{8 T'^{\frac{3}{2}}} + \sum_{n=3}^{\infty} \frac{T^n w_i^n}{T'^{n-\frac{1}{2}}} \binom{\frac{1}{2}}{n} \tag{4.87}$$

Since $0 < w_i < 1$ and since $T' > T > 0$, the series converges. Similarly, the Taylor expansion of $\sqrt{w_i^2 S_I T + T'}$ is

$$\sqrt{T'} + \frac{T S_I w_i^2}{2\sqrt{T'}} - \frac{T^2 S_I^2 w_i^4}{8 T'^{\frac{3}{2}}} + \sum_{n=3}^{\infty} \frac{T^n S_I^n w_i^{2n}}{T'^{n-\frac{1}{2}}} \binom{\frac{1}{2}}{n} \tag{4.88}$$

Choose $T' > T \max_I \{S_I\}$. Since $0 < w_i < 1$, the series converges for all $I \in \mathcal{I}_i$. We first show that

$$\sum_{I \in \mathcal{I}_i} \left(\sum_{n=3}^{\infty} \frac{T^n S_I^n w_i^{2n}}{T'^{n-\frac{1}{2}}} \binom{\frac{1}{2}}{n} - \sum_{n=3}^{\infty} \frac{T^n w_i^n}{T'^{n-\frac{1}{2}}} \binom{\frac{1}{2}}{n} \right) \geq 0 \tag{4.89}$$

$$\sum_{n=3}^{\infty} \Big(\sum_{I \in \mathcal{I}_i} (S_I^n w_i^n - 1) \frac{T^n w_i^n}{T'^{n-\frac{1}{2}}} \binom{\frac{1}{2}}{n} \Big) \geq 0 \tag{4.90}$$

Since $w_i < 1$ and $T' > T$, so

$$\Big| \frac{T^3 w_i^3}{T'^{3-\frac{1}{2}}} \binom{\frac{1}{2}}{3} \Big| \geq \Big| \frac{T^4 w_i^4}{T'^{4-\frac{1}{2}}} \binom{\frac{1}{2}}{4} \Big|$$

Thus, it is sufficient to show $\sum_{I \in \mathcal{I}_i} (S_I^n w_i^n) \geq |\mathcal{I}_i|$ for $n \geq 3$ in order to prove (4.90). From (4.83) and the definition of S_I, we see this holds for $n \geq 1$. Thus (4.90) holds and so it would be sufficient to find w_i such that

$$\sum_{I \in \mathcal{I}_i} \big(\sqrt{T'} + \frac{T S_I w_i^2}{2\sqrt{T'}} - \frac{T^2 S_I^2 w_i^4}{8 T'^{\frac{3}{2}}} \big) \leq |\mathcal{I}_i| \big(\sqrt{T'} + \frac{T w_i}{2\sqrt{T'}} - \frac{T^2 w_i^2}{8 T'^{\frac{3}{2}}} \big)$$

Simplifying, this is equivalent to

$$\sum_{I \in \mathcal{I}_i} \big(S_I w_i - \frac{T S_I^2 w_i^3}{4T'} \big) \leq |\mathcal{I}_i| \big(1 - \frac{T w_i}{4T'} \big)$$

Rearranging terms, this is equivalent to

$$w_i \sum_{I \in \mathcal{I}_i} S_I \ + w_i \sum_{I \in \mathcal{I}_i} \big(\frac{T}{4T'} - \frac{T S_I^2 w_i^2}{4T'}\big) \leq |\mathcal{I}_i| \tag{4.91}$$

We first show that $\sum_{I \in \mathcal{I}_i} \big(\frac{T}{4T'} - \frac{T S_I^2 w_i^2}{4T'}\big) \leq 0$. Simplifying, this is equivalent to

$$\big(|\mathcal{I}_i| - w_i^2 \sum_{I \in \mathcal{I}_i} S_I^2\big) \leq 0$$

Substituting for w_i, we wish to show

$$\big(|\mathcal{I}_i| - \frac{|\mathcal{I}_i|^2 \sum_{I \in \mathcal{I}_i} S_I^2}{\big(\sum_{I \in \mathcal{I}_i} S_I\big)^2} \leq 0 \tag{4.92}$$

From Lemma 4 in Lanctot et al. [96], we know that

$$\big(\sum_{I \in \mathcal{I}_i} S_I\big)^2 \leq |\mathcal{I}_i| \sum_{I \in \mathcal{I}_i} S_I^2$$

and therefore (4.92) holds. Going back to equation (4.91), replacing (4.92) with 0 , and substituting (4.83) for w_i, we get

$$\frac{|\mathcal{I}_i| \sum_{I \in \mathcal{I}_i} S_I}{\sum_{I \in \mathcal{I}_i} S_I} + 0 \leq |\mathcal{I}_i|$$

and therefore (4.83) satisfies the theorem.

Finally, we prove that if the theorem is satisfied for some w_i, then the theorem is satisfied for any w_i' where $0 \leq w_i' \leq w_i$. Assume we satisfied (4.86) for some w_i, so that

$$\sum_{I \in \mathcal{I}_i} \sqrt{w_i^2 S_I T + X} \leq |\mathcal{I}_i| \sqrt{w_i T + X}$$

for all $X > \max\{T, \frac{\max_I \sum_{a \in A(I)} R_+^T(I,a)^2}{L^2 |A|}\}$. Then

$$\sum_{I \in \mathcal{I}_i} \sqrt{w_i^2 S_I T + \frac{w_i}{w_i'} T'} \leq |\mathcal{I}_i| \sqrt{w_i T + \frac{w_i}{w_i'} T'}$$

so

$$\sum_{I \in \mathcal{I}_i} \sqrt{\frac{w_i}{w_i'}} \sqrt{w_i w_i' S_I T + T'} \leq |\mathcal{I}_i| \sqrt{\frac{w_i}{w_i'}} \sqrt{w_i' T + T'}$$

Canceling out $\sqrt{\frac{w_i}{w_i'}}$ and recognizing $w_i' \leq w_i$, this gives

$$\sum_{I \in \mathcal{I}_i} \sqrt{w_i'^2 S_I T + T'} \leq |\mathcal{I}_i| \sqrt{w_i' T + T'}$$

Thus, w_i' satisfies the sufficient condition given in (4.86). □

Chapter 5

Search for Imperfect-Information Games

Prior to the popularization of search-based methods such as the ones described in this chapter, the standard approach to computing strategies in large imperfect-information games was to first generate an abstraction, which is a smaller version of the game that retains as much as possible the strategic characteristics of the original game [126, 127, 128]. For example, a continuous action space might be discretized. Techniques for abstraction were discussed in detail in Chapter 4. The abstract game was solved offline and its solution was used when playing the full game. We refer to the solution of an abstraction (or more generally any approximate solution to a game, such as those resulting from Deep CFR 4.1) as a **blueprint** strategy.

In heavily abstracted games, a blueprint may be far from the true solution. **Search** attempts to improve upon the blueprint by computing a more accurate equilibrium solution for an encountered subgame, while fitting its solution within the overarching blueprint. Search has led to dramatic improvements in performance for imperfect-information game AI agents, and the development of theoretically sound and empirically effective search techniques was critical to the creation of Libratus and Pluribus, which defeated top humans in no-limit Texas hold'em poker. However, as we will show, conducting search in an imperfect-information game is far more difficult than in a perfect-information game.

In this chapter we describe methods for conducting search in imperfect-information games. Section 5.1 introduces new technique for **safe** search, in which we wish to guarantee we will achieve a closer approximation of a full-game equilibrium by doing search compared to just playing the blueprint. However, Section 5.1 assumes we always search to the end of the game. Section 5.2 and Section 5.3 describe algorithms, compatible with the ones in Section 5.1, that may be used for depth-limited search.

5.1 Safe and Nested Search

In perfect-information games, determining the optimal strategy at a decision point only requires knowledge of the game tree's current node and the remaining game tree beyond that node (the *subgame* rooted at that node). This fact has been leveraged by nearly every AI for perfect-information games, including AIs that defeated top humans in chess [34] and Go [140]. In checkers, the ability to decompose the game into smaller independent subgames was even used

to solve the entire game [131].

However, determining the optimal strategy for a decision point in an imperfect-information game is a far more complex process because the game tree's exact node is typically unknown. Instead, the optimal strategy going forward depends on both players' beliefs about the state of the world. In this section we begin by showing that simply assuming all players have played a particular Nash equilibrium strategy up to the current decision point and then computing an optimal strategy going forward can lead to highly exploitable strategies. We then introduce approaches for computing strategies going forward that are provably no more exploitable than the pre-computed blueprint strategy, by incorporating information about what values are achievable in blueprint subgames. We also introduce techniques for conducting search repeatedly as play continues down the game tree, in a technique we call nested search.

An alternative approach to safe search is presented as part of ReBeL in Section 5.3.4. When ReBeL's safe search technique can be applied, it may be preferable. However, that safe search technique is, so far, only known to be theoretically sound when used with a particular form of depth-limited search (specifically, depth-limited search with a public belief state value function), whereas the search techniques described in this section can be applied whenever there is a blueprint strategy for the game.

5.1.1 Example Game: Coin Toss

In this section we provide intuition for why an imperfect-information subgame cannot be solved in isolation without information about the root probability distribution over nodes or about the values of other subgames. We demonstrate this in a simple game we call Coin Toss, shown in Figure 5.1a, which will be used as a running example throughout the chapter.

Coin Toss is played between players P_1 and P_2. The figure shows rewards only for P_1; P_2 always receives the negation of P_1's reward. A coin is flipped and lands either Heads or Tails with equal probability, but only P_1 sees the outcome. P_1 then chooses between actions "Sell" and "Play." The Sell action leads to a subgame whose details are not important, but the **expected value** (EV) of choosing the Sell action will be important. (For simplicity, one can equivalently assume *in this subsection* that Sell leads to an immediate terminal reward, where the value depends on whether the coin landed Heads or Tails). If the coin lands Heads, it is considered lucky and P_1 receives an EV of $\$0.50$ for choosing Sell. On the other hand, if the coin lands Tails, it is considered unlucky and P_1 receives an EV of $-\$0.50$ for action Sell. (That is, P_1 must on average pay $\$0.50$ to get rid of the coin). If P_1 instead chooses Play, then P_2 may guess how the coin landed. If P_2 guesses correctly, then P_1 receives a reward of $-\$1$. If P_2 guesses incorrectly, then P_1 receives $\$1$. P_2 may also forfeit, which should never be chosen but will be relevant in later sections. We wish to determine the optimal strategy for P_2 in the subgame S that occurs after P_1 chooses Play, shown in Figure 5.1a.

Were P_2 to always guess Heads, P_1 would receive $\$0.50$ for choosing Sell when the coin lands Heads, and $\$1$ for Play when it lands Tails. This would result in an average of $\$0.75$ for P_1. Alternatively, were P_2 to always guess Tails, P_1 would receive $\$1$ for choosing Play when the coin lands Heads, and $-\$0.50$ for choosing Sell when it lands Tails. This would result in an average reward of $\$0.25$ for P_1. However, P_2 would do even better by guessing Heads with 25% probability and Tails with 75% probability. In that case, P_1 could only receive $\$0.50$ (on

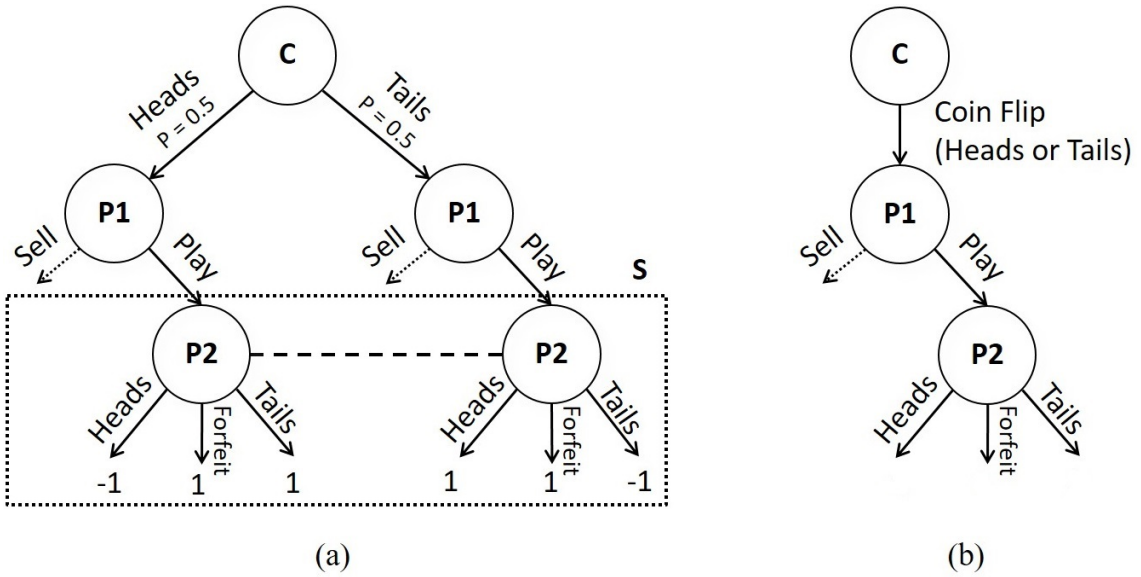


Figure 5.1: (a) The example game of Coin Toss. "C" represents a chance node. S is a Player 2 (P_2) subgame. The dotted line between the two P_2 nodes means that P_2 cannot distinguish between them. (b) The public game tree of Coin Toss. The two outcomes of the coin flip are only observed by P_1.

average) by choosing Play when the coin lands Heads—the same value received for choosing Sell. Similarly, P_1 could only receive $-\$0.50$ by choosing Play when the coin lands Tails, which is the same value received for choosing Sell. This would yield an average reward of $\$0$ for P_1. It is easy to see that this is the best P_2 can do, because P_1 can average $\$0$ by always choosing Sell. Therefore, choosing Heads with 25% probability and Tails with 75% probability is an optimal strategy for P_2 in the "Play" subgame.

Now suppose the coin is considered lucky if it lands Tails and unlucky if it lands Heads. That is, the expected reward for selling the coin when it lands Heads is now $-\$0.50$ and when it lands Tails is now $\$0.50$. It is easy to see that P_2's optimal strategy for the "Play" subgame is now to guess Heads with 75% probability and Tails with 25% probability. This shows that a player's optimal strategy in a subgame can depend on the strategies and outcomes in other parts of the game. Thus, one cannot solve a subgame using information about that subgame alone. This is a central challenge of imperfect-information games as opposed to perfect-information games.

5.1.2 Prior Approaches to Search in Imperfect-Information Games

This section reviews prior techniques for search in imperfect-information games, which we build upon. Throughout this section, we refer to the Coin Toss game shown in Figure 5.1a.

As discussed earlier, a standard approach to dealing with large imperfect-information games is to solve an abstraction of the game. The abstract solution is a (probably suboptimal) strategy profile in the full game. We refer to this full-game strategy profile as the blueprint. The goal of search is to improve upon the blueprint by changing the strategy only in a subgame. While the blueprint is frequently a Nash equilibrium (or approximate Nash equilibrium) in some abstraction of the full game, our techniques do not assume this. The blueprint can in fact be any arbitrary

strategy in the full game.

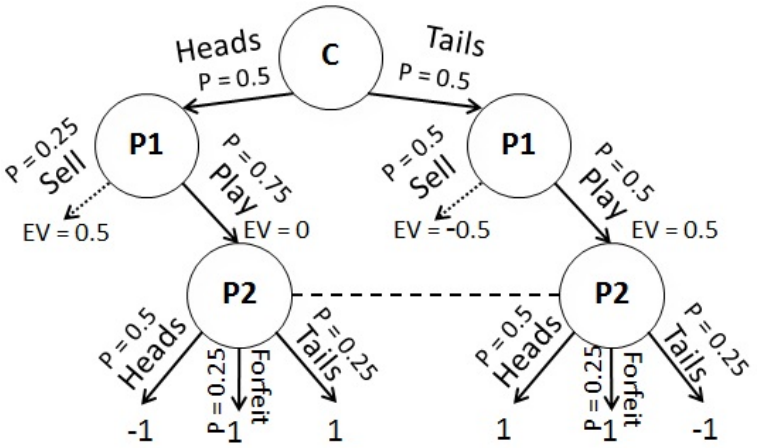


Figure 5.2: The blueprint we refer to in the game of Coin Toss. The Sell action leads to a subgame that is not displayed. Probabilities are shown for all actions. The dotted line means the two P_2 nodes share an infoset. The EV of each P_1 action is also shown.

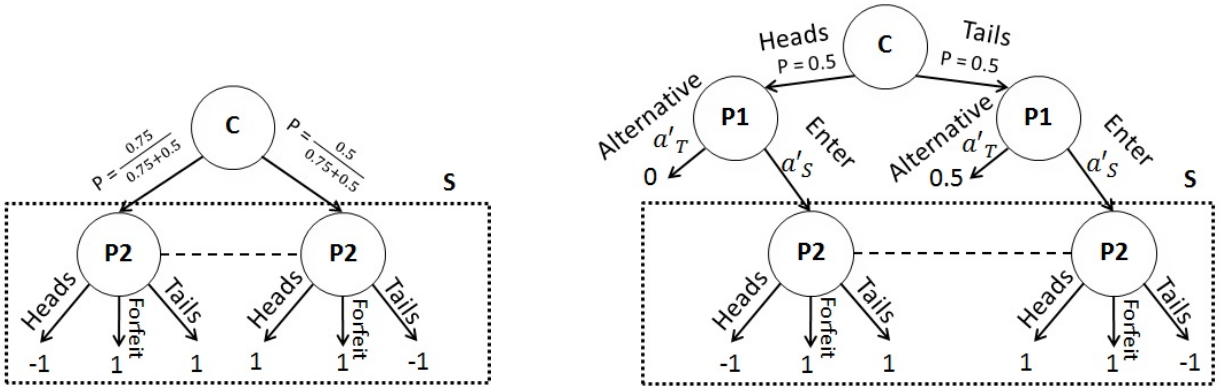
Assume that a blueprint σ (shown in Figure 5.2) has already been computed for Coin Toss in which P_1 chooses Play $\frac{3}{4}$ of the time with Heads and $\frac{1}{2}$ of the time with Tails, and P_2 chooses Heads $\frac{1}{2}$ of the time, Tails $\frac{1}{4}$ of the time, and Forfeit $\frac{1}{4}$ of the time after P_1 chooses Play. [1] The details of the blueprint in the Sell subgame are not relevant in this section, but the EV for choosing the Sell action *is* relevant. We assume that if P_1 chose the Sell action and played optimally thereafter, then she would receive an expected payoff of 0.5 if the coin is Heads, and -0.5 if the coin is Tails. We will attempt to improve P_2's strategy in the subgame S that follows P_1 choosing Play.

Additional Notation

We begin by introducing additional notation that will be useful when describing the search techniques in this section.

Recall that a **counterfactual best response (CBR)**, is a best response that additionally plays optimally even in unreached sequences. That is, given a strategy σ_i, a counterfactual best response to it, $CBR(\sigma_i)$, is a best response to σ_i with the additional condition that for every player $-i$ infoset I, $v_{-i}^{\langle CBR(\sigma_i),\sigma_i\rangle}(I) = \max_{a \in A(I)} v_{-i}^{\langle CBR(\sigma_i),\sigma_i\rangle}(I,a)$. In other words, a CBR must always choose an action resulting in maximum expected value in every infoset, regardless of whether that infoset is actually reached as part of a best response. In a regular best response, an infoset need only choose a maximum-EV action if that infoset is reached with positive probability (i.e., if earlier actions lead to it with positive probability). Recall also that a **counterfactual best response value** is defined as $CBV^{\sigma_{-i}}(I,a) = \min_{\sigma'_i} v^{\langle \sigma'_i,\sigma_{-i}\rangle}(I,a)$ and $CBV^{\sigma_{-i}}(I) = \min_{\sigma'_i} v^{\langle \sigma'_i,\sigma_{-i}\rangle}(I)$.

[1] In many large games the blueprint is far from optimal either because the equilibrium-finding algorithm did not sufficiently converge, or because the game was too large and had to be abstracted or function approximation was used. Clearly the example blueprint shown here could be trivially improved; we use it for simplicity of exposition.



5.3a Unsafe search

5.3b Resolving search

Figure 5.3: The augmented subgames solved to find a P_2 strategy in the Play subgame of Coin Toss.

We define S_{top} for a subgame S as the set of earliest-reachable histories in S. Specifically, $h \in S_{\text{top}}$ if $h \in S$ and $h' \notin S$ for any $h' \sqsubset h$.

Unsafe Search

We first review the most intuitive form of search, which we refer to as **unsafe search** [9, 51, 57, 58]. This form of search assumes both players played according to the blueprint prior to reaching the subgame. That defines a probability distribution over the nodes at the root of the subgame S, presenting the probability that the true game state matches that node. A strategy for the subgame is then calculated which assumes that this distribution is correct.

In all the search algorithms we describe, an **augmented subgame** containing S and a few additional nodes is solved to determine the strategy for S. Applying unsafe search to the blueprint in Coin Toss (after P_1 chooses Play) means solving the augmented subgame shown in Figure 5.3a.

Specifically, the augmented subgame consists of only an initial chance node and S. The initial chance node reaches $h \in S_{top}$ with probability $\frac{\pi^\sigma(h)}{\sum_{h' \in S_{top}} \pi^\sigma(h')}$. The augmented subgame is solved and its strategy for P_2 is used in S rather than the blueprint strategy.

Unsafe search lacks theoretical solution quality guarantees and there are many situations where it performs extremely poorly. Indeed, if it were applied to the blueprint of Coin Toss then P_2 would always choose Heads—which P_1 could exploit severely by only choosing Play with Tails. Despite the lack of theoretical guarantees and potentially bad performance, unsafe search is simple and can *sometimes* produce low-exploitability strategies, as we show later.

We now move to discussing *safe* search techniques, that is, ones that ensure that the exploitability of the strategy is no higher than that of the blueprint strategy.

Subgame Resolving

In **Subgame Resolving** [31], a safe strategy is computed for P_2 in the subgame by solving the augmented subgame shown in Figure 5.3b, producing an equilibrium strategy σ^S. This augmented subgame differs from unsafe search by giving P_1 the option to "opt out" from entering S

and instead receive the EV of playing optimally against P_2's blueprint strategy in S.

Specifically, the augmented subgame for Resolving differs from unsafe search as follows. For each $h_{top} \in S_{top}$ we insert a new P_1 node h_r, which exists only in the augmented subgame, between the initial chance node and h_{top}. The set of these h_r nodes is S_r. The initial chance node connects to each node $h_r \in S_r$ in proportion to the probability that player P_1 could reach h_{top} if P_1 tried to do so (that is, in proportion to $\pi^{\sigma}_{-1}(h_{top})$). At each node $h_r \in S_r$, P_1 has two possible actions. Action a'_S leads to h_{top}, while action a'_T leads to a terminal payoff that awards the value of playing optimally against P_2's blueprint strategy, which is $CBV^{\sigma_2}(I_1(h_{top}))$. In the blueprint of Coin Toss, P_1 choosing Play after the coin lands Heads results in an EV of 0, and $\frac{1}{2}$ if the coin is Tails. Therefore, a'_T leads to a terminal payoff of 0 for Heads and $\frac{1}{2}$ for Tails. After the equilibrium strategy σ^S is computed in the augmented subgame, P_2 plays according to the computed subgame strategy σ^S_2 rather than the blueprint strategy when in S. The P_1 strategy σ^S_1 is not used.

Clearly P_1 cannot do worse than always picking action a'_T (which awards the highest EV P_1 could achieve against P_2's blueprint). But P_1 also cannot do *better* than always picking a'_T, because P_2 could simply play according to the blueprint in S, which means action a'_S would give the same EV to P_1 as action a'_T (if P_1 played optimally in S). In this way, the strategy for P_2 in S is pressured to be no worse than that of the blueprint. In Coin Toss, if P_2 were to always choose Heads (as was the case in unsafe search), then P_1 would always choose a'_T with Heads and a'_S with Tails.

Resolving guarantees that P_2's exploitability will be no higher than the blueprint's (and may be better). However, it may miss opportunities for improvement. For example, if we apply Resolving to the example blueprint in Coin Toss, one solution to the augmented subgame is the blueprint itself, so P_2 may choose Forfeit 25% of the time even though Heads and Tails dominate that action. Indeed, the original purpose of Resolving was not to *improve* upon a blueprint strategy in a subgame, but rather to compactly store it by keeping only the EV at the root of the subgame and then reconstructing the strategy in real time when needed rather than storing the whole subgame strategy.

Maxmargin Search

Maxmargin search [109] is similar to Resolving, except that it seeks to improve P_2's strategy in the subgame strategy as much as possible. While Resolving seeks a strategy for P_2 in S that would simply dissuade P_1 from entering S, Maxmargin search additionally seeks to punish P_1 as much as possible if P_1 nevertheless chooses to enter S. A *subgame margin* is defined for each infoset in S_r, which represents the difference in value between entering the subgame versus choosing the alternative payoff. Specifically, for each infoset $I_1 \in S_{top}$, the *subgame margin* is

$$M^{\sigma^S}(I_1) = CBV^{\sigma_2}(I_1) - CBV^{\sigma^S_2}(I_1) \tag{5.1}$$

In Maxmargin search, a Nash equilibrium σ^S for the augmented subgame described in Resolving search is computed such that the minimum margin over all $I_1 \in S_{top}$ is maximized. Aside from maximizing the minimum margin, the augmented subgames used in Resolving and Maxmargin search are identical.

Given our blueprint strategy in Coin Toss, Maxmargin search would result in P_2 choosing Heads with probability $\frac{5}{8}$, Tails with probability $\frac{3}{8}$, and Forfeit with probability 0.

The augmented subgame can be solved in a way that maximizes the minimum margin by using a standard LP solver. In order to use iterative algorithms such as the Excessive Gap Technique [61, 91, 114] or Counterfactual Regret Minimization (CFR) [163], one can use the **gadget game**, described in Section 5.1.10, which was introduced by Moravcik et al. [109]. Our experiments used CFR.

Maxmargin search is safe. Furthermore, it guarantees that if every Player 1 best response reaches the subgame with positive probability through some infoset(s) that have positive margin, then exploitability is strictly lower than that of the blueprint strategy. While the theoretical guarantees are stronger, Maxmargin may lead to worse practical performance relative to Resolving when combined with the techniques discussed in Section 5.1.4, due to Maxmargin's greater tendency to overfit to assumptions in the model.

5.1.3 Reach Search

All of the search techniques described in Section 5.1.2 only consider the target subgame in isolation, which can lead to suboptimal strategies. For example, Maxmargin solving applied to S in Coin Toss results in P_2 choosing Heads with probability $\frac{5}{8}$ and Tails with $\frac{3}{8}$ in S. This results in P_1 receiving an EV of $-\frac{1}{4}$ by choosing Play in the Heads state, and an EV of $\frac{1}{4}$ in the Tails state. However, P_1 could simply always choose Sell in the Heads state (earning an EV of 0.5) and Play in the Tails state and receive an EV of $\frac{3}{8}$ for the entire game. In this section we introduce *Reach search*, an improvement to past search techniques that considers *what the opponent could have alternatively received from other subgames*.[2] For example, a better strategy for P_2 would be to choose Heads with probability $\frac{3}{4}$ and Tails with probability $\frac{1}{4}$. Then P_1 is indifferent between choosing Sell and Play in both cases and overall receives an expected payoff of 0 for the whole game.

However, that strategy is only optimal if P_1 would indeed achieve an EV of 0.5 for choosing Sell in the Heads state and -0.5 in the Tails state. That would be the case if P_2 played according to the blueprint in the Sell subgame (which is not shown), but in reality we would apply search to the Sell subgame if the Sell action were taken, which would change P_2's strategy there and therefore P_1's EVs. Applying search to any subgame encountered during play is equivalent to applying it to all subgames independently. Thus, we must consider that the EVs from other subgames may differ from what the blueprint says because search would be applied to them as well.

As an example of this issue, consider the game shown in Figure 5.4 which contains two identical subgames S_1 and S_2 where the blueprint has P_2 pick Heads and Tails with 50% probability. The Sell action leads to an EV of 0.5 from the Heads state, while Play leads to an EV of 0. If we were to solve just S_1, then P_2 could afford to always choose Tails in S_1, thereby letting P_1 achieve an EV of 1 for reaching that subgame from Heads because, due to the chance node C_1, S_1 is only reached with 50% probability. Thus, P_1's EV for choosing Play would be 0.5 from

[2]Other search methods have also considered the cost of reaching a subgame [74, 159]. However, those approaches are not correct in theory when applied in real time to any subgame reached during play.

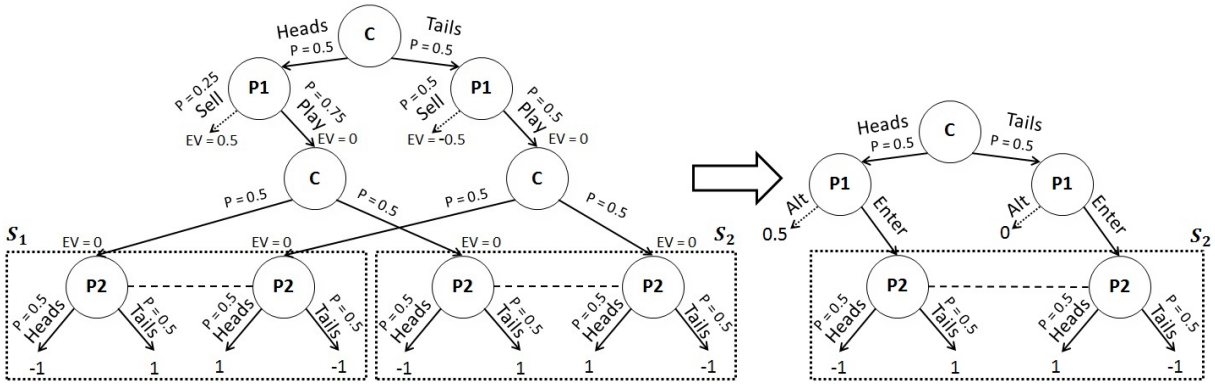


Figure 5.4: **Left:** A modified game of Coin Toss with two subgames. The nodes C_1 and C_2 are public chance nodes whose outcomes are seen by both P_1 and P_2. **Right:** An augmented subgame for one of the subgames according to Reach search. If only one of the subgames is being solved, then the alternative payoff for Heads can be at most 1. However, if both are solved independently, then the gift must be split among the subgames and must sum to at most 1. For example, the alternative payoff in both subgames can be 0.5.

Heads and -0.5 from Tails, which is optimal. We can achieve this strategy in S_1 by solving an augmented subgame in which the alternative payoff for Heads is 1. In that augmented subgame, P_2 always choosing Tails would be a solution (though not the only solution).

However, if the same reasoning were applied independently to S_2 as well, then P_2 might always choose Tails in both subgames and P_1's EV for choosing Play from Heads would become 1 while the EV for Sell would only be 0.5. Instead, we could allow P_1 to achieve an EV of 0.5 for reaching each subgame from Heads (by setting the alternative payoff for Heads to 0.5). In that case, P_1's overall EV for choosing Play could only increase to 0.5, even if both S_1 and S_2 were solved independently.

We capture this intuition by considering for each $I_1 \in S_{top}$ all the infosets and actions $I_1' \cdot a' \sqsubset I_1$ that P_1 would have taken along the path to I_1. If, at some $I_1' \cdot a' \sqsubset I_1$ where P_1 acted, there was a different action $a^* \in A(I_1')$ that leads to a higher EV, then P_1 would have taken a suboptimal action if they reached I_1. The difference in value between a^* and a' is referred to as a *gift*. We can afford to let P_1's value for I_1 increase beyond the blueprint value (and in the process lower P_1's value in some other infoset in S_{top}), so long as the increase to I_1's value is small enough that choosing actions leading to I_1 is still suboptimal for P_1. Critically, we must ensure that the increase in value is small enough even when the potential increase across all subgames is summed together, as in Figure 5.4.[3]

A complicating factor is that gifts we assumed were present may actually not exist. For example, in Coin Toss, suppose applying search to the Sell subgame results in P_1's value for Sell

[3]In this paper and in our experiments, we allow any infoset that descends from a gift to increase by the size of the gift (e.g., in Figure 5.4 the gift from Heads is 0.5, so we allow P_1's value for Heads in both S_1 and S_2 to increase by 0.5). However, any division of the gift among subgames is acceptable so long as the potential increase across all subgames (multiplied by the probability of P_1 reaching that subgame) does not exceed the original gift. For example in Figure 5.4 if we only apply Reach search to S_1, then we could allow the Heads state in S_1 to increase by 1 rather than just by 0.5. In practice, some divisions may do better than others. The division we use in this paper (applying gifts equally to all subgames) did well in practice.

from the Heads state decreasing from 0.5 to 0.25. If we independently solve the Play subgame, we have no way of knowing that P_1's value for Sell is lower than the blueprint suggested, so we may still assume there is a gift of 0.5 from the Heads state based on the blueprint. Thus, in order to guarantee a theoretical result on exploitability that is as strong as possible, we use in our theory and experiments a *lower bound* on what gifts could be after search was applied to all other subgames.

Formally, let σ_2 be a P_2 blueprint and let σ_2^{-S} be the P_2 strategy that results from applying search independently to a set of disjoint subgames other than S. Since we do not want to compute σ_2^{-S} in order to apply search to S, let $\lfloor g^{\sigma_2^{-S}}(I_1', a') \rfloor$ be a lower bound of $CBV^{\sigma_2^{-S}}(I_1') - CBV^{\sigma_2^{-S}}(I_1', a')$ that does not require knowledge of σ_2^{-S}. In our experiments we use $\lfloor g^{\sigma_2^{-S}}(I_1', a') \rfloor = \max_{a \in A_z(I_1') \cup \{a'\}} CBV^{\sigma_2}(I_1', a) - CBV^{\sigma_2}(I_1', a')$ where $A_z(I_1') \subseteq A(I_1')$ is the set of actions leading immediately to terminal nodes. Reach search modifies the augmented subgame in Resolving and Maxmargin by increasing the alternative payoff for infoset $I_1 \in S_{top}$ by $\sum_{I_1' \cdot a' \sqsubseteq I_1 | P(I_1') = P_1} \lfloor g^{\sigma_2^{-S}}(I_1', a') \rfloor$. Formally, we define a *reach margin* as

$$M_r^{\sigma^S}(I_1) = M^{\sigma^S}(I_1) + \sum_{I_1' \cdot a' \sqsubseteq I_1 | P(I_1') = P_1} \lfloor g^{\sigma_2^{-S}}(I_1', a') \rfloor \tag{5.2}$$

This margin is larger than or equal to the one for Maxmargin, because $\lfloor g^{\sigma_2^{-S}}(I', a') \rfloor$ is nonnegative. We refer to the improved algorithms as Reach-Resolve and Reach-Maxmargin.

Intuitively, the alternative payoff in an augmented subgame determines how important it is that P_2 "defend" against that P_1 infoset. If the alternative payoff is increased, then P_1 is more likely to choose the alternative payoff rather than enter the subgame, so P_2 can instead focus on lowering the value of other P_1 infosets in S_{top}.

Using a lower bound on gifts is not necessary to guarantee safety. So long as we use a gift value $g^{\sigma'}(I_1', a') \leq CBV^{\sigma_2}(I_1') - CBV^{\sigma_2}(I_1', a')$, the resulting strategy will be safe. However, using a lower bound further guarantees a reduction to exploitability when a P_1 best response reaches with positive probability an infoset $I_1 \in S_{top}$ that has positive margin, as proven in Theorem 20. In practice, it may be best to use an accurate estimate of gifts. One option is to use $\hat{g}^{\sigma_2^{-S}}(I_1', a') = C\tilde{B}V^{\sigma_2}(I_1') - C\tilde{B}V^{\sigma_2}(I_1', a')$ in place of $\lfloor g^{\sigma_2^{-S}}(I_1', a') \rfloor$, where $C\tilde{B}V^{\sigma_2}$ is the closest P_1 can get to the value of a counterfactual best response while P_1 is constrained to playing within the abstraction that generated the blueprint. Using estimates is covered in more detail in Section 5.1.4.

Theorem 20 shows that when subgames are solved independently and using lower bounds on gifts, Reach-Maxmargin search has exploitability lower than or equal to past safe techniques. The theorem statement is similar to that of Maxmargin [109], but the margins are now larger (or equal) in size.

Theorem 20. *Given a strategy σ_2 in a two-player zero-sum game, a set of disjoint subgames $\mathbb{S}$, and a strategy σ_2^S for each subgame $S \in \mathbb{S}$ produced via Reach-Maxmargin search using lower bounds for gifts, let σ_2' be the strategy that plays according to σ_2^S for each subgame $S \in \mathbb{S}$, and σ_2 elsewhere. Moreover, let σ_2^{-S} be the strategy that plays according to σ_2' everywhere except for P_2 nodes in S, where it instead plays according to σ_2. If $\pi_1^{BR(\sigma_2')}(I_1) > 0$ for some $I_1 \in S_{top}$, then $exp(\sigma_2') \leq exp(\sigma_2^{-S}) - \sum_{h \in I_1} \pi_{-1}^{\sigma_2}(h) M_r^{\sigma^S}(I_1)$.*

So far the described techniques have guaranteed a reduction in exploitability over the blueprint by setting the value of a'_T equal to the value of P_1 playing optimally to P_2's blueprint. Relaxing this guarantee by instead setting the value of a'_T equal to an *estimate* of P_1's value when *both* players play optimally leads to far lower exploitability in practice. We discuss this approach in the next section.

5.1.4 Estimates for Alternative Payoffs

In this section we consider the case where we have a good estimate of what the values of subgames would look like in a Nash equilibrium. Unlike previous sections, exploitability might be *higher* than the blueprint when using this method; the solution quality ultimately depends on the accuracy of the estimates used. In practice this approach leads to significantly lower exploitability.

When solving multiple P_2 subgames, there is a minimally-exploitable strategy σ_2^* that could, in theory, be computed by changing only the strategies in the subgames. (σ_2^* may not be a Nash equilibrium because P_2's strategy outside the subgames is fixed, but it is the closest that can be achieved by changing the strategy only in the subgames). However, σ_2^* can only be guaranteed to be produced by solving all the subgames together, because the optimal strategy in one subgame depends on the optimal strategy in other subgames.

Still, suppose that we know $CBV^{\sigma_2^*}(I_1)$ for every infoset $I_1 \in S_{top}$ for every subgame S. Let $I_{r,1}$ be the infoset in S_r that leads to I_1. By setting the P_1 alternative payoff for $I_{r,1}$ to $v(I_{r,1}, a'_T) = CBV^{\sigma_2^*}(I_1)$, safe search guarantees a strategy will be produced with exploitability no worse than σ_2^*. Thus, achieving a strategy equivalent to σ_2^* does not require knowledge of σ_2^*; rather, it only requires knowledge of $CBV^{\sigma_2^*}(I_1)$ for infosets I_1 in the top of the subgames.

While we do not know $CBV^{\sigma_2^*}(I_1)$ exactly without knowing σ_2^* itself, we may nevertheless be able to produce (or learn) good *estimates* of $CBV^{\sigma_2^*}(I_1)$. For example, in Section 5.1.8 we compute the solution to the game of No-Limit Flop Hold'em (NLFH), and find that in perfect play P_2 can expect to win about 37 mbb/h[4] (that is, if P_1 plays perfectly against the computed P_2 strategy, then P_1 earns -37; if P_2 plays perfectly against the computed P_1 strategy, then P_2 earns 37). An abstraction of the game which is only 0.02% of the size of the full game produces a P_1 strategy that can be beaten by 112 mbb/h, and a P_2 strategy that can be beaten by 21 mbb/h. Still, the abstract strategy estimates that at equilibrium, P_2 can expect to win 35 mbb/h. So even though the abstraction produces a very poor estimate of the *strategy* σ^*, it produces a good estimate of the *value* of σ^*. In our experiments, we estimate $CBV^{\sigma_2^*}(I_1)$ by calculating a P_1 counterfactual best response *within the abstract game* to P_2's blueprint. We refer to this strategy as $C\tilde{B}R(\sigma_2)$ and its value in an infoset I_1 as $C\tilde{B}V^{\sigma_2}(I_1)$. We then use $C\tilde{B}V^{\sigma_2}(I_1)$ as the alternative payoff of I_1 in an augmented subgame. In other words, rather than calculate a P_1 counterfactual best response in the full game to P_2's blueprint strategy (which would be $CBR(\sigma_2)$), we instead calculate P_1's counterfactual best response where P_1 is constrained by the abstraction.

If the blueprint was produced by conducting T iterations of CFR in an abstract game, then

[4]In poker, the performance of one strategy against another depends on how much money is being wagered. For this reason, expected value and exploitability are measured in milli big blinds per hand (mbb/h). A big blind is the amount of money one of the players is required to put into the pot at the beginning of each hand.

one could instead simply use the final iteration's strategy σ_1^T, as this converges to a counterfactual best response within the abstract game. This is what we use in our experiments in this paper.

Theorem 21 proves that if we use estimates of $CBV^{\sigma_2^*}(I_1)$ as the alternative payoffs in Maxmargin search, then we can bound exploitability by the distance of the estimates from the true values. This is in contrast to the previous algorithms which guaranteed exploitability no worse than the blueprint.

Theorem 21. *Let $\mathbb{S}$ be a set of disjoint subgames being solved in a game with no private actions. Let σ be a blueprint and let σ_2^* be a minimally-exploitable P_2 strategy that differs from σ_2 only in $\mathbb{S}$. Let $\Delta = \max_{S \in \mathbb{S}, I_1 \in S_{top}} |CBV^{\sigma_2^*}(I_1) - CBV^{\sigma_2}(I_1)|$. Applying Maxmargin solving to each subgame using σ as the blueprint produces a P_2 strategy with exploitability no higher than $exp(\sigma_2^*) + 2\Delta$.*

Using estimates of the values of σ^* tends to be do better than the theoretically safe options described in Section 5.1.2.[5]

Although Theorem 21 uses Maxmargin in the proof, in practice Resolve does far better with estimates than Maxmargin, even though Maxmargin should do better if the estimates are accurate. This is possibly because Maxmargin may "overfit" to the estimates. Theorem 21 easily extends to Reach-Maxmargin as well, and Reach-Resolve does better than Resolve regardless of whether estimates are used.

Section 5.1.5 discusses an improvement, which we refer to as Distributional Alternative Payoffs, that leads to even better performance by making the algorithm more robust to errors in the blueprint estimates.

5.1.5 Distributional Alternative Payoffs

One problem with existing safe search techniques is that they may "overfit" to the alternative payoffs, even when we use estimates. Consider for instance a subgame with two different P_1 infosets I_1 and I_1' at the top. Assume P_1's value for I_1 is estimated to be 1, and for I_1' is 10. Now suppose during search, P_2 has a choice between two different strategies. The first sets P_1's value in the subgame for I_1 to 0.99 and for I_1' to 9.99. The second slightly increases P_1's value for the subgame for I_1 to 1.01 but dramatically lowers the value for I_1' to 0. The safe search methods described so far would choose the first strategy, because the second strategy leaves one of the margins negative. However, intuitively, the second strategy is likely the better option, because it is more robust to errors in the model. For example, perhaps we are not confident that 10 is the exact value, but instead believe its true value is normally distributed with 10 as the mean and a standard deviation of 1. In this case, we would prefer the strategy that lowers the value for I_1' to 0.

To address this problem, we introduce a way to incorporate the modeling uncertainty into the game itself. Specifically, we introduce a new augmented subgame that makes search more robust to errors in the model. This augmented subgame changes the augmented subgame used in subgame Resolving (shown in Figure 5.3b) so that the alternative payoffs are random variables, and P_1 is informed at the start of the augmented subgame of the values drawn from the

[5]It is also possible to combine the safety of past approaches with some of the better performance of using estimates by adding the original Resolve conditions as additional constraints.

random variables (but P_2 is not). The augmented subgame is otherwise identical. A visualization of this change is shown in Figure 5.5. As the distributions of the random variables narrow, the augmented subgame converges to the Resolve augmented subgame (but still maximizes the minimum margin when all margins are positive). As the distributions widen, P_2 seeks to maximize the sum over all margins, regardless of which are positive or negative.

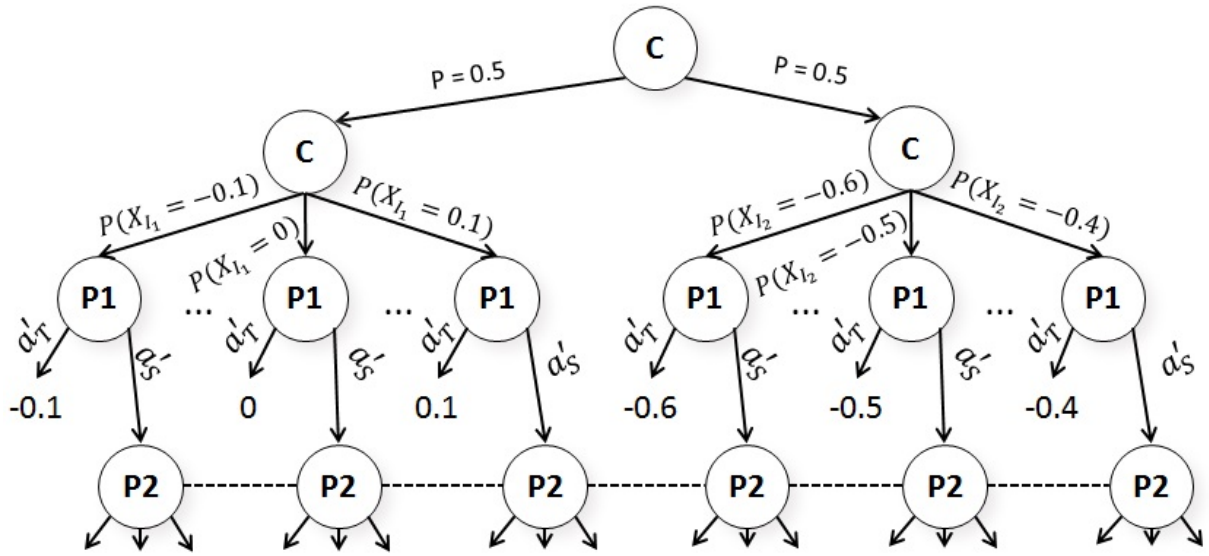


Figure 5.5: A visualization of the change in the augmented subgame from Figure 5.3b when using distributional alternative payoffs.

This modification makes the augmented subgame infinite in size because the random variables may be real-valued and P_1 could have a unique strategy for each outcome of the random variable. Fortunately, the special structure of the game allows us to arrive at a P_2 Nash equilibrium strategy for this infinite-sized augmented subgame by solving a much simpler gadget game.

The gadget game is identical to the augmented subgame used in Resolve search (shown in Figure 5.3b), except at each initial P_1 infoset $I_{r,1} \in S_r$, P_1 chooses action a'_S (that is, chooses to enter the subgame rather than take the alternative payoff) with probability $P\big(X_{I_1} \leq v(I_{r,1}, a'_S)\big)$, where $v(I_{r,1}, a'_S)$ is the expected value of action a'_S. (When solving via CFR, it is the expected value on each iteration, as described in CFR-BR [80]). This leads to Theorem 22, which proves that solving this simplified gadget game produces a P_2 strategy that is a Nash equilibrium in the infinite-sized augmented subgame illustrated in Figure 5.5.

Theorem 22. *Let S' be a Resolve augmented subgame and S'_r its root. Let S be a Distributional augmented subgame similar to S', except at each infoset $I_{r,1} \in S_r$, P_1 observes the outcome of a random variable X_{I_1} and the alternative payoff is equal to that outcome. If CFR is used to solve S' except that the action leading to S' is taken from each $I_{r,1} \in S'_r$ with probability $P\big(X_{I_1} \leq v^t(I_{r,1}, a'_S)\big)$, where $v^t(I_{r,1}, a'_S)$ is the value on iteration t of action a'_S, then the resulting P_2 strategy $\sigma_2^{S'}$ in S' is a P_2 Nash equilibrium strategy in S.*

Another option which also solves the game but has better empirical performance relies on the *softmax* (also known as *Hedge*) algorithm [104]. This gadget game is more complicated, and is

described in detail in Section 5.1.6. We use the softmax gadget game in our experiments.

The correct distribution to use for the random variables ultimately depends on the actual unknown errors in the model. In our experiments for this technique, we set $X_{I_1} \sim \mathcal{N}\left(\mu_{I_1}, s_{I_1}^2\right)$, where μ_I is the blueprint value (plus any gifts). s_{I_1} is set as the difference between the blueprint value of I_1, and the true (that is, unabstracted) counterfactual best response value of I_1. Our experiments show that this heuristic works well, and future research could yield even better options.

5.1.6 Hedge for Distributional Search

In this paper we use CFR [163] with Hedge in S_r, which allows us to leverage a useful property of the Hedge algorithm [104] to update all the infosets resulting from outcomes of X_{I_1} simultaneously.[6] When using Hedge, action a'_S in infoset $I_{r,1}$ in the augmented subgame is chosen on iteration t with probability $\frac{e^{\eta_t \hat{v}(I_{r,1}, a'_S)}}{e^{\eta_t \hat{v}(I_{r,1}, a'_S)} + e^{\eta_t \hat{v}(I_{r,1}, a'_T)}}$. Where $\hat{v}(I_{r,1}, a'_T)$ is the observed expected value of action a'_T, $\hat{v}(I_{r,1}, a'_S)$ is the observed expected value of action a'_S, and η_t is a tuning parameter. Since, action a'_S leads to identical play by both players for all outcomes of X, $\hat{v}(I_{r,1}, a'_S)$ is identical for all outcomes of X. Moreover, $\hat{v}(I_{r,1}, a'_T)$ is simply the outcome of X_{I_1}. So the probability that a'_S is taken across all infosets on iteration t is

$$\int_{-\infty}^{\infty} \frac{e^{\eta_t \hat{v}(I_{r,1}, a'_S)}}{e^{\eta_t \hat{v}(I_{r,1}, a'_S)} + e^{\eta_t x}} f_{X_{I_1}}(x) dx \tag{5.3}$$

where $f_{X_{I_1}}(x)$ is the pdf of X_{I_1}. In other words, if CFR is used to solve the augmented subgame, then the game being solved is identical to Figure 5.3b except that action a'_S is always chosen in infoset I_1 on iteration t with probability given by (5.3). In our experiments, we set the Hedge tuning parameter η as suggested in [26]: $\eta_t = \frac{\sqrt{\ln(|A(I_1)|)}}{3\sqrt{VAR(I_1)_t}\sqrt{t}}$, where $VAR(I_1)_t$ is the observed variance in the payoffs the infoset has received across all iterations up to t. In the subgame that follows S_r, we use CFR+ as the solving algorithm.

5.1.7 Nested Search

As we have discussed, large games must be abstracted to reduce the game to a tractable size. This is particularly common in games with large or continuous action spaces. Typically the action space is discretized by action abstraction so that only a few actions are included in the abstraction. While we might limit ourselves to the actions we included in the abstraction, an opponent might choose actions that are not in the abstraction. In that case, the *off-tree* action can be mapped to an action that is in the abstraction, and the strategy from that in-abstraction action can be used. For example, in an auction game we might include a bid of $\$100$ in our abstraction. If a player bids $\$101$, we simply treat that as a bid of $\$100$. This is referred to as **action translation** [49, 60, 133]. Action translation is the state-of-the-art prior approach to dealing with this issue. It has been used, for example, by all the leading competitors in the Annual Computer Poker Competition (ACPC).

[6] Another option is to apply CFR-BR [80] only at the initial P_1 nodes when deciding between a'_T and a'_S.

In this section, we develop techniques for applying search to calculate responses to opponent off-tree actions, thereby obviating the need for action translation. That is, rather than simply treat a bid of $\$101$ as $\$100$, we calculate in real time a unique response to the bid of $\$101$. This can also be done in a nested fashion in response to subsequent opponent off-tree actions. We present two methods that dramatically outperform the leading action translation technique. Additionally, these techniques can be used to solve finer-grained models as play progresses down the game tree. For exposition, we assume that P_2 wishes to respond to P_1 choosing an off-tree action.

We refer to the first method as the *inexpensive* method.[7] When P_1 chooses an off-tree action a, a subgame S is generated following that action such that for any infoset I_1 that P_1 might be in, $I_1 \cdot a \in S_{top}$. This subgame may itself be an abstraction. A solution σ^S is computed via search, and σ^S is combined with σ to form a new blueprint σ' in the expanded abstraction that now includes action a. The process repeats whenever P_1 again chooses an off-tree action.

To conduct safe search in response to off-tree action a, we could calculate $CBV^{\sigma_2}(I_1, a)$ by defining, via action translation, a P_2 blueprint following a and best responding to it [16]. However, that could be computationally expensive and would likely perform poorly in practice because, as we show later, action translation is highly exploitable. Instead, we relax the guarantee of safety and use $C\tilde{B}V^{\sigma_2}(I_1)$ for the alternative payoff, where $C\tilde{B}V^{\sigma_2}(I_1)$ is the value in I_1 of P_1 playing as close to optimal as possible while constrained to playing in the blueprint abstraction (which excludes action a). In this case, exploitability depends on how well $C\tilde{B}V^{\sigma_2}(I_1)$ approximates $CBV^{\sigma_2^*}(I_1)$, where σ_2^* is an optimal P_2 strategy (see Section 5.1.4).[8] In general, we find that only a small number of near-optimal actions need to be included in the blueprint abstraction for $C\tilde{B}V^{\sigma_2}(I_1)$ to be close to $CBV^{\sigma_2^*}(I_1)$. We can then approximate a near-optimal response to any opponent action. This is particularly useful in very large or continuous action spaces.

The "inexpensive" approach cannot be combined with Unsafe search because the probability of reaching an action outside of a player's abstraction is undefined. Nevertheless, a similar approach is possible with Unsafe search (as well as all the other search techniques) by starting the search at h rather than at $h \cdot a$. In other words, if action a taken in node h is not in the abstraction, then Unsafe search is conducted in the smallest subgame containing h (and action a is added to that abstraction). This increases the size of the subgame compared to the inexpensive method because a strategy must be recomputed for every action $a' \in A(h)$ in addition to a. For example, if an off-tree action is chosen by the opponent as the first action in the game, then the strategy for the entire game must be recomputed. We therefore call this method the *expensive* method. We present experiments with both methods.

5.1.8 Experiments

Our experiments were conducted on action-abstracted heads-up no-limit Texas hold'em (HUNL) (described in Section 2.4.3, as well as action-abstracted no-limit flop hold'em (NLFH) (described in Section 2.4.4) and action-abstracted no-limit turn hold'em (NLTH) (which is just like HUNL

[7]Following our study, the AI DeepStack used a technique similar to this form of nested search [110].

[8]We estimate $CBV^{\sigma_2^*}(I_1)$ rather than $CBV^{\sigma_2^*}(I_1, a)$ because $CBV^{\sigma_2^*}(I_1) - CBV^{\sigma_2^*}(I_1, a)$ is a gift that may be added to the alternative payoff anyway.

but ends after the third betting round with only four community cards ever revealed). For equilibrium finding, we used CFR+ [150].

Our first experiment compares the performance of the search techniques when applied to information abstraction (which is card abstraction in the case of poker). Specifically, we solve NLFH with no information abstraction on the preflop. On the flop, there are 1,286,792 infosets for each betting sequence; the abstraction buckets them into 200, 2,000, or 30,000 abstract ones (using a leading information abstraction algorithm [50]). We then apply search immediately after the flop community cards are dealt.

We experiment with two versions of the game, one small and one large, which include only a few of the available actions in each infoset. We also experimented on abstractions of NLTH. In that case, we solve NLTH with no information abstraction on the preflop or flop. On the turn, there are 55,190,538 infosets for each betting sequence; the abstraction buckets them into 200, 2,000, or 20,000 abstract ones. We apply search immediately after the turn community card is dealt.

Tables 5.1, 5.2, and 5.3 show the performance of each technique. In all our experiments, exploitability is measured in the standard units used in this field: milli big blinds per hand (mbb/h).

Small Flop Hold'em Flop Buckets:	**200**	**2,000**	**30,000**
Trunk Strategy	88.69	37.374	9.128
Unsafe	14.68	3.958	0.5514
Resolve	60.16	17.79	5.407
Maxmargin	30.05	13.99	4.343
Reach-Maxmargin	29.88	13.90	4.147
Reach-Maxmargin (not split)	24.87	9.807	2.588
Estimate	11.66	6.261	2.423
Estimate + Distributional	10.44	6.245	3.430
Reach-Estimate + Distributional	10.21	5.798	2.258
Reach-Estimate + Distributional (not split)	9.560	4.924	1.733

Table 5.1: Exploitability (evaluated in the game with no information abstraction) of search in small flop Texas hold'em.

In the above experiments, Estimate is the technique introduced in Section 5.1.4 (added on top of Resolving) and Distributional is the technique introduced in Section 5.1.5. We use a normal distribution in the Distributional search experiments, with standard deviation determined by the heuristic presented in Section 5.1.5.

Since search begins immediately after a chance node with an extremely high branching factor ($1,755$ in NLFH), the gifts for the Reach algorithms are divided among subgames inefficiently. Many subgames do not use the gifts at all, while others could make use of more. The result is that the theoretically safe version of Reach allocates gifts very conservatively. In the experiments we show results both for the theoretically safe splitting of gifts, as well as a more aggressive version where gifts are scaled up by the branching factor of the chance node ($1,755$). This weakens the theoretical guarantees of the algorithm, but in general did better than splitting gifts in a theoretically correct manner. However, this is not universally true. Section 5.1.11 shows that in

Large Flop Hold'em Flop Buckets:	**200**	**2,000**	**30,000**
Trunk Strategy	283.7	165.2	41.41
Unsafe	65.59	38.22	396.8
Resolve	179.6	101.7	23.11
Maxmargin	134.7	77.89	19.50
Reach-Maxmargin	134.0	72.22	18.80
Reach-Maxmargin (not split)	130.3	66.79	16.41
Estimate	52.62	41.93	30.09
Estimate + Distributional	49.56	38.98	10.54
Reach-Estimate + Distributional	49.33	38.52	9.840
Reach-Estimate + Distributional (not split)	49.13	37.22	8.777

Table 5.2: Exploitability (evaluated in the game with no information abstraction) of search in large flop Texas hold'em.

Turn Hold'em Turn Buckets:	**200**	**2,000**	**20,000**
Trunk Strategy	684.6	465.1	345.5
Unsafe	130.4	85.95	79.34
Resolve	454.9	321.5	251.8
Maxmargin	427.6	299.6	234.4
Reach-Maxmargin	424.4	298.3	233.5
Reach-Maxmargin (not split)	333.4	229.4	175.5
Estimate	120.6	89.43	76.44
Estimate + Distributional	119.4	87.83	74.35
Reach-Estimate + Distributional	116.8	85.80	72.59
Reach-Estimate + Distributional (not split)	113.3	83.24	70.68

Table 5.3: Exploitability (evaluated in the game with no information abstraction) of search in turn Texas hold'em.

at least one case, exploitability increased when gifts were scaled up too aggressively. In all cases, using Reach search in at least the theoretical safe method led to lower exploitability.

Despite lacking theoretical guarantees, Unsafe search did surprisingly well in most games. However, it did substantially worse in Large NLFH with 30,000 buckets. This exemplifies its variability. Among the safe methods, all of the changes we introduce show improvement over past techniques. The Reach-Estimate + Distributional algorithm generally resulted in the lowest exploitability among the various choices, and in most cases beat Unsafe search.

In all but one case, using estimated values lowered exploitability more than Maxmargin and Resolve search. Also, in all but one case using distributional alternative payoffs lowered exploitability.

The second experiment evaluates nested search, and compares it to action translation. In order to also evaluate action translation, in this experiment, we create an NLFH game that includes 3 bet sizes at every point in the game tree (0.5, 0.75, and 1.0 times the size of the pot); a player can also decide not to bet. Only one bet (i.e., no raises) is allowed on the preflop, and three bets are

allowed on the flop. There is no information abstraction anywhere in the game. We also created a second, smaller abstraction of the game in which there is still no information abstraction, but the $0.75\times$ pot bet is never available. We calculate the exploitability of one player using the smaller abstraction, while the other player uses the larger abstraction. Whenever the large-abstraction player chooses a $0.75\times$ pot bet, the small-abstraction player generates and solves a subgame for the remainder of the game (which again does not include any subsequent $0.75\times$ pot bets) using the nested search techniques described above. This subgame strategy is then used as long as the large-abstraction player plays within the small abstraction, but if she chooses the $0.75\times$ pot bet again later, then the search is used again, and so on.

Table 5.4 shows that all the search techniques substantially outperform action translation. Resolve, Maxmargin, and Reach-Maxmargin use inexpensive nested search, while Unsafe and "Reach-Maxmargin (expensive)" use the expensive approach. In all cases, we used estimates for the alternative payoff as described in Section 5.1.7. We did not test distributional alternative payoffs in this experiment, since the calculated best response values are likely quite accurate. Reach-Maxmargin performed the best, outperforming Maxmargin and Unsafe search. These results suggest that nested search is preferable to action translation (if there is sufficient time to solve the subgame).

	mbb/h
Randomized Pseudo-Harmonic Mapping	1,465
Resolve	150.2
Reach-Maxmargin (Expensive)	149.2
Unsafe (Expensive)	148.3
Maxmargin	122.0
Reach-Maxmargin	119.1

Table 5.4: Exploitability of the various search techniques in nested search. The performance of the pseudo-harmonic action translation is also shown.

We used the techniques presented in this paper in our AI Libratus (described in Section 6.4), which competed against four top human specialists in heads-up no-limit Texas hold'em in the January 2017 Brains vs. AI competition. Libratus was constructed by first solving an abstraction of the game via a new variant of Monte Carlo CFR [96] that prunes negative-regret actions [15, 17, 19]. Libratus applied nested search (solved with CFR+ [150]) upon reaching the third betting round, and in response to every subsequent opponent bet thereafter. This allowed Libratus to avoid information abstraction during play, and leverage nested search's far lower exploitability in response to opponent off-tree actions.

5.1.9 Conclusions

Search has been critical for achieving superhuman performance in perfect-information games such as backgammon [152], chess [34], and go [140]. However, applying search in imperfect-information games in a way that is theoretically sound has been a major challenge.

In this section we described search techniques for imperfect-information games that have stronger theoretical guarantees and better practical performance than prior search methods. We

presented results on exploitability of both safe and unsafe search techniques. We also introduced a method for nested search in response to the opponent's off-tree actions, and demonstrated that this leads to dramatically better performance than the usual approach of action translation. These search techniques were ultimately used in Libratus and Pluribus, which for the first time defeated top humans in no-limit Texas hold'em poker.

5.1.10 Description of Gadget Game

Solving the augmented subgame described in Maxmargin solving and Reach-Maxmargin solving will not, by itself, necessarily maximize the minimum margin. While LP solvers can easily handle this objective, the process is more difficult for iterative algorithms such as Counterfactual Regret Minimization (CFR) and the Excessive Gap Technique (EGT). For these iterative algorithms, the augmented subgame can be modified into a *gadget game* that, when solved, will provide a Nash equilibrium to the augmented subgame and will also maximize the minimum margin [109]. This gadget game is unnecessary when using distributional alternative payoffs, which is introduced in section 5.1.5.

The gadget game differs from the augmented subgame in two ways. First, all P_1 payoffs that are reached from the initial infoset of $I_1 \in S_r$ are shifted by the alternative payoff of I_1, and there is longer an alternative payoff. Second, rather than the game starting with a chance node that determines P_1's starting infoset, P_1 decides for herself which infoset to begin the game in. Specifically, the game begins with a P_1 node where each action in the node corresponds to an infoset I_1 in S_r. After P_1 chooses to enter an infoset I_1, chance chooses the precise node $h \in I_1$ in proportion to $\pi^{\sigma}_{-1}(h)$.

By shifting all payoffs in the game by the size of the alternative payoff, the gadget game forces P_1 to focus on improving the performance of each infoset over some baseline, which is the goal of Maxmargin and Reach-Maxmargin solving. Moreover, by allowing P_1 to choose the infoset in which to enter the game, the gadget game forces P_2 to focus on maximizing the minimum margin.

Figure 5.6 illustrates the gadget game used in Maxmargin and Reach-Maxmargin.

5.1.11 Scaling of Gifts

To retain the theoretical guarantees of Reach search, one must ensure that the gifts assigned to reachable subgames do not (in aggregate) exceed the original gift. That is, if $g(I_1)$ is a gift at infoset I_1, we must ensure that $CBV^{\sigma^*_2}(I_1) \leq CBV^{\sigma_2}(I_1) + g(I_1)$. In this paper we accomplish this by increasing the margin of an infoset I'_1, where $I_1 \sqsubseteq I'_1$, by at most $g(I_1)$. However, empirical performance may improve if the increase to margins due to gifts is scaled up by some factor. In most games we experimented on, exploitability decreased the further the gifts were scaled. However, Figure 5.7 shows one case in which we observe the exploitability increasing when the gifts are scaled up too far. The graph shows exploitability when the gifts are scaled by various factors. At 0, the algorithm is identical to Maxmargin. at 1, the algorithm is the theoretically correct form of Reach-Maxmargin. Optimal performance in this game occurs when the gifts are scaled by a factor of about $1,000$. Scaling the gifts by $100,000$ leads to performance that is worse than Maxmargin search. This empirically demonstrates that while scaling up gifts

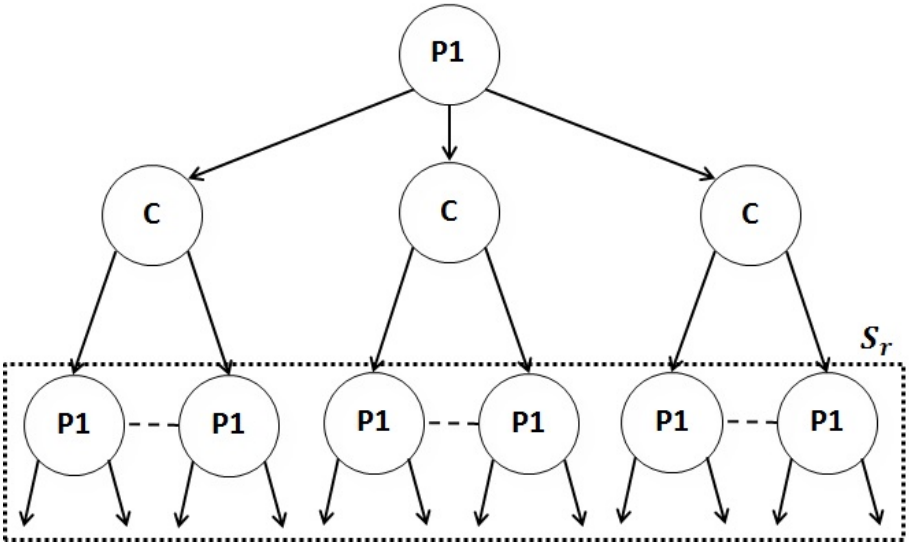


Figure 5.6: An example of a gadget game in Maxmargin refinement. P_1 picks the initial infoset she wishes to enter S_r in. Chance then picks the particular node of the infoset, and play then proceeds identically to the augmented subgame, except all P_1 payoffs are shifted by the size of the alternative payoff and the alternative payoff is then removed from the augmented subgame.

may lead to better performance in some cases (because an entire gift is unlikely to be used in every subgame that receives one), it may also lead to far worse performance in some cases.

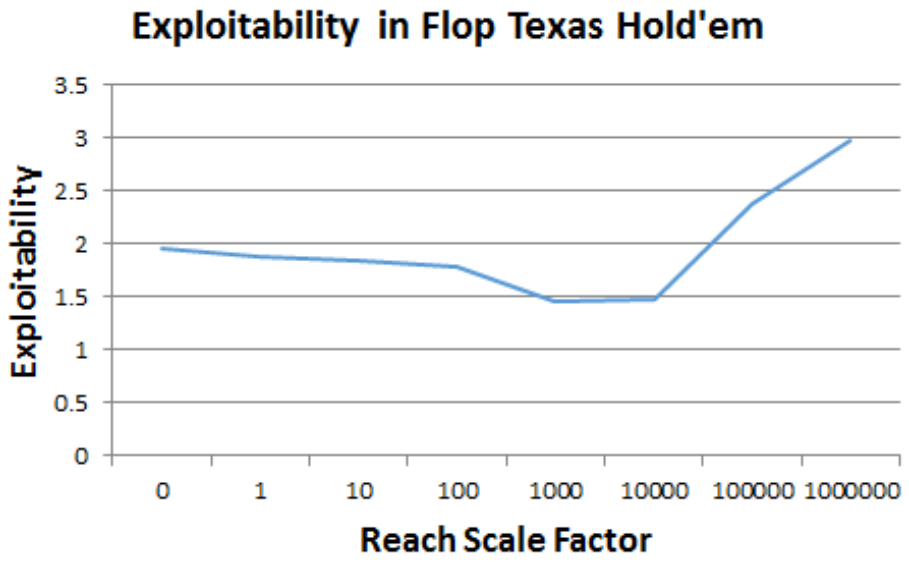


Figure 5.7: Exploitability in Flop Texas Hold'em of Reach-Maxmargin as we scale up the size of gifts.

5.1.12 Proofs of Theoretical Results

Proof of Theorem 20

Proof. Assume $M_r^{\sigma^S}(I_1) \geq 0$ for every infoset I_1 and assume $\pi_1^{BR(\sigma_2')}(I_1^*) > 0$ for some $I_1^* \in S_{top}$ and let $\epsilon = M_r(I_1^*)$. Define $\pi_{-1}^{\sigma}(I_1) = \sum_{h \in I_1} \pi_{-1}^{\sigma}(h)$ and define $\pi_{-1}^{\sigma}(I_1, I_1') = \sum_{h \in I_1, h' \in I_1'} \pi_{-1}^{\sigma}(h, h')$.

We show that for every P_1 infoset $I_1 \sqsubseteq I_1^*$ where $P(I_1) = P_1$,

$$CBV^{\sigma_2'}(I_1) \leq CBV^{\sigma_2^{-S}}(I_1) + \sum_{I_1'' \cdot a'' \sqsubseteq I_1 | P(I_1'') = P_1} \left(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'') \rfloor \right) - \sum_{h \in I_1, h^* \in I_1^*} \pi_{-1}^{\sigma_2}(h, h^*)\epsilon \quad (5.4)$$

By the definition of $M_r^{\sigma^S}(I_1^*)$ this holds for I_1^* itself. Moreover, the condition holds for every other $I_1 \in S_{top}$, because by assumption every margin is nonnegative and $\pi_{-1}^{\sigma_2}(I_1, I_1^*) = 0$ for any $I_1 \in S_{top}$ where $I_1 \neq I_1^*$. The condition also clearly holds for any I_1 with no descendants in S because then $\pi_{-1}^{\sigma_2}(I_1, I_1^*) = 0$ and $\sigma_2'(h) = \sigma_2^{-S}(h)$ in all P_2 nodes following I_1. This satisfies the base step. We now move on to the inductive step.

Let $Succ(I_1, a)$ be the set of earliest-reachable P_1 infosets following I_1 such that $P(I_1') = P_1$ for $I' \in Succ(I_1, a)$. Formally, $I_1' \in Succ(I_1, a)$ if $P(I_1') = P_1$ and $I_1 \cdot a \sqsubseteq I_1'$ and for any other $I_1'' \in Succ(I_1, a)$, $I_1'' \not\sqsubset I_1'$. Then

$$CBV^{\sigma_2'}(I_1, a) = CBV^{\sigma_2^{-S}}(I_1, a) + \sum_{I_1' \in Succ(I_1, a)} \pi_{-1}^{\sigma_2'}(I_1, I_1')(CBV^{\sigma_2'}(I_1') - CBV^{\sigma_2^{-S}}(I_1')) \quad (5.5)$$

Assume that every $I_1' \in Succ(I_1, a)$ satisfies (5.4). Then

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^{-S}}(I_1, a) - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1' \in Succ(I_1, a)} \pi_{-1}^{\sigma_2}(I_1, I_1') \Big(\sum_{I_1'' \cdot a'' \sqsubseteq I_1' | P(I_1'') = P_1} \left(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'') \rfloor \right) \Big)$$

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^{-S}}(I_1) - \Big(CBV^{\sigma_2^{-S}}(I_1) - CBV^{\sigma_2^{-S}}(I_1, a) \Big) - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1' \in Succ(I_1, a)} \pi_{-1}^{\sigma_2}(I_1, I_1') \Big(\sum_{I_1'' \cdot a'' \sqsubseteq I_1' | P(I_1'') = P_1} \left(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'') \rfloor \right) \Big)$$

Since $\lfloor CBV^{\sigma_2^{-S}}(I_1) - CBV^{\sigma_2^{-S}}(I_1, a) \rfloor \leq CBV^{\sigma_2^{-S}}(I_1) - CBV^{\sigma_2^{-S}}(I_1, a_1)$ so we get

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^{-S}}(I_1) - \lfloor (CBV^{\sigma_2^{-S}}(I_1) - CBV^{\sigma_2^{-S}}(I_1, a) \rfloor - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1' \in Succ(I_1, a)} \pi_{-1}^{\sigma_2}(I_1, I_1') \Big(\sum_{I_1'' \cdot a'' \sqsubseteq I_1' | P(I_1'') = P_1} \left(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'') \rfloor \right) \Big)$$

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^{-S}}(I_1) - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1' \in Succ(I_1,a)} \pi_{-1}^{\sigma_2}(I_1, I_1')\Big(\sum_{I_1'' \cdot a'' \sqsubseteq I_1 | P(I_1'')=P_1} \big(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'')\rfloor\big)\Big)$$

$$CBV^{\sigma_2'}(I_1, a_1) \leq CBV^{\sigma_2^{-S}}(I_1) - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1'' \cdot a'' \sqsubseteq I_1 | P(I_1'')=P_1} \big(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a_1'')\rfloor\big)$$

Since $\pi_1^{BR(\sigma_2')}(I_1^*) > 0$, and action a leads to I_1^*, so by definition of a best response, $CBV^{\sigma_2'}(I_1, a) = CBV^{\sigma_2'}(I_1)$. Thus,

$$CBV^{\sigma_2'}(I_1) \leq CBV^{\sigma_2^{-S}}(I_1) - \pi_{-1}^{\sigma_2}(I_1, I_1^*)\epsilon + \sum_{I_1'' \cdot a'' \sqsubseteq I_1 | P(I_1'')=P_1} \big(\lfloor CBV^{\sigma_2^{-S}}(I_1'') - CBV^{\sigma_2^{-S}}(I_1'', a'')\rfloor\big)$$

which satisfies the inductive step.

Applying this reasoning to the root of the entire game, we arrive at $exp(\sigma_2') \leq exp(\sigma_2^{-S}) - \pi_{-1}^{\sigma_2}(I_1^*)\epsilon$. □

Proof of Theorem 21

Proof. Without loss of generality, we assume that it is player P_2 who conducts search. We define a node h in a subgame S as *earliest-reachable* if there does not exist a node $h' \in S$ such that $h' \prec h$. For each earliest-reachable node $h \in S$, let h_r be its parent and a_S be the action leading to h such that $h_r \cdot a_S = h$. We require h_r to be a P_1 node; if it is not, then we can simply insert a P_1 node with only a single action between h_r and h. Let S_r be the set of all h_r for S.

Applying search to subgames as they are reached during play is equivalent to applying search to every subgame before play begins, so we can phrase what follows in the context of all subgames being solved before play begins. Let σ_2' be the P_2 strategy produced after search is applied to every subgame. We show inductively that for any P_1 infoset $I_1 \notin \mathcal{S}$ where it is P_1's turn to move (i.e., $P(I_1) = P_1$), the counterfactual best response values for P_1 satisfy

$$CBV^{\sigma_2'}(I_1) \leq CBV^{\sigma_2^*}(I_1) + 2\Delta \tag{5.6}$$

Define *Succ*(I_1, a) as the set of infosets belonging to P_1 that follow action a in I_1 and where it is P_1's turn and where P_1 has not had a turn since a, as well as terminal nodes follow action a in I_1 without P_1 getting a turn. Formally, a terminal node $z \in Z$ is in *Succ*(I_1, a) if there exists a history $h \in I_1$ such that $h \cdot a \preceq z$ and there does not exist a history h' such that $P(h') = P_1$ and $h \cdot a \preceq h' \prec z$. Additionally, an infoset I_1' belonging to P_1 is in *Succ*(I_1, a) if $P(I_1') = P_1$ and $I_1 \cdot a \preceq I_1'$ and there does not exist an earlier infoset I_1'' belonging to P_1 such that $P(I_1'') = P_1$ and $I' \cdot a \preceq I_1'' \prec I_1'$. Define *Succ*$(I_1)$ as $\cup_{a \in A(I_1)}$*Succ*(I_1, a). Similarly, we define *Succ*(h, a) as the set of histories belonging to $P(h)$, or terminals, that follow action a and where $P(h)$ has not had a turn since a. Formally, $h' \in$ *Succ*(h, a) if either $P(h') = P(h)$ or $P(h') \in Z$ and $h \cdot a \preceq h'$ and there does not exist a history h'' such that $P(h'') = P(h)$ and $h \cdot a \preceq h'' \prec h'$.

Now we define a level L for each P_1 infoset where it is P_1's turn and the infoset is not in the set of subgames $\mathcal{S}$.

- For immediate parents of subgames we define the level to be zero: for all $I_1 \in S_r$ for any subgame $S \in \mathcal{S}$, $L(I_1) = 0$.
- For infoset that are not ancestors of subgames, we define the level to be zero: $L(I_1) = 0$ for any infoset I_1 that is not an ancestor of a subgame in $\mathcal{S}$.
- For all other infosets, the level is one greater than the greatest level of its successors: $L(I_1) = \ell + 1$ where $\ell = \max_{I_1' \in \mathit{Succ}(I_1)} L(I_1')$ where $L(z) = 0$ for terminal nodes z.

Base case of induction

First consider infosets $I_1 \in S_r$ for some subgame $S \in \mathcal{S}$. We define $M^{\sigma_2'}(I_1) = v^{\sigma}(I_1, a_S) - CBV^{\sigma_2'}(I_1, a_S)$. Consider a subgame $S \in \mathcal{S}$. Estimated-Maxmargin search arrives at a strategy σ_2' such that $\min_{I_1 \in S_r} M^{\sigma_2'}(I_1)$ is maximized. By the assumption in the theorem statement, $|v^{\sigma}(I_1, a_S) - CBV^{\sigma_2^*}(I_1, a_S)| \leq \Delta$ for all $I_1 \in S_r$. Thus, σ_2^* satisfies $\min_{I_1 \in S_r} M^{\sigma_2^*}(I_1) \geq -\Delta$ and therefore $\min_{I_1 \in S_r} M^{\sigma_2'}(I_1) \geq -\Delta$, because Estimated-Maxmargin search could, at least, arrive at $\sigma_2' = \sigma_2^*$. From the definition of $M^{\sigma_2'}(I_1)$, this implies that for all $I_1 \in S_r$, $CBV^{\sigma_2'}(I_1, a_S) \leq v^{\sigma}(I_1, a_S) + \Delta$. Since by assumption $v^{\sigma}(I_1, a_S) \leq CBV^{\sigma_2^*}(I_1, a_S) + \Delta$, this gives us $CBV^{\sigma_2'}(I_1, a_S) \leq CBV^{\sigma_2^*}(I_1, a_S) + 2\Delta$.

Now consider infosets I_1 that are not ancestors of any subgame in $\mathcal{S}$. By definition, for all h such that $h \succeq I_1$ or $I_1 \succeq h$, and $P(h) = P_2$, $\sigma_2^*(I_2(h)) = \sigma_2(I_2(h)) = \sigma_2'(I_2(h))$. Therefore, $CBV^{\sigma_2'}(I_1) = CBV^{\sigma_2^*}(I_1)$.

So, we have shown that (5.6) holds for any I_1 such that $L(I_1) = 0$.

Inductive step

Now assume that (5.6) holds for any P_1 infoset I_1 where $P(I_1) = P_1$ and $I_1 \notin \mathcal{S}$ and $L(I_1) \leq \ell$. Consider an I_1 such that $P(I_1) = P_1$ and $I_1 \notin \mathcal{S}$ and $L(I_1) = \ell + 1$.

From the definition of $CBV^{\sigma_2'}(I_1, a)$, we have that for any action $a \in A(I_1)$,

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{h \in I_1} \Big(\big(\pi_{-1}^{\sigma_2'}(h)\big) \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h \cdot a)\big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{5.7}$$

Since for any $h \in I_1$ there is no P_1 action between a and reaching any $h' \in \mathit{Succ}(h, a)$, so $\pi_1^{\sigma_2'}(h \cdot a, h') = 1$. Thus,

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{h \in I_1} \Big(\pi_{-1}^{\sigma_2'}(h) \sum_{h' \in \mathit{Succ}(h,a)} \pi_{-1}^{\sigma_2'}(h, h') \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h')\big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{5.8}$$

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{h \in I_1} \sum_{h' \in \mathit{Succ}(h,a)} \Big(\big(\pi_{-1}^{\sigma_2'}(h')\big) v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h') \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{5.9}$$

Since the game is perfect recall, $\sum_{h \in I_1} \sum_{h' \in \mathit{Succ}(h,a)} f(h') = \sum_{I_1' \in \mathit{Succ}(I_1, a)} \sum_{h' \in I_1'} f(h')$ for any function f. Thus,

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{I_1' \in \mathit{Succ}(I_1,a)} \sum_{h' \in I_1'} \Big(\big(\pi_{-1}^{\sigma_2'}(h')\big) \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h')\big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{5.10}$$

From the definition of $CBV^{\sigma'_2}(I'_1)$ we get

$$CBV^{\sigma'_2}(I_1, a) = \Big(\sum_{I'_1 \in Succ(I_1,a)} \big(CBV^{\sigma'_2}(I'_1) \sum_{h' \in I'_1} \pi^{\sigma'_2}_{-1}(h') \big) \Big) / \sum_{h \in I_1} \pi^{\sigma'_2}_{-1}(h) \tag{5.11}$$

Since (5.6) holds for all $I'_1 \in Succ(I_1, a)$, so

$$CBV^{\sigma'_2}(I_1, a) \leq \Big(\sum_{I'_1 \in Succ(I_1,a)} \big((CBV^{\sigma^*_2}(I'_1) + 2\Delta) \sum_{h' \in I'_1} \pi^{\sigma'_2}_{-1}(h') \big) \Big) / \sum_{h \in I_1} \pi^{\sigma'_2}_{-1}(h) \tag{5.12}$$

Since P_2's strategy is fixed according to σ_2 outside of $\mathcal{S}$, so for all $I_1 \notin \mathcal{S}$, $\pi^{\sigma'}_{-1}(I_1) = \pi^{\sigma}_{-1}(I_1) = \pi^{\sigma^*}_{-1}(I_1)$. Therefore,

$$CBV^{\sigma'_2}(I_1, a) \leq \Big(\sum_{I'_1 \in Succ(I_1,a)} \big((CBV^{\sigma^*_2}(I'_1) + 2\Delta) \sum_{h' \in I'_1} \pi^{\sigma^*_2}_{-1}(h') \big) \Big) / \sum_{h \in I_1} \pi^{\sigma^*_2}_{-1}(h) \tag{5.13}$$

Pulling out the 2Δ constant and applying equation (5.11) for $CBV^{\sigma^*_2}(I_1, a)$ we get

$$CBV^{\sigma'_2}(I_1, a) \leq CBV^{\sigma^*}(I_1, a) + 2\Delta \Big(\big(\sum_{I'_1 \in Succ(I_1,a)} \sum_{h' \in I'_1} \pi^{\sigma^*_2}_{-1}(h') \big) / \sum_{h \in I_1} \pi^{\sigma^*_2}_{-1}(h) \Big) \tag{5.14}$$

Since $\big(\sum_{I'_1 \in Succ(I_1,a)} \sum_{h' \in I'_1} \pi^{\sigma^*_2}_{-1}(h') \big) = \sum_{h \in I_1} \pi^{\sigma^*_2}_{-1}(h)$ we arrive at

$$CBV^{\sigma'_2}(I_1, a) \leq CBV^{\sigma^*}(I_1, a) + 2\Delta \tag{5.15}$$

Thus, (5.6) holds for I_1 as well and the inductive step is satisfied. Extending (5.6) to the root of the game, we see that $exp(\sigma'_2) \leq exp(\sigma^*_2) + 2\Delta$. □

Proof of Theorem 22

Proof. We prove inductively that using CFR in S' while choosing the action leading to S' from each $I_1 \in S'_r$ with probability $P\big(X_{I_1} \leq v^t(I_1, a'_S)\big)$ results in play that is identical to CFR in S and CFR-BR [80] in S_r, which converges to a Nash equilibrium.

For each P_2 infoset I'_2 in S' where $P(I'_2) = P_2$, there is exactly one corresponding infoset I_2 in S that is reached via the same actions, ignoring random variables. Each P_1 infoset I'_1 in S' where $P(I'_1) = P_1$ corresponds to a set of infosets in S that are reached via the same actions, where the elements in the set differ only by the outcome of the random variables. We prove that on each iteration, the instantaneous regret for these corresponding infosets is identical (and therefore the average strategy played in the P_2 infosets over all iterations is identical).

At the start of the first iteration of CFR, all regrets are zero. Therefore, the base case is trivially true. Now assume that on iteration t, regrets are identical for all corresponding infosets. Then the strategies played on iteration t in S are identical as well.

First, consider an infoset I'_1 in S' and a corresponding infoset I_1 in S. Since the remaining structure of the game is identical beyond I'_1 and I_1, and because P_2's strategies are identical in all P_2 infosets encountered, so the immediate regret for I'_1 and I_1 is identical as well.

Next, consider a P_1 infoset $I_{1,x}$ in S_r in which the random variable X_{I_1} has an observed value of x. Let the corresponding P_1 infoset in S'_r be I'_1. Since CFR-BR is played in this infoset, and since action a'_T leads to a payoff of x, so P_1 will choose action a'_S with probability 1 if $x \geq a'_T$ and with probability 0 otherwise. Thus, for all infosets in S_r corresponding to I'_1, action a'_S is chosen with probability $P\big(X_{I_1} \leq v(I_1, a'_S)\big)$.

Finally, consider a P_2 infoset I_2 in S and its corresponding infoset I'_2 in S'. Since in both cases action a'_T is taken in S_r with probability $P\big(X_{I_1} \leq v(I_1, a'_S)\big)$, and because P_1 plays identically between corresponding infosets in S and S', and because the structure of the game is otherwise identical, so the immediate regret for I'_1 and I_1 is identical as well. □

5.2 Depth-Limited Search via Multi-Valued States

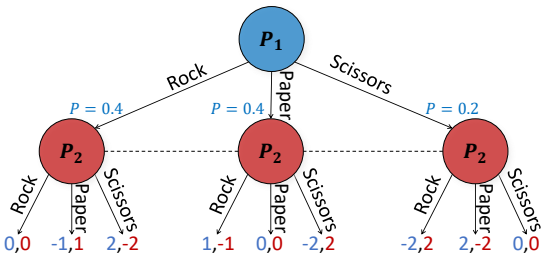
The key breakthrough that led to superhuman performance in HUNL poker was nested search, described in Section 5.1, in which the agent repeatedly calculates a finer-grained strategy in real time (for just a portion of the full game) as play proceeds down the game tree [20, 21, 110].

However, real-time search was too expensive for *Libratus* in the first half of the game because the subgame that *Libratus* searched in real time always extended to the end of the game. Instead, for the first half of the game *Libratus* pre-computed a fine-grained strategy that was used as a lookup table. While this pre-computed strategy was successful, it required millions of core hours and terabytes of memory to calculate. Moreover, in deeper sequential games the computational cost of this approach would be even more expensive because either longer subgames or a larger pre-computed strategy would need to be solved. A more general approach would be to solve **depth-limited subgames**, which may not extend to the end of the game. These could be solved even in the early portions of a game.

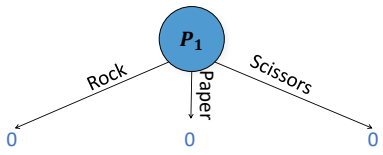
When conducting depth-limited search, a primary challenge is determining what values to substitute at the leaf nodes of the depth-limited subgame. In perfect-information depth-limited subgames, the value substituted at leaf nodes is simply an estimate of the state's value when all players play an equilibrium [125, 137]. For example, this approach was used to achieve superhuman performance in backgammon [153], chess [34], and Go [140, 141]. The same approach is also widely used in single-agent settings such as heuristic search [64, 102, 115, 117]. Indeed, in single-agent and perfect-information multi-agent settings, knowing the values of states when all agents play an equilibrium is sufficient to reconstruct an equilibrium. However, this does not work in imperfect-information games, as we demonstrate in the next section.

This section describes a way to solve depth-limited imperfect-information subgames which we call the **multi-valued states** approach. The crux of the approach is to assign multiple values to leaf nodes corresponding to different combinations of blueprint strategies that players might play for the remainder of the game. When a leaf node is reached, each player simultaneously selects one of these blueprint strategies and a reward is returned based on the joint choice. This technique was ultimately used in Pluribus to defeat top humans in multiplayer no-limit Texas hold'em poker for the first time (specifically, six-player NLTH). Additionally, Pluribus was orders of magnitude computationally more efficient that Libratus, despite the game being orders of magnitude larger. This is efficiency is primarily due to depth-limited search via multi-valued states.

Another way to solve depth-limited subgames, described in Section 5.3, involves a public



5.8a Rock-Paper-Scissors+ shown with the optimal P_1 strategy. The terminal values are shown first for P_1, then P_2. The red lines between the P_2 nodes means they are indistinguishable to P_2.



5.8b A depth-limited subgame of Rock-Paper-Scissors+ with state values determined from the equilibrium.

belief state (PBS) value function. This PBS value function was originally used by the poker AI DeepStack [110], and was also used by the poker AI ReBeL [29]. However, training and using a PBS value function is orders of magnitude more expensive than the multi-valued states approach, especially in the way used by DeepStack. Nevertheless, ReBeL has some advantages over the multi-valued states approach. Section 5.4 contains a thorough comparison of the two techniques.

5.2.1 The Challenge of Depth-Limited Search in Imperfect-Information Games

In imperfect-information games (also referred to as partially observable games), an optimal strategy cannot be determined in a subgame simply by knowing the values of states (i.e., game-tree nodes) when all players play an equilibrium strategy. A simple demonstration is in Figure 5.8a, which shows a sequential game we call Rock-Paper-Scissors+ (RPS+). RPS+ is identical to traditional Rock-Paper-Scissors, except if either player plays Scissors, the winner receives 2 points instead of 1 (and the loser loses 2 points). Figure 5.8a shows RPS+ as a sequential game in which P_1 acts first but does not reveal the action to P_2 [31, 51]. The optimal strategy (Minmax strategy, which is also a Nash equilibrium in two-player zero-sum games) for both players in this game is to choose Rock and Paper each with 40% probability, and Scissors with 20% probability. In this equilibrium, the expected value to P_1 of choosing Rock is 0, as is the value of choosing Scissors or Paper. In other words, all the red states in Figure 5.8a have value 0 in the equilibrium. Now suppose P_1 conducts a depth-limited search with a depth of one in which the equilibrium values are substituted at that depth limit. This depth-limited subgame is shown in Figure 5.8b. Clearly, there is not enough information in this subgame to arrive at the optimal strategy of 40%, 40%, and 20% for Rock, Paper, and Scissors, respectively.

In the RPS+ example, the core problem is that we incorrectly assumed P_2 would always play a fixed strategy. If indeed P_2 were to always play Rock, Paper, and Scissors with probability $\langle 0.4, 0.4, 0.2 \rangle$, then P_1 could choose any arbitrary strategy and receive an expected value of 0. However, by assuming P_2 is playing a fixed strategy, P_1 may not find a strategy that is robust to P_2 adapting. In reality, P_2's optimal strategy depends on the probability that P_1 chooses Rock,

Paper, and Scissors. In general, in imperfect-information games a player's optimal strategy at a decision point depends on the player's belief distribution over states as well as the strategy of all other agents beyond that decision point.

In this section we introduce a method for depth-limited search that ensures a player is robust to such opponent adaptations. Rather than simply substitute a single state value at a depth limit, we instead allow the opponent one final choice of action at the depth limit, where each action corresponds to a strategy the opponent will play in the remainder of the game. The choice of strategy determines the value of the state. The opponent does not make this choice in a way that is specific to the state (in which case he would trivially choose the maximum value for himself). Instead, naturally, the opponent must make the same choice at all states that are indistinguishable to him. We prove that if the opponent is given a choice between a sufficient number of strategies at the depth limit, then any solution to the depth-limited subgame is part of a Nash equilibrium strategy in the full game. We also show experimentally that when only a few choices are offered (for computational speed), performance of the method is extremely strong.

5.2.2 Multi-Valued States in Imperfect-Information Games

We now describe our new method for depth-limited solving in imperfect-information games, which we refer to as **multi-valued states**. Our general approach is to first precompute an approximate Nash equilibrium for the entire game. We refer to this precomputed strategy profile as a **blueprint**. Since the blueprint is precomputed for the entire game, it is likely just a coarse approximation of a true Nash equilibrium. Our goal is to compute a better approximation in real time for just a depth-limited subgame S that we find ourselves in during play. For the remainder of the description of our technique, we assume that player P_1 is attempting to approximate a Nash equilibrium strategy in S.

Let σ^* be an exact Nash equilibrium. To present the intuition for our approach, we begin by considering what information about σ^* would, in theory, be sufficient in order to compute a P_1 Nash equilibrium strategy in S. For ease of understanding, when considering the intuition for multi-valued states we suggest the reader first focus on the case where S is rooted at the start of the game (that is, no prior actions have occurred).

As explained in Section 5.2.1, knowing the values of leaf nodes in S when both players play according to σ^* (that is, $v_i^{\sigma^*}(h)$ for leaf node h and player P_i) is insufficient to compute a Nash equilibrium in S (even though this is sufficient in perfect-information games), because it assumes P_2 would not adapt their strategy outside S. But what if P_2 could adapt? Specifically, suppose hypothetically that P_2 could choose *any* strategy in the entire game, while P_1 could only play according to σ_1^* outside of S. In this case, what strategy should P_1 choose in S? Since σ_1^* is a Nash equilibrium strategy and P_2 can choose any strategy in the game (including a best response to P_1's strategy), so by definition P_1 cannot do better than playing σ_1^* in S. Thus, P_1 should play σ_1^* (or some equally good Nash equilibrium) in S.

Another way to describe this setup is that upon reaching a leaf node h in infoset I in subgame S, rather than simply substituting $v_2^{\sigma^*}(h)$ (which assumes P_2 plays according to σ_2^* for the remainder of the game), P_2 could instead choose any mixture of pure strategies for the remainder of the game. So if there are N possible pure strategies following I, P_2 would choose among N actions upon reaching I, where action n would correspond to playing pure strategy σ_2^n for the

remainder of the game. Since this choice is made separately at each infoset I and since P_2 may mix between pure strategies, so this allows P_2 to choose *any* strategy below S.

Since the choice of action would define a P_2 strategy for the remainder of the game and since P_1 is known to play according to σ_1^* outside S, so the chosen action could immediately reward the expected value $v_i^{\langle \sigma_1^*, \sigma_2^n \rangle}(h)$ to P_i. Therefore, in order to reconstruct a P_1 Nash equilibrium in S, it is sufficient to know for every leaf node the expected value of every pure P_2 strategy against σ_1^* (stated formally in Proposition 3). This is in contrast to perfect-information games, in which it is sufficient to know for every leaf node just the expected value of σ_2^* against σ_1^*. Critically, it is not necessary to know the *strategy* σ_1^*, just the *values* of σ_1^* played against every pure opponent strategy in each leaf node.

Proposition 3 adds the condition that we know $v_2^{\langle \sigma_1^*, BR(\sigma_1^*) \rangle}(I)$ for every root infoset $I \in S$. This condition is used if S does not begin at the start of the game. Knowledge of $v_2^{\langle \sigma_1^*, BR(\sigma_1^*) \rangle}(I)$ is needed to ensure that any strategy σ_1 that P_1 computes in S cannot be exploited by P_2 changing their strategy earlier in the game. Specifically, we add a constraint that $v_2^{\langle \sigma_1, BR(\sigma_1^*) \rangle}(I) \leq v_2^{\langle \sigma_1^*, BR(\sigma_1^*) \rangle}(I)$ for all P_2 root infosets I. This makes our technique *safe*:

Proposition 3. *Assume P_1 has played according to Nash equilibrium strategy σ_1^* prior to reaching a depth-limited subgame S of a two-player zero-sum game. In order to calculate the portion of a P_1 Nash equilibrium strategy that is in S, it is sufficient to know $v_2^{\langle \sigma_1^*, BR(\sigma_1^*) \rangle}(I)$ for every root P_2 infoset $I \in S$ and $v_1^{\langle \sigma_1^*, \sigma_2 \rangle}(h)$ for every pure undominated P_2 strategy σ_2 and every leaf node $h \in S$.*

Other safe search techniques have been developed in recent papers, but those techniques require solving to the end of the full game [20, 21, 31, 74, 109] (except one [110], which we will compare to in Section 5.4).

Of course, it is impractical to know the expected value in every state of every pure P_2 strategy against σ_1^*, especially since we do not know σ_1^* itself. To deal with this, we first compute a blueprint strategy $\hat{\sigma}^*$ (that is, a precomputed approximate Nash equilibrium for the full game). Next, rather than consider every pure P_2 strategy, we instead consider just a small number of different P_2 strategies (that may or may not be pure). Indeed, in many complex games, the possible opponent strategies at a decision point can be approximately grouped into just a few "meta-strategies", such as which highway lane a car will choose in a driving simulation. In our experiments, we find that excellent performance is obtained in poker with fewer than ten opponent strategies. In part, excellent performance is possible with a small number of strategies because the choice of strategy beyond the depth limit is made separately at each leaf infoset. Thus, if the opponent chooses between ten strategies at the depth limit, but makes this choice independently in each of 100 leaf infosets, then the opponent is actually choosing between 10^{100} different strategies. We now consider two questions. First, how do we compute the blueprint strategy $\hat{\sigma}_1^*$? Second, how do we determine the set of P_2 strategies? We answer each of these in turn.

There exist several methods for constructing a blueprint. One option, which achieves the best empirical results and is what we use, involves first abstracting the game by bucketing together similar situations [50, 80] and then applying the iterative algorithm Monte Carlo Counterfactual Regret Minimization [96]. Several alternatives exist that do not use a distinct abstraction step [16,

35, 69]. The agent will never actually play according to the blueprint $\hat{\sigma}^*$. It is only used to estimate $v^{\langle \sigma_1^*, \sigma_2 \rangle}(h)$.

We now discuss two different ways to select a set of P_2 strategies. Ultimately we would like the set of P_2 strategies to contain a diverse set of intelligent strategies the opponent might play, so that P_1's solution in a subgame is robust to possible P_2 adaptation. One option is to bias the P_2 blueprint strategy $\hat{\sigma}_2^*$ in a few different ways. For example, in poker the blueprint strategy should be a mixed strategy involving some probability of folding, calling, or raising. We could define a new strategy σ_2' in which the probability of folding is multiplied by 10 (and then all the probabilities renormalized). If the blueprint strategy $\hat{\sigma}^*$ were an exact Nash equilibrium, then any such "biased" strategy σ_2' in which the probabilities are arbitrarily multiplied would still be a best response to $\hat{\sigma}_1^*$. In our experiments, we use this biasing of the blueprint strategy to construct a set of four opponent strategies on the second betting round. We refer to this as the *bias approach*.

Another option is to construct the set of P_2 strategies via self-play. The set begins with just one P_2 strategy: the blueprint strategy $\hat{\sigma}_2^*$. We then solve a depth-limited subgame rooted at the start of the game and going to whatever depth is feasible to solve, giving P_2 only the choice of this P_2 strategy at leaf infosets. That is, at leaf node h we simply substitute $v_i^{\hat{\sigma}^*}(h)$ for P_i. Let the P_1 solution to this depth-limited subgame be σ_1. We then approximate a P_2 best response assuming P_1 plays according to σ_1 in the depth-limited subgame and according to $\hat{\sigma}_1^*$ in the remainder of the game. Since P_1 plays according to this fixed strategy, approximating a P_2 best response is equivalent to solving a Markov Decision Process, which is far easier to solve than an imperfect-information game. This P_2 approximate best response is added to the set of strategies that P_2 may choose at the depth limit, and the depth-limited subgame is solved again. This process repeats until the set of P_2 strategies grows to the desired size. This self-generative approach bears some resemblance to the double oracle algorithm [106] and recent work on generation of opponent strategies in multi-agent RL [98]. In our experiments, we use this self-generative method to construct a set of ten opponent strategies on the first betting round. We refer to this as the *self-generative approach*.

One practical consideration is that since $\hat{\sigma}_1^*$ is not an exact Nash equilibrium, a generated P_2 strategy σ_2 may do better than $\hat{\sigma}_2^*$ against $\hat{\sigma}_1^*$. In that case, P_1 may play more conservatively than σ_1^* in a depth-limited subgame. To correct for this, one can balance the players by also giving P_1 a choice between multiple strategies for the remainder of the game at the depth limit.

Once a P_1 strategy $\hat{\sigma}_1^*$ and a set of P_2 strategies have been generated, we need some way to calculate and store $v_2^{\langle \hat{\sigma}_1^*, \sigma_2 \rangle}(h)$. Calculating the state values can be done by traversing the entire game tree once. However, that may not be feasible in large games. Instead, one can use Monte Carlo simulations to approximate the values. For storage, if the number of states is small (such as in the early part of the game tree), one could simply store the values in a table. More generally, one could train a function to predict the values corresponding to a state, taking as input a description of the state and outputting a value for each P_2 strategy. Alternatively, one could simply store $\hat{\sigma}_1^*$ and the set of P_2 strategies. Then, in real time, the value of a state could be estimated via Monte Carlo rollouts. We present results for both of these approaches in Section 5.2.3.

5.2.3 Experiments

We conducted experiments on the games of heads-up no-limit Texas hold'em poker (HUNL) (described in Section 2.4.3) and heads-up no-limit flop hold'em poker (NLFH) (described in Section 2.4.4). Performance is measured in terms of mbb/g, which is a standard win rate measure in the literature. It stands for milli-big blinds per game and represents how many thousandths of a big blind (the initial money a player must commit to the pot) a player wins on average per hand of poker played.

Exploitability Experiments in No-Limit Flop Hold'em

Our first experiment measured the *exploitability* of our technique in NLFH. Exploitability of a strategy in a two-player zero-sum game is how much worse the strategy would do against a best response than a Nash equilibrium strategy would do against a best response. Formally, the exploitability of σ_1 is $\min_{\sigma_2} u_1(\sigma_1^*, \sigma_2) - \min_{\sigma_2} u_1(\sigma_1, \sigma_2)$, where σ_1^* is a Nash equilibrium strategy.

We considered the case of P_1 betting $0.75\times$ the pot at the start of the game, when the action abstraction only contains bets of $0.5\times$ and $1\times$ the pot. We compared our depth-limited solving technique to the randomized pseudoharmonic action translation (RPAT) [49], in which the bet of $0.75\times$ is simply treated as either a bet of $0.5\times$ or $1\times$. RPAT is the lowest-exploitability known technique for responding to off-tree actions that does not involve real-time computation.

We began by calculating an approximate Nash equilibrium in an action abstraction that does not include the $0.75\times$ bet. This was done by running the CFR+ equilibrium-approximation algorithm [150] for 1,000 iterations, which resulted in less than 1 mbb/g of exploitability within the action abstraction. Next, values for the states at the end of the first betting round within the action abstraction were determined using the self-generative method discussed in Section 5.2.2. Since the first betting round is a small portion of the entire game, storing a value for each state in a table required just 42 MB.

To determine a P_2 strategy in response to the $0.75\times$ bet, we constructed a depth-limited subgame rooted after the $0.75\times$ bet with leaf nodes at the end of the first betting round. The values of a leaf node in this subgame were set by first determining the in-abstraction leaf nodes corresponding to the exact same sequence of actions, except P_1 initially bets $0.5\times$ or $1\times$ the pot. The leaf node values in the $0.75\times$ subgame were set to the average of those two corresponding value vectors. When the end of the first betting round was reached and the board cards were dealt, the remaining game was solved using safe subgame solving.

Figure 5.8 shows how exploitability decreases as we add state values (that is, as we give P_1 more best responses to choose from at the depth limit). When using only one state value at the depth limit (that is, assuming P_1 would always play according to the blueprint strategy for the remainder of the game), it is actually better to use RPAT. However, after that our technique becomes significantly better and at 16 values its performance is close to having had the $0.75\times$ action in the abstraction in the first place.

While one could have calculated a (slightly better) P_2 strategy in response to the $0.75\times$ bet by solving to the end of the game, that subgame would have been about $10{,}000\times$ larger than the subgames solved in this experiment. Thus, depth-limited solving dramatically reduces the

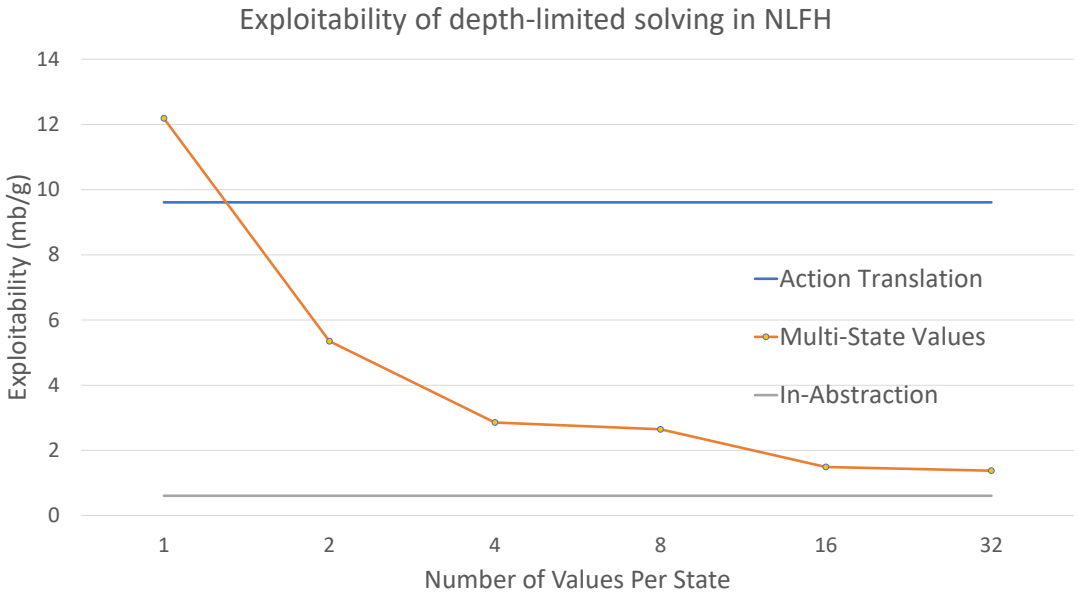


Figure 5.8: Exploitability of depth-limited solving in response to an opponent off-tree action as a function of number of state values. We compare to action translation and to having had the off-tree action included in the action abstraction (which is a lower bound on the exploitability achievable with 1,000 iterations of CFR+).

computational cost of nested subgame solving while giving up very little solution quality.

Experiments Against Top AIs in Heads-Up No-Limit Texas Hold'em

Our main experiment uses depth-limited solving to produce a master-level HUNL poker AI called *Modicum* (described in detail in Section 6.5) using computing resources found in a typical laptop. We test *Modicum* against Baby Tartanian8 [17], the winner of the 2016 Annual Computer Poker Competition, and against Slumbot [76], the winner of the 2018 Annual Computer Poker Competition. Neither Baby Tartanian8 nor Slumbot uses real time computation; their strategies are a precomputed lookup table. Baby Tartanian8 used about 2 million core hours and 18 TB of RAM to compute its strategy. Slumbot used about 250,000 core hours and 2 TB of RAM to compute its strategy. In contrast, *Modicum* used just 700 core hours and 16GB of RAM to compute its strategy and can play in real time at the speed of human professionals (an average of 20 seconds for an entire hand of poker) using just a 4-core CPU.

The blueprint strategy for *Modicum* was constructed by first generating an abstraction of HUNL using state-of-the-art abstraction techniques [50, 82]. Storing a strategy for this abstraction as 4-byte floats requires just 5 GB. This abstraction was approximately solved by running Monte Carlo Counterfactual Regret Minimization for 700 core hours [96].

HUNL consists of four betting rounds. We conduct depth-limited solving on the first two rounds by solving to the end of that round using MCCFR. Once the third betting round is reached, the remaining game is small enough that we solve to the end of the game using an enhanced form of CFR+ described in the appendix.

We generated 10 values for each state at the end of the first betting round using the self-

generative approach. The first betting round was small enough to store all of these state values in a table using 240 MB. For the second betting round, we used the bias approach to generate four opponent best responses. The first best response is simply the opponent's blueprint strategy. For the second, we biased the opponent's blueprint strategy toward folding by multiplying the probability of fold actions by 10 and then renormalizing. For the third, we biased the opponent's blueprint strategy toward checking and calling. Finally for the fourth, we biased the opponent's blueprint strategy toward betting and raising. To estimate the values of a state when the depth limit is reached on the second round, we sample rollouts of each of the stored best-response strategies.

The performance of *Modicum* is shown in Table 5.5. For the evaluation, we used AIVAT to reduce variance [32]. Our new agent defeats both Baby Tartanian8 and Slumbot with statistical significance. For comparison, Baby Tartanian8 defeated Slumbot by 36 ± 12 mbb/g, *Libratus* defeated Baby Tartanian8 by 63 ± 28 mbb/g, and *Libratus* defeated top human professionals by 147 ± 77 mbb/g.

	Baby Tartanian8	Slumbot
Blueprint (No real-time solving)	-57 ± 13	-11 ± 8
Naïve depth-limited solving	-10 ± 8	-1 ± 15
Depth-limited solving	6 ± 5	11 ± 9

Table 5.5: Head to head performance of our new agent against Baby Tartanian8 and Slumbot with 95% confidence intervals shown. Our new agent defeats both opponents with statistical significance. Naïve depth-limited solving means states are assumed to have just a single value, which is determined by the blueprint strategy.

In addition to head-to-head performance against prior top AIs, we also tested *Modicum* against two versions of **Local Best Response (LBR)** [103]. An LBR agent is given full access to its opponent's full-game strategy and uses that knowledge to exactly calculate the probability the LBR agent is in each possible state. Given that probability distribution and a heuristic for how the opposing agent will play thereafter, the LBR agent chooses a best response action. LBR is a way to calculate a lower bound on exploitability and has been shown to be effective in exploiting agents that do not use real-time solving.

In the first version of LBR we tested against, the LBR agent was limited to either folding or betting $0.75\times$ the pot on the first action, and thereafter was limited to either folding or calling. *Modicum* beat this version of LBR by 570 ± 42 mbb/g. The second version of LBR we tested against could bet 10 different amounts on the flop that *Modicum* did not include in its blueprint strategy. Much like the experiment in Section 5.2.3, this was intended to measure how vulnerable *Modicum* is to unanticipated bet sizes. The LBR agent was limited to betting $0.75\times$ the pot for the first action of the game and calling for the remaining actions on the preflop. On the flop, the LBR agent could either fold, call, or bet 0.33×2^x times the pot for $x \in \{0, 1, ..., 10\}$. On the remaining rounds the LBR agent could either fold or call. *Modicum* beat this version of LBR by 1377 ± 115 mbb/g. In contrast, similar forms of LBR have been shown to defeat prior top poker AIs that do not use real-time solving by hundreds or thousands of mbb/g [103].

While our new agent is probably not as strong as *Libratus*, it was produced with less than

0.1% of the computing resources and memory, and is never vulnerable to off-tree opponent actions.

While the rollout method used on the second betting round worked well, rollouts may be significantly more expensive in deeper games. To demonstrate the generality of our approach, we also trained a deep neural network (DNN) to predict the values of states at the end of the second betting round as an alternative to using rollouts. The DNN takes as input a 34-float vector of features describing the state, and outputs four floats representing the values of the state for the four possible opponent strategies (represented as a fraction of the size of the pot). The DNN was trained using 180 million examples per player by optimizing the Huber loss with Adam [84], which we implemented using PyTorch [120]. In order for the network to run sufficiently fast on just a 4-core CPU, the DNN has just 4 hidden layers with 256 nodes in the first hidden layer and 128 nodes in the remaining hidden layers. This achieved a Huber loss of 0.02. Using a DNN rather than rollouts resulted in the agent beating Baby Tartanian8 by 2 ± 9 mbb/g. However, the average time taken using a 4-core CPU increased from 20 seconds to 31 seconds per hand. Still, these results demonstrate the generality of our approach.

5.2.4 Conclusions

Search has been critical for achieving superhuman performance in perfect-information games and was the key breakthrough that allowed Libratus to defeat top humans in imperfect-information games. However, in order to make search truly general, it must be possible to conduct depth-limited search.

In this section we described the multi-valued states approach for depth-limited search in imperfect-information games and proved that it is theoretically sound in the limit. Experimental results show that it leads to stronger performance than the best precomputed-strategy AIs in HUNL while using orders of magnitude less computational resources, and is also orders of magnitude more efficient than past approaches that use real-time solving. Additionally, the method exhibits low exploitability.

The multi-valued states approach was ultimately used to develop Pluribus [23], which for the first time defeated top human professionals in multiplayer poker, and did so with orders of magnitude fewer computational resources compared to Libratus despite multiplayer no-limit Texas hold'em poker being orders of magnitude larger than heads-up no-limit Texas hold'em poker.

5.2.5 Proofs of Theoretical Results

Proof of Proposition 3

Proof. Consider the augmented subgame S' structured as follows. S' contains S and all its descendants. Additionally, for every root node $h \in S$ (that is, a node whose parent is not in S), S' contains a node h' belonging to P_2. If h_1 and h_2 are root nodes in S and h_1 and h_2 share an infoset, then h'_1 and h'_2 share an infoset. S' begins with an initial chance node that reaches h' with probability proportional to the probability of reaching h if P_2 tried to do so (that is, the probability of reaching it according to P_1's strategy and chance's probabilities).

At node h', P_2 has two actions. The "alt" action leads to a terminal node that awards $v_2^{\langle \sigma_1^*, BR(\sigma_1^*)\rangle}(I)$. The "enter" action leads to h. From Theorem 1 in [31], a solution to S' is part of a P_1 Nash equilibrium strategy in the full game.

Now consider the depth-limited augmented subgame S'' that is similar to S' but does not contain the descendants of S. We show that knowing $v_1^{\langle \sigma_1^*, \sigma_2\rangle}(h)$ for every pure undominated P_2 strategy σ_2 and every leaf node $h \in S$ is sufficient to calculate the portion of a P_1 Nash equilibrium strategy for S' that is in S''. That, in turn, gives a strategy in S that is a Nash equilibrium strategy in the full game.

We modify S'' so that, after P_1's strategy is chosen, P_2 chooses a probability distribution over the N pure undominated strategies where the probability of pure undominated strategy σ_2^n is represented as $p(n)$. This mixture of pure strategies defines a strategy $\sigma_2^m = \sum_{n \leq N} \left(p(n)\sigma_2^n\right)$. In this way, P_2 can pick any undominated strategy because every undominated strategy is a mixture of pure undominated strategies. Upon reaching a leaf node h, P_1 receives a reward of $\sum_{n \leq N} \left(p(n) v_1^{\langle \sigma_1^*, \sigma_2^n\rangle}(h)\right) = v_1^{\langle \sigma_1^*, \sigma_2^m\rangle}(h)$. Clearly P_1 can do no better than playing σ_1^*, because it is a Nash equilibrium and P_2 can play any undominated strategy. Thus, any strategy P_1 plays in S'', when combined with σ_1^* outside of S'', must do at least as well as playing σ_1^* in the full game. □

5.3 Depth-Limited Search and Deep Reinforcement Learning via Public Belief States

Combining reinforcement learning with search at both training and test time **(RL+Search)** has led to a number of major successes in AI in recent years. For example, the AlphaZero algorithm achieves state-of-the-art performance in the perfect-information games of Go, chess, and shogi [142]. However, prior RL+Search algorithms such as AlphaZero do not work in imperfect-information games because they make a number of assumptions that no longer hold in these settings. An example of this was presented in Figure 5.8a and Figure 5.8b in Section 5.2.1. Recent AI breakthroughs in imperfect-information games have highlighted the importance of search at test time [21, 23, 99, 110], but combining RL and search during training in imperfect-information games has been an open problem.

This section describes **ReBeL (Recursive Belief-based Learning)**, a general RL+Search algorithm that converges to a Nash equilibrium in two-player zero-sum games. At a high level, ReBeL resembles past RL+Search algorithms used in perfect-information games [1, 134, 141, 142, 151]. These algorithms train a value network through self play. During training, a search algorithm is used in which the values of leaf nodes are determined via the value function. Additionally, a policy network may be used to guide search. However, ReBeL differs from RL+Search algorithms for perfect-information games in that the notion of "state" is expanded to include the probabilistic belief distribution of all agents about what state they may be in, given the available common knowledge information and the policies of all agents. This expanded notion of state, which we refer to as a **public belief state (PBS)** originated in work on decentralized multi-agent POMDPs [43, 113, 118] and has since been used in work on imperfect-information games more broadly [46, 73, 110, 136].

ReBeL builds upon the idea of using a PBS value function, which was used in the poker AI DeepStack [110]. However, DeepStack's value function was trained not through self-play RL, but rather by generating random PBSs, including random probability distributions, and estimating their values using search. This would be like learning a value function for Go by randomly placing stones on the board. This is not an efficient way of learning a value function because the vast majority of randomly generated situations would not be relevant in actual play. DeepStack coped with this by using handcrafted features to reduce the dimensionality of the public belief state space, by sampling PBSs from a distribution based on expert domain knowledge, and by using domain-specific abstractions to circumvent the need for a value network when close to the end of the game.

Section 5.2 introduced multi-valued states, an alternative sound method for conducting depth-limited search in imperfect-information games that was used in the Pluribus poker AI to defeat elite human professionals in multiplayer poker for the first time [23, 27]. However, multi-valued states only uses search at test time and therefore requires a strong blueprint as a starting point. Section 5.4 discusses in detail the relative strengths and weaknesses of ReBeL and multi-valued states.

Our goal in developing ReBeL was not to chase state-of-the-art performance by any means necessary. Instead, our goal was to develop a simple, flexible, effective algorithm that leverages as little expert domain knowledge as possible. Experimental results show that despite its simplicity, ReBeL is effective in large-scale two-player zero-sum imperfect-information games and defeats a top human professional with statistical significance in the benchmark game of heads-up no-limit Texas hold'em poker while using far less expert domain knowledge than any previous poker AI. In perfect-information games, ReBeL simplifies to an algorithm similar to AlphaZero, with the major difference being in type of search algorithm used.

5.3.1 Notation and Background

Since ReBeL most closely resembles a reinforcement learning algorithm, we describe it using reinforcement learning notation based on factored observation games [88], which is a modification of partially observable stochastic games [63] that distinguishes between private and public observations. We consider a game with $\mathcal{N} = \{1, 2, ..., N\}$ agents. We provide theoretical and empirical results only for when $|N| = 2$, though related techniques have been shown to be successful in practice in certain settings with more agents [23].

We assume throughout this section that the rules of the game, as well as our agent's policy (including any algorithms used to generate the policy) are common knowledge. However, the outcome of stochastic algorithms (i.e., the random seeds) are not known. This is a common assumption in game theory. One argument for it is that in repeated play an adversary would eventually determine an agent's policy. Another motivation is that in a Nash equilibrium there is no incentive to change one's policy even if the other players' policies are known.

A **world state** $w \in \mathcal{W}$ is a state in the game. $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N$ is the space of joint actions. $\mathcal{A}_i(w)$ denotes the legal actions for agent i at w and $a = (a_1, a_2, ..., a_N) \in \mathcal{A}$ denotes a joint action. After a joint action a is chosen, a transition function $\mathcal{T}$ determines the next world state w' drawn from the probability distribution $\mathcal{T}(w, a) \in \Delta\mathcal{W}$. After joint action a, agent i receives a reward $\mathcal{R}_i(w, a)$.

Upon transition from world state w to w' via joint action a, agent i receives a **private observation** from a function $\mathcal{O}_{\text{priv}(i)}(w, a, w')$. Additionally, all agents receive a **public observation** from a function $\mathcal{O}_{\text{pub}}(w, a, w')$. Public observations may include observations of publicly taken actions by agents. For example, in many recreational games, including poker, all betting actions are public.

A **history** (also called a trajectory) is a finite sequence of legal actions and world states, denoted $h = (w^0, a^0, w^1, a^1, ..., w^t)$. An **infostate** (or **action-observation history (AOH)**) for agent i is a sequence of an agent's observations and actions $s_i = (O_i^0, a_i^0, O_i^1, a_i^1, ..., O_i^t)$ where

$$O_i^k = \big(\mathcal{O}_{\text{priv}(i)}(w^{k-1}, a^{k-1}, w^k), \mathcal{O}_{\text{pub}}(w^{k-1}, a^{k-1}, w^k)\big)$$

The unique infostate corresponding to a history h for agent i is denoted $s_i(h)$. The set of histories that correspond to s_i is denoted $\mathcal{H}(s_i)$.

A **public state** is a sequence $s_{\text{pub}} = (O_{\text{pub}}^0, O_{\text{pub}}^1, ..., O_{\text{pub}}^t)$ of public observations. The unique public state corresponding to a history h and an infostate s_i is denoted $s_{\text{pub}}(h)$ and $s_{\text{pub}}(s_i)$, respectively. The set of histories that match the sequence of public observation of s_{pub} is denoted $\mathcal{H}(s_{\text{pub}})$.

As an example, consider a game in which two players roll two six-sided dice each. One die of each player is publicly visible, but the other die is only observed by the player who rolled it. Suppose player 1 rolls a 3 and a 4 (with 3 being the hidden die), and player 2 rolls a 5 and a 6 (with 5 being the hidden die). The history is $h = \big((3,4),(5,6)\big)$. The set of histories corresponding to player 2's infostate is $\mathcal{H}(s_2) = \big\{\big((x,4),(5,6)\big) \mid x \in \{1,2,3,4,5,6\}\big\}$, so $|\mathcal{H}(s_2)| = 6$. The set of histories corresponding to s_{pub} is $\mathcal{H}(s_{\text{pub}}) = \big\{\big((x,4),(y,6)\big) \mid x, y \in \{1,2,3,4,5,6\}\big\}$, so $|\mathcal{H}(s_{\text{pub}})| = 36$.

Public states provide an easy way to reason about common knowledge in a game. All agents observe the same public sequence s_{pub}, and therefore it is common knowledge among all agents that the true history is some $h \in \mathcal{H}(s_{\text{pub}})$.[9]

An agent's **policy** π_i is a function mapping from an infostate to a probability distribution over actions. A **policy profile** π is a tuple $(\pi_1, \pi_2, ..., \pi_N)$. We also define a policy for a history h as $\pi_i(h) = \pi_i(s_i(h))$ and $\pi(h) = (\pi_1(s_1(h)), \pi_2(s_2(h)), ..., \pi_N(s_N(h)))$. The expected sum of future rewards (also called the **expected value (EV)**) for agent i in history h when all agents play policy profile π is denoted $v_i^\pi(h)$. The EV for the entire game is denoted $v_i(\pi)$.

5.3.2 From World States to Public Belief States

As shown in Section 5.2.1, world states and histories do not necessarily have unique values in imperfect-information games because their values depend not just on the true state of the world, but also on what each agent knows, what each agent knows about what the other agents know, etc. In other words, it depends on the **common knowledge** [2] among the agents.

In this section we describe a mechanism for converting an imperfect-information game into a continuous state space perfect-information game where the state description contains the common knowledge of all agents. In this way, techniques that have been applied to perfect-information games can also be applied to imperfect-information games (with some modifications).

[9] As explained in [88], it may be possible for agents to infer common knowledge beyond just public observations. However, doing this additional reasoning is inefficient both theoretically and practically.

For intuition, consider a game such as poker in which one of 52 cards is privately dealt to each player. The players choose actions based on their private card, and eventually receive a reward once the game ends. Now consider a modification of this game in which the players cannot see their private cards; instead, their cards are seen by a "referee". When a player acts, they announce the probability they would take each action with each possible private card. The referee then chooses an action on the player's behalf according to the announced probability distribution for the player's true private card. At the beginning of this game, players do not know their private card and assign equal probability to each card. However, after each action by the referee, players can update their belief distribution about which card they are holding via Bayes' Rule. Likewise, players can update their belief distribution about an *opponent's* private card through the same operation. Thus, the probability that each player is holding each private card is common knowledge among all players at all times in this game.

The critical insight for this section is that *these two games are strategically identical*, but the latter game contains no private information and is instead a continuous state space perfect-information game. While players do not announce their action probabilities for each possible card in the first game, we assume (as stated in Section 5.3.1) that all players' policies are common knowledge, and therefore the probability that a player would choose each action for each possible card is indeed known by all players.

Of course, at test time (i.e., when our agent actually plays against a human opponent) the opponent does not actually announce their entire policy and therefore our agent does not know the true probability distribution over opponent cards. We later address this problem in Section 5.3.4.

Going even further, consider a similar game but in which players have no single private card. Instead, players have a weight (or, equivalently, a probability) $p(s)$ for each private card s such that the weights of all cards sum to 1. When a player acts, the environment (i.e., the referee) chooses an action a with probability $\sum_s \pi(a|s)p(s)$, where $\pi(a|s)$ is the player's announced probability for choosing action a with private card s. After an action is chosen, the weight for each card is updated according to Bayes' Rule. Again, this game is strategically identical to the two already described.

This intuition can be formalized by defining a **public belief state** (PBS) PBS β as a common-knowledge joint probability distribution over the agents' possible infostates.[10] Formally, let $S_i(s_{\text{pub}})$ be the set of infostates that player i may be in given a public state s_{pub}. Then PBS $\beta = (\Delta S_1(s_{\text{pub}}), ..., \Delta S_N(s_{\text{pub}}))$ where $\Delta S_1(s_{\text{pub}})$ denotes a probability distribution over the elements of $S_1(s_{\text{pub}})$.[11] In perfect-information games, PBSs reduce to histories, which in two-player zero-sum games effectively reduce to world states.

We can generalize the notion of "state value" to imperfect-information games by defining a **belief subgame** (which we refer to simply as a subgame for the rest of this paper) to be rooted

[10]One could alternatively define a PBS as a probability distribution over histories in $\mathcal{H}(s_{\text{pub}})$ for some public state s_{pub}. However, any PBS that can arise in play (i.e., that can arise from the agents playing some policy profile π) can always be described by a joint probability distribution over the agents' possible infostates [118, 135], so we use this latter definition for simplicity.

[11]Frequently, a PBS can be compactly summarized by discarding parts of the history that are no longer relevant. For example, in poker we do not need to track the entire history of actions, but just the amount of money each player has in the pot, the public board cards, and whether there were any bets in the current round.

at a PBS.[12] Just as in perfect-information subgames, the optimal agent policies in a subgame rooted at a PBS do not depend on anything (policies, observations, etc.) that came before the PBS. Thus, a two-player zero-sum subgame rooted at a PBS β has a unique value $V_1(\beta) = \sum_{h \in \mathcal{H}(s_{\text{pub}}(\beta))} \left(p(h) v_1^{\pi^*}(h)\right)$, where $s_{\text{pub}}(\beta)$ is the public state corresponding to β and π^* is a Nash equilibrium in the subgame. Since the game is zero-sum, $V_1(\beta) = -V_2(\beta)$. Just as one can compute an optimal policy in perfect-information games via depth-limited search by learning a value function for world states, we show in this paper that one can compute an optimal policy in imperfect-information games via search by learning a value function $V_1 : \mathcal{B} \rightarrow \mathbb{R}$, where $\mathcal{B}$ is the continuous state space of PBSs.

However, existing depth-limited search algorithms operate on values similar to the *gradient* of V_1 rather than on V_1 itself. We therefore learn these gradient-like values directly rather than learning V_1.

Specifically, existing depth-limited search algorithms for imperfect-information games require the EVs of *infostates* for PBSs [31, 110]. The EV of infostate s_i in β assuming π^* is played is

$$v_i^{\pi^*}(s_i|\beta) = \sum_{h \in \mathcal{H}(s_i)} p(h|s_i, \beta) v_i^{\pi^*}(h)$$

Theorem 23 proves that infostate EVs for β under some Nash equilibrium π^* can be derived from V_i.

Theorem 23. *Let $\tilde{V}_i$ be the extension of V_i to unnormalized belief distributions. For any (normalized) belief β and any subgradient $-\bar{g}_i$ of $-\tilde{V}_i(\beta)$ with respect to β, $V_i(\beta) + \bar{g}_i \cdot \hat{s}_i = v_i^{\pi^*}(s_i|\beta)$ for some Nash equilibrium policy π^*, where $\hat{s}_i$ is the unit vector in direction s_i.*

Therefore, we learn an infostate-value function $\hat{v} : \mathcal{B} \rightarrow \mathbb{R}^{|S_1|+|S_2|}$ that directly approximates for each s_i the average of the sampled $v_i^{\pi^*}(s_i|\beta)$ values produced by our RL+Search procedure at β.

Unlike the PBS value $V_1(\beta)$, the infostate values may not be unique and depend on which Nash equilibrium is played in the subgame. Each execution k of our RL+Search algorithm may converge to a different $\pi^{*,k}$ in β, so the samples of $v_i^{\pi^{*,k}}(\beta)$ may not be identical. Nevertheless, Theorem 24 states that the average of valid samples of $v_i^{\pi_k^*}(\beta)$ corresponds to $v_i^{\pi^*}(\beta)$ for some other equilibrium policy π^*. Therefore $\hat{v}$ should approximate $v_i^{\pi^*}$ for some equilibrium policy.

Theorem 24. *Let X be the vector of infostate EVs in PBS β corresponding to minimax policy profile $\pi^{*,X}$, and let Y be the vector of infostate EVs in β corresponding to minimax policy profile $\pi^{*,Y}$. Then $\lambda X + (1-\lambda)Y$ is the vector of infostate EVs in β corresponding to minimax policy profile $\lambda\pi^{*,X} + (1-\lambda)\pi^{*,Y}$ for $0 \leq \lambda \leq 1$.*

Given that agents play according to policy profile π, the PBS that arises at public state s_{pub}, the infostate s_i, and the history h is denoted $\beta^\pi_{s_{\text{pub}}}$, $\beta^\pi_{s_i}$, and β^π_h, respectively.

A **depth-limited belief subgame** (which we refer to simply as a depth-limited subgame) is a belief subgame that extends only for some limited number of actions into the future. In this paper, search is performed over a fixed-size depth-limited subgame (as opposed to Monte Carlo Tree Search, which grows the subgame as more search iterations are performed [53]), and we assume

[12] Past work defines a subgame to be rooted at a public state [16, 20, 31, 87, 88, 109, 110, 135, 147]. However, imperfect-information subgames rooted at a public state do not have well-defined values.

that all histories sharing a public state are either all in the subgame or all not in the subgame. A history z that has children in the full game but does not have children in the subgame is a **leaf node**, and agent i receives a reward of $\hat{v}(s_i(z)|\beta_z^\pi)$ at such a history, where π is the policy profile in the subgame. This means that the value of a leaf node is conditional on the beliefs at that leaf node, which in turn are conditional on the policy in the subgame.

5.3.3 Self Play Reinforcement Learning and Search for Public Belief States

At a high level, ReBeL, shown in Algorithm 4, is similar to RL+Search algorithms used for perfect-information games, but operating on PBSs rather than world states. At the start of the game, a depth-limited subgame rooted at the initial PBS β_r is generated. This subgame is solved (i.e., a Nash equilibrium is approximated) by running T iterations of an iterative equilibrium-finding algorithm and using the learned value network $\hat{v}$ to approximate leaf values on every iteration. The infostate values at β_r are added as training examples for $\hat{v}$ and (optionally) the policies in the subgame are added as training examples for the policy network. Finally, a leaf node z is sampled and the process repeats with the PBS at z being the new subgame root. Detailed pseudocode is provided in Section 5.3.7.

Algorithm 4 ReBeL: RL and Search for Imperfect-Information Games

function SELFPLAY($\beta_r, \theta^v, \theta^\pi, D^v, D^\pi$) ▷ β_r is the current PBS
  while !ISTERMINAL(β_r) **do**
    $G \leftarrow$ CONSTRUCTSUBGAME(β_r)
    $\bar{\pi}, \pi^{t_{\text{warm}}} \leftarrow$ INITIALIZEPOLICY(G, θ^π) ▷ $t_{\text{warm}} = 0$ and π^0 is uniform if no warm start
    $G \leftarrow$ SETLEAFVALUES($G, \bar{\pi}, \pi^{t_{\text{warm}}}, \theta^v$)
    $v(\beta_r) \leftarrow$ COMPUTEEV($G, \pi^{t_{\text{warm}}}$)
    $t_{sample} \sim \text{unif}\{t_{\text{warm}} + 1, T\}$ ▷ Sample an iteration
    for $t = (t_{\text{warm}} + 1)..T$ **do**
      $\pi^t \leftarrow$ UPDATEPOLICY(G, π^{t-1})
      $\bar{\pi} \leftarrow \frac{t-1}{t}\bar{\pi} + \frac{1}{t}\pi^t$
      $G \leftarrow$ SETLEAFVALUES($G, \bar{\pi}, \pi^t, \theta^v$)
      $v(\beta_r) \leftarrow \frac{t-1}{t}v(\beta_r) + \frac{1}{t}$ COMPUTEEV(G, π^t)
      if $t = t_{sample}$ **then**
        $\beta'_r \leftarrow$ SAMPLELEAF(G, π^t) ▷ Sample a leaf PBS according to the new policies
    Add $\{\beta_r, v(\beta_r)\}$ to D^v ▷ Add to value net training data
    for $\beta \in G$ **do** ▷ Loop over the PBS at every public state in G
      Add $\{\beta, \bar{\pi}(\beta)\}$ to D^π ▷ Add to policy net training data (optional)
    $\beta_r \leftarrow \beta'_r$

Search in a depth-limited imperfect-information subgame

There exist a number of iterative algorithms for solving imperfect-information games [13, 72, 92, 93, 163]. Our framework is flexible with respect to the choice of a search algorithm.

We assume that the search algorithm used is an iterative self-play algorithm. On each iteration t, the algorithm determines a policy profile π^t. Next, the value of every leaf node z is set according to $\hat{v}(s_i(z)|\beta_z^{\bar{\pi}^t})$ or $\hat{v}(s_i(z)|\beta_z^{\pi^t})$, depending on the algorithm, where $\bar{\pi}^t$ denotes the average policy profile over iterations 1 to t. Given π^t and the leaf node values, each infostate in β_r has a well-defined value. This vector of values, denoted $v^{\pi^t}(\beta_r)$, is computed and stored. Next, the algorithm chooses a new policy profile π^{t+1}, and the process repeats for T iterations. For many algorithms, including CFR, the *average* policy profile $\bar{\pi}^T$ converges to a Nash equilibrium as $T \to \infty$.

After solving a subgame rooted at PBS β_r with an iterative algorithm that has run for T iterations, the value vector $(\sum_{t=1}^{T} v^{\pi^t}(\beta_r))/T$ is added to the training data for $\hat{v}(\beta_r)$.[13]

Prior work on search in imperfect-information games has used the **CFR Decomposition (CFR-D)** algorithm [31, 110]. Section 5.3.9 describes **CFR-AVG**, a modification of CFR-D that sets the value of a leaf node z based on $\bar{\pi}^t$ rather than π^t, which addresses some weaknesses of CFR-D. Section 5.3.5 also shows experimental results for **fictitious play (FP)** [13].

Self-play reinforcement learning

Algorithm 4 learns values for PBSs through self play. After solving a subgame rooted at PBS β_r, the value vector for the root infostates is added to the training dataset for $\hat{v}$. Next, a leaf PBS β'_r is sampled and a new subgame rooted at β'_r is solved. This process repeats until the game ends.

Since the subgames are solved using an iterative algorithm, we want $\hat{v}$ to be accurate for leaf PBSs on every iteration. Therefore, a leaf node z is sampled according to π^t on a random iteration $t \sim \text{unif}\{0, T-1\}$, where T is the number of iterations of the search algorithm.[14] To ensure sufficient exploration, one agent samples random actions with probabilility $\epsilon > 0$.[15] In CFR-D $\beta'_r = \beta_z^{\pi^t}$, while in CFR-AVG and FP $\beta'_r = \beta_z^{\bar{\pi}^t}$.

Eventually, a subgame rooted at β_r^* is reached near the end of the game that does not contain leaf nodes (i.e., the subgame is not depth-limited). $\hat{v}$ will therefore learn correct values $v^{\pi^*}(s_i|\beta_r^*)$ for every root infostate s_i and for some Nash equilibrium π^* (except for an error term that disappears as $T \to \infty$). In the future, when β_r^* is a leaf PBS of a different subgame, it will be possible to more accurately compute the value of that subgame. In this way, accurate PBS values will "bubble up" the game tree and $\hat{v}$ will increase in accuracy over time.

Theorem 25 states that, with perfect function approximation, running Algorithm 4 will produce a value network whose error is bounded by $\mathcal{O}(\frac{1}{\sqrt{T}})$ after a finite amount of time for any PBS that could be encountered during play, where T is the number of CFR iterations being run in subgames.

Theorem 25. *Consider an idealized value approximator that returns the most recent sample of the value for sampled PBSs, and 0 otherwise. Running Algorithm 4 with T iterations of CFR in each subgame will, after a finite amount of time, produce a value approximator that has error of at most $\frac{C}{\sqrt{T}}$ for any PBS that could be encountered during play, where C is a game-dependent constant.*

[13] For some algorithms, including CFR-AVG and FP, an alternative is to add the value vector $v^{\bar{\pi}^T}(\beta_r)$.

[14] For FP, we pick a random agent i and sample according to $(\pi_i^t, \bar{\pi}_{-i}^t)$ to reflect the search operation.

[15] The algorithm is still correct if all agents sample random actions with probability ϵ, but that is less efficient because the value of a leaf node that can only be reached if both agents go off policy is irrelevant.

Adding a policy network

Algorithm 4 will result in $\hat{v}$ converging correctly even if a policy network is not used. However, if a policy network is not used then in the first several iterations t of running a search algorithm in the subgames, $\bar{\pi}^t$ may be very far from an equilibrium. Since, in some search algorithms, $\bar{\pi}^t$ determines the PBSs at the leaf nodes of a subgame, not using a policy network means the learned value network must be accurate over a wide domain of PBSs. Additionally, adding an accurate policy network will reduce the number of search iterations necessary to closely approximate a Nash equilibrium.

Algorithm 4 can train a policy network $\hat{\Pi} : \beta \rightarrow (\Delta\mathcal{A})^{|S_1|+|S_2|}$ by adding $\bar{\pi}^T(\beta)$ for each PBS β in the subgame to a training dataset each time a subgame is solved (i.e., T iterations of CFR have been run in the subgame).

In our experiments, we use a simplified form of the warm start technique described in Section 3.2 to warm start CFR from the trained policy network. The simplifications we make are that we use an exact best response rather than a weakened best response, and we always warm start to 15 CFR iterations.

Algorithm behavior in perfect-information games

Perfect-information games can be viewed as a special case of imperfect-information games in which public states are equivalent to histories, and therefore have the same value as world states. Since the value of a leaf node in a perfect-information subgame does not depend on the policy in the subgame, only one search iteration is required to solve a subgame.

Thus, in perfect-information games Algorithm 4 reduces to an algorithm similar to AlphaZero. The major differences are that AlphaZero plays just a single action before solving a new subgame while ReBeL plays the subgame policy until reaching a leaf node, and AlphaZero grows the size of the subgame during search, and AlphaZero trains on the final reward received at the end of the game.

5.3.4 Safe Search with Public Belief States

As discussed in subsection 5.3.2, we assume during training that the players' entire policies are common knowledge. This, in turn, means that the current public belief state is always known by all players. In self-play training, all players' policies are indeed common knowledge. However, at test time (such as when playing against an actual human) the opponent obviously does not make their entire policy public. What, then, should ReBeL assume the opponent's policy is?

This question was examined in detail in Section 5.1. As explained in that section, **unsafe** search, in which we assume the opponent plays a specific Nash equilibrium policy, may be highly exploitable because the opponent might take advantage of our assumption and simply play a different policy. Section 5.1 described several prior and new **safe** search techniques which are not as exploitable.

However, using those safe search techniques requires already having access to a trained value network $\hat{v}$, which makes them difficult to use during self-play training, and using safe search at test time when it was not used during training might lead to novel PBSs for which the value

network might not be accurate. Moreover, safe search tends to do worse than unsafe search in terms of head-to-head performance, and for this reason every past competitive agent used unsafe search either partially or entirely. [21, 23, 27, 110, 136].

We now introduce a new safe search technique that can be used when conducting depth-limited search using a PBS value network. Unlike all previous safe search techniques, this new technique does not any additional constraints. Instead, it simply runs the exact same ReBeL algorithm at test time that was used during self-play training. Specifically, when conducting search at test time it picks a *random* iteration and assumes all agents play according to that iteration's policy profile (or, in the case of CFR-AVG, the average policy profile up to that iteration) for the entire subgame. This leads to a PBS leaf node, which defines a new subgame, and the process repeats. This is very similar to unsafe search, except unsafe search always uses the average policy profile of the *final* iteration. By choosing a random iteration, the opponent does not know what belief distribution we are assuming and therefore is unable to exploit our assumption.

This leads to a counter-intuitive result in which the policy output by our search algorithm may be extremely pure and highly exploitable, and yet the *algorithm itself* is not exploitable. For example, in the modified Rock-Paper-Scissors subgame shown in Figure 5.8a, our algorithm may output a policy of 100% Rock for player 2. However, the algorithm is still unexploitable so long as the probability it outputs that policy is 40% because the opponent would not know when our policy is to play Rock with 100% probability. Thus, there is a distinction between the policy our algorithm samples for a specific "episode" of play, versus the "true" policy which is played by the algorithm in expectation.

Theorem 26, the proof of which is in Section 5.3.11, states that this new safe search algorithm approximates a Nash equilibrium. Specifically, Theorem 26 states that once a value network is trained according to Theorem 25, using Algorithm 4 at test time (without off-policy exploration) will approximate a Nash equilibrium.

Theorem 26. *If Algorithm 4 is run at test time with no off-policy exploration, a value network with error at most δ for any leaf PBS that was trained to convergence as described in Theorem 25, and with T iterations of CFR being used to solve subgames, then the algorithm plays a $(\delta C_1 + \frac{\delta C_2}{\sqrt{T}})$-Nash equilibrium, where C_1, C_2 are game-specific constants.*

In other words, the same algorithm we describe for training also approximates a Nash equilibrium at test time. This result applies regardless of how the value network was trained and therefore can be applied to prior algorithms that use PBS value functions [110, 136].

Since a random iteration is selected, there is a risk that we may select a very early iteration, or even the first iteration, in which the policy is extremely poor. This can be mitigated by using modern equilibrium-finding algorithms, such as Linear CFR or Discounted CFR [22], that assign little or no weight to the early iterations that are played while still converging to a Nash equilibrium. Theorem 26 still holds so long as the equilibrium-finding algorithm that is used converges to a Nash equilibrium.

5.3.5 Experiments

We measure **exploitability** of a policy π^*, which is $\sum_{i \in \mathcal{N}} \max_\pi v_i(\pi, \pi^*_{-i})/|\mathcal{N}|$. All CFR experiments use alternating-updates Linear CFR [22]. All FP experiments use alternating-updates

Linear Optimistic FP, which is a novel variant described in Section 5.3.8.

We evaluate on the benchmark imperfect-information games of heads-up no-limit Texas hold'em poker (HUNL), which is described in Section 2.4.3, and Liar's Dice, which is described in Section 2.4.6. We also evaluate our techniques on turn endgame hold'em (TEH), a variant of no-limit Texas hold'em in which both players automatically check/call for the first two of the four betting rounds in the game.

In HUNL and TEH, we reduce the action space to at most nine actions using domain knowledge of typical bet sizes. However, our agent responds to any "off-tree" action at test time by adding the action to the subgame [23, 27]. The bet sizes and stack sizes are randomized during training. For TEH we train on the full game and measure exploitability on the case of both players having $20,000, unperturbed bet sizes, and the first four board cards being 3♠7♡T♢K♠.

For HUNL, our agent uses far less domain knowledge than any prior competitive AI agent. Additionally, our AI agent is trained on all stack sizes between 5,000 and 25,000 chips, rather than just the standard 20,000. Section 6.7 discusses the poker domain knowledge leveraged in ReBeL.

We approximate the value and policy functions using artificial neural networks. Both networks are multilayer perceptrons with Gaussian Error Linear Unit [71] activation functions and LayerNorm [3]. Both networks are trained with Adam [84]. We use pointwise Huber loss as the criterion for the value function and mean squared error (MSE) over probabilities for the policy. In preliminary experiments we found MSE for the value network and cross entropy for the policy network did worse. See Section 5.3.10 for the hyperparameters.

We use PyTorch [120] to train the networks. We found data generation to be the bottleneck due to the sequential nature of the FP and CFR algorithms and the evaluation of all leaf nodes on each iteration. For this reason we use a single machine for training and up to 128 machines with 8 GPUs each for asynchronous data generation. Figure 5.9 shows ReBeL, using a learned value network, reaches a level of exploitability in TEH equivalent to running about 125 iterations of full-game tabular linear CFR. For context, top poker agents typically use between 100 and 1,000 tabular CFR iterations [12, 21, 23, 27, 110].

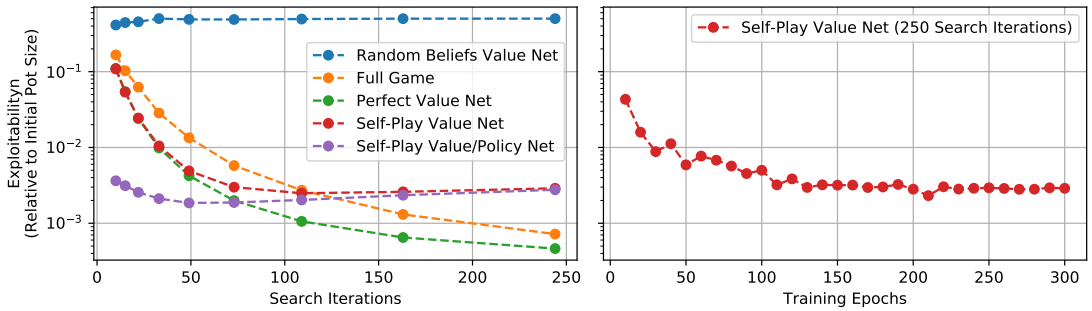


Figure 5.9: Convergence of different techniques in TEH. All subgames are solved using CFR-AVG. Perfect Value Net uses an oracle function to return the exact value of leaf nodes on each iteration. Self-Play Value Net uses a value function trained through self play. Self-Play Value/Policy Net additionally uses a policy network to warm start CFR. Random Beliefs trains the value net on random PBSs.

Table 5.6 shows results for ReBeL in HUNL. We compare ReBeL to BabyTartanian8 [17]

and Slumbot, prior champions of the Computer Poker Competition, and the local best response (LBR) [103] algorithm. We also present results against Dong Kim, a top human HUNL expert that did best among the four top humans that played against Libratus. Kim played 7,500 hands. Variance was reduced by using AIVAT [32]. ReBeL played faster than 2 seconds per hand and never needed more than 5 seconds for a decision.

Bot Name	**Slumbot**	**BabyTartanian8**	**LBR**	**Top Humans**
DeepStack	-	-	383 ± 112	-
Libratus	-	63 ± 14	-	147 ± 39
Modicum	11 ± 5	6 ± 3	-	-
ReBeL	45 ± 5	9 ± 4	881 ± 94	165 ± 69

Table 5.6: Head-to-head results of ReBeL versus BabyTartanian8 [17] and Slumbot, as well as top human expert Dong Kim, measured in thousandths of a big blind per game. We also show performance against LBR [103] where the LBR agent must call for the first two betting rounds, and can either fold, call, bet $1\times$ pot, or bet all-in on the last two rounds. The $\pm$ shows one standard deviation. For Libratus, we list the aggregate score against all top humans; Libratus beat Dong Kim by 29 with an estimated $\pm$ of 78.

Table 5.7 shows ReBeL also converges to an approximate Nash in several versions of Liar's Dice. Of course, tabular CFR does better than ReBeL when using the same number of CFR iterations, but tabular CFR quickly becomes intractable to run as the game grows in size.

Algorithm	**1x4f**	**1x5f**	**1x6f**	**2x3f**
Full-game FP	0.012	0.024	0.039	0.057
Full-game CFR	0.001	0.001	0.002	0.002
ReBeL FP	0.041	0.020	0.040	0.020
ReBeL CFR-D	0.017	0.015	0.024	0.017

Table 5.7: Exploitability of different algorithms of 4 variants of Liar's Dice: 1 die with 4, 5, or 6 faces and 2 dice with 3 faces. The top two rows represent baseline numbers when a tabular version of the algorithms is run on the entire game for 1,024 iterations. The bottom 2 lines show the performance of ReBeL operating on subgames of depth 2 with 1,024 search iterations. For exploitability computation of the bottom two rows, we averaged the policies of 1,024 playthroughs and thus the numbers are upper bounds on exploitability.

5.3.6 Conclusions

We describe ReBeL, an algorithm that generalizes the paradigm of self-play reinforcement learning and search to imperfect-information games. We prove that ReBeL computes to an approximate Nash equilibrium in two-player zero-sum games and demonstrate that it produces superhuman performance in the benchmark game of heads-up no-limit Texas hold'em.

ReBeL has some limitations that present avenues for future research. Most prominently, the input to its value and policy functions currently grows linearly with the number of infostates in a public state. This is intractable in games such as Recon Chess [116] that have strategic depth but very little common knowledge. ReBeL's theoretical guarantees are also limited only to two-player zero-sum games.

Nevertheless, ReBeL achieves low exploitability in benchmark games and superhuman performance in heads-up no-limit Texas hold'em while leveraging far less expert knowledge than any prior bot. We view this as a major step toward developing universal techniques for multi-agent interactions.

5.3.7 Pseudocode for ReBeL

Algorithm 5 presents ReBeL in more detail.

We define the average of two policies to be the policy that is, in expectation, identical to picking one of the two policies and playing that policy for the entire game. Formally, if $\pi = \alpha\pi_1 + (1-\alpha)\pi_2$, then $\pi(s_i) = \frac{\left(x_i^{\pi_1}(s_i)\alpha\right)\pi_1(s_i) + \left(x_i^{\pi_2}(s_i)(1-\alpha)\right)\pi_2(s_i)}{x_i^{\pi_1}(s_i)\alpha + x_i^{\pi_2}(s_i)(1-\alpha)}$ where $x_i^{\pi_1}(s_i)$ is the product of the probabilities for all agent i actions leading to s_i. Formally, $x_i^{\pi}(s_i)$ of infostate $s_i = (O_i^0, a_i^0, O_i^1, a_i^1, ..., O_i^t)$ is $x_i^{\pi}(s_i) = \Pi_t(a_i^t)$.

5.3.8 Fictitious Linear Optimistic Play

Fictitious Play (FP) [13] is an extremely simple iterative algorithm that is proven to converge to a Nash equilibrium in two-player zero-sum games. However, in practice it does so at an extremely slow rate. On the first iteration, all agents choose a uniform policy π_i^0 and the average policy $\bar{\pi}_i^0$ is set identically. On each subsequent iteration t, agents compute a best response to the other agents' average policy $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \bar{\pi}_{-i}^{t-1})$ and update their average policies to be $\bar{\pi}_i^t = \frac{t-1}{t}\bar{\pi}_i^{t-1} + \frac{1}{t}\pi_i^t$. As $t \to \infty$, $\bar{\pi}^t$ converges to a Nash equilibrium in two-player zero-sum games.

It has also been proven that a family of algorithms similar to FP known as **generalized weakened fictitious play (GWFP)** also converge to a Nash equilibrium so long as they satisfy certain properties [100, 154], mostly notably that in the limit the policies on each iteration converge to best responses.

In this section we introduce a novel variant of FP we call **Fictitious Linear Optimistic Play (FLOP)** which is a form of GWFP. FLOP is inspired by related variants in CFR, in particular Linear CFR [22]. FLOP converges to a Nash equilibrium much faster than FP while still being an extremely simple algorithm. However, variants of CFR such as Linear CFR and Discounted CFR [22] still converge much faster in most large-scale games.

In FLOP, the initial policy π_i^0 is uniform. On each subsequent iteration t, agents compute a best response to an *optimistic* [40, 122, 148] form of the opponent's average policy in which π_{-i}^{t-1} is given extra weight: $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \frac{t}{t+2}\bar{\pi}_{-i}^{t-1} + \frac{2}{t+2}\pi_{-i}^{t-1})$. The average policy is updated to be $\bar{\pi}_i^t = \frac{t-1}{t+1}\bar{\pi}_i^{t-1} + \frac{2}{t+1}\pi_i^t$. Theorem 27 proves that FLOP is a form of GWFP and therefore converges to a Nash equilibrium as $t \to \infty$.

Theorem 27. *FLOP is a form of Generalized Weakened Fictitious Play.*

Algorithm 5 ReBeL

function REBEL-LINEAR-CFR-D($\beta_r, \theta^v, \theta^\pi, D^v, D^\pi$) ▷ β_r is the current PBS
  while !ISTERMINAL(β_r) **do**
    $G \leftarrow$ CONSTRUCTSUBGAME(β_r)
    $\bar{\pi}, \pi^{t_{\text{warm}}} \leftarrow$ INITIALIZEPOLICY(G, θ^π) ▷ $t_{\text{warm}} = 0$ and π^0 is uniform if no warm start
    $G \leftarrow$ SETLEAFVALUES($\beta_r, \pi^{t_{\text{warm}}}, \theta^v$)
    $v(\beta_r) \leftarrow$ COMPUTEEV($G, \pi^{t_{\text{warm}}}$)
    $t_{sample} \sim \text{linear}\{t_{\text{warm}} + 1, T\}$ ▷ Probability of sampling iteration t is proportional to t
    for $t = (t_{\text{warm}} + 1)..T$ **do**
      $\pi^t \leftarrow$ UPDATEPOLICY(G, π^{t-1})
      $\bar{\pi} \leftarrow \frac{t-1}{t+1}\bar{\pi} + \frac{2}{t+1}\pi^t$
      $G \leftarrow$ SETLEAFVALUES(β_r, π^t, θ^v)
      $v(\beta_r) \leftarrow \frac{t-1}{t+1}v(\beta_r) + \frac{2}{t+1}$ COMPUTEEV(G, π^t)
      if $t = t_{sample}$ **then**
        $\beta'_r \leftarrow$ SAMPLELEAF(G, π^t) ▷ Sample a leaf PBS according to the new policies
    Add $\{\beta_r, v(\beta_r)\}$ to D^v ▷ Add to value net training data
    for $\beta \in G$ **do** ▷ Loop over the PBS at every public state in G
      Add $\{\beta, \bar{\pi}(\beta)\}$ to D^π ▷ Add to policy net training data (optional)
    $\beta_r \leftarrow \beta'_r$

function SETLEAFVALUES(β, π, θ^v)
  if ISLEAF(β) **then**
    for $s_i \in \beta$ **do** ▷ For each infostate s_i corresponding to β
      $v(s_i) = \hat{v}(s_i|\beta, \theta^v)$
  else
    for $a \in \mathcal{A}(\beta)$ **do**
      SETLEAFVALUES($\mathcal{T}(\beta, \pi, a), \pi, \theta^v$)

function SAMPLELEAF(G, π)
  $i^* \sim \text{unif}\{1, N\}$, $h \sim \beta_r$ ▷ Sample a history randomly from the root PBS and a random player
  while !ISLEAF(h) **do**
    $c \sim \text{unif}[0, 1]$
    for $i = 1..N$ **do**
      if $i == i^*$ and $c < \epsilon$ **then** ▷ we set $\epsilon = 0.25$ during training, $\epsilon = 0$ at test time
        sample an action a_i uniform random
      else
        sample an action a_i according to $\pi_i(s_i(h))$
    $h \sim \tau(h, a)$
  return β_h ▷ Return the PBS corresponding to leaf node h

Proof. Assume that the range of payoffs in the game is M. Since $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \frac{t}{t+2}\bar{\pi}_{-i}^{t-1} + \frac{2}{t+2}\pi_{-i}^{t-1})$, so π_i^t is an ϵ_t-best response to $\bar{\pi}_{-i}^{t-1}$ where $\epsilon_t < M\frac{2}{t+2}$ and $\epsilon_t \to 0$ as $t \to \infty$. Thus, FLOP is a form of GWFP with $\alpha_t = \frac{2}{t}$. □

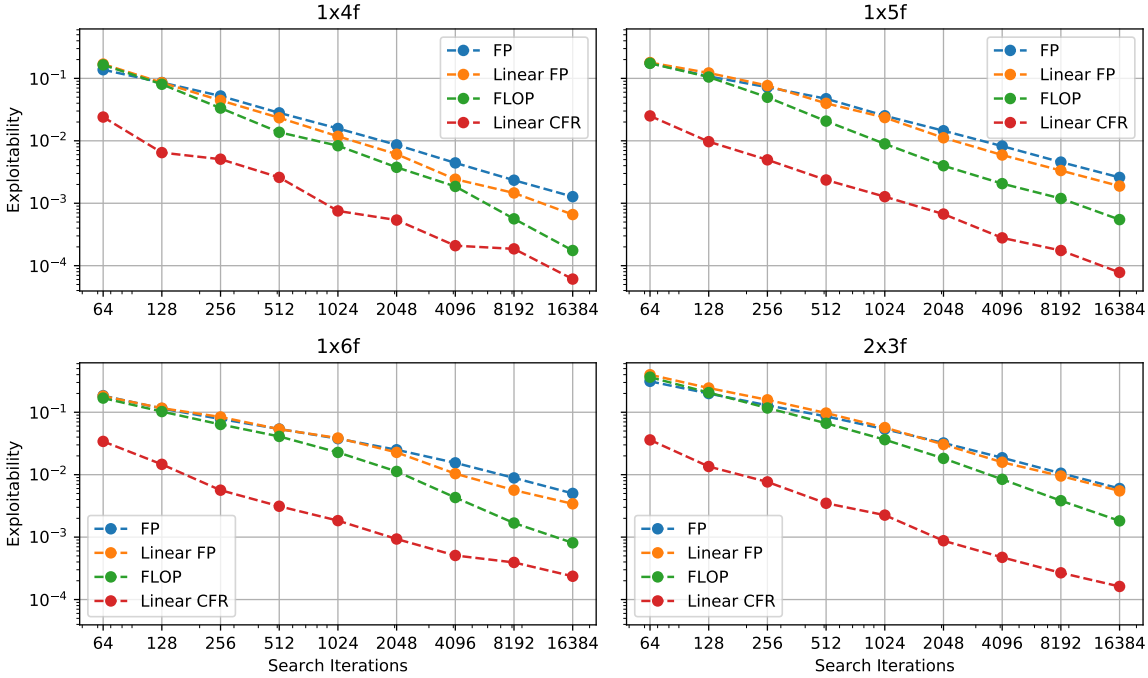


Figure 5.10: Exploitability of different algorithms of 4 variants of Liar's Dice: 1 die with 4, 5, or 6 faces and 2 dice with 3 faces. For all games FLOP outperforms Linear FP, but does not match the quality of Linear CFR.

5.3.9 CFR-AVG: CFR Decomposition using Average Strategy

On each iteration t of CFR-D, the value of every leaf node z is set to $\hat{v}(s_i(z)|\beta_z^{\pi^t})$. Other than changing the values of leaf nodes every iteration, CFR-D is otherwise identical to CFR. If T iterations of CFR-D are conducted with a value network that has error at most δ for each infostate value, then $\bar{\pi}^T$ has exploitability of at most $k_1\delta + k_2/\sqrt{T}$ where k_1 and k_2 are game-specific constants [110].

Since it is the *average* policy profile $\bar{\pi}^t$, not π^t, that converges to a Nash equilibrium as $t \to \infty$, and since the leaf PBSs are set based on π^t, the input to the value network $\hat{v}$ may span the entire domain of inputs even as $t \to \infty$. For example, suppose in a Nash equilibrium π^* the probability distribution at $\beta_z^{\pi^*}$ was uniform. Then the probability distribution at $\beta_z^{\pi^t}$ for any individual iteration t could be *anything*, because regardless of what the probability distribution is, the average over all iterations could still be uniform in the end. Thus, $\hat{v}$ may need to be accurate over the entire domain of inputs rather than just the subspace near $\beta_z^{\pi^*}$.

In **CFR-AVG**, leaf values are instead set according to the *average policy* $\bar{\pi}^t$ on iteration t. When a leaf PBS is sampled, the leaf node is sampled with probability determined by π^t, but the PBS itself is defined using $\bar{\pi}^t$.

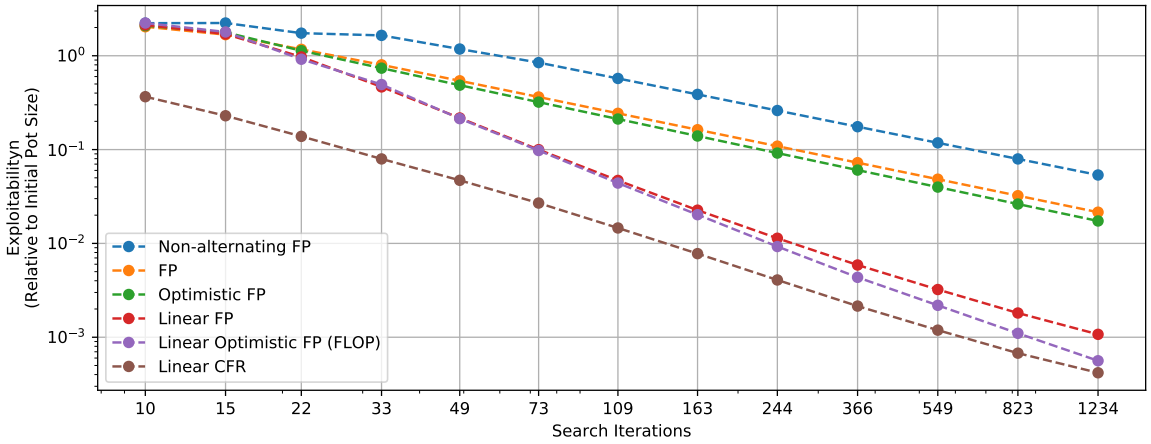


Figure 5.11: Exploitability of different algorithms for Turn Endgame HoldâĂŹem.

We first describe the tabular form of CFR-D [31]. Consider a game G' and a depth-limited subgame G, where both G' and G share a root but G extends only a limited number of actions into the future. Suppose that T iterations of a modified form of CFR are conducted in G'. On each iteration $t \leq T$, the policy $\pi(s_i)$ is set according to CFR for each $s_i \in G$. However, for every infostate $s'_i \in G' \setminus G$, the policy is set differently than what CFR would call for. At each leaf public state s'_{pub} of G, we solve a subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$ by running T' iterations of CFR. For each s'_i in the subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$, we set $\pi^t(s'_i) = \bar{\pi}^T(s'_i)$ (where $\pi^t(s'_i)$ is the policy for the infostate in G and $\bar{\pi}^T(s'_i)$ is the policy for the infostate in the subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$). It is proven that as $T' \to \infty$, CFR-D converges to a $O(\frac{1}{\sqrt{T}})$-Nash equilibrium [31].

CFR-AVG is identical to CFR-D, except the subgames that are solved on each iteration t are rooted at $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ rather than $\beta^{\pi^t}_{s'_{\text{pub}}}$. Theorem 28 proves that CFR-AVG achieves the same bound on convergence to a Nash equilibrium as CFR-D.

Theorem 28. *Suppose that T iterations of CFR-AVG are run in a depth-limited subgame, where on each iteration $t \leq T$ the subgame rooted at each leaf PBS $\beta^{\bar{\pi}^t}_{s'_{pub}}$ is solved completely. Then $\bar{\pi}^T$ is a $\frac{C}{\sqrt{T}}$-Nash equilibrium for a game-specific constant C.*

CFR-AVG has a number of potential benefits over CFR-D:

- Since $\bar{\pi}^t$ converges to a Nash equilibrium as $t \to \infty$, CFR-AVG allows $\hat{v}$ to focus on being accurate over a more narrow subspace of inputs.
- When combined with a policy network (as introduced in Section 5.3.3), CFR-AVG may allow $\hat{v}$ to focus on an even more narrow subspace of inputs.
- Since $\bar{\pi}^{t+1}$ is much closer to $\bar{\pi}^t$ than π^{t+1} is to π^t, in practice as t becomes large one can avoid querying the value network on every iteration and instead recycle the values from a previous iteration. This may be particularly valuable for Monte Carlo versions of CFR.

While CFR-AVG is theoretically sound, we modify its implementation in our experiments to make it more efficient in a way that has not been proven to be theoretically sound. The reason for this is that while the *input* to the value network is $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ (i.e., the leaf PBS corresponding to $\bar{\pi}^t$ being played in G, the *output* needs to be the value of each infostate s_i given that π^t is played in

G. Thus, unlike CFR-D and FP, in CFR-AVG there is a mismatch between the input policy and the output policy.

One way to cope with this is to have the input consist of both $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ and $\beta^{\pi^t}_{s'_{\text{pub}}}$. However, we found this performed relatively poorly in preliminary experiments when trained through self play.

Instead, on iteration $t-1$ we store the output from $\hat{v}(s_i|\beta^{\bar{\pi}^{t-1}}_{s'_{\text{pub}}})$ for each s_i and on iteration t we set $v^t(s_i)$ to be $t\hat{v}(s_i|\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}) - (t-1)\hat{v}(s_i|\beta^{\bar{\pi}^{t-1}}_{s'_{\text{pub}}})$ (in vanilla CFR). The motivation for this is that $\pi^t = t\bar{\pi}^t - (t-1)\bar{\pi}^{t-1}$. If $v^t(h) = v^{t-1}(h)$ for each history h in the leaf PBS, then this modification of CFR-AVG is sound. Since $v^t(h) = v^{t-1}(h)$ when h is a full-game terminal node (i.e., it has no actions), this modified form of CFR-AVG is identical to CFR in a non-depth-limited game. However, that is not the case in a depth-limited subgame, and it remains an open question whether this modified form of CFR-AVG is theoretically sound in depth-limited subgames. Empirically, however, we found that it converges to a Nash equilibrium in turn endgame hold'em for every set of parameters (e.g., bet sizes, stack sizes, and initial beliefs) that we tested.

Figure 5.12 shows the performance of CFR-D, CFR-AVG, our modified form of CFR-AVG, and FP in TEH when using an oracle function for the value network. It also shows the performance of CFR-D, our modified form of CFR-AVG, and FP in TEH when using a value network trained through self-play. Surprisingly, the theoretically sound form of CFR-AVG does worse than CFR-D when using an oracle function. However, the modified form of CFR-AVG does better than CFR-D when using an oracle function and also when trained through self play.

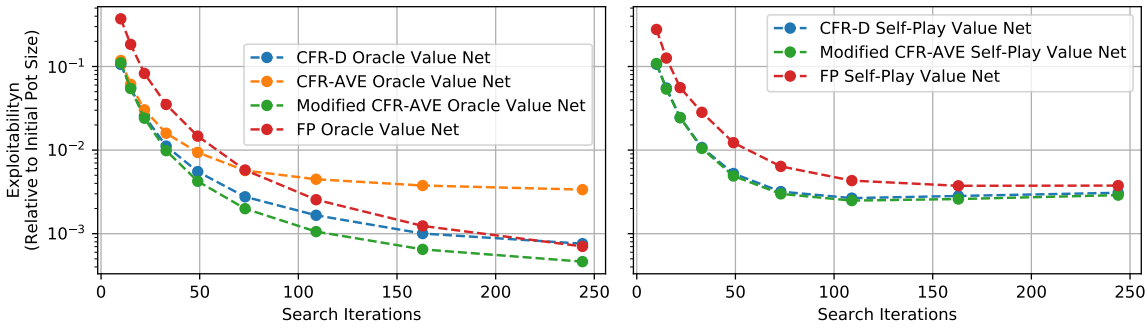


Figure 5.12: **Left:** comparison of CFR-D, CFR-AVG, modified CFR-AVG, and FP using an oracle value network which returns exact values for leaf PBSs. **Right:** comparison of CFR-D, modified CFR-AVG, and FP using a value network learned through 300 epochs of self play.

We also trained a model on HUNL with training parameters that were identical to the one reported in Section 5.3.5, but using CFR-D rather than CFR-AVG. That model lost to BabyTartanian8 by 10 ± 3 whereas the CFR-AVG model won by 9 ± 4. The CFR-D model also beat Slumbot by 39 ± 6 whereas the CFR-AVG model won by 45 ± 5.

Proof of Theorem 28

Our proof closely follows from [31] and [110].

Proof. Let $R^t(s_i)$ be the (cumulative) regret of infostates s_i on iteration t. We show that the regrets of all infostates in G' are bounded by $O(\sqrt{T})$ and therefore the regret of the entire game is bounded by $O(\sqrt{T})$.

First, consider the infostates in G. Since their policies are chosen according to CFR each iteration, their regrets are bounded by $O(\sqrt{T})$ regardless of the policies played in descendant infostates.

Next consider an infostate $s_i \in G' \setminus G$. We prove inductively that $R^t(s_i) \leq 0$. Let β^{π^t} be the PBS at the root of the subgame containing s_i in CFR-D, and $\beta^{\bar{\pi}^t}$ be the PBS at the root of the subgame containing s_i in CFR-AVG. On the first iteration, $\beta^{\pi^t} = \beta^{\bar{\pi}^t}$. Since we assume CFR-AVG computes an exact equilibrium in the subgame rooted at $\beta^{\bar{\pi}^t} = \beta^{\pi^t}$, so $R^t(s_i) = 0$ on the first iteration.

Next, we prove $R^{t+1}(s_i) \leq R^t(s_i)$. We define $a^{*,t}$ as

$$a_i^{*,t} = \operatorname*{argmax}_{a_i} \sum_{t'=0}^{t} v^{t'}(s_i, a_i) \tag{5.16}$$

By definition of regret,

$$R^{t+1}(s_i) = \sum_{t'=0}^{t+1} \left(v^{t'}(s_i, a_i^{*,t+1}) - v^{t'}(s_i) \right) \tag{5.17}$$

Separating iteration $t+1$ from the summation we get

$$R^{t+1}(s_i) = \sum_{t'=0}^{t} \left(v^{t'}(s_i, a_i^{*,t+1}) - v^{t'}(s_i) \right) + \left(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i) \right) \tag{5.18}$$

By definition of $a_i^{*,t}$ we know $\sum_{t'=0}^{t} v^{t'}(s_i, a_i^{*,t+1}) \leq \sum_{t'=0}^{t} v^{t'}(s_i, a_i^{*,t})$, so

$$R^{t+1}(s_i) \leq \sum_{t'=0}^{t} \left(v^{t'}(s_i, a_i^{*,t}) - v^{t'}(s_i) \right) + \left(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i) \right) \tag{5.19}$$

Since $\sum_{t'=0}^{t} \left(v^{t'}(s_i, a_i^{*,t}) - v^{t'}(s_i) \right)$ is the definition of $R^t(s_i)$ we get

$$R^{t+1}(s_i) \leq R^t(s_i) + \left(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i) \right) \tag{5.20}$$

Since $\pi^{t+1} = \pi^{*,t+1}$ in the subgame where $\pi^{*,t+1}$ is an exact equilibrium of the subgame rooted at $\beta^{\bar{\pi}^{t+1}}$, so π^{t+1} is a best response to $\bar{\pi}^{t+1}$ in the subgame and therefore $v^{t+1}(s_i, a_i^{*,t+1}) = v^{t+1}(s_i)$. Thus,

$$R^{t+1}(s_i) \leq R^t(s_i) \tag{5.21}$$

□

5.3.10 Hyper parameters

In this section we provide details of the value and policy networks and the training procedures.

We approximate the value and policy functions using artificial neural networks. The input to the value network consists of three components for both games: agent index, representation of the public state, and a probability distribution over infostates for both agents. For poker, the

public state representation consists of the board cards and the common pot size divided by stack size; for Liar's Dice it is the last bid and the acting agent. The output of the network is a vector of values for each possible infostate of the indexed agent, e.g., each possible poker hand she can hold.

We trained a policy network only for poker. The policy network state representation additionally contains pot size fractions for both agents separately as well as a flag for whether there have been any bets so far in the round. The output is a probability distribution over the legal actions for each infostate.

As explained in section 5.3.5 we use Multilayer perceptron with GeLU [71] activation functions and LayerNorm [3] for both value and policy networks.

For poker we represent the public state as a concatenation of a vector of indices of the board cards, current pot size relative to the stack sizes, and binary flag for the acting player. The size of the full input is

$$1(\text{agent index}) + 1(\text{acting agent}) + 1(\text{pot}) + 5(\text{board}) + 2 \times 1326(\text{infostate beliefs})$$

We use card embedding for the board cards similar to [28] and then apply MLP. Both the value and the policy networks contain 6 hidden layers with 1536 layers each. For all experiments we set the probability to explore a random action to $\epsilon = 25\%$ (see Section 5.3.3). To store the training data we use a simple circular buffer of size 12M and sample uniformly. Since our action abstraction contains at most 9 legal actions, the size of the target vector for the policy network is 9 times bigger than one used for the value network. In order to make it manageable, we apply linear quantization to the policy values. As initial data is produced with a random value network, we remove half of the data from the replay buffer after 20 epochs.

For the full game we train the network with the Adam optimizer with learning rate 3×10^{-4} and halved the learning rate every 800 epochs. One epoch is 2,560,000 examples and the batch size 1024. We used 90 DGX-1 machines, each with 8 $\times$ 32GB Nvidia V100 GPUs for data generation. We report results after 1,750 epochs. For TEH experiments we use higher initial learning rate 4×10^{-4}, but halve it every 100 epochs. We report results after 300 epochs.

For Liar's Dice we represent the state as a concatenation of a one hot vector for the last bid and binary flag for the acting player. The size of the full input is

$$1(\text{agent index}) + 1(\text{acting agent}) + n_{\text{dice}} n_{\text{faces}}(\text{last bid}) + 2{n_{\text{faces}}}^{n_{\text{dice}}}(\text{infostate beliefs}).$$

The value network contains 2 hidden layers with 256 layers each. We train the network with Adam optimizer with learning rate 3×10^{-4} and halved the learning rate every 400 epochs. One epoch is 25,600 examples and the batch size 512. During both training and evaluation we run the search algorithm for 1024 iterations. We use single GPU for training and 60 CPU threads for data generation. We trained the network for 1000 epochs. To reduce the variance in RL+Search results, we evaluated the three last checkpoints and reported averages in table 5.7.

Human Experiments for HUNL

We evaluated our HUNL agent against Dong Kim, a top human professional specializing in HUNL. Kim was one of four humans that played against Libratus [21] in the man-machine

competition which Libratus won. Kim lost the least to Libratus. However, due to high variance, it is impossible to statistically compare the performance of the individual humans that participated in the competition.

A total of 7,500 hands were played between Kim and the bot. Kim was able to play from home at his own pace on any schedule he wanted. He was also able to play up to four games simultaneously against the bot. To incentivize strong play, Kim was offered a base compensation of $\$1 \pm \$0.05x$ for each hand played, where x signifies his average win/loss rate in terms of big blinds per hundred hands played. Kim was guaranteed a minimum of $0.75 per hand and could earn no more than $2 per hand. Since final compensation was based on the variance-reduced score rather than the raw score, Kim was not aware of his precise performance during the experiment.

The bot played at an extremely fast pace. No decision required more than 5 seconds, and the bot on average plays faster than 2 seconds per hand in self play. To speed up play even further, the bot cached subgames it encountered on the preflop. When the same subgame was encountered again, it would simply reuse the solution it had already computed previously.

Kim's variance-reduced score, which we report in Section 5.3.5, was a loss of 165 ± 69 where the $\pm$ indicates one standard error. His raw score was a loss of 358 ± 188.

5.3.11 Proofs of Theoretical Results

We start by proving some preliminary Lemmas. For simplicity, we will sometimes prove results for only one player, but the results hold WLOG for both players.

For some policy profile $\pi = (\pi_1, \pi_2)$, let $v_i^\pi(s_1|\beta) : \mathcal{B} \to \mathbb{R}^{|S_i|}$ be a function that takes as input a PBS and outputs infoset values for player i at infoset s_1.

Lemma 16. *For fixed β and π_2, $v_1^{(\pi_1,\pi_2)}(s_1|\beta)$ is identical for any π_1 that is a BR to π_2 if $\beta_1(s_1) > 0$.*

Proof. π_1^* is a BR therefore it must maximize $V_1 = \sum_{s_1} p(s_1) v_1^\pi(s_1)$. It can only do so by achieving the unique maximum at each infoset s_1 that occurs with positive probability. □

Lemma 17. *Let $V_1^{\pi_2}(\beta)$ be player 1's BR value at β assuming that player 2 plays π_2. $V_1^{\pi_2}(\beta)$ is linear in β_1.*

Proof. This follows directly from Lemma 16 along with the definition of V_1,

$$V_1^{\pi_2}(\beta) = \sum_{s_1 \in S_1(s_{\text{pub}})} \beta_1(s_1) v_1(s_1|\beta, (BR(\pi_2), \pi_2))$$

. □

Lemma 18. *$V_1(\beta) = \min_{\pi_2} V_1^{\pi_2}(\beta)$, and the set of π_2 that attain $V_1(\beta)$ at β_0 are precisely the Nash equilibrium policies at β_0. This also implies that $V_1(\beta)$ is concave.*

Proof. By definition, the Nash equilibrium at β is the minimum among all choices of π_2 of the value to player 1 of her BR to π_2. Any π_2 that achieves this NE value when playing a BR is a NE policy.

From Lemma 17, we know that each $V_1^{\pi_2}(\beta)$ is linear, which implies that $V_1(\beta)$ is concave since any function that is the minimum of linear functions is concave. □

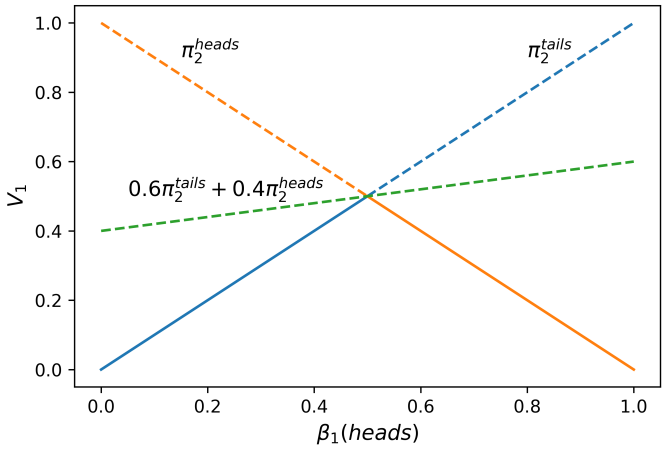


Figure 5.13: Illustration of Lemma 18. In this simple example, the subgame begins with some probability $\beta(heads)$ of a coin being heads-up, which player 1 observes. Player 2 then guesses if the coin is heads or tails, and wins if he guesses correctly. The payoffs for Player 2's pure strategies are shown as the lines marked π_2^{heads} and π_2^{tails}. The payoffs for a mixed strategy is a linear combination of the pure strategies. The value for player 1 is the minimum among all the lines corresponding to player 2 strategies, denoted by the solid lines.

Lemma 19. *At any β, the set of maps $v_1 : S_1 \to \mathbb{R}$ corresponding to Nash equilibrium policies π^* forms a convex set.*

Proof. A mixture of Nash equilibrium policy profile is a coarse correlated equilibrium, which means it's a Nash equilibrium since the game is two-player zero-sum. Therefore the set of Nash equilibrium *policies* is convex on the simplex.

Now, consider the map from infosets to values, using a normal form representation of the subgame:

$$v_1^{\pi^*}(s_1|\beta) = \sum_{h \in s_1} p(h|\beta)\pi_1^*(a_1)\pi_2^*(a_2)v_1(h|a_1, a_2) \tag{5.22}$$

This map is continuous in π^*, so the set of maps must also be convex. □

Now we can turn to proving the Theorem.

Consider a function $\tilde{V}_1$ that is an extension of V_1 to unnormalized probability distributions over S_1 and S_2; i.e. $\tilde{V}_i((s_{pub}, b_1, b_2)) = V_i((s_{\text{pub}}, b_1/|b_1|_1, b_2/|b_2|_1))$. $\tilde{V}_i = V_i$ on the simplex of valid beliefs, but we extend it in this way to $\mathbb{R}_{\geq 0}^{|s_1|} \setminus \vec{0}$ so that we can consider gradients w.r.t. $p(s_1)$.

We will use the term 'supergradient' to be the equivalent of the subgradient for concave functions. Formally, g is a supergradient of concave function F at x_0 iff for any x in the domain of F,

$$F(x) - F(x_0) \leq g \cdot (x - x_0).$$

Also, $superg(F) = -subg(-F)$.

Theorem (Restatement of Theorem 23)**.** *For any belief β_1 and any supergradient $\bar{g}$ of $\tilde{V}_1(\beta)$ with respect to β_1,*

$$v_1^{\pi^*}(s_1|\beta) = V_1(\beta) + \bar{g} \cdot \hat{s}_1 \tag{5.23}$$

for some Nash equilibrium policy π^*, where $\hat{s}_1$ is the unit vector in direction s_1.

Proof. Lemma 18 shows that $V_1(\beta)$ is a concave function of β, and its extension $\tilde{V}$ off the simplex is constant perpendicular to the simplex, so $\tilde{V}$ is concave as well. Therefore the notion of a supergradient is well-defined.

Furthermore, V_1 is the minimum of a set of linear functions $V_1^{\pi_2}$ (Lemma 18), so at each point β, V_1 is equal to $V_1^{\pi_2}$ for one or more policies π_2 which are exactly the set of equilibrium policies at β. The gradient of $V_1^{\pi_2}$ is

$$\nabla_{\beta_1} V_1^{\pi_2}(\beta) = \nabla_{\beta_1} \sum_{s_1 \in S_1(s_{\text{pub}})} \beta_1(s_1) v_1^{(BR(\pi_2),\pi_2)}(s_1|\beta) \tag{5.24}$$

$$= \sum_{s_1 \in S_1(s_{\text{pub}})} \hat{s}_1 v_1^{(BR(\pi_2),\pi_2)}(s_1|\beta) \tag{5.25}$$

$$\tag{5.26}$$

If there is only a single $V_1^{\pi_2}(\beta)$ plane that intersects $V_1(\beta)$, then V_1 lies on this plane and has a single supergradient which is simply the gradient of $V_1^{\pi_2}$ at this point[16].

Otherwise, $V_1(\beta)$ lies at an 'edge' defined by the intersection of planes corresponding to $V_1^{\pi_2}(\beta)$ for different equilibrium policies. The tangent plane for any supergradient at β_1 lies within the convex hull of these intersecting planes. By Lemma 19, the set of $V_1^{\pi_2}$ planes is convex so any plane in this convex hull corresponds to the value for some equilibrium π^*. Therefore, any supergradient of V corresponds to a $\nabla_{\beta_1} V_1^{\pi^*}(\beta_1)$ for some NE π^* in the subgame.

Finally, let's compute $g \cdot \hat{s}_1$ at some β_1 on the simplex (i.e. $|\beta_1|_1 = 1$).

[16]The supergradient of a differentiable function is only equal to its gradient in the interior, which is why we exclude the boundary from the result, i.e. we only consider $\mathbb{R}_{>0}^{|S_1|}$. This could be corrected with a more detailed proof, but we don't care about the boundary since CFR-AVG never assigns a probability of exactly 0 to any state.

$$g = \nabla_{\beta_1/|\beta_1|_1} V_1^{\pi^*}(s_{\text{pub}}, \beta_1/|\beta_1|, \beta_2) \cdot \frac{d}{d\beta_1}\left(\frac{\beta_1}{|\beta_1|_1}\right) \qquad \text{(chain rule)} \tag{5.27}$$

$$= \left(\sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta)\right) \cdot (|\beta_1|_1 - \beta_1)/(|\beta_1|_1)^2 \qquad \text{(Eq. 5.26)} \tag{5.28}$$

$$= \left(\sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta)\right) \cdot (1 - \beta_1) \qquad \text{(since } |\beta_1|_1 = 1\text{)} \tag{5.29}$$

$$= \sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) - \sum_{s_1' \in S_1(s_{\text{pub}})} \beta_1(s_1') v_1^{\pi^*}(s_1'|\beta) \tag{5.30}$$

$$= \sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) - V_1(\beta) \tag{5.31}$$

$$g \cdot \hat{s}_1 = v_1^{\pi^*}(s_1|\beta) - V_1(\beta) \tag{5.32}$$

And we're done.

□

Theorem (Restatement of Theorem 24)**.** *Let X be the vector of infostate EVs in PBS β corresponding to minimax policy profile π_X^*, and let Y be the vector of infostate EVs in β corresponding to minimax policy profile π_Y^*. Then $\lambda X + (1-\lambda)Y$ is the vector of infostate EVs in β corresponding to minimax policy profile $\lambda\pi_X^* + (1-\lambda)\pi_Y^*$ for $0 \leq \lambda \leq 1$.*

Proof. We focus on a single infoset EV, $v_1(s_1|\beta, \pi^*)$, and consider a normal form representation of the subgame.

$$v_1^{\lambda\pi_X^* + (1-\lambda)\pi_Y^*}(s_1|\beta) = \sum_{h \in s_1} p(h|\beta)\left(\lambda\pi_X^*(a) + (1-\lambda)\pi_Y^*(a)\right) v_1(h|a) \tag{5.33}$$

$$= \lambda \sum_{h \in s_1} p(h|\beta)\pi_X^*(a) v_1(h|a) + (1-\lambda) \sum_{h \in s_1} p(h|\beta)\pi_Y^*(a) v_1(h|a) \tag{5.34}$$

$$= \lambda v_1^{\pi_X^*}(s_1|\beta) + (1-\lambda) v_1^{\pi_Y^*}(s_1|\beta) \tag{5.35}$$

This mixed joint policy can be played independently by each agent in a two-player zero-sum game, since all coarse correlated equilibria are Nash equilibria. The interpolated policy is a minimax (Nash) strategy, due to Lemma 19. □

Theorem (Restatement of Theorem 25)**.** *Consider an idealized value approximator that returns the most recent sample of the value for sampled PBSs, and 0 otherwise. Running Algorithm 4 with T iterations of CFR in each subgame will, after a finite amount of time, produce a value approximator that produces values that correspond to a $\frac{C}{\sqrt{T}}$-equilibrium policy for any PBS that could be encountered during play, where C is a game-dependent constant.*

Proof. CFR [163] is an iterated self play algorithm whose average policy across iterations converges to a Nash equilibrium. The key idea behind CFR is that it decomposes the regret minimization in the full game into independent regret minimization problems at each information state. At each infostate I, CFR minimizes the regret over the *counterfactual value*, that is the EV of taking action a at I weighted by the probability of reaching I_i assuming player i plays to reach I and the opponent and chance play their policies at iteration t. The central result of [163] is that the total regret R_i^T in the game is bounded by the sum of the counterfactual regrets at each infostate $R_i^T(I)$. [163] then proposes an independent regret matching policy [65] of

$$\pi_i^{t+1}(s_i, a_i) = \begin{cases} \frac{\max\{0, R_i^t(s_i,a_i)\}}{\sum_{a_i' \in \mathcal{A}_i(s_i)} \max\{0, R_i^t(s_i,a_i')\}} & \text{if} \sum_{a_i' \in \mathcal{A}_i(s_i)} \max\{0, R_i^t(s_i, a_i')\} > 0 \\ \frac{1}{|\mathcal{A}_i(s_i)|} & \text{otherwise} \end{cases} \tag{5.36}$$

at each infostate, whose external regret after T iterations is bounded by $O(1/\sqrt{T})$. This leads to the CFR bound

$$R_i^T \leq \Delta |\mathcal{I}_i| \sqrt{|A_i|} / \sqrt{T},$$

where Δ is the range of payoffs, $|\mathcal{I}_i|$ is the number of infostates, and $|A_i|$ is the max number of actions for player i.

A crucial property of CFR is that each regret minimization at I only depends on the counterfactual values of each action at I. It doesn't matter exactly what policy is performed at other infostates as long as they have low regret.

Suppose we compute an ϵ-Nash equilibrium in a PBS β^{π^t}. Then the values for the PBS correspond to an (average) policy in β^{π^t} that achieves at most ϵ regret in I at time t.

Consider CFR run in a depth-limited subgame $\mathcal{G}$. Let I_i^L be the infostates in leaf L, and $I_i^{\mathcal{G}^*}$ be the infostates of $\mathcal{G}$ not in any leaf subgame. If the total regret at each leaf PBS $\beta_L^{\pi^t}$ at each iteration t is bounded by $|I_i^L|/\sqrt{T}$ then the total regret in $\mathcal{G}$ will be bounded by

$$\begin{aligned} R_{i,\mathcal{G}}^T &\leq |I_i^{\mathcal{G}^*}|/\sqrt{T} + \frac{1}{T} \sum_{L \in \mathcal{G}} \sum_{t=1}^{T} R_{i,\beta_L^{\pi^t}}^T && (5.37) \\ &= |I_i^{\mathcal{G}^*}|/\sqrt{T} + \frac{1}{T} \sum_{L \in \mathcal{G}} \sum_{t=1}^{T} |I_i^L|/\sqrt{T} && (5.38) \\ &= |I_i^{\mathcal{G}^*}|/\sqrt{T} + \sum_{L \in \mathcal{G}} |I_i^L|/\sqrt{T} && (5.39) \\ &= |I_i^{\mathcal{G}}|/\sqrt{T}. && (5.40) \end{aligned}$$

In other words, if the $O(1/\sqrt{T})$ regret bound holds at each leaf PBS $\beta_L^{\pi^t}$ encountered during the search in $\beta^{\mathcal{G}}$, then the regret bound also holds for the values computed in β. So now we must show inductively that valid bounds are computed for each relevant PBS that may be encountered during play.

Consider the naive algorithm that at each leaf node in a depth-limited subgame, recursively runs the same CFR procedure in each leaf subgame β_L^t. This algorithm would would clearly

obey the regret bound in Equation 5.40 by sequentially solving $O(t^{|s_{pub}|})$ PBSs. But what about the sampling approach in Algorithm 4?

Every PBS solved during this naive scheme can be specified by a pair of a public state s_{pub} and a sequence $\tau = (t_1, t_2, ..., t_k))$, denoting that this is the PBS at s_{pub} with beliefs that arise from π^{t_1} at the root PBS, π^{t_2} at the second subgae on the path to s_{pub}, and so on. We call τ an **iter-sequence**.

We define the "natural ordering" of iter-sequences to be the one that places all suffixes of τ before τ, and orders all prefixes lexicographially. E.g. for $T = 2$ and depth of 3, the natural ordering would be

$$(1,1,1),(1,1,2),(1,1),(1,2,1),(1,2,2),(1,2),(1),(2,1,1),(2,1,2),(2,1),(2,2,1),(2,2,2),(2,2),(2)$$

Consider a PBS β with some iter-sequence τ. Suppose the value function for every PBS with an iter sequence $\tau' < \tau$ is "valid" at iteration j of Algorithm 4. Then CFR will be correct in all subgames leading to β up to the relevant iteration. So with positive probability, at iteration j of Algorithm 4, the sequence of leaf PBSs leading to β will be reached, and CFR will be evaluated in β. Furthermore, the PBSs for all subgames of β are suffixes in the natural ordering, so values for all leaf nodes in β will be "valid". Therefore, with positive probability, a correct value of β will be inserted into the value function at this iteration of Algorithm 1. And all future computations of the value of β will also be correct, so the value of β will always be valid after this point.

Order all PBSs encountered during this procedure $(\beta_1, ..., \beta_N)$ by the natural ordering of their iter-sequences. Suppose on iter j of Algorithm 4, the value function is valid for $(\beta_1, ..., \beta_k)$. Then $P(\beta_{k+1}$ sampled on iter m) is positive and independent on each iter $m > j$. Therefore by the second Borel-Cantelli Lemma,

$$\lim_{M \to \infty} Pr(\beta_{k+1} \text{ sampled on some iter j} < \text{m} < \text{M}) = 1$$

So each β will eventually be sampled and produce a valid value estimate.

The only thing left to show is that any β encountered during play against the final policies generated by this procedure will be in the set of $\{\beta\}$ computed by Algorithm 4. The set of possible test-time PBSs consist of those where the agent plays π^T and the opponent plays an arbitrary policy. As described in Section 5.3.3, leaf PBSs are sampled at a random $t \leq T$ for a public state reached by one player playing π^t and the other playing a uniform policy with probability ϵ (and π^t otherwise). So it will sample every $\beta_L^{\pi^T}$ for any L as long as it's in the support of π^T for at least one player. This is a superset of all leaf nodes that may be encountered when the agent plays π^T at test time. □

Theorem (Restatement of Theorem 26)**.** *If Algorithm 4 is run at test time with no off-policy exploration, a value network that has error at most δ for any leaf PBS, and with T iterations of CFR being used to solve subgames, then the algorithm plays a $(\delta C_1 + \frac{\delta C_2}{\sqrt{T}})$-Nash equilibrium, where C_1, C_2 are game-specific constants.*

Proof. We prove the theorem inductively. Consider first a subgame near the end of the game that is not depth-limited. I.e., it has no leaf nodes. Clearly, the policy π^* that Algorithm 4 using CFR plays in expectation is a $\frac{k_1}{\sqrt{T}}$-Nash equilibrium for game-specific constant k_1 in this subgame.

Rather than play the average policy over all T iterations $\bar{\pi}^T$, one can equivalently pick a random iteration $t \sim \text{uniform}\{1, T\}$ and play according to π^t, the policy on iteration t. This algorithm is also a $\frac{k_1}{\sqrt{T}}$-Nash equilibrium in expectation.

Next, consider a depth-limited subgame G such that for any leaf PBS β^t on any CFR iteration t, the policy that Algorithm 4 plays in the subgame rooted at β^t is in expectation a δ-Nash equilibrium in the subgame. If one computes a policy for G using tabular CFR-D [31] (or, as discussed in Section 5.3.9, using CFR-AVG), then by Theorem 2 in [31], the average policy over all iterations is $k_2\delta + \frac{k_3}{\sqrt{T}}$-Nash equilibrium.

Just as before, rather than play according to this average policy $\bar{\pi}^T$, one can equivalently pick a random iteration $t \sim \text{uniform}\{1, T\}$ and play according to π^t. Doing so would also result in a $k_2\delta + \frac{k_3}{\sqrt{T}}$-Nash equilibrium in expectation. This is exactly what Algorithm 4 does.

Since there are a finite number of "levels" in a game, which is a game-specific constant, Algorithm 4 plays according to a $\delta C_1 + \frac{\delta C_2}{\sqrt{T}}$-Nash equilibrium.

□

5.4 Comparison of Search via Multi-Valued States versus Public Belief States

This chapter described two methods for conducting depth-limited search: multi-valued states (MVSs) (described in Section 5.2 and public belief states (PBSs) (described in Section 5.3). There are benefits and drawbacks to both approaches, which we now describe in detail. The right choice may depend on the domain and future research may change the competitiveness of either approach.

The most significant drawback of the PBS approach is that evaluating PBSs with a function approximator is more expensive and less scalable to large games than MVSs. The input to a function that predicts the value of a MVS is simply the state description (for example, the sequence of actions), and the output is several values. In our experiments on HUNL, the input was 34 floating-point numbers and the output was 4 floating-point numbers. In contrast, the input to a function that predicts the values of a PBS is a probability vector for each player over the possible infostates they may be in. For example, in HUNL, the input is more than 2,652 floating-point numbers and the output is more than 1,326 floating-point numbers. The input would be even larger in games with more infostates per infoset and in a game such as Stratego would be prohibitively large using current techniques.

Moreover, because a PBS is partly defined by a player's belief distribution, the values of the leaf infosets in the PBS approach must be recalculated each time the strategy in the subgame changes. With the best domain-specific iterative algorithms, this requires recalculating the leaf infosets between 100 and 1,000 times. In contrast, the MVS approach requires only a single function call for each leaf node regardless of the number of iterations conducted.

Another drawback of the PBS approach is that learning a mapping from PBSs to infostate values is computationally more expensive than learning a mapping from states to a set of values. For example, Modicum required less than 1,000 core hours to create this mapping. In contrast, DeepStack required over 1,000,000 core hours to create its mapping. ReBeL substantially im-

proved upon DeepStack by learning through self play rather than by generating random PBSs, but even ReBeL was orders of magnitude more expensive to train than Modicum.

On the other hand, the MVS approach as it exists so far requires knowledge of a blueprint strategy that is already an approximate Nash equilibrium. A benefit of the PBS approach is that, as we describe in Section 5.3, it can be combined with self-play to completely avoid the need for a blueprint.

Another benefit of the PBS approach is that in many games (but not all) it obviates the need to keep track of the sequence of actions played. For example, in poker if there are two different sequences of actions that result in the same amount of money in the pot and all players having the same belief distribution over what their opponents' cards are, then the optimal strategy in both of those situations is the same. This is similar to how in Go it is not necessary to know the exact sequence of actions that were played. Rather, it is only necessary to know the current configuration of the board (and, in certain situations, also the last few actions played).

A further benefit of the PBS approach is that its run-time complexity does not increase with the degree of precision in approximating a Nash equilibrium, other than needing a better (possibly more computationally expensive) function approximator. In contrast, for the MVS approach the computational complexity of finding a solution to a depth-limited subgame grows linearly with the number of values per state.

In short, there are benefits and drawbacks to using PBSs versus MVSs and neither approach is clearly superior in all settings. Currently, the most important factor in deciding which technique to use is the number of infosets per public state. If the number of infosets per public state is small, then the PBS approach used in ReBeL is likely the superior choice. Otherwise, if there are a lot of infosets per public state, then the MVSs approach is likely the superior choice.

Chapter 6

Empirical Evaluation via Poker AI Agents

Poker served for decades as a grand challenge problem for the fields of AI and game theory. The foundational papers on game theory, first written nearly a century ago, used poker games as examples when describing their work [112, 156] and poker is the most popular and well-known imperfect-information game in the world. There are many variants of poker, but no-limit Texas hold'em in particular is the most popular variant and is the variant played in the World Series of Poker Main Event, the most prestigious poker competition in the world. Despite AI successes in many perfect-information games, no-limit Texas hold'em poker proved stubbornly resistant to prior AI approaches.

The algorithms described in this thesis are designed from the ground up to be domain-independent and applicable to a wide variety of settings. However, in order to measure the effectiveness of these techniques relative to others, we evaluated primarily on the common benchmark of no-limit Texas hold'em poker. This chapter describes large-scale poker agents that were created in order to evaluate the techniques described in this thesis.

In particular, Libratus [21] (described in Section 6.4) was the first agent to defeat top humans in heads-up (i.e., two-player) no-limit Texas hold'em poker. Libratus leveraged several new techniques, with the most important being the newly developed safe and nested search algorithms described in Section 5.1. Libratus demonstrated the importance of search even in an imperfect-information game. Prior to Libratus, the vast majority of poker agents did not use search, and those that did only used it sparingly and not in a theoretically sound way. Additionally, Pluribus [23] (described in Section 6.6) was the first agent to defeat top humans in multi-player (specifically, six-player) no-limit Texas hold'em poker, the most popular format of poker in the world. Pluribus also leveraged several new techniques, with the most important being the depth-limited search techniques described in Section 5.2, which allowed search to scale to much larger games by reducing the computational cost by orders of magnitude. In fact, Pluribus's training cost was the equivalent of only $144 at 2019 cloud computing spot instance rates.

6.1 Tartanian7

Tartanian7 [24] was our entry for the 2014 Annual Computer Poker Competition's heads-up no-limit Texas hold'em events. It was the clear winner, beating each of the next three competitors

by at least 20 thousandths of a big blind per hand of poker (20 mbb/g).

Tartanian7 was distinguished by its large-scale distributed CFR algorithm. While past bots had potentially up to 5,000 buckets per round for their information abstraction, Tartanian7 had 30,000 buckets. It also had a much larger action abstraction than prior bots. Tartanian7 did not use search.

Tartanian7 used a new hierarchical abstraction algorithm that would first divide hands into N "clusters" based only on public information (community cards). It would then further divide these clusters into separate buckets based on private and public information [24]. In this way, the abstraction was a mix or perfect and imperfect recall. Once a hand was assigned to a cluster, any future hand on later rounds would be in the same cluster. In Tartanian7, there were 60 clusters and 500 buckets per cluster.

Since hands always remained in a single cluster for the remainder of the game, it was possible to use a distributed form of MCCFR for equilibrium finding. Specifically, one node was designated a "head" node that would handle the preflop part of the computation. Upon reaching the flop, the head node would sample one flop from each cluster and have each cluster in parallel update the regrets for the remainder of the MCCFR iteration. Since there were 60 clusters and one head node, we used 61 blades of the Pittsburgh Supercomputing Center's Blacklight supercomputer for this purpose.

6.2 Claudico

Claudico[1] was an AI bot that played against four top human specialist professionals in heads-up no-limit Texas hold'em poker in 2015. The bot played 80,000 hands in total.

Claudico was based on Tartanian7. We used a larger information abstraction (60,000 and 90,000 buckets for the big and small blind position, respectively) and ran MCCFR for longer. We also used an asymmetric action abstraction rather than Tartanian7's symmetric abstraction.

For part of the competition, we also used search in the final betting round. However, this search algorithm only solved the remainder of the game once upon reaching the final round. If the opponent selected an off-tree action after that point, Claudico would use action translation to map the bet to a size already in the abstraction. The search algorithm also used card abstraction.

Claudico ultimately lost by 92 mbb/g, but the standard error was 52.8 mbb/g, so the result was not statistically significant at the 95% confidence level. By the end of the competition, it was clear that the humans it was playing against had found weaknesses they could exploit. This included a recognition that the bot could not distinguish certain hands due to its information abstraction, and also a recognition that the bot would incorrectly interpret certain bet sizes due to action translation. The humans' successful discovery and exploitation of these weaknesses motivated research into better search techniques, and ultimately led to Libratus, which defeated top humans in the same game less than two years later (see Section 6.4).

[1]Claudico is Latin and means *I limp*. Its name was chosen due to the bot's tendency to limp (that is, call for the initial action in the game rather than raise), which went against conventional poker wisdom at the time.

6.3 Baby Tartanian8

Baby Tartanian8 [17] was the winner of both events in the 2016 Annual Computer Poker Competition (there was no 2015 competition). Like Tartanian7, which won the 2014 competition, Baby Tartanian8 was an abstraction-based non-searching agent. However, it used a much finer abstraction than any other bot. It also used a form of regret-based pruning (described in Section 3.3) designed for MCCFR that effectively increased the size of the abstraction even further.

Baby Tartanian8 is likely the strongest non-searching poker AI bot for HUNL ever developed, and remains a challenging benchmark even five years later for state-of-the-art searching agents. The major weakness of Baby Tartanian8 is that due to its use of action translation and post-processing, it is much more exploitable than modern search-based agents. However, unlike search-based bots, Baby Tartanian8 acts instantaneously.

To facilitate distributed equilibrium finding, Baby Tartanian8's abstraction algorithm decomposes the game tree into disjoint parts after the early stage of the game. Specifically, we defined this "early stage" as the preflop in poker, and separated the remaining game tree into disjoint sets by conditioning on the flop community cards. During equilibrium finding on a distributed architecture, the early stage of the game is assigned to one head node, while each remaining disjoint part is assigned to a different child node. This ensures that each machine can access memory locally and run independently, other than one message to and from the head node on each iteration. This abstraction approach was used for the top three agents in the ACPC in 2016, and was also used by our champion agent Tartanian7 in the 2014 competition [24] (described in Section 6.1).

Baby Tartanian8 uses an asymmetric action abstraction, in which more actions are allowed for the opponent than for ourselves [5]. This allows us to leverage domain knowledge to eliminate suboptimal actions for ourselves, while still being able to respond intelligently in case the opponent chooses suboptimal actions. Actions were selected by examining the equilibrium strategies of smaller agents and choosing the actions that were most commonly used.

Equilibrium Finding

For equilibrium finding, we used a distributed variant of MCCFR based on the algorithm used by Tartanian7 (see Section 6.1). Our agent also employs a novel sampling algorithm based on regret-based pruning (RBP) (see Section 3.3). RBP allows an agent to avoid exploring actions in the game tree on every iteration if those actions have performed poorly in the past, while still guaranteeing convergence to a Nash equilibrium within the same number of iterations. Thus, while the number of iterations needed to arrive within a certain ϵ of a Nash equilibrium does not change, each iteration is performed far more quickly. The number of iterations for which an action may be skipped depends on how negative the regret is for that action—the action must be explored again at the earliest iteration on which its regret could turn positive.

Our *sampled* implementation of RBP, which we refer to as **regret-based sampling (RBS)**, has not been proven to converge to a Nash equilibrium. Nevertheless, preliminary experiments on smaller-scale hardware, as shown in Figure 6.1, demonstrated a substantial increase in performance in both small and large games. Due to this strong empirical performance in large-scale experiments, we used RBS in the equilibrium finding of Baby Tartanian8, despite lacking theo-

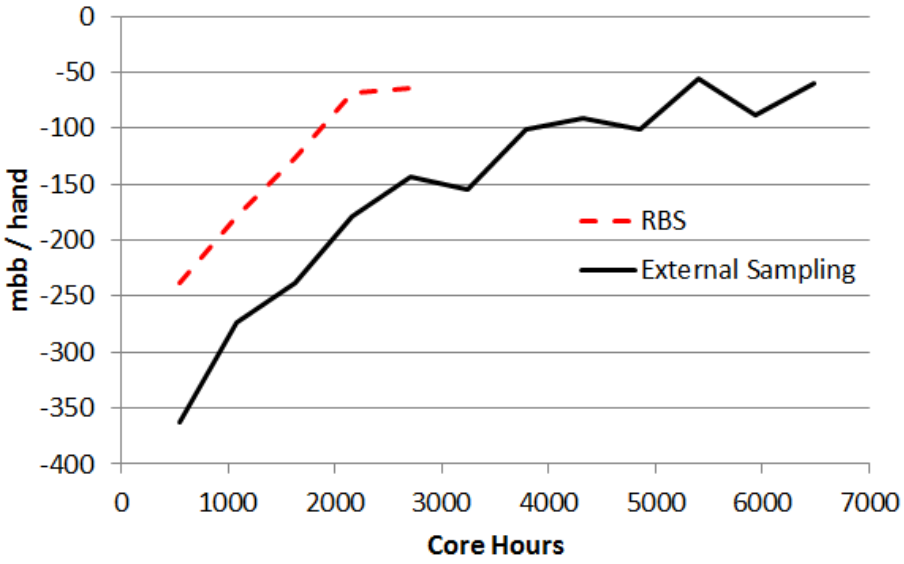


Figure 6.1: Performance of RBS compared to external-sampling MCCFR in a smaller-scale preliminary experiment. Both algorithms were used to train a strategy based on identical abstractions using 64 cores. Performance in milli-big blinds per hand (mbb / hand) is shown against *Tartanian7*, the winner of the 2014 ACPC no-limit hold'em competition.

retical guarantees. Although RBS likely improved our early convergence rate, there was some evidence that our implementation of RBS may have led to decreased performance when closer to convergence. For this reason, we turned off RBS for the final 5 days of the equilibrium computation.

Agent Construction

Our equilibrium finding was run offline at the San Diego Supercomputing Center on the Comet supercomputer. We used 3,408 cores (142 blades with 24 cores each) for about 600 hours, for a total of about 2 million core hours. Each node had 128 GB of RAM.

Since we used an asymmetric abstraction, we solved two separate abstractions (using about 1 million core hours for each abstraction) and used half of each solution for our final agent (i.e., the first mover's strategy or the second mover's strategy). Each abstraction had $1.6 \cdot 10^{14}$ nodes in its game tree, and the final strategy required 16 TB to store as doubles.

The submission size limit for the ACPC was 200 GB. To satisfy this constraint, the final strategy was purified so that the agent would take a single action with probability one [24]. The purified strategy was then compressed so that each situation would use only $\lceil \ln(|A|) \rceil$ bits to represent which action should be played, where $|A|$ is the number of possible actions in a situation. To reduce the possibility of an opponent exploiting our deterministic strategy, we did not purify or compress the early part of the game (the preflop), which requires only 170 KB to store uncompressed. The size constraints resulted in our submission of a "Baby" version of Tartanian8.

6.4 Libratus

In this section we describe Libratus,[2] an AI bot that, in a 20-day, 120,000-hand competition featuring a $200,000 prize pool, became the first AI to defeat top humans in HUNL poker.[3]

Libratus was inspired by the weaknesses that were seen in Claudico (see Section 6.2). In particular, information abstraction and action translation were two major issues in Claudico. It was clear that both of these could be addressed through search[4], though determining exactly how was not an easy process. For this reason, there was a heavy emphasis on search in Libratus, and especially **nested search** (described in Section 5.1), which distinguishes it from previous bots.

Libratus featured three modules: the first was a blueprint strategy computed offline using a variant of MCCFR. The second was a real-time search algorithm that improved upon the blueprint when playing against actual humans. The last was a "self-improvement" module that would augment the blueprint strategy with additional actions offline, where the actions were determined by the most common bet sizes used by the humans during the competition.

Blueprint Strategy

Most of the actions included in Libratus's action abstraction were determined by analyzing the most common bet sizes at various points in the game taken by prior top AIs in the **Annual Computer Poker Competition (ACPC)**[5]. However, bet sizes for the first two actions in the game were determined by the parameter optimization algorithm described in Section 4.2, which converged to a locally optimal set of bet sizes.

Libratus does not use any card abstraction on the first and second betting rounds. The last two betting rounds, which are exponentially larger, are abstracted only in the blueprint strategy. The 55 million different hand possibilities on the third round are algorithmically grouped into 2.5 million abstract buckets, and the 2.4 billion different possibilities on the fourth round are algorithmically grouped into 1.25 million abstract buckets. However, the AI does not follow the blueprint strategy in these rounds and instead applies a nested search algorithm, described in the next section, which does not use any card abstraction. Thus, each poker hand is considered individually during actual play. The card abstraction algorithm that we used was similar to that used in our prior AIs Baby Tartanian8 (covered in Section 6.3) and Tartanian7 (covered in Section 6.1).

Once the abstraction was constructed, we computed the blueprint strategy for Libratus via a variant of MCCFR. Our variant of MCCFR traverses a smaller portion of the game tree on

[2]Libratus is Latin and means *balanced* (for approximating Nash equilibrium) and *forceful* (for its powerful play style and strength).

[3]While past AIs have challenged humans in poker, no other AI has demonstrated the ability to defeat top humans in HUNL. Polaris previously defeated human specialists in heads-up *limit* Texas hold'em, and the game was later essentially solved [12], but heads-up limit Texas hold'em is a far simpler game than HUNL. It is widely believed that AIs have operated profitably in online poker games, including HUNL games, but at generally low stakes playing against average players. The AI DeepStack beat non-specialist humans in HUNL [110], but those players were used to primarily playing other poker variants, are not considered strong in HUNL, and were offered little incentive.

[4]Claudico did use search on the final betting round, but that search algorithm still used information abstraction and action translation.

[5]`www.computerpokercompetition.org`

each iteration by employing a form of regret-based pruning (covered in Section 3.3). Intuitively, there are many clearly suboptimal actions in the game, and repeatedly exploring them wastes computational resources that could be better used improving the strategy elsewhere. Rather than explore every hypothetical alternative action to see what its reward would have been, our algorithm probabilistically skips over unpromising actions that have very negative regret as it proceeds deeper into the tree during a game.

Formally, an action a with regret $R(a)$ that is below a threshold C (where C is negative) is sampled with probability $K/\big(K + C - R(a)\big)$, where K is a positive constant. This sampling is only done for about the last half of iterations to be run; the first half is conducted using traditional external-sampling MCCFR. Other formulas can also be used.

This led to a factor of three speedup of MCCFR in practice and allowed us to solve larger abstractions than were previously possible. This skipping also mitigates the problems of **imperfect recall**. Imperfect-recall abstractions involve intentionally forgetting some aspects of the cards on the path of play so far in order to be able to computationally afford to have a more refined abstraction of the present state of cards [17, 24, 25, 161]. Since all infosets in a single abstract card bucket share the same strategy, updating the strategy for one of them leads to updating the strategy for all of them. This is not an issue if all of them share the same optimal strategy at the solution reached, but in practice there are differences between their optimal strategies and they effectively "fight" to push the bucket's strategy toward their own optimal strategy. Skipping negative-regret actions means that infosets that will never be reached in actual play will no longer have their strategies updated, thereby allowing the infosets that will actually occur during play to move the bucket's strategy closer to their optimal strategies.

Nested Search

While purely abstraction-based approaches have produced strong AIs for poker [17, 25, 60, 80], abstraction alone has not been enough to reach superhuman performance in HUNL. In addition to abstraction, Libratus uses search to compute a more detailed strategy for the particular part of the game it finds itself in. Compared to prior bots such as Claudico, Libratus uses a much more advanced form of search.

Libratus plays according to the abstract blueprint strategy only in the early parts of HUNL, where the number of possible states is relatively small and we can afford the abstraction to be extremely detailed. Upon reaching the third betting round or situations where no additional bets or raises can be made, Libratus constructs a new subgame, without any card abstraction, that is solved in real time. The subgame's solution is used as the strategy for Libratus going forward.

Libratus uses unsafe search upon reaching the third betting round for the first time. Thereafter, it uses Nested Reach Maxmargin search (described in Section 5.1.3) after every opponent action. This allowed Libratus to avoid action translation [49], which was a major weakness in Claudico.

Since the subgame is solved in real time, the abstraction in the subgame can also be decided upon in real time and change between hands. Libratus leveraged this feature by changing, at the first point of subgame solving, the bet sizes it would use in every subgame, thereby forcing the opponent to continually adapt to new bet sizes and strategies. Specifically, Libratus increased or decreased all its bet sizes by a percentage chosen uniformly at random between 0% and 8%.

Self-Improvement

The third module of Libratus is the self-improver, which improves the blueprint strategy in the background. The way machine learning has typically been used in game playing is to try to build an opponent model, find mistakes in the opponent's strategy, and exploit those mistakes [7, 48, 52]. The downside is that trying to exploit the opponent opens oneself to being exploited. Therefore, Libratus generally did not do opponent exploitation. Instead, it used the data of how the opponents played against it to suggest where the holes in its own strategy were, and algorithmically filled those holes in its blueprint strategy in the background. This is similar to the Simultaneous Abstraction and Equilibrium Finding algorithm described in Section 4.3, except the actions to be added were chosen based on the human opponents' play rather than by computing a best response. In other words, Libratus used the humans as the best response algorithms.

In most situations that can occur in the first two betting rounds, real-time search would be prohibitively slow in Libratus because it did not use depth-limited search. In those rounds there are many actions in the abstraction, so the error from rounding to a nearby size is small. Still, there is some error, and this could be reduced by including more actions in the abstraction. To fill in these gaps in the abstraction, Libratus analyzed the gaps between actions and which gaps were most heavily exploited by its opponents each day. It chose the top k gaps on the first betting round where the gaps were scored based on how frequently opponent actions were played in the gap and how far those actions were from an existing action in the abstraction. Based on the available computing resources, we chose $k = 3$ so that the algorithms could typically fix three holes to reasonable accuracy in 24 hours. For each of those gaps, a subgame was solved overnight for the bet size roughly at the midpoint of the gap. If those actions were used by the opponent the following day, then the subgame's solution was used rather than rounding to an action farther away in the abstraction. In this way Libratus was able to progressively narrow its gaps as the competition proceeded by leveraging the humans' ability to find weaknesses in an opponent. Furthermore, these fixes to its strategy are universal: they work against all opponents, not just the opponents that Libratus has faced.

Specifically, Libratus used two versions of the self-improver. The "defensive" version attempted to fill in gaps in Libratus's abstraction to make it more difficult for the opponent to exploit the AI. The "offensive" version attempted to exploit its opponents to an extent if they consistently used the same bet size. Both versions were only applied to actions in the first betting round. We now describe both techniques in detail. In all cases, bet sizes are measured as fractions of the size of the pot.

Each evening after the day's games were over, Libratus determined two opponent bet sizes to solve with the defensive self-improver, and one bet size to solve with the offensive self-improver. Defensive bet sizes were determined by scoring each "gap" between existing bet sizes in the abstraction. A gap is defined by two neighboring bet sizes A and B already in the abstraction. If, during the day, an opponent chose a bet size x such that $A < x < B$, then the gap's score would increase by the distance of x from A or B. Formally, the score would increase by $\min\{x - A, B - x\}$. The pseudo-harmonic midpoint [49] of the two highest-scoring gaps would be solved using defensive self-improvement each night. Specifically, if a gap between bet sizes A and B was selected, then defensive self-improvement would be applied to the bet size $(A + B +$

$2AB)/(A + B + 2)$. One opponent bet size was also solved overnight using offensive self-improvement. This was determined by simply choosing the most common opponent bet size in the previous day.

In the offensive self-improver, a subgame was solved using unsafe subgame solving, which means we assume both players play according to the blueprint strategy for all moves preceding the subgame. In the subgame the opponent was given the choice between folding, checking, or calling (except in the first action of the game, where calling was not provided as a valid option), or betting the self-improvement bet size. After the subgame was solved, the strategy in the subgame of the response to the self-improvement bet size was used if the opponent consistently bet that particular size in the future. Specifically, if the opponent bet that particular size for each of the last eight times they bet in that particular situation, then we would use the offensive self-improvement strategy to respond to the bet size. The offensive self-improver enabled Libratus to exploit an opponent in a fairly safe way if they were not playing a balanced strategy. However, during the *Brains vs. AI* competition the human opponents changed the bet sizes they used almost every day in order to prevent Libratus from calculating an effective response to their strategy. As a result, the offensive self-improver played little role in the competition.

In the defensive self-improver, a subgame was solved in a manner similar to the offensive self-improver, but with the addition of at least one "control" bet size that was commonly played in the blueprint strategy that we had computed in advance of the competition using the first module of Libratus described in the body of this paper. The sole role of including the control action was to determine a balanced strategy in the game tree following the self-improvement bet size: we do not want to assume that the opponent uses the selected new bet size for all private cards. Therefore, for computational speed, our algorithm used a significantly coarser abstraction following the control action, and its strategy was discarded after subgame solving finished. Using unsafe subgame solving with control actions still has theoretical bounds on exploitability when applied to the first action of the game—because there is no assumption being made about the opponent's prior play. For situations other than the first action of the game, unsafe subgame solving lacks theoretical guarantees, but empirically we found it to produce competitive strategies with generally low exploitability—as shown, for example, in the experiments discussed in the body of this paper. The strategy in the subgame of the response to the self-improvement bet size (but not the control bet size) was added to the overall blueprint strategy of Libratus. If an opponent chose an action that was close to the self-improvement bet size, then Libratus would use the self-improvement strategy as a response. This is in contrast to the offensive self-improver, which only used the strategy from the subgame if the opponent used the same bet size consistently. The defensive self-improver played a more significant role in the competition. By the end of the competition, roughly half of all the hands played by Libratus were played using a strategy determined by defensive self-improvement.

Results

No-limit Texas hold'em is the most popular form of poker in the world and has been the primary benchmark challenge for AI in imperfect-information games. The competition was played over the course of 20 days, and involved 120,000 hands of poker. A prize pool of $200,000 was split among the four humans based on their performance against the AI to incentivize strong play. The

AI decisively defeated the team of human players by a margin of 147 mbb / hand, with 99.98% statistical significance (see Figure 6.2), and defeating each human individually. This was the first time an AI defeated top humans in heads-up no-limit Texas hold'em poker.

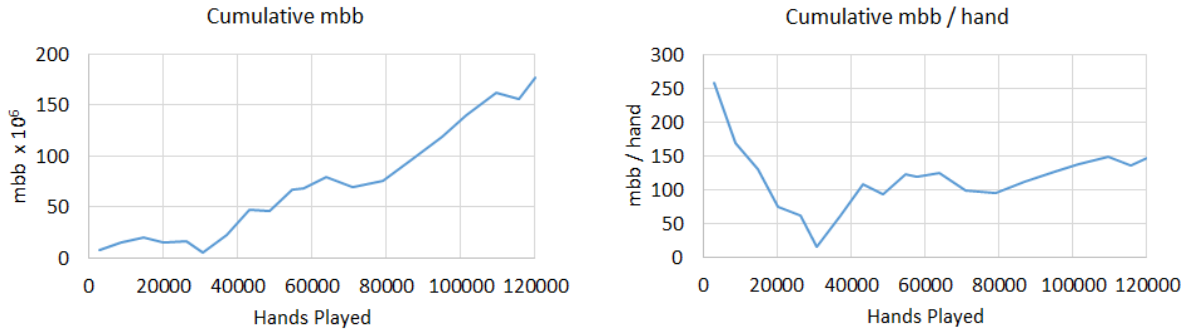


Figure 6.2: *Libratus*'s performance over the course of the 2017 *Brains vs AI* competition.

6.5 Modicum

Modicum was developed in 2018 to play heads-up no-limit Texas hold'em. While Modicum is substantially weaker than Libratus, it is far less expensive to run. Its training and operation requires only a 4-core CPU and 16GB of memory. We achieved this dramatic reduction in computational cost and memory usage by leveraging the depth-limited search techniques described in Section 5.2. In a sense, Modicum is the precursor to Pluribus, which went on to defeat elite human professionals in six-player no-limit Texas hold'em.

Modicum was shown to beat the benchmark bots Slumbot and Baby Tartanian8 with statistical significance, shown in Table 5.5. Modicum was also shown to be unexploitable by the local best response algorithm [103]. While Modicum was never tested against human professionals, we think it is likely that it plays at a superhuman level in HUNL.

As with Libratus and Pluribus, Modicum first trains a blueprint policy using MCCFR. It then conducts search at test time.

The blueprint abstraction treats every poker hand separately on the first betting round (where there are 169 strategically distinct hands). On the remaining betting rounds, the hands are grouped into 30,000 buckets using prior information abstraction techniques [24, 50, 82]. The action abstraction was chosen primarily by observing the most common actions used by prior top agents. We made a conscious effort to avoid actions that would likely not be in Baby Tartanian8's and Slumbot's action abstraction, so that we do not actively exploit their use of action translation. This makes our experimental results relatively conservative.

We used unsafe nested search on the first and second betting rounds, as well as for the first subgame on the third betting round. Unsafe search is described in Section 5.1.2, and nested search is described in Section 5.1.7. Unsafe search lacks theoretical guarantees because the opponent need not play according to the specific equilibrium we compute, and may actively exploit our assumption that they are playing according to a specific strategy. Nevertheless, in

practice unsafe search typically achieves strong performance and exhibits low exploitability in poker, particularly in large games [20].

Since the first betting round (called the **preflop**) is extremely small, whenever the opponent takes an action that we have not previously observed, we add it to the action abstraction for the preflop, solve the whole preflop again, and cache the solution. When the opponent chooses an action that they have taken in the past, we simply load the cached solution rather than solve the subgame again. This results in the preflop taking a negligible amount of time on average.

To determine the values of leaf nodes on the first and second betting round, whenever a subgame was constructed we mapped each leaf node in the subgame to a leaf node in the blueprint abstraction (based on similarity of the action sequence). The values of a leaf node in the subgame (as a fraction of the pot) were set to its corresponding blueprint abstraction leaf node. In the case of rollouts, this meant conducting rollouts in the blueprint strategy starting at the blueprint leaf node.

As explained in Section 5.2, we tried two methods for determining state values at the end of the second betting round. The first method involves storing the four opponent approximate best responses and doing rollouts in real time whenever the depth limit is reached. The second involves training a deep neural network (DNN) to predict the state values determined by the four approximate best responses.

For the rollout method, it is not necessary to store the best responses as 4-byte floats. That would use $32|A|$ bits per abstract infoset, where $|A|$ is the number of actions in an infoset. If one is constrained by memory, an option is to randomize over the actions in an abstract infoset ahead of time and pick a single action. That single action can then be stored using a minimal number of bits. This means using only $\lceil log_2(|A|) \rceil$ bits per infoset. This comes at a slight cost of precision, particularly if the strategy is small, because it would mean always picking the same action in an infoset whenever it is sampled. Since we were not severely memory constrained, we instead stored the approximate best responses using a single byte per abstract infoset action. In order to reduce variance and converge more quickly, we conduct multiple rollouts upon reaching a leaf node. We found the optimal number of rollouts to be three given our memory access speeds.

For the DNN approach, whenever a subgame on the second round is generated we evaluate each leaf node using the DNN before solving begins. The state values are stored (using about 50 MB). This takes between 5 and 10 seconds depending on the size of the subgame.

Starting on the third betting round, we always solve to the end of the game using an algorithm similar to Discounted CFR (Discounted CFR was not yet developed). We use unsafe search the first time the third betting round is reached. Subsequent subgames are solved using safe nested search (specifically, Reach search (described in Section 5.1.3) where the alternative payoffs are based on the expected value from the previously-solved subgame [20]).

Our CFR variant ignores the first 50% of iterations when determining the average strategy. Also, for the first 30 iterations, it discounts the regrets after each iteration by $\frac{\sqrt{T}}{\sqrt{T+1}}$ where T indicates the iteration. This reduces exploitability in the subgame by about a factor of three compared to CFR+.

The number of CFR iterations and the amount of time we run MCCFR varies depending on the size of the pot. For the preflop, we always run MCCFR for 30 seconds to solve a subgame (though this is rarely done due to caching). On the flop, we run MCCFR for 10 to 30 seconds

depending on the pot size. On the turn, we run between 150 and 1,000 iterations of our modified form CFR. On the river, we run between 300 and 2,000 iterations of our modified form of CFR.

6.6 Pluribus

Pluribus [23] was the first bot to defeat elite human professionals in multiplayer (i.e., more than two-player) poker. Specifically, it defeated a set of 15 elite human professionals in six-player no-limit Texas hold'em poker. Alongside Libratus's victory in two-player no-limit Texas hold'em, this is considered a major milestone achievement in the field of AI.

Multiplayer poker posed several difficult challenges beyond HUNL. First, since the game is not two-player zero-sum, there were theoretical questions of what should even be computed. Two-player zero-sum games are a special class of games in which Nash equilibria also have an extremely useful additional property: any player who chooses to use a Nash equilibrium is guaranteed to not lose in expectation no matter what the opponent does (as long as one side does not have an intrinsic advantage under the game rules, or the players alternate sides). In other words, a Nash equilibrium strategy is unbeatable in two-player zero-sum games. For this reason, to "solve" a two-player zero-sum game means to find an exact Nash equilibrium.

While a Nash equilibrium strategy is guaranteed to exist in any finite game, efficient algorithms for finding one are only proven to exist for special classes of games, among which two-player zero-sum games are the most prominent. For multiplayer or general-sum games, computing a Nash equilibrium is PPAD-complete [39, 41]. Even approximating a Nash equilibrium is difficult (except in special cases) in theory [124].

Moreover, even if a Nash equilibrium could be computed efficiently in a game with more than two players, it is not clear that playing such an equilibrium strategy would be wise. If each player in such a game independently computes and plays a Nash equilibrium, the joint strategy that they play (one strategy per player) may not be a Nash equilibrium and players might have an incentive to deviate to a different strategy. Two-player zero-sum games are a special case where even if the players independently compute and select Nash equilibria, the joint strategy is still a Nash equilibrium.

The shortcomings of Nash equilibria outside of two-player zero-sum games, and the failure of any other game-theoretic solution concept to convincingly overcome them, have raised the question of what the right goal should even be in such games. In the case of six-player poker, we took the viewpoint that our goal should not be a specific game-theoretic solution concept, but rather to create an AI that empirically consistently defeats human opponents, including elite human professionals.

Beyond just the game theory challenges, six-player NLTH is also much larger than HUNL. For example, if the same search algorithms that were used in Libratus were used in Pluribus, they would have required roughly 100,000$\times$ as much memory and time.

Despite the massive increase in the size of the game, Pluribus used far fewer resources for both training and search than Libratus. The blueprint strategy for Pluribus was computed in 8 days on a 64-core server for a total of 12,400 CPU core hours. It required less than 512 GB of memory. At cloud computing spot instance rates in 2019, this would have cost about $144 to train. This is in sharp contrast to all the other recent superhuman AI milestones for games,

which used large numbers of servers and/or farms of GPUs. When conducting real-time search, Pluribus used a 28-core CPU and no more than 128 GB of memory.

Like Libratus, Pluribus used a blueprint strategy that was trained offline using a variant of MCCFR. This blueprint strategy was improved upon at test time using depth-limited search. The search technique was based on the search algorithm described in Section 5.2. Libratus's self-improvement component was not used in Pluribus, though using it likely would have led to even better performance.

Depth-limited search

Depth-limited search was the most important improvement that made Pluribus's victory in six-player poker possible. It reduces the computational resources and memory needed by probably at least five orders of magnitude. Libratus [21] always solved to the end of the game when real-time search was used (Libratus started using real-time search on the third betting round—the half-way point in the game—and in certain situations even earlier). However, additional players increase the size of subgames exponentially. Conducting beneficial real-time search on the flop (the second betting round) with more than three players involved is likely infeasible without depth-limited search. Pluribus used a depth-limited search technique similar to Modicum, but it was generalized to more than two players and it included a number of additional technical contributions as follows.

- Modicum's depth-limited search had the searcher play the blueprint strategy while the opponent chose among multiple continuation strategies. This is theoretically sound (in two-player zero-sum games), but in practice gives the opponents more power and therefore makes the searcher relatively defensive/conservative. Pluribus addressed this weakness by having the searcher also choose among continuous strategies (which is still theoretically sound in two-player zero-sum games). Modicum instead penalized the opponent to try to balance the players, but our new approach is more effective, easier, and more elegant.
- Previous nested search algorithms either used nested safe search or nested unsafe search (described in detail in Section 5.1). Nested unsafe search is not theoretically sound, and in some cases may do extremely poorly, but on average tends to perform better in practice. For Pluribus we used a new form of nested unsafe search in which we always solve starting at the beginning of the current betting round rather than starting at the most recent decision point. Our player's strategy is held fixed for actions that it has already chosen in the betting round. However, the approach allows the other players to have changed strategies anywhere in the current betting round, that is, even above the current decision point. Taking that possibility into account mitigates the potential for unsafe search to be exploitable, while still retaining its practical benefits.

Equilibrium finding

Pluribus also has innovations in the equilibrium finding algorithms that were used in the blueprint computation and within the depth-limited search. We summarize them here.

- We used Linear MCCFR rather than traditional MCCFR. This was the first time Linear MCCFR was implemented and tested at scale. We suspect this sped up convergence by

about a factor of 3.

- Our Linear MCCFR algorithm used a form of dynamic pruning that skipped actions with extremely negative regret in 95% of iterations. A similar idea was used in Libratus and in Baby Tartanian8, but in those cases the skipping was done everywhere. In contrast, in Pluribus, in order to reduce the potential inaccuracies involved with pruning, we do not skip actions on the last betting round (because on the last betting round one does not get the benefits of effectively increasing the card abstraction through pruning) or actions leading to terminal payoffs (because the cost of examining those payoffs is minor anyway). Additionally, whether or not to skip in Libratus/BabyTartanian8 was decided separately for each action rather than deciding for the entire iteration; the latter is cheaper due to fewer calls to a random number generator, so we do the latter in Pluribus. We suspect that these changes contributed about a factor of 2 speedup.

Memory usage

To conserve memory, Pluribus allocated memory for the regrets in an action sequence only when the sequence was encountered for the first time (except on the first betting round which is small and allocated up front). This is particularly useful in six-player poker, in which there are many action sequences that never occur. This reduced memory usage by more than a factor of 2.

Experimental evaluation

We evaluated Pluribus against elite human professionals in two formats: five human professionals playing with one copy of Pluribus (5H+1AI), and one human professional playing with five copies of Pluribus (1H+5AI). All human participants have won more than $1 million playing poker professionally.

In all experiments, we used the variance-reduction technique AIVAT [32] to reduce the luck factor in the game[6] and measured statistical significance at the 95% confidence level using a one-tailed t-test for whether Pluribus is profitable.

The human participants in the 5H+1AI experiment were Jimmy Chou, Seth Davies, Michael Gagliano, Anthony Gregg, Dong Kim, Jason Les, Linus Loeliger, Daniel McCaulay, Greg Merson, Nicholas Petrangelo, Sean Ruane, Trevor Savage, and Jacob Toole. In this experiment, 10,000 hands of poker were played over 12 days. Each day, five volunteers from the pool of professionals were selected to participate based on availability. The participants were not told who else was participating in the experiment. Instead, each participant was assigned an alias that remained constant throughout the experiment. The alias of each player in each game was known, so that players could track the tendencies of each player throughout the experiment. $50,000 was divided among the human participants based on their performance to incentivize them to play their best. Each player was guaranteed a minimum of $0.40 per hand for participating, but this could increase to as much as $1.60 per hand based on performance. After applying AIVAT, Pluribus won an average of 48 mbb/game (with a standard error of 25 mbb/game). This is

[6]Due to the presence of AIVAT and because the players did not know each others' scores during the experiment, there was no incentive for the players to play a risk-averse or risk-seeking strategy in order to outperform the other human.

considered a very high win rate in six-player no-limit Texas hold'em poker, especially against a collection of elite professionals, and implies that Pluribus is much better than the human players. Pluribus was determined to be profitable with a p-value of 0.028, which is statistically significant. The performance of Pluribus over the course of the experiment is shown in Figure 6.3.

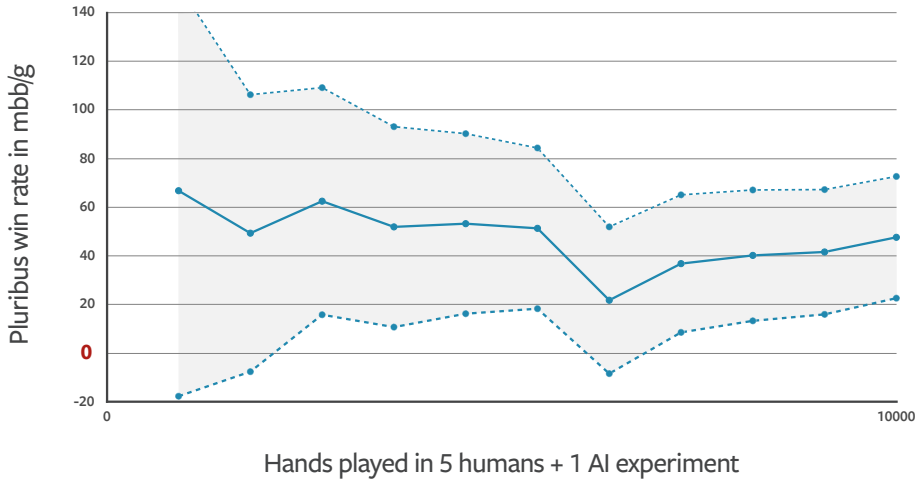


Figure 6.3: Performance of Pluribus in the 5 humans + 1 AI experiment. The dotted lines show the win rate plus or minus the standard error. The relatively steady performance of Pluribus over the course of the 10,000-hand experiment suggests the humans were unable to find exploitable weaknesses in the bot.

The human participants in the 1H+5AI experiment were Chris "Jesus" Ferguson and Darren Elias. Each of the two humans separately played 5,000 hands of poker against five copies of Pluribus. Pluribus does not adapt its strategy to its opponents and does not know the identity of its opponents, so the copies of Pluribus could not intentionally collude against the human player. To incentivize strong play, we offered each human $2,000 for participation and an additional $2,000 if he performed better against the AI than the other human player did. The players did not know who the other participant was and were not told how the other human was performing during the experiment. For the 10,000 hands played, Pluribus beat the humans by an average of 32 mbb/game (with a standard error of 15 mbb/game). Pluribus was determined to be profitable with a p-value of 0.014, which is statistically significant. (Darren Elias was behind Pluribus by 40 mbb/game with a standard error of 22 mbb/game and a p-value of 0.033, and Chris Ferguson was behind Pluribus by 25 mbb/game with a standard error of 20 mbb/game and a p-value of 0.107. Ferguson's lower loss rate may be due to variance, skill, and/or the fact that he used a more conservative strategy that was biased toward folding in unfamiliar difficult situations.)

Since Pluribus's strategy was determined entirely from self-play without any human data, it also provides an outside perspective on what optimal play should look like in multi-player no-limit Texas hold'em. Pluribus confirms the conventional human wisdom that limping (calling the

"big blind" rather than folding or raising) is suboptimal for any player except the "small blind" player who already has half the big blind in the pot by the rules, and thus has to invest only half as much as the other players to call. While Pluribus initially experimented with limping when computing its blueprint strategy offline through self play, it gradually discarded this action from its strategy as self play continued. However, Pluribus disagrees with the folk wisdom that "donk betting" (starting a round by betting when one ended the previous betting round with a call) is a mistake; Pluribus does this far more often than professional humans do.

6.7 ReBeL

The ReBeL algorithm (described in Section 5.3) combines deep reinforcement learning and search (RL+Search) at both training and test time. Unlike prior RL+Search algorithms such as AlphaZero [142], which are intended for perfect-information games, ReBeL converges to a Nash equilibrium in two-player zero-sum imperfect-information games in addition to solving perfect-information games and single-agent settings.

We evaluated the ReBeL algorithm in part by developing a bot for HUNL poker (which we also call ReBeL). Our goal was not to develop the strongest possible poker AI agent, and indeed ReBeL is not as strong as Libratus (described in Section 6.4) and is far more expensive than Modicum (described in Section 6.5). Instead, our goal for ReBeL was to develop a relatively simple, flexible algorithm that leverages as little domain knowledge as possible and still achieves superhuman performance. Indeed, the bot achieved very strong results against prior benchmark bots and even defeated a top human professional by a wide margin (see Table 5.6 in Section 5.3.5). To show that the algorithm can be easily deployed in games other than poker, we applied it to the imperfect-information game of Liar's Dice and showed that it achieves excellent performance in that domain as well.

The most prominent form of domain knowledge in the ReBeL poker AI agent is the simplification of the action space during self play so that there are at most 8 actions at each decision point. The bet sizes are hand-chosen based on conventional poker wisdom and are fixed fractions of the pot, though each bet size is perturbed by $\pm 0.1\times$ pot during training to ensure diversity in the training data.

We specifically chose not to leverage domain knowledge that has been widely used in previous poker AI agents:

- All prior top poker agents, including DeepStack [110], Libratus [21], and Pluribus [23], have used *information abstraction* to bucket similar infostates together based on domain-specific features [24, 50, 80]. Even when computing an exact policy, such as during search or when solving a poker game in its entirety [12, 56], past agents have used *lossless abstraction* in which strategically identical infostates are bucketed together. For example, a flush of spades may be strategically identical to a flush of hearts.

  Our agent does not use any information abstraction, whether lossy or lossless. The agent computes a unique policy for each infostate. The agent's input to its value and policy network is a probability distribution over pairs of cards for each player, as well as all public board cards, the amount of money in the pot relative to the stacks of the players, and a flag for whether a bet has occurred on this betting round yet.

- DeepStack trained its value network on random PBSs. In order to make this tractable, DeepStack reduced the dimensionality of its value network input by using information abstraction and sampled PBSs according to a handcrafted algorithm that would sample more realistic PBSs compared to sampling uniform random. We show in Section 5.3.5 that training on PBSs sampled uniformly randomly without information abstraction results in extremely poor performance in a value network.

  Our agent collects training data purely from self play without any additional heuristics guiding which PBSs are sampled, other than an exploration hyperparameter that was set to $\epsilon = 0.25$ in all experiments.

- Past no-limit poker bots were designed specifically to play with 200 big blinds (20,000 chips). This was the standard format in the Annual Computer Poker Competition. In order to run an agent like Libratus effectively with a different number of chips, a separate blueprint strategy would need to be computed. ReBeL, in contrast, is trained to handle any number of chips between 5,000 and 25,000.

- In cases where both players bet all their chips before all board cards are revealed, past poker AIs compute the exact expected value of all possible remaining board card outcomes. This is expensive to do in real time on earlier rounds, so past agents pre-compute this expected value and look it up during training and testing. Using the exact expected value reduce variance and makes learning easier.

  Our agent does not use this shortcut. Instead, the agent learns these "all-in" expected values on its own. When both agents have bet all their chips, the game proceeds as normal except neither player is allowed to bet.

- The search space in DeepStack [110] extends to the start of the next betting round, except for the third betting round (out of four) where it instead extends to the end of the game. Searching to the end of the game on the third betting round was made tractable by using information abstraction on the fourth betting round (see above). Similarly, Libratus [20], Modicum [27], and Pluribus [23] all search to the end of the game when on the third betting round. Searching to the end of the game has the major benefit of not requiring the value network to learn values for the end of the third betting round. Thus, instead of the game being three "levels" deep, it is only two levels deep. This reduces the potential for propogation of errors.

  Our agent always solves to the end of the current betting round, regardless of which round it is on.

- The depth-limited subgames in DeepStack extended to the start of the next betting round on the second betting round. On the first betting round, it extended to the end of the first betting round for most of training and to the start of the next betting round for the last several CFR iterations. Searching to the start of the next betting round was only tractable due to the abstractions mentioned previously and due to careful optimizations, such as implementing CFR on a GPU.

  Our agent always solves to the end of the current betting round regardless of which round it is on. We implement CFR only on a single-thread CPU and avoid any abstractions. Since a subgame starts at the beginning of a betting round and ends at the start of the next

betting round, our agent must learn six "layers" of values (end of first round, start of second round, end of second round, start of third round, end of third round, start of fourth round) compared to three for DeepStack (end of first round, start of second round, start of third round).

- DeepStack used a separate value network for each of the three "layers" of values (end of first round, start of second round, start of third round). Our agent uses a single value network for all situations.

Chapter 7

Conclusions and Future Research

This thesis described advances in equilibrium finding for large adversarial imperfect-information games along three tracks: improvements to counterfactual regret minimization (CFR), new forms of abstraction and ways of combining function approximation with CFR, and finally new search techniques for imperfect-information games. Together, these improvements for the first time enabled an AI to defeat top human professionals in both two-player no-limit Texas hold'em poker and multiplayer no-limit Texas hold'em poker, which were considered grand challenges for the fields of AI and game theory. However, poker is only a benchmark for progress in this field. The techniques themselves are domain-independent and can be deployed in adversarial imperfect-information games beyond just poker.

Chapter 3 introduced substantial improvements to CFR in three areas: discounting variants that achieve state-of-the-art performance among equilibrium-finding algorithms both with and without sampling, the first theoretically sound and empirically effective warm-starting technique, and pruning techniques that lead to more than an order of magnitude speedup in large games. However, there are still opportunities for large improvements.

While Discounted CFR (DCFR) and Linear CFR (LCFR) achieve remarkable speedups over CFR and CFR+, LCFR does relatively better in games with wide distributions in payoffs, while DCFR does better in other games. Surprisingly, combining the changes of both algorithms into a single algorithm leads to poor performance. One of the biggest unanswered questions in this space is whether there exists an algorithm that can achieve similar performance to LCFR in games with wide distributions in payoffs, while still being competitive with DCFR in other games.

Section 3.2 introduced the first theoretically sound method for warm starting CFR from arbitrary strategies. This warm starting technique provably maintains the convergence bound of CFR. However, in practice CFR converges much faster than its theoretical bound. While the warm start technique allows for a range of parameter values to be used for warm starting, it is unclear which choice of parameter values will lead to the best empirical performance. Developing a heuristic for selecting this parameter value, or proving a tighter bound on convergence for CFR, would allow the warm starting algorithm to be even more effective.

Another important open question is whether regret-based pruning or best-response pruning can be adapted to Monte Carlo variants of CFR in a theoretically sound way. Currently, in theory both pruning techniques require exact CFR iterations and are therefore incompatible with

sampling. While forms of regret-based pruning for Monte Carlo CFR were used in BabyTartanian8, Libratus, and Pluribus, which led to large speedups, its use was not theoretically sound. A provably sound form of pruning may lead to even larger improvements in algorithms with sampling.

Chapter 4 introduced new ways of conducting abstraction. Regret transfer and simultaneous abstraction and equilibrium-finding (SAEF) provide the first theoretically sound approaches for action abstraction, and make it possible to converge to a Nash equilibrium even in games with continuous action spaces. Deep CFR combines CFR with neural network function approximation and for the first time made it possible to use CFR in large games without tabular tracking of regrets and strategies. The recently introduced DREAM algorithm [146] builds upon Deep CFR and can be deployed even in settings without an explicit model of the environment.

However, there is still much work to be done along the path of scaling CFR to larger domains such as 3-dimensional continuous action space environments. A major obstacle is that existing CFR algorithms cannot cope with continuous action spaces. Developing such an algorithm would be a major step toward universal reinforcement learning algorithms that could be successful in large imperfect-information games as well as perfect-information games and single-agent settings.

Chapter 5 introduced new theoretically sound and empirically effective search techniques for imperfect-information games. Safe search techniques, which prevent the opponent from exploiting any assumption we might make about the opponent's strategy, have been known for many years. However, they performed poorly in practice compared to unsafe search techniques that lacked theoretical guarantees and for that reason the unsafe search was used either entirely or partially in competitive agents. Reach search and ReBeL are two major improvements to safe search that for the first time make safe search empirically competitive with unsafe search. Nested search, described in Section 5.1.7, presented a way for both safe and unsafe search techniques to be applied repeatedly as play proceeds down the game tree. Section 5.2 introduced depth-limited search via multi-valued states, which made depth-limited search far cheaper and more scalable than past approaches. Finally, ReBeL, introduced in Section 5.3, made it possible for the first time to combined depth-limited search and reinforcement learning at both training and test time in imperfect-information games in a theoretically sound way. This was a major step toward bridging the gap between the historically separate research threads of perfect-information games and imperfect-information games.

Perhaps the biggest remaining open research question related to search in two-player zero-sum imperfect-information games is how to apply search in a theoretically sound way in games that have little or no common knowledge. All existing safe search techniques for imperfect-information games operate by decomposing the game into pieces that are divided based on common knowledge and then solving just one of those pieces rather than the entire game. However, if there is no common knowledge, then existing search techniques view the whole game as a single atomic piece and those search techniques provide no benefit over simply solving the entire game offline. While a lack of common knowledge is relatively rare in recreational games, it is quite common in real-world applications, which makes this challenge important for the deployment of equilibrium-finding techniques in the real world. Still, humans are able to search in games even when there is not explicit common knowledge, which suggests there should be a path forward on this challenge.

Most importantly, there is a question of how to extend the techniques presented in this thesis to general-sum and multiplayer settings. While two-player zero-sum interactions are common among recreational games, they are very rare in the real world. Instead, real-world strategic interactions, such as business negotiations, politics, and traffic navigation, typically have a complex mixture of adversarial and cooperative elements. Pluribus demonstrated that the techniques in this thesis extend to multiplayer poker and likely to a broader class of multiplayer interactions, but there are certainly situations in which existing CFR variants and existing search techniques are not sufficient for computing an effective strategy.

Nevertheless, this thesis advances the state of the art for equilibrium-finding in large adversarial imperfect-information games. Just ten years ago, the prospect of developing AI algorithms capable of defeating top humans in a complex imperfect-information game like no-limit poker seemed to be a distant dream. Now those techniques, and much more, are a reality.

Bibliography

[1] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Neural Information Processing Systems (NIPS)*, pages 5360–5370, 2017. 5.3

[2] Robert Aumann. Agreeing to disagree. *The Annals of Statistics*, 4, 1976. 5.3.2

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 5.3.5, 5.3.10

[4] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749, 2005. 4.1.1

[5] Nolan Bard, Michael Johanson, and Michael Bowling. Asymmetric abstractions for adversarial settings. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 501–508, 2014. 6.3

[6] Kimmo Berg and Tuomas Sandholm. Exclusion method for finding nash equilibrium in multiplayer games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017. 2.2

[7] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 493–499, Madison, WI, 1998. 6.4

[8] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002. 1

[9] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003. 4.3, 5.1.2

[10] David Blackwell. An analog of the minmax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6:1–8, 1956. 3.4.5

[11] Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8(Jun):1307–1324, 2007. 2.3

[12] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, January 2015. 2.2, 2.4.2, 3.1, 3.1.4, 3.5, 5.3.5, 3, 6.7

[13] George W. Brown. Iterative solutions of games by fictitious play. In Tjalling C. Koopmans,

editor, *Activity Analysis of Production and Allocation*, pages 374–376. John Wiley & Sons, 1951. 4.1, 5.3.3, 5.3.8

[14] Noam Brown and Tuomas Sandholm. Regret transfer and parameter optimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 594–601, 2014. 3.1.2, 3.1.2, 3.4.5

[15] Noam Brown and Tuomas Sandholm. Regret-based pruning in extensive-form games. In *Neural Information Processing Systems (NIPS)*, pages 1972–1980, 2015. 3.1.2, 5.1.8

[16] Noam Brown and Tuomas Sandholm. Simultaneous abstraction and equilibrium finding in games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015. 3.1.2, 3.5, 5.1.7, 5.2.2, 12

[17] Noam Brown and Tuomas Sandholm. Baby Tartanian8: Winning agent from the 2016 annual computer poker competition. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4238–4239, 2016. (document), 5.1.8, 5.2.3, 5.3.5, 5.6, 6.3, 6.4, 6.4

[18] Noam Brown and Tuomas Sandholm. Strategy-based warm starting for regret minimization in games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 432–438, 2016. 3.5, 3.5.1

[19] Noam Brown and Tuomas Sandholm. Reduced space and faster convergence in imperfect-information games via pruning. In *International Conference on Machine Learning (ICML)*, 2017. 3.1.2, 3.5.1, 5.1.8

[20] Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *Neural Information Processing Systems (NIPS)*, pages 689–699, 2017. 3.1, 3.1.3, 5.2, 5.2.2, 12, 6.5, 6.7

[21] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733, 2017. 2.2, 3.1, 3.1.3, 3.1.4, 3.1.5, 5.2, 5.2.2, 5.3, 5.3.4, 5.3.5, 5.3.10, 6, 6.6, 6.7

[22] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. 4.1.3, 5.3.4, 5.3.5, 5.3.8

[23] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, page eaay2400, 2019. 5.2.4, 5.3, 5.3.1, 5.3.4, 5.3.5, 6, 6.6, 6.7

[24] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit texas hold'em agent. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 7–15, 2015. 4.1.2, 4.3, 6.1, 6.3, 6.3, 6.4, 6.5, 6.7

[25] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Tartanian7: A champion two-player no-limit Texas hold'em poker-playing program. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 4270–4271, 2015. Demo Track paper. 6.4, 6.4

[26] Noam Brown, Christian Kroer, and Tuomas Sandholm. Dynamic thresholding and pruning for regret minimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 421–

429, 2017. 5.1.6

[27] Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Neural Information Processing Systems (NeurIPS)*, pages 7663–7674, 2018. 3.1.3, 4.1, 5.3, 5.3.4, 5.3.5, 6.7

[28] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International Conference on Machine Learning (ICML)*, pages 793–802, 2019. 5.3.10

[29] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. In *Neural Information Processing Systems (NeurIPS)*, 2020. 5.2

[30] Neil Burch. *Time and Space: Why Imperfect Information Games are Hard*. PhD thesis, University of Alberta, 2017. 2.3.8, 3.1.5

[31] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 602–608, 2014. 4.3.1, 5.1.2, 5.2.1, 5.2.2, 5.2.5, 5.3.2, 12, 5.3.3, 5.3.9, 5.3.9, 5.3.11

[32] Neil Burch, Martin Schmid, Matej Moravcik, Dustin Morill, and Michael Bowling. AIVAT: A new variance reduction technique for agent evaluation in imperfect information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 5.2.3, 5.3.5, 6.6

[33] Neil Burch, Matej Moravcik, and Martin Schmid. Revisiting CFR+ and alternating updates. *Journal of Artificial Intelligence Research*, 64:429–443, 2019. 2.3.5

[34] Murray Campbell, A Joseph Hoane, and Feng-Hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002. 1, 2.2, 5.1, 5.1.9, 5.2

[35] Jiří Čermák, Viliam Lisỳ, and Branislav Bošanskỳ. Automated construction of bounded-loss imperfect-recall abstractions in extensive-form games. *Artificial Intelligence*, 282: 103248, 2020. 4.3.7, 5.2.2

[36] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006. 2.3.1, 2.3.1, 2.3.3, 3.1.2, 3.4.3, 3.4.5, 3.4.5, 3.4.5, 4.2.8

[37] Nicolo Cesa-Bianchi, Yishay Mansour, and Gilles Stoltz. Improved second-order bounds for prediction with expert advice. *Machine Learning*, 66(2-3):321–352, 2007. 2.3.3

[38] Kamalika Chaudhuri, Yoav Freund, and Daniel J Hsu. A parameter-free hedging algorithm. In *Neural Information Processing Systems (NIPS)*, pages 297–305, 2009. 2.3.1, 3.4.3

[39] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 2009. 2.2, 6.6

[40] Chao-Kai Chiang, Tianbao Yang, Chia-Jung Lee, Mehrdad Mahdavi, Chi-Jen Lu, Rong Jin, and Shenghuo Zhu. Online optimization with gradual variations. In *Conference on Learning Theory*, pages 6–1, 2012. 5.3.8

[41] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1), 2009. 2.2,

6.6

[42] Constantinos Daskalakis, Alan Deckelbaum, and Anthony Kim. Near-optimal no-regret algorithms for zero-sum games. *Games and Economic Behavior*, 92:327–348, 2015. 3.2.6, 3.4.4

[43] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. Optimally solving dec-POMDPs as continuous-state MDPs. *Journal of Artificial Intelligence Research*, 55:443–497, 2016. 5.3

[44] Gabriele Farina, Christian Kroer, Noam Brown, and Tuomas Sandholm. Stable-predictive optimistic counterfactual regret minimization. In *International Conference on Machine Learning*, pages 1853–1862, 2019. 3.5.2

[45] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Online convex optimization for sequential decision processes and extensive-form games. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 1917–1925, 2019. 2.3.5

[46] Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1942–1951, 2019. 5.3

[47] Yoav Freund and Robert 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. 3.4

[48] Sam Ganzfried and Tuomas Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 533–540, 2011. 2.2, 6.4

[49] Sam Ganzfried and Tuomas Sandholm. Action translation in extensive-form games with large action spaces: axioms, paradoxes, and the pseudo-harmonic mapping. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 120–128, 2013. 4.3.3, 5.1.7, 5.2.3, 6.4, 6.4

[50] Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 682–690. AAAI Press, 2014. 4.1.2, 4.3, 5.1.8, 5.2.2, 5.2.3, 6.5, 6.7

[51] Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 37–45, 2015. 5.1.2, 5.2.1

[52] Sam Ganzfried and Tuomas Sandholm. Safe opponent exploitation. *ACM Transaction on Economics and Computation (TEAC)*, 3(2):8:1–28, 2015. Best of EC-12 special issue. 2.2, 6.4

[53] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *International Conference on Machine learning (ICML)*, pages 273–280, 2007. 5.3.2

[54] Richard Gibson. *Regret Minimization in Games and the Development of Champion Mul-*

tiplayer Computer Poker-Playing Agents. PhD thesis, University of Alberta, 2014. 4.3.1

[55] Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, and Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1355–1361, 2012. 2.3.6, 3.1.5

[56] Andrew Gilpin and Tuomas Sandholm. Optimal Rhode Island hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1684–1685, Pittsburgh, PA, 2005. AAAI Press / The MIT Press. Intelligent Systems Demonstration. 2.3.8, 4, 6.7

[57] Andrew Gilpin and Tuomas Sandholm. A competitive Texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1007–1013, 2006. 4.3, 5.1.2

[58] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas hold'em poker. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1168–1175, 2007. 5.1.2

[59] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM*, 54(5), 2007. 4.3

[60] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. A heads-up no-limit Texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 911–918, 2008. 5.1.7, 6.4

[61] Andrew Gilpin, Javier Peña, and Tuomas Sandholm. First-order algorithm with $\mathcal{O}(\ln(1/\epsilon))$ convergence for ϵ-equilibrium in two-person zero-sum games. *Mathematical Programming*, 133(1–2):279–298, 2012. Conference version appeared in AAAI-08. 3.4, 5.1.2

[62] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957. 2.2

[63] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004. 5.3.1

[64] Peter E Hart, Nils J Nilsson, and Bertram Raphael. Correction to "a formal basis for the heuristic determination of minimum cost paths". *ACM SIGART Bulletin*, pages 28–29, 1972. 5.2

[65] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000. 2.3.1, 2.3.5, 5.3.11

[66] Junichi Hashimoto, Akihiro Kishimoto, Kazuki Yoshizoe, and Kokolo Ikeda. Accelerated UCT and its application to two-player games. In *Advances in computer games*, pages 1–12. Springer, 2011. 3.1.2

[67] John Hawkin, Robert Holte, and Duane Szafron. Automated action abstraction of imperfect information extensive-form games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 681–687, 2011. 4.2, 4.2.7, 4.3

[68] John Hawkin, Robert Holte, and Duane Szafron. Using sliding windows to generate action abstractions in extensive-form games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1924–1930, 2012. 4.2, 4.2.7, 4.3

[69] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016. 4.1, 2, 5.2.2

[70] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning (ICML)*, pages 805–813, 2015. 3.5, 3.5.5

[71] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 5.3.5, 5.3.10

[72] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010. Conference version appeared in WINE-07. 2.3.8, 3.2.6, 3.4, 3.4.3, 3.5.2, 3.5.5, 5.3.3

[73] Karel Horák and Branislav Bošanskỳ. Solving partially observable stochastic games with public observations. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 2029–2036, 2019. 5.3

[74] Eric Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2014. 4.3.1, 2, 5.2.2

[75] Eric Griffin Jackson. Compact CFR. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2016. 4.1.1

[76] Eric Griffin Jackson. Targeted CFR. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2017. 2.3.6, 3.1.5, 5.2.3

[77] Peter H Jin, Sergey Levine, and Kurt Keutzer. Regret minimization for partially observable deep reinforcement learning. *arXiv preprint arXiv:1710.11424*, 2017. 4.1

[78] Michael Johanson. Measuring the size of large no-limit poker games. Technical report, University of Alberta, 2013. 4.3

[79] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 258–265, 2011. 3.1.5, 3.2, 3.2.5, 3.3.7, 3.5.5, 4.3.4

[80] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1371–1379. AAAI Press, 2012. 3.2, 3.3.1, 5.1.5, 6, 5.1.12, 5.2.2, 6.4, 6.7

[81] Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 837–846, 2012. 2.3.6

[82] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. In *International Conference on Au-*

tonomous Agents and Multiagent Systems (AAMAS), pages 271–278, 2013. 4.1.2, 4.2.6, 4.3, 5.2.3, 6.5

[83] Michael Bradley Johanson. *Robust Strategies and Counter-Strategies: From Superhuman to Optimal Play*. PhD thesis, University of Alberta, 2016. 4.1.3

[84] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4.1.2, 5.2.3, 5.3.5

[85] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, July 1997. 2.3.8

[86] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996. 2.3.8

[87] Vojtěch Kovařík and Viliam Lisỳ. Problems with the EFG formalism: a solution attempt using observations. *arXiv preprint arXiv:1906.06291*, 2019. 12

[88] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisỳ. Rethinking formal models of partially observable multiagent decision making. *arXiv preprint arXiv:1906.11110*, 2019. 5.3.1, 9, 12

[89] Christian Kroer and Tuomas Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 621–638. ACM, 2014. 4.3

[90] Christian Kroer, Kevin Waugh, Fatma Kılınç-Karzan, and Tuomas Sandholm. Faster first-order methods for extensive-form game solving. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 817–834. ACM, 2015. 2.3.8, 3.2.6, 3.4

[91] Christian Kroer, Kevin Waugh, Fatma Kılınç-Karzan, and Tuomas Sandholm. Theoretical and practical advances on smoothing for extensive-form games. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2017. 5.1.2

[92] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Solving large sequential games with the excessive gap technique. In *Neural Information Processing Systems (NeurIPS*, pages 864–874, 2018. 2.3.8, 5.3.3

[93] Christian Kroer, Kevin Waugh, Fatma Kılınç-Karzan, and Tuomas Sandholm. Faster algorithms for extensive-form game solving via improved smoothing functions. *Mathematical Programming Series A*, 2020. 2.3.8, 3.4.3, 3.4.4, 3.5.2, 5.3.3

[94] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017. 1

[95] Marc Lanctot. *Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games*. PhD thesis, University of Alberta, University of Alberta, Computing Science, 116 St. and 85 Ave., Edmonton, Alberta T6G 2R3, June 2013. 2.3.6, 2, 15, 4.1.5, 4.1.5, 4.1.5, 4.1.5, 4.1.5

[96] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Neural Information Processing Systems (NeurIPS)*, pages 1078–1086, 2009. 2.3.6, 2.3.6, 3.1.5, 3.2.5, 3.3, 3.4.5, 4.1.1, 4.1.3,

4.1.5, 4.3.8, 4.3.8, 5.1.8, 5.2.2, 5.2.3

[97] Marc Lanctot, Richard Gibson, Neil Burch, Martin Zinkevich, and Michael Bowling. No-regret learning in extensive-form games with imperfect recall. In *International Conference on Machine Learning (ICML)*, pages 65–72, 2012. 4.3

[98] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Neural Information Processing Systems (NIPS)*, pages 4190–4203, 2017. 2.2, 5.2.2

[99] Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020. 5.3

[100] David S Leslie and Edmund J Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006. 5.3.8

[101] Hui Li, Kailiang Hu, Shaohua Zhang, Yuan Qi, and Le Song. Double neural counterfactual regret minimization. In *International Conference on Learning Representations (ICLR)*, 2019. 4.1

[102] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965. 5.2

[103] Viliam Lisy and Michael Bowling. Equilibrium approximation quality of current no-limit poker bots. In *AAAI Workshop on Computer Poker and Imperfect Information Games*, 2017. (document), 5.2.3, 5.3.5, 5.6, 6.5

[104] Nick Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. 3.4, 5.1.5, 5.1.6

[105] Haipeng Luo and Robert E Schapire. A drifting-games analysis for online learning and applications to boosting. In *Neural Information Processing Systems (NIPS)*, pages 1368–1376, 2014. 3.4

[106] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning (ICML)*, pages 536–543, 2003. 5.2.2

[107] 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, 2015. 4.1

[108] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016. 4.1.1

[109] Matej Moravcik, Martin Schmid, Karel Ha, Milan Hladik, and Stephen J Gaukrodger. Refining subgames in large imperfect information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 572–578. AAAI Press, 2016. 5.1.2, 5.1.2, 5.1.3, 5.1.10, 5.2.2,

12

[110] 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. 3.1, 3.1.3, 3.1.4, 4.1, 7, 5.2, 5.2.2, 5.3, 5.3.2, 12, 5.3.3, 5.3.4, 5.3.4, 5.3.5, 5.3.9, 5.3.9, 3, 6.7

[111] Dustin R Morrill. Using regret estimation to solve games compactly. Master's thesis, University of Alberta, 2016. 4.1.5

[112] John Nash. Non-cooperative games. *Annals of Mathematics*, 54:289–295, 1951. 1, 2.2, 6

[113] Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. Decentralized stochastic control with partial history sharing: A common information approach. *IEEE Transactions on Automatic Control*, 58(7):1644–1658, 2013. 5.3

[114] Yurii Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005. 3.2.6, 3.4, 3.4.3, 5.1.2

[115] Allen Newell and George Ernst. The search for generality. In *Proc. IFIP Congress*, volume 65, pages 17–24, 1965. 5.2

[116] Andrew J Newman, Casey L Richardson, Sean M Kain, Paul G Stankiewicz, Paul R Guseman, Blake A Schreurs, and Jeffrey A Dunne. Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management. In *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, volume 9842, page 984209. International Society for Optics and Photonics, 2016. 5.3.6

[117] Nils J Nilsson. Problem-solving methods in. *Artificial Intelligence*, 1971. 5.2

[118] Frans Adriaan Oliehoek. Sufficient plan-time statistics for decentralized pomdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013. 5.3, 10

[119] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994. 2.1

[120] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019. 5.2.3, 5.3.5

[121] François Pays. An interior point approach to large games of incomplete information. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2014. 3.4, 3.4.4

[122] Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. In *Conference on Learning Theory*, pages 993–1019, 2013. 5.3.8

[123] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962. 2.3.8

[124] Aviad Rubinstein. Inapproximability of nash equilibrium. *SIAM Journal on Computing*, 47(3):917–959, 2018. 2.2, 6.6

[125] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959. 5.2

[126] Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Magazine*, pages 13–32, Winter 2010. Special issue on Algorithmic Game Theory. 5

[127] Tuomas Sandholm. Abstraction for solving large incomplete-information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 4127–4131, 2015. Senior Member Track. 5

[128] Tuomas Sandholm. Solving imperfect-information games. *Science*, 347(6218):122–123, 2015. 5

[129] Tuomas Sandholm and Satinder Singh. Lossy stochastic game abstraction with bounds. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 880–897. ACM, 2012. 4.2, 4.2.7, 4.3

[130] Jonathan Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, New York, 1997. 1, 2.2

[131] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007. 5.1

[132] Martin Schmid, Matej Moravcik, and Milan Hladik. Bounding the support size in extensive form games with imperfect information. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 784–790, 2014. 3.5

[133] David Schnizlein, Michael Bowling, and Duane Szafron. Probabilistic state translation in extensive games with large action sets. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 278–284, 2009. 5.1.7

[134] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019. 5.3

[135] Dominik Seitz, Vojtech Kovarík, Viliam Lisỳ, Jan Rudolf, Shuo Sun, and Karel Ha. Value functions for depth-limited solving in imperfect-information games beyond poker. *arXiv preprint arXiv:1906.06412*, 2019. 10, 12

[136] Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. Finding friend and foe in multi-agent games. In *Neural Information Processing Systems (NeurIPS)*, pages 1249–1259, 2019. 5.3, 5.3.4, 5.3.4

[137] Claude E Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950. 5.2

[138] A Shapiro and Y Wardi. Convergence analysis of gradient descent stochastic algorithms. *Journal of optimization theory and applications*, 91(2):439–454, 1996. 4.2.8

[139] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 333–345, London, UK, 2002. Springer-Verlag. 4.3

[140] 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. 1, 2.2, 5.1, 5.1.9, 5.2

[141] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017. 4.1, 5.2, 5.3

[142] 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. 4.1, 5.3, 6.7

[143] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, July 2005. 2.4.1, 3.3.5

[144] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *Neural Information Processing Systems (NeurIPS)*, pages 3426–3439, 2018. 4.1

[145] Eric Steinberger. Single deep counterfactual regret minimization. *arXiv preprint arXiv:1901.07621*, 2019. 4.1.1

[146] Eric Steinberger, Adam Lerer, and Noam Brown. DREAM: Deep regret minimization with advantage baselines and model-free learning. *arXiv preprint arXiv:2006.10410*, 2020. 4.1, 7

[147] Michal Šustr, Vojtěch Kovařík, and Viliam Lisỳ. Monte carlo continual resolving for online strategy computation in imperfect information games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 224–232, 2019. 12

[148] Vasilis Syrgkanis, Alekh Agarwal, Haipeng Luo, and Robert E Schapire. Fast convergence of regularized learning in games. In *Neural Information Processing Systems (NIPS)*, pages 2989–2997, 2015. 3.1.2, 5.3.8

[149] Oskari Tammelin. Solving large imperfect information games using CFR+. *arXiv preprint arXiv:1407.5042*, 2014. 2.3.2, 3.1, 3.3.4, 3

[150] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 645–652, 2015. 2.3.2, 2.3.7, 3.1, 3.1.7, 3.1.7, 3.1.7, 3.2, 3.5, 4, 5.1.8, 5.1.8, 5.2.3

[151] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994. 5.3

[152] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995. 1, 5.1.9

[153] Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199, 2002. 5.2

[154] Ben Van der Genugten. A weakened form of fictitious play in two-person zero-sum games. *International Game Theory Review*, 2(04):307–328, 2000. 5.3.8

[155] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 4.1.1

[156] John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100: 295–320, 1928. 1, 6

[157] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996. 2.3.8

[158] Kevin Waugh and Drew Bagnell. A unified view of large-scale zero-sum equilibrium computation. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2015. 3.2.6

[159] Kevin Waugh, Nolan Bard, and Michael Bowling. Strategy grafting in extensive games. In *Neural Information Processing Systems (NeurIPS)*, 2009. 4.3.1, 2

[160] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 781–788, 2009. 2.3.4, 4.3.4

[161] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael Bowling. A practical use of imperfect recall. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 2009. 6.4

[162] Kevin Waugh, Dustin Morrill, Drew Bagnell, and Michael Bowling. Solving games with functional regret estimation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015. 4.1

[163] Martin Zinkevich, Michael Johanson, Michael H Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Neural Information Processing Systems (NeurIPS)*, pages 1729–1736, 2007. 1, 2.3.5, 4.2.8, 4.2.8, 5.1.2, 5.1.6, 5.3.3, 5.3.11

[164] Martin A Zinkevich, Michael Bowling, and Michael Wunder. The lemonade stand game competition: solving unsolvable games. *ACM SIGecom Exchanges*, 10(1):35–38, 2011. 2.2