

**FACULTY  
OF MATHEMATICS  
AND PHYSICS**  
Charles University

**DOCTORAL THESIS**

Matej Moravčík

**Bridging the Gap: Towards Unified Approach to  
Perfect and Imperfect Information Games**

Department of Applied Mathematics

Supervisor of the doctoral thesis: Prof. Mgr. Milan Hladík, Ph.D.

Study programme: Computer Science

Study branch: Discrete Models and Algorithms

Prague 2023

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....

signature of the author

Title: Bridging the Gap: Towards Unified Approach to Perfect and Imperfect Information Games

Author: Matej Moravčík

Department: Department of Applied Mathematics

Supervisor: Prof. Mgr. Milan Hladík, Ph.D., Department of Applied Mathematics

Abstract: From the onset of AI research, games have played an important part, serving as a benchmark for progress in artificial intelligence. Recent approaches using search in combination with learning from self-play have shown strong performance and the ability to generalize across a wide range of perfect information games. In contrast, the leading algorithms for imperfect information traditionally used a small, abstract version of a game and solved this abstraction in one go. This thesis introduces a chain of improvements for imperfect information algorithms that culminates in two significant milestones that helped bridge the gap between perfect and imperfect information games. The first milestone is DeepStack — the first agent that successfully used a combination of sound search and a learned value function in imperfect information games. This led to the first AI to achieve victory over human professional players in no-limit poker. The second milestone is Player of Games — a universal algorithm that can master both perfect and imperfect information games starting from scratch.

Keywords: game theory, search, imperfect information, games, DeepStack, Player of Games

# Acknowledgments

There are many people to whom I am deeply grateful.

Thanks to my supervisor, Milan Hladik, I was able to do exciting research in algorithmic game theory and large-scale games, areas that were not standard at our university when I began my PhD studies. He trusted me even when my research goals were very ambitious and gave me unparalleled freedom in my pursuit of these goals. He also introduced me to seminars at the Society for the Optimization of Consumption, which I thoroughly enjoyed.

I was also very lucky to meet a lot of great friends and colleagues that I was able to work and do research with. I have learned a lot from every one of you.

I met my friend, Martin Schmid, during the first year of undergrad and have been working with him ever since. I still vividly remember us creating our first naive poker bots during the summer break that followed. We submitted our first paper and built our first entry for the Annual Poker Competition a few years later, and that marked the beginning of my research career. Since then, we've had tons of fun — the right kind of fun, involving unhealthy doses of caffeine and lots of code — working on big projects like Player Of Games and DeepStack.

DeepStack brings me to Michael Bowling. Mike allowed us to pursue our dream of surpassing human-level play in poker using deep learning and search at the renowned Computer Poker Research Group at the University of Alberta, even before AlphaGo demonstrated that such an accomplishment was possible in Go.

My parents, Zuzana and Stefan, have always been there for me. From when I was young and curious, to my years studying, they've supported me every step of the way. I wouldn't be who I am without them.

Common misconception that; that fun is relaxing. If it is, you're not doing it right.

---

*Iain M. Banks, The Player of Games*

# Contents

<b>1</b>	<b>Introduction</b>	<b>8</b>
1.1	Types of Games . . . . .	8
1.1.1	Number of Players . . . . .	8
1.1.2	Perfect and Imperfect Information . . . . .	8
1.2	History of AI In Games . . . . .	8
1.3	Perfect Information Games . . . . .	9
1.3.1	Turing chess . . . . .	9
1.3.2	Samuel's Checkers . . . . .	9
1.3.3	TD-Gammon . . . . .	9
1.3.4	Deep Blue . . . . .	9
1.3.5	AlphaGo . . . . .	9
1.3.6	AlphaGo Zero . . . . .	10
1.3.7	AlphaZero . . . . .	10
1.3.8	Summary . . . . .	10
1.4	Imperfect Information . . . . .	11
1.4.1	Matrix Games . . . . .	11
1.4.2	Sequential Decision Making . . . . .	11
1.4.3	Computer Poker . . . . .	12
1.4.4	Annual Computer Poker Competition . . . . .	13
1.4.5	Game Abstraction . . . . .	13
1.4.6	Success of Classical Techniques . . . . .	13
1.4.7	Limitations of Classical Techniques . . . . .	14
1.4.8	Search Based Techniques . . . . .	14
1.5	Author's Contribution . . . . .	16
1.5.1	Theoretical Advancements . . . . .	16
1.5.2	Novel Algorithms . . . . .	16
<b>2</b>	<b>Background</b>	<b>19</b>
2.1	Extensive Form Games . . . . .	19
2.2	Strategies and Equilibrium . . . . .	19
<b>3</b>	<b>Revisiting CFR+ and Alternating Updates</b>	<b>22</b>
3.1	Introduction . . . . .	22
3.2	Definitions . . . . .	22
3.2.1	Regret-Matching and Regret-Matching+ . . . . .	22
3.2.2	CFR and CFR+ . . . . .	23
3.3	Theoretical Results . . . . .	23
3.3.1	Regret-Matching and Regret-Matching+ Properties . . . . .	24
3.3.2	CFR and CFR+ Properties . . . . .	27
3.4	Conclusions . . . . .	31
3.5	Author's contributions . . . . .	31
<b>4</b>	<b>Bounding the Support Size in Extensive Form Games with Imperfect Information</b>	<b>32</b>
4.1	Introduction . . . . .	32
4.2	Public Game Tree . . . . .	33

4.3	Strategies and Equilibrium . . . . .	34
4.3.1	Support . . . . .	34
4.3.2	Overall Regret . . . . .	35
4.3.3	Counterfactual Values, Regret and Equilibrium . . . . .	35
4.4	Main Theorem . . . . .	36
4.5	Overview of our Approach . . . . .	36
4.5.1	Optimality of the New Strategy . . . . .	36
4.6	Equilibrium Preserving Transformations . . . . .	37
4.6.1	Information Sets after Public Set . . . . .	37
4.6.2	Other Information Sets . . . . .	38
4.7	System of Linear Equations . . . . .	39
4.7.1	First System of Equations . . . . .	39
4.7.2	Second System of Equations . . . . .	39
4.7.3	Final System of Equations . . . . .	39
4.8	Example Games . . . . .	40
4.8.1	Bayesian Extensive Games with Observable Actions . . . . .	40
4.8.2	No-limit Texas Hold'em Poker . . . . .	41
4.9	Conclusion . . . . .	41
4.10	Author's contributions . . . . .	41
<b>5</b>	<b>Sound Algorithms in Imperfect Information Games</b>	<b>42</b>
5.1	Introduction . . . . .	42
5.2	Background . . . . .	43
5.3	Online Algorithm . . . . .	44
5.3.1	Coordinated Matching Pennies . . . . .	44
5.3.2	Naive Tabularization . . . . .	45
5.3.3	Online Settings . . . . .	46
5.3.4	Soundness of Online Algorithm . . . . .	46
5.3.5	Response Game . . . . .	47
5.3.6	Tabularized Strategy . . . . .	47
5.4	Relating soundness and Nash . . . . .	48
5.4.1	Local Consistency . . . . .	48
5.4.2	Global Consistency . . . . .	48
5.4.3	Strong Global Consistency . . . . .	49
5.5	Relating soundness and regret . . . . .	50
5.6	Experiments . . . . .	50
5.7	Related literature . . . . .	51
5.8	Conclusion . . . . .	52
5.9	Author's contributions . . . . .	52
<b>6</b>	<b>Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines</b>	<b>53</b>
6.1	Introduction . . . . .	53
6.2	Related Work . . . . .	54
6.3	Background . . . . .	55
6.3.1	Augmented Information Sets . . . . .	55
6.3.2	Counterfactual Regret Minimization . . . . .	55
6.3.3	Monte Carlo CFR . . . . .	56
6.3.4	Control Variates . . . . .	56
6.3.5	Reinforcement Learning Mapping . . . . .	57

6.4	Monte Carlo CFR with Baselines . . . . .	57
6.4.1	Recursive Bootstrapping . . . . .	58
6.4.2	Choice of Baselines . . . . .	59
6.4.3	Summary of the Full Algorithm . . . . .	60
6.5	Experimental Results . . . . .	60
6.5.1	Convergence . . . . .	61
6.5.2	Observed Variance . . . . .	61
6.5.3	Evaluation of Bootstrapping and Baseline Dependence on Actions . . . . .	61
6.6	Conclusions . . . . .	64
6.7	Author’s contributions . . . . .	65
<b>7</b>	<b>Refining Subgames in Large Imperfect Information Games</b>	<b>66</b>
7.1	Introduction . . . . .	66
7.2	Previous Work . . . . .	67
7.3	Background and Notation . . . . .	68
7.3.1	Counterfactual Best Response . . . . .	68
7.3.2	Subgame . . . . .	68
7.3.3	Formulating Subgame Refinement using Optimization . . . . .	68
7.3.4	Endgame Solving . . . . .	69
7.3.5	Re-solving . . . . .	69
7.3.6	Discussion . . . . .	70
7.4	Our Technique . . . . .	71
7.4.1	Subgame Margin . . . . .	71
7.4.2	Optimization Formulation . . . . .	72
7.4.3	Gadget Game . . . . .	72
7.4.4	Gadget Game Construction . . . . .	72
7.5	Experiments . . . . .	73
7.6	Conclusion . . . . .	74
7.7	Author’s contributions . . . . .	75
<b>8</b>	<b>AIVAT: A New Variance Reduction Technique for Agent Evaluation in Imperfect Information Games</b>	<b>76</b>
8.1	Introduction . . . . .	76
8.2	Value Estimation . . . . .	77
8.3	MIVAT and Imaginary Observations . . . . .	77
8.4	AIVAT . . . . .	78
8.4.1	AIVAT Correction Terms . . . . .	78
8.4.2	AIVAT Base Value . . . . .	80
8.4.3	AIVAT Value Estimate . . . . .	80
8.5	Unbiased Value Estimate . . . . .	81
8.6	Experimental Results . . . . .	82
8.6.1	Leduc Hold’em . . . . .	83
8.6.2	No-limit Texas Hold’em . . . . .	84
8.7	Conclusions . . . . .	84
8.8	Author’s contributions . . . . .	85
<b>9</b>	<b>DeepStack: Expert-level artificial intelligence in heads-up no-limit poker</b>	<b>86</b>
9.1	Introduction . . . . .	86
9.2	DeepStack . . . . .	87
9.2.1	Continual re-solving . . . . .	88



9.2.2	Limited depth lookahead via intuition . . . . .	89
9.2.3	Sound reasoning . . . . .	90
9.2.4	Sparse lookahead trees . . . . .	90
9.2.5	Relationship to heuristic search in perfect information games . . . . .	90
9.2.6	Relationship to abstraction-based approaches . . . . .	91
9.3	Deep Counterfactual Value Networks . . . . .	91
9.3.1	Architecture . . . . .	92
9.3.2	Training . . . . .	92
9.4	Evaluating DeepStack . . . . .	93
9.4.1	Exploitability . . . . .	94
9.5	Discussion . . . . .	94
9.6	Author’s contributions . . . . .	95
<b>10</b>	<b>Player of Games</b>	<b>96</b>
10.1	Introduction . . . . .	96
10.2	Background and Terminology . . . . .	97
10.2.1	Tree Search and Machine Learning . . . . .	98
10.2.2	Game-Theoretic Reasoning and Counterfactual Regret Minimization . . . . .	99
10.2.3	Imperfect Information Search, Decomposition, and Re-Solving . . . . .	100
10.3	Player of Games . . . . .	101
10.3.1	Counterfactual Value-and-Policy Networks . . . . .	101
10.3.2	Search via Growing-Tree CFR . . . . .	102
10.3.3	Data Generation via Sound Self-play . . . . .	104
10.3.4	Training Process . . . . .	105
10.3.5	Bringing it all Together: Full Algorithm Overview . . . . .	107
10.4	Evaluation . . . . .	107
10.4.1	Exploitability in Leduc Poker and Small Scotland Yard Map . . . . .	108
10.4.2	Results in Challenge Domains . . . . .	109
10.5	Related Work . . . . .	112
10.6	Conclusion . . . . .	113
10.7	Author’s contributions . . . . .	114
<b>11</b>	<b>Conclusion</b>	<b>115</b>
11.1	Potential Applications . . . . .	115
11.2	Future Work . . . . .	115
	<b>Bibliography</b>	<b>116</b>
	<b>List of Figures</b>	<b>130</b>
	<b>List of Tables</b>	<b>132</b>
	<b>List of Publications</b>	<b>133</b>
<b>A</b>	<b>Attachments for Chapter 4</b>	<b>134</b>
A.1	Proof of Lemma 4.5 . . . . .	134
A.2	Proof of Lemma 4.6 . . . . .	134
A.3	Proof of Lemma 4.7 . . . . .	136

<b>B</b>	<b>Attachments for Chapter 5</b>	<b>138</b>
B.1	Consistency Examples . . . . .	138
B.2	Tabularization . . . . .	138
B.3	Proofs of Theorems . . . . .	139
B.4	Experiment details . . . . .	143
<b>C</b>	<b>Attachments for Chapter 6</b>	<b>146</b>
C.1	MCCFR and MCCFR+ comparison . . . . .	146
C.2	Vector Form of CFR . . . . .	146
C.3	Proofs . . . . .	147
C.3.1	Proof of Lemma 6.1 . . . . .	147
C.3.2	Proof of Lemma 6.2 . . . . .	147
C.3.3	Proof of Lemma 6.3 . . . . .	149
C.4	Kuhn Example . . . . .	150
<b>D</b>	<b>Attachments for Chapter 7</b>	<b>153</b>
<b>E</b>	<b>Attachments for Chapter 9</b>	<b>154</b>
E.1	Game of Heads-Up No-Limit Texas Hold'em . . . . .	154
E.2	Poker Glossary . . . . .	155
E.3	Performance Against Professional Players . . . . .	156
E.4	Local Best Response of DeepStack . . . . .	156
E.5	DeepStack Implementation Details . . . . .	159
E.5.1	Continual Re-Solving . . . . .	159
E.5.2	Deep Counterfactual Value Networks . . . . .	162
E.6	Proof of Theorem 9.1 . . . . .	164
E.7	Best-response Values Versus Self-play Values . . . . .	169
E.8	Pseudocode . . . . .	170
<b>F</b>	<b>Attachments for Chapter 10</b>	<b>173</b>
F.1	An example of Factored-Observation Stochastic Game . . . . .	173
F.2	Player of Games Algorithm Details . . . . .	174
F.2.1	Re-solving Process . . . . .	174
F.2.2	Complexity of the Algorithm . . . . .	175
F.2.3	Network Architecture and Optimization . . . . .	176
F.2.4	Hyperparameters . . . . .	176
F.2.5	Pseudocode . . . . .	177
F.2.6	Implementation . . . . .	178
F.2.7	Poker Betting Abstraction . . . . .	178
F.3	Evaluation Details and Additional Experimental Results . . . . .	179
F.3.1	Description of Leduc poker . . . . .	179
F.3.2	Custom Glasses Map for Scotland Yard . . . . .	181
F.3.3	Full Results of Go Agent Tournament . . . . .	181
F.3.4	Reinforcement Learning and Search in Imperfect Information Games	181
F.4	Scotland Yard example . . . . .	184
F.5	Proofs of Theorems . . . . .	185
F.5.1	Value Functions for Subgames . . . . .	185
F.5.2	Growing Trees . . . . .	187
F.5.3	Self-play Values as Re-solving Constraints . . . . .	191
F.5.4	Continual Re-solving . . . . .	193

# 1. Introduction

Games have played a key role in AI history, engaging top minds and serving as important benchmarks. They can model a wide range of real-world situations, have well-defined objectives, and the performance of agents can be directly compared with that of humans. Also, games are fun to play and even more fun to do research on.

## 1.1 Types of Games

Game theory includes a diverse set of games, each with unique structures, rules, and outcomes, ranging from simple games like tic-tac-toe to complex scenarios like economic markets. This diversity demands different agents or algorithms for different types of games. There are few categories that can help us determine algorithmic properties of a game.

### 1.1.1 Number of Players

The definition of optimal policy for a single player game is straightforward - it is a set of actions that maximize the reward in each reachable state.

However, when multiple players are involved, the reward is not just based on the actions of one player, but also on the actions of all other players in the game. Since the policies of the opponents are unknown, determining the optimal policy becomes more complex. The objective remains the same - to maximize the reward - but without knowledge of the opponents' policies, this becomes much more challenging to achieve.

Fortunately, the case of two-player games can be relatively simple. If the game is strictly competitive (i.e., zero-sum, as the rewards of both players sum to zero) and symmetrical, such that players get to play both positions (e.g., small and big blind in poker, or black and white in chess), in expectation, an optimal policy cannot lose to any opponent and often can still benefit substantially from opponent mistakes. This allows us to disregard what the exact policy of the opponent will be and to focus on near-optimal policies instead. In the rest of this thesis, we will consider only two-player zero-sum games.

### 1.1.2 Perfect and Imperfect Information

In perfect information games, the information available to players is symmetrical - if one player knows a piece of information, all the other players know it too. This makes reasoning about the optimal play relatively straightforward - it is always possible to choose a single best action to play in any state of the game. Common examples of perfect information games are checkers, chess, and Go.

In imperfect information games, some information can be known only by a subset of players, while being hidden from the rest. This asymmetric nature of the information makes reasoning about the optimal play significantly more complex. Players have to play stochastically and carefully mix their actions to reduce the leakage of hidden information. Canonical examples for such a game are poker or rock-paper-scissors.

## 1.2 History of AI In Games

From the 1950s to the present, there have been significant developments in algorithms resulting in multiple milestones. Until recently, this development was largely separate for

perfect and imperfect information games. In this section, we will first look at the historical development of perfect information games, then at imperfect information games, and finally at algorithms that unify approaches from both areas. Given the extent of the research in this area, the mentioned milestones are by no means exhaustive; we will only look at a few of the most prominent and illustrative results.

## **1.3 Perfect Information Games**

### **1.3.1 Turing chess**

One of the earliest examples was Turochamp , a chess program developed by Alan Turing and David Champernowne in 1948 [Copeland, 2004]. Though it was not executed on a real computer due to its complexity for contemporary machines, it was run manually step by step by Turing himself, showing it could handle a full game against a human. While this was a first attempt, it already had important concepts that would repeatedly appear in later algorithms - namely, search and heuristic evaluation function.

### **1.3.2 Samuel's Checkers**

The next important milestone was Samuel's checkers program [Samuel, 1959]. It improved both the search and the value function. For the search, it used minimax search with alpha-beta pruning — an algorithm that is still used today in top chess engines [The Stockfish Development Team, 2021]. Even more important and interesting are the improvements in the heuristic value function. Instead of hard coding, the function was trained using self-play and machine learning. This was one of the first big successes of machine learning. The final version of the program achieved a strong amateur level in checkers - much better than the author himself.

### **1.3.3 TD-Gammon**

The concept of using machine learning with self-play to learn value was then taken one step further by TD-Gammon, a Backgammon player program developed by Gerald Tesauro [Tesauro, 1995]. It used neural networks to approximate the value function and a TD-update rule borrowed from reinforcement learning, combined again with search. It was one of the first programs approaching the level of top human players in a large game.

### **1.3.4 Deep Blue**

In 1997, almost 50 years after Samuel's checkers program, Deep Blue became the first computer program to defeat a reigning world champion after winning a match against Garry Kasparov [Campbell et al., 2002]. This came after a close defeat in the previous year. The program used alpha-beta search in combination with a sophisticated value function. While the value function was largely hand-crafted, it was tuned using a large database of human games. The program was also accelerated using special chess chips.

### **1.3.5 AlphaGo**

Even after mastering chess, the game of Go remained a long-standing challenge for computer players. Two sources of difficulty hindered classical search approaches. The first was the

large branching factor of Go - this made delving deeper into the search tree exponentially harder. The other significant problem was the absence of a known strong value function. AlphaGo solved both these problems with the help of machine learning using deep networks [Silver et al., 2016].

The search was based on Monte Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006], previously used successfully for Go, but the space of searched actions was greatly reduced thanks to the use of a policy network that suggested the most promising actions to investigate. The evaluation of positions was composed of a combination of two different approaches. Firstly, there was a value function implemented by a convolutional neural network that took a representation of the board and returned a corresponding value. The other evaluation approach used a fast policy network to quickly unroll the game - simulating the actions of both players until the eventual end of the game. This end value was then returned as a final estimate. A linear combination of both these approaches was then used to provide a single value estimate for MCTS.

To train the agent, a large dataset of human Go games was first used for supervised training of the policy and value functions. These functions were then improved using self-play training.

AlphaGo was able to defeat Lee Sedol, one of the world's best Go players.

### **1.3.6 AlphaGo Zero**

AlphaGo Zero was a successor of AlphaGo [Silver et al., 2017b]. It demonstrated that even such a complex game as Go could be trained in a zero-knowledge fashion - that means without human data, using only self-play, the rules of the game, and minimal prior knowledge. Not only was it able to surpass the performance of the original AlphaGo, but it did so using a simpler and more general algorithm.

In contrast to AlphaGo, value estimation came only from the value network, and both policy and value estimation were trained directly from self-play.

### **1.3.7 AlphaZero**

Most high-performance computer programs were designed to play just a single game. For example, Deep Blue would not be able to play Go or checkers. However, the general architecture of AlphaGo Zero allowed one to simply take the same algorithm without any big modifications and train it to play two more games. The resulting agent — AlphaZero — achieved state-of-the-art performance on chess, Go and shogi, all of this using the same network architecture and almost identical hyper-parameters [Silver et al., 2017a].

### **1.3.8 Summary**

Most successful algorithms for perfect information games share a few common traits. The first is the use of search methods, either minimax or MCTS, which allows for real-time reasoning about complex situations as they occur during gameplay. The second is the utilization of a heuristic value function at the leaves of the search tree. More general approaches that use less expert knowledge also share the use of self-play — a technique from reinforcement learning where an AI agent repeatedly plays games against itself, using the outcomes of these games as learning input. As a result, both the value and the policy can be learned without human input.

## 1.4 Imperfect Information

There is a small but crucial difference between perfect information games such as chess and Go, and imperfect information games such as poker or rock-paper-scissors. In chess, all necessary information is known by both players. In contrast, poker players don't know what cards their opponents hold. This allows chess players to simply choose the single best action to play an optimal maximin strategy. It's easy to see that this cannot be done in games like rock-paper-scissors or poker. Instead, the player has to act strategically and mix actions to carefully conceal information. Moreover, in contrast to perfect information games, where one can just examine possible future actions, in imperfect information games, the optimal policy also depends on the past actions of the players and their opponent. Because of this, classical approaches solve the whole game at once using some optimization technique.

Let's now consider some examples of these games, alongside the classical techniques used.

### 1.4.1 Matrix Games

Matrix games, also known as normal form games, represent the simplest form of imperfect information games. They depict a situation where all players make decisions simultaneously. In the case of two-player, zero-sum games, such a game can be described by a single payoff matrix. The possible actions for Player 1 are to choose a row from the matrix, while Player 2 must choose a column. The value of the corresponding matrix element is then equal to the utility of Player 1 at the end of the game. Since we are considering zero-sum games, this value also corresponds to the negative utility of Player 2. A simple example of such a game is rock-paper-scissors. Note that actions can be stochastic — each player can choose a probability distribution over their actions.

In 1928, Von Neumann developed the minimax theorem, which has become a foundational principle of game theory, demonstrating what the optimal solution of matrix games looks like [Neumann, 1928]. In 1951, Dantzig showed the equivalence of zero-sum games and linear programs [Dantzig, 1951]. This allows for efficient solutions of large normal form games, using any available LP solver.

### 1.4.2 Sequential Decision Making

#### Extensive form Games

In many real-world situations, players do not act simultaneously but instead take sequences of actions. This is the case for the majority of board games, including Checkers, Chess, Go, and Poker. The extensive form game formalism represents all possible action sequences using a game tree. The leaves of the tree correspond to terminal states where the game's terminal utility is defined. Nodes in the tree represent decision points for the players and the edges represent players' actions. If the game involves imperfect information, as in Poker, a player must apply the same strategy in all states that he cannot distinguish between. These states form an information set. If there's a stochastic element in the game, such as dealing cards in Poker, an additional player called a "chance player" is introduced to model this stochasticity. This player acts according to a known fixed probability distribution. The formal definition of the extensive form game is introduced in chapter 2.

## **Shortcomings of the Formalism**

One of the shortcomings of the extensive game formalism is that it is too general. It allows the definition of games where players are forced to forget their past actions. If this situation occurs, the game is called an imperfect recall game. Imperfect recall is not only unrealistic, but it also has unfortunate algorithmic consequences. The existence of the Nash equilibrium becomes an NP-hard problem [Hansen et al., 2007], and even the computation of the best strategy given a fixed opponent becomes very complicated [Piccione et al., 1996, Piccione and Rubinstein, 1997].

Thus, in this thesis, we will only consider perfect recall games, and when mentioning an extensive form game, we mean a two-player, zero-sum, perfect recall extensive form game.

## **Factored-Observation Stochastic Games**

The Factored-Observation Stochastic Game formalism is similar to the Extensive Form Game formalism, but there are a few significant changes that make the definition of modern algorithms easier [Kovářík et al., 2021]. The observability of the states is described more clearly: there is a private component that only a player can see, and a public component that all players can observe. There is also a notion of a player's state even when he is not acting.

Virtually all board games can be described as Factored-Observation Stochastic Games.

## **Solution Techniques**

### **Conversion to Normal Form**

It is possible to convert any extensive form game to a normal form game simply by enumerating every possible combination of decisions for each information set and for each player. Once this conversion is done, one can use an LP solver to find the optimal policy. Unfortunately, this conversion can be exponential in the size of the game tree, so it is usable only for very small games.

### **Sequence Form**

A more efficient solution is to use the sequence form - an LP formulation that is linear in size to the game tree [Nisan et al., 2007]. While this approach allows for solving much larger games, it requires memory proportional to the number of all game states. For games like poker, where the number of possible game states is several orders of magnitude higher than the number of information sets, the memory requirement is still too large.

### **Counterfactual Regret Minimization**

Recently, most of the successful solving techniques for large extensive form games have been using some version of Counterfactual Regret Minimization (CFR) [Zinkevich et al., 2007]. It is an iterative algorithm that operates directly on information sets, thus requiring several orders of magnitude less memory. In each iteration, both players update their strategies, and the average of these strategies provably converges to a Nash equilibrium. The algorithm can be stopped at any time.

### **1.4.3 Computer Poker**

Poker is the canonical game of imperfect information where players cannot see their opponent's cards. Strong play involves bluffing and insights into potential opponent strategies,

qualities that have traditionally not been considered computer-like. In their groundbreaking work "Theory of Games and Economic Behavior," von Neumann and Morgenstern dedicated an entire section (over 30 pages) to poker [Morgenstern and Von Neumann, 1953].

Over the years, there has been a substantial body of research in imperfect information games, with poker game variants being the only domain used for evaluating the algorithms.

### **Kuhn Poker**

The Kuhn poker is a two-player, zero-sum game used to model the decision-making process in scenarios where complete information is not available. It was first introduced by Harold W. Kuhn in the 1950s [Kuhn, 1950]. In Kuhn poker, each player is dealt one card from a three-card deck and must choose to either bet or pass. If both players pass, the player with the higher card wins the pot. If one player bets and the other passes, the betting player wins the pot. If both players bet, the player with the higher card wins the pot. The game is very simple, yet it contains all the essential elements of a more complex game, including incomplete information and sequential decision-making. It's easy to verify that there isn't a deterministic optimal strategy. Kuhn demonstrated in his paper that there is a continuum of optimal stochastic strategies for the first player and a single optimal strategy for the second player.

#### **1.4.4 Annual Computer Poker Competition**

The Annual Computer Poker Competition [Bard et al., 2013] was started in 2006 as an effort to develop a system to evaluate poker agents that were being developed by the University of Alberta and Carnegie Mellon University. It has been held annually since 2006 until 2018, open to all competitors, in conjunction with top-tier artificial intelligence conferences: AAAI and IJCAI. Multiple university teams and individuals participated each year, submitting dozens of poker agents.

#### **1.4.5 Game Abstraction**

Classical solution approaches for imperfect information games require reasoning about the entire game tree at once and producing a complete strategy prior to play. Since a lot of poker variants, like Heads-Up No Limit Texas Hold'em, are too large to be solved directly, the common technique is to solve a smaller, abstracted game that is similar to the original game. To play the original game, one must first translate actions from the original game to the abstracted game, then choose an action based on the abstracted game's policy, and finally translate this action back to the original game. This entire process is called game abstraction.

The majority of the top Annual Computer Poker Competition entries used game abstraction along with counterfactual regret minimization.

#### **1.4.6 Success of Classical Techniques**

The combination of the counterfactual regret minimization and abstraction resulted in important milestones for imperfect information games.



## **Polaris**

In 2007 and 2008, the Computer Poker Research Group at the University of Alberta organized the Man-vs-Machine Poker Championships, using the game of Heads-Up Limit Texas Hold'em [Bowling et al., 2009]. In 2007, a poker agent named Polaris competed against human professional players but lost narrowly. In 2008, the improved versions of Polaris narrowly won. This was the first time that a poker-playing AI defeated human professionals.

## **Cepheus**

In January of 2015, the poker agent Cepheus reached another milestone by essentially solving the entire game of Heads-Up Limit Texas Hold'em [Bowling et al., 2015]. While other large games, such as checkers or Connect Four, had been solved previously, this was the first time that any large imperfect information game played professionally by humans was solved.

### **1.4.7 Limitations of Classical Techniques**

While abstraction techniques were very successful in Limit Heads-Up Texas Hold'em poker, their success in No-Limit Texas Hold'em poker, a more complex but also more popular version of poker, was modest. In 2015, the abstraction-based computer program Claudico lost to a team of professional poker players in a No-Limit Texas Hold'em poker match by a margin of 91 mbb/g, which is considered a 'huge margin of victory' [Moravčík et al., 2017]. Furthermore, the local best-response technique showed that abstraction-based programs from the Annual Computer Poker Competition have massive flaws, and moreover, these flaws are relatively easy to find. All evaluated abstraction-based programs lost by at least 3,000 mbb/g against the local best response, which is four times more than if they had simply folded each game [Lisý and Bowling, 2017b]. To illustrate the naivety of the abstraction-based approach for use in large extensive games, one can imagine its application to chess. It would require constructing an abstracted version of chess small enough to solve directly and then mapping states and actions between this abstraction and the original game [Schmid, 2021].

### **1.4.8 Search Based Techniques**

In perfect information, the combination of decision-time search with a heuristic value function leads to strong performance. Some popular perfect information search methods, like Monte Carlo tree search can be also used in imperfect information settings [Whitehouse, 2014]. Unfortunately they fail to produce optimal policies even in very small games. Until recently, it had been even thought that sound search is impossible in imperfect information games [Frank et al., 1998, Lisý et al., 2015]. Fortunately, a significant milestone in computational game theory has been reached recently — a sound search in imperfect information games.

## **DeepStack**

DeepStack was the first algorithm to introduce the combination of sound decision-time search and heuristic value function for imperfect information games [Moravčík et al., 2017]. This was made possible by a technique called continual-resolving. It is analogous to the search in perfect information games, but with a few important modifications that make it theoretically sound. The lookahead tree contains not only states in which the player is acting, but also all the states that are sharing the same publicly known information. This

allows for coordination of the policy between states that either player cannot distinguish. In poker, this means that the search always takes into consideration all possible private cards a player and his opponent could hold. To reason within this complex search tree, the value function also has to be more intricate. In contrast to perfect information games, it outputs a vector of values for each player. The root of the lookahead tree is also significantly modified—it forms a gadget game. This gadget ensures that if the policy is optimal in the lookahead tree, it is also optimal in the whole game. The search algorithm must be able to compute a precise stochastic policy, therefore DeepStack uses a version of CFR instead of simple minimax. DeepStack’s value function was implemented by a neural network and trained using a large number of examples generated from random poker situations. Continual resolving allowed DeepStack to ditch the abstraction and reason about situations independently as they arise during play, which led to a significant improvement over prior methods. In December of 2016, DeepStack became the first program to beat professional human players in no-limit Texas hold’em poker. In contrast to previous abstraction-based agents, DeepStack is unexploitable by the local best response.

## **Libratus**

Subsequent to DeepStack, the computer program Libratus defeated a team of four professional heads-up poker specialists in a HUNL competition held in January 2017 [Brown and Sandholm, 2018]. Libratus could be described as a hybrid approach. Near the end of the game, it used a ‘nested endgame solving’ technique similar to the continuous re-solving used by DeepStack. Since it did not utilize a value function, it couldn’t execute the search in the early stages of the game and instead used classical abstraction techniques. The use of abstraction resulted in weaknesses in the strategy. To address this, the abstract strategy was augmented with the help of human analysis during match breaks.

Both DeepStack and Libratus demonstrated that real-time decision-making is crucial to achieving high-level performance.

## **Player of Games**

Even after the introduction of decision time search, the worlds of perfect and imperfect games remained separate. Imperfect information agents were usually designed to handle just a single specific game. The Player of Games (PoG) bridges this gap [Schmid et al., 2021]. It was the first algorithm to achieve strong empirical performance in large perfect information games — chess and Go, as well as in imperfect information games — poker and Scotland Yard. This marked an important step toward creating general algorithms for arbitrary environments. The algorithm combines ideas from DeepStack and AlphaZero in a theoretically sound fashion. Continual-resolving introduced by DeepStack is used to ensure that the policy is consistent during online play. Growing-tree counterfactual regret minimization (GT-CFR) builds a lookahead tree non-uniformly, expanding the tree toward the most relevant states similarly to the MCTS used by AlphaZero. Sound self-play is used to train the policy and value network. Both networks are specified in a “zero-like” fashion with minimal domain-specific knowledge

## **Limitations of Sound Search**

The main limitation of the sound search used by DeepStack and Player of Games is the need to enumerate all possible information states contained in a public state, which can be prohibitively expensive for some games.

This could be an interesting area for future research; one possible solution is to use sampling of the information states.

## 1.5 Author’s Contribution

The remaining sections of this Ph.D. thesis delve into my contributions to the field of algorithmic game theory. These contributions can be broadly categorized into two main areas: theoretical advancements and novel algorithms.

### 1.5.1 Theoretical Advancements

#### Revisiting CFR+ and Alternating Updates

Many successful imperfect information game agents, such as Polaris, Libratus, DeepStack, and Player of Games, leverage a variant of the Counterfactual Regret Minimization (CFR) algorithm. A recent and widely adopted version of CFR is CFR+ due to its superior empirical performance across various problem domains, making it one of the key factors contributing to the success of Cepheus [Burch, 2017]. Although CFR+ was initially introduced with a theoretical upper bound on solution error, subsequent research revealed an error in one of the proof steps [Farina et al., 2019]. We provide updated proofs to recover the original bound [Burch et al., 2019].

#### Bounding the Support Size in Extensive Form Games with Imperfect Information

Optimal play in imperfect information games often necessitates the use of stochastic policies. This stands in contrast to perfect information games, where a simple deterministic optimal strategy always exists. Naturally, one may wonder about the impact on the number of optimal actions as the level of uncertainty increases.

We have established a linear relationship between the level of uncertainty and the support size, which refers to the number of actions with non-zero probability [Schmid et al., 2014].

#### Sound Algorithms in Imperfect Information Games

The concept of Nash equilibrium is traditionally defined for a fixed offline set of strategies. However, extending this concept to online settings, where the entire strategy is not computed in advance, is not a straightforward task. Naively attempting to do so may result in the disappearance of certain guarantees for two-player zero-sum games. To tackle this issue, we introduced the concept of a consistency hierarchy, which enables the analysis of algorithms that perform online search, such as those employed in DeepStack or Player of Games [Šustr et al., 2020].

### 1.5.2 Novel Algorithms

#### Variance Reduction in Monte Carlo Counterfactual Regret Minimization for Extensive Form Games Using Baselines

Monte Carlo Counterfactual Regret Minimization (MCCFR) [Lanctot et al., 2009] is a family of game-solving algorithms designed for imperfect information games. In contrast to the vanilla CFR implementation, MCCFR doesn’t require traversing the entire game tree in each iteration. Instead, it samples a limited number of trajectories, similar to algorithms

used in reinforcement learning. While MCCFR still offers good probabilistic convergence guarantees, the introduction of sampling introduces variance in value estimates, which can considerably slow down the convergence speed [Burch, 2017].

In the realm of reinforcement learning, this variance issue has traditionally been addressed using baselines in policy-based methods. In VR-MCCFR (Variance-Reduced MCCFR) [Schmid et al., 2019], we employed similar ideas to obtain unbiased value estimates and reduce variance. In the ideal scenario of perfect estimates, the variance can be reduced to zero. In experimental evaluations, VR-MCCFR achieved an order of magnitude speedup and decreased empirical variance by three orders of magnitude.

### **Refining Subgames in Large Imperfect Information Games**

Traditionally, state-of-the-art game algorithms have employed an abstraction approach, where they solve a smaller, abstracted version of the game and then map the strategy from this reduced game back to the original game at decision time. However, to enable online improvement of strategies, particularly in situations close to the game end where the problem is more tractable, we introduced the concept of safe refinement of sub-games [Moravčík et al., 2016]. The ideas from this work have since been utilized by Libratus, DeepStack, and Player of Games.

### **AIVAT: A New Variance Reduction Technique for Agent Evaluation in Imperfect Information Games**

Evaluation of agents in imperfect information games is inherently noisy, especially when compared to perfect information games. Traditionally, evaluating agents in computer poker competitions required millions of matches to obtain statistically significant results. While this approach worked for simple, abstraction-based agents that don't require complex computation at run-time, it becomes computationally expensive when agents employ decision-time search. Furthermore, comparing agent performance to human players exacerbates the problem. To address these challenges, we developed AIVAT, a provably unbiased method that significantly reduces variance during evaluation [Burch et al., 2018].

### **Deepstack: Expert-level artificial intelligence in heads-up no-limit poker**

DeepStack was the first algorithm to use a theoretically sound combination of limited search depth and machine learning for imperfect information games [Moravčík et al., 2017]. This approach reduced the gap between approaches for perfect and imperfect information. Significantly, it was also the first AI system to outperform professional poker players in No-Limit Texas Hold'em, representing a major accomplishment in the field of AI.

### **Player of Games**

The Player of Games represents the culmination of our efforts to unify the domains of perfect and imperfect information games [Schmid et al., 2021]. It synthesizes techniques employed in both DeepStack and AlphaZero, demonstrating strong empirical performance in both types of games. This achievement is a significant step towards developing truly general algorithms for arbitrary environments. The approach gradually builds a search tree, similar to Monte Carlo Tree Search, and learns from self-play with minimal prior information, similar to AlphaZero. At the same time, it incorporates sound game theoretic reasoning, akin to DeepStack. We have proved that the Player of Games is theoretically sound, and have evaluated its performance in two perfect information games — chess and Go - and two

imperfect information games — Heads-Up No-Limit Texas Hold'em Poker and Scotland Yard.

## 2. Background

Definitions in this chapter are based on [Burch et al., 2019].

### 2.1 Extensive Form Games

An **extensive form game** [Von Neumann and Morgenstern, 1947] is a sequential decision-making problem where players have imperfect (asymmetric) information. The formal description of an extensive form game is given by a tuple  $\langle H, P, p, \sigma_c, u, \mathcal{I} \rangle$ .

$H$  is the set of all states  $h$ , which are a **history** of **actions** from the beginning of the game  $\emptyset$ . Given a history  $h$  and an action  $a$ ,  $ha$  is the new state reached by taking action  $a$  at  $h$ . To denote a descendant relationship, we say  $h \sqsubseteq j$  if  $j$  can be reached by some (possibly empty) sequence of actions from  $h$ , and  $h \sqsubset j \iff h \sqsubseteq j, h \neq j$ .

We will use  $Z := \{h \in H \mid \nexists j \in H \text{ s.t. } h \sqsubset j\}$  to denote the set of **terminal histories**, where the game is over. We will use  $Z(h) := \{z \in Z \mid h \sqsubseteq z\}$  to refer to the set of terminal histories that can be reached from some state  $h$ .

$A(h)$  gives the set of valid actions at  $h \in H \setminus Z$ . We assume some fixed ordering  $a_1, a_2, \dots, a_{|A|}$  of the actions, so we can speak about a vector of values or probabilities across actions.  $a \prec b$  denotes that action  $a$  precedes  $b$ , with  $a \prec b \iff a_i = a, a_j = b, i < j$ .

$P$  (often also denoted as  $N$ ) is the **set of players**, and  $p : H \setminus Z \rightarrow P \cup \{c\}$  gives the **acting player** for state  $h$ , or the special chance player  $c$  for states where a chance event occurs according to probabilities specified by  $\sigma_c(h) \in \Delta^{|A(h)|}$ .  $\sigma_c$  is often also denoted as  $f_c$ . The scope of this thesis is restricted to **two-player games**, so we will say  $P = \{1, 2\}$ . The only exception is Chapter 4 that presents results that hold for an arbitrary number of players  $P = \{1, \dots, n\}$ .

The **utility** of a terminal history  $z$  for Player  $p$  is given by  $u_p(z)$ . We will restrict ourselves to **zero-sum** games, where  $\sum_{p \in P} u_p(z) = 0$ .

A player's imperfect information about the game state is represented by a partition  $\mathcal{I}$  of states  $H$  based on player knowledge. For all **information sets**  $I \in \mathcal{I}$  and all states  $h, j \in I$  are indistinguishable to Player  $p(h) = p(j)$ , with the same legal actions  $A(h) = A(j)$ . Given this equality, we can reasonably talk about  $p(I) := p(h)$  and  $A(I) := A(h)$  for any  $h \in I$ . For any  $h$ , we will use  $I(h) := I \in \mathcal{I}$  such that  $h \in I$  to refer to the information set containing  $h$ . It is convenient to group information sets by the acting player, so we will use  $\mathcal{I}_p := \{I \in \mathcal{I} \mid p(I) = p\}$  to refer to Player  $p$ 's information sets.

We will also restrict ourselves to extensive form games where players have **perfect recall**. Informally, Player  $p$  has perfect recall if they do not forget anything they once knew: for all states  $h, j$  in some information set, both  $h$  and  $j$  passed through the same sequence of Player  $p$  information sets from the beginning of the game  $\emptyset$ , and made the same Player  $p$  actions.

### 2.2 Strategies and Equilibrium

A **strategy**  $\sigma_p : \mathcal{I}_p \rightarrow \Delta^{|A(I)|}$  for Player  $p$  gives a probability distribution  $\sigma_p(I)$  over legal actions for Player  $p$  information sets. For convenience, let  $\sigma_p(h) := \sigma_p(I(h))$ . A **strategy profile**  $\sigma := (\sigma_1, \sigma_2)$  is a tuple of strategies for both players. Given a profile  $\sigma$ , we will use  $\sigma_{-p}$  to refer to the strategy of  $p$ 's opponent.

Because states are sequences of actions, we frequently need to refer to various products of strategy action probabilities. Given a strategy profile  $\sigma$ ,

$$\pi^\sigma(h) := \prod_{ia \sqsubseteq h} \sigma_{p(i)}(h)_a \quad (2.1)$$

refers to a **reach probability**—the probability of a game reaching state  $h$  when players sample actions according to  $\sigma$  and chance events occur according to  $\sigma_c$ .

$$\pi^\sigma(h | j) := \prod_{\substack{ia \sqsubseteq h \\ j \sqsubseteq i}} \sigma_{p(i)}(h)_a \quad (2.2)$$

refers to the probability of a game reaching  $h$  given that  $j$  was reached.

$$\begin{aligned} \pi_p^\sigma(h) &:= \prod_{\substack{ia \sqsubseteq h \\ p(h)=p}} \sigma_{p(i)}(h)_a \\ \pi_{-p}^\sigma(h) &:= \prod_{\substack{ia \sqsubseteq h \\ p(h) \neq p}} \sigma_{p(i)}(h)_a \end{aligned} \quad (2.3)$$

refer to probabilities of Player  $p$  or all actors but  $p$  making the actions to reach  $h$ , given that  $p$ 's opponent and chance made the actions in  $h$ . Note that there is a slight difference in the meaning of the label  $-p$  here, with  $\pi_{-p}^\sigma$  considering actions by both Player  $p$ 's opponent and chance, whereas  $\sigma_{-p}$  refers to the strategy of  $p$ 's opponent.

$$\pi_p^\sigma(h | j) := \prod_{\substack{ia \sqsubseteq h \\ j \sqsubseteq i \\ p(h)=p}} \sigma_{p(i)}(h)_a \quad (2.4)$$

refers to the probability of Player  $p$  making the actions to reach  $h$ , given  $j$  was reached and  $p$ 's opponent and chance make the actions to reach  $h$ . There are a few useful relationships:

$$\begin{aligned} \pi^\sigma(h) &= \pi_p^\sigma(h) \pi_{-p}^\sigma(h) \\ \forall j \sqsubseteq h, \pi^\sigma(h) &= \pi^\sigma(j) \pi^\sigma(h | j) \end{aligned} \quad (2.5)$$

The **expected utility** of a strategy profile  $\sigma$  is

$$u_p^\sigma := \sum_{z \in Z} \pi^\sigma(z) u_p(z) \quad (2.6)$$

The **counterfactual value** of a history or information set are defined as

$$\begin{aligned} v_p^\sigma(h) &:= \sum_{z \in Z(h)} \pi_{-p}^\sigma(z) \pi_p^\sigma(z | h) u_p(z) \\ \mathbf{v}^\sigma(I) &:= \sum_{h \in I} (v_{p(h)}^\sigma(h a_1), \dots, v_{p(h)}^\sigma(h a_{|A(I)|})) \end{aligned} \quad (2.7)$$

For later convenience, we will assume that for each player there exists an information set  $I_p^\emptyset$  at the beginning of the game, containing a single state with a single action, leading to the rest of the game. This lets us say that  $u_p^\sigma = v_p^\sigma(I_p^\emptyset)_{a_0}$ .

Given a sequence  $\sigma_p^0, \dots, \sigma_p^t$  of strategies, we denote the **average strategy** from  $a$  to  $b$  as

$$\bar{\sigma}_p^{[a,b]} := \sum_{i=a}^b \frac{\sigma_p^i}{b - a + 1} \quad (2.8)$$

Given a sequence  $\sigma^0, \dots, \sigma^{t-1}$  of strategy profiles, we denote the average Player  $p$  regret as

$$\begin{aligned} r_p^t &:= \max_{\sigma_p^*} \sum_{i=0}^{t-1} (u_p^{(\sigma_p^*, \sigma_{-p}^i)} - u_p^{\sigma^i}) / t \\ &= \max_{\sigma_p^*} u_p^{(\sigma_p^*, \bar{\sigma}_{-p}^{[0, t-1]})} - \sum_{i=0}^{t-1} u_p^{\sigma^i} / t \end{aligned} \quad (2.9)$$

The **exploitability** of a strategy profile  $\sigma$  is a measurement of how much expected utility each player could gain by switching their strategy:

$$\begin{aligned} \text{expl}(\sigma) &:= \max_{\sigma_1^*} u_1^{(\sigma_1^*, \sigma_2)} - u_1^\sigma + \max_{\sigma_2^*} u_2^{(\sigma_1, \sigma_2^*)} - u_2^\sigma \\ &= \max_{\sigma_1^*} u_1^{(\sigma_1^*, \sigma_2)} + \max_{\sigma_2^*} u_2^{(\sigma_1, \sigma_2^*)} \quad \text{by zero-sum} \end{aligned} \quad (2.10)$$

Achieving zero exploitability – a Nash equilibrium [Nash, 1950] – is possible. In two player, zero-sum games, finding a strategy with low exploitability is a reasonable goal for good play.



# 3. Revisiting CFR+ and Alternating Updates

This chapter is based on [Burch et al., 2019].

## 3.1 Introduction

CFR<sup>+</sup> was introduced [Tammelin, 2014] as an algorithm for approximately solving imperfect information games, and was subsequently used to essentially solve the game of heads-up limit Texas Hold'em poker [Bowling et al., 2015]. Another paper associated with the poker result gives a correctness proof for CFR<sup>+</sup>, showing that approximation error approaches zero [Tammelin et al., 2015].

CFR<sup>+</sup> is a variant of the CFR algorithm [Zinkevich et al., 2007], with much better empirical performance than CFR. One of the CFR<sup>+</sup> changes is switching from simultaneous updates to alternately updating a single player at a time. A crucial step in proving the correctness of both CFR and CFR<sup>+</sup> is linking regret, a hindsight measurement of performance, to exploitability, a measurement of the solution quality.

Later work pointed out a problem with the CFR<sup>+</sup> proof [Farina et al., 2019], noting that the CFR<sup>+</sup> proof makes reference to a folk theorem making the necessary link between regret and exploitability, but fails to satisfy the theorem's requirements due to the use of alternating updates in CFR<sup>+</sup>. Farina [Farina et al., 2019] give an example of a sequence of updates which lead to zero regret for both players, but high exploitability.

We state a version of the folk theorem that links alternating update regret and exploitability, with an additional term in the exploitability bound relating to strategy improvement. By proving that CFR and CFR<sup>+</sup> generate improved strategies, we can give a new correctness proof for CFR<sup>+</sup>, recovering the original bound on approximation error.

## 3.2 Definitions

We need a fairly large collection of definitions to get to the correctness proof. CFR and CFR<sup>+</sup> make use of the regret-matching algorithm [Hart and Mas-Colell, 2000] and regret-matching<sup>+</sup> algorithm [Tammelin, 2014], respectively, and we need to show some properties of these component algorithms. Both CFR and CFR<sup>+</sup> operate on extensive form games defined in chapter 2.

### 3.2.1 Regret-Matching and Regret-Matching<sup>+</sup>

Regret-matching is an algorithm for solving the online regret minimisation problem. External regret is a hindsight measurement of how well a policy did, compared to always selecting some action. Given a set of possible actions  $A$ , a sequence of value functions  $v^t \in \mathbb{R}^{|A|}$ , and sequence of policies  $\sigma^t \in \Delta^{|A|}$ , the regret for an action is

$$\begin{aligned} r^{t+1} &:= r^t + v^t - \sigma^t \cdot v^t \\ r^0 &:= 0 \end{aligned} \tag{3.1}$$

An online regret minimisation algorithm specifies a policy  $\sigma^t$  based on past value functions and policies, such that  $\max_a r_a^t/t \rightarrow 0$  as  $t \rightarrow \infty$ .

Let  $x^+ := \max(x, 0)$ ,  $\mathbf{x}^+ := [x_1^+, \dots, x_n^+]$ , and

$$\sigma_{\text{rm}}(\mathbf{x}) := \begin{cases} \mathbf{x}^+ / (\mathbf{1} \cdot \mathbf{x}^+) & \text{if } \exists a \text{ s.t. } x_a > 0 \\ \mathbf{1}/|A| & \text{otherwise} \end{cases} \quad (3.2)$$

Then for any  $t \geq 0$ , regret-matching uses a policy

$$\sigma^t := \sigma_{\text{rm}}(\mathbf{r}^t) \quad (3.3)$$

Regret-matching<sup>+</sup> is a variant of regret-matching that stores a set of non-negative regret-like values

$$\begin{aligned} \mathbf{q}^{t+1} &:= (\mathbf{q}^t + \mathbf{v}^t - \sigma^t \cdot \mathbf{v}^t)^+ \\ \mathbf{q}^0 &:= \mathbf{0} \end{aligned} \quad (3.4)$$

and uses the same regret-matching mapping from stored values to policy

$$\sigma^t := \sigma_{\text{rm}}(\mathbf{q}^t) \quad (3.5)$$

### 3.2.2 CFR and CFR<sup>+</sup>

CFR and its variant CFR<sup>+</sup> are both algorithms for finding an extensive form game strategy with low exploitability. They are all iterative self-play algorithms that track the average of a current strategy that is based on many loosely coupled regret minimisation problems.

CFR and CFR<sup>+</sup> track regret-matching values  $\mathbf{r}^t(I)$  or regret-matching<sup>+</sup> values  $\mathbf{q}^t(I)$  respectively, for all  $I \in \mathcal{I}$ . At time  $t$ , CFR and CFR<sup>+</sup> use strategy profile  $\sigma^t(I) := \sigma_{\text{rm}}(\mathbf{r}^t(I))$  and  $\sigma^t(I) := \sigma_{\text{rm}}(\mathbf{q}^t(I))$ , respectively. When doing alternating updates, with the first update done by Player 1, the values used for updating regrets are

$$\mathbf{v}^t(I) := \begin{cases} \mathbf{v}^{\sigma^t(I)} & \text{if } p(I) = 1 \\ \mathbf{v}^{(\sigma_1^{t+1}, \sigma_2^t)}(I) & \text{if } p(I) = 2 \end{cases} \quad (3.6)$$

and the output of CFR is the profile of average strategies  $(\bar{\sigma}_1^{[1,t]}, \bar{\sigma}_2^{[0,t-1]})$ , while the output of CFR<sup>+</sup> is the profile of weighted average strategies  $(\frac{2}{t^2+t} \sum_{i=1}^t i \sigma_1^i, \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \sigma_2^i)$ .

## 3.3 Theoretical Results

The CFR<sup>+</sup> proof of correctness [Tammelin et al., 2015] references a folk theorem that links regret and exploitability. [Farina et al., 2019] show that the folk theorem only applies to simultaneous updates, not alternating updates, giving an example of a sequence of alternating updates with no regret but constant exploitability [Farina et al., 2019]. Their observation is reproduced below using the definitions from this work.

**Observation 3.1.** *Let  $P = \{X, Y\}$ ,  $A = \{0, 1\}$ , and  $Z = \{00, 01, 10, 11\}$ . A game consists of each player selecting one action. Let  $u_X(11) = 1$ , and  $u_X(z) = 0$  for all  $z \neq 11$ . Consider the sequence of strategies  $\sigma_X^t = \sigma_Y^t = t \bmod 2$ , with Player X regrets computed using  $\mathbf{v}^{(\sigma_X^t, \sigma_Y^t)}$  and Player Y regrets computed using  $\mathbf{v}^{(\sigma_X^{t+1}, \sigma_Y^t)}$ . Then at any time  $2T$  the accumulated regret for both players is 0 and the average strategy is  $\bar{\sigma}_X^{[1,2T]} = \bar{\sigma}_Y^{[0,2T-1]} = \mathbf{0.5}$ , with exploitability  $\text{expl}(\bar{\sigma}_X^{[1,2T]}, \bar{\sigma}_Y^{[0,2T-1]}) = 0.5$ . So both players have 0 regret, but the exploitability does not approach 0.*

As a first step in correcting the CFR<sup>+</sup> proof, we introduce an analogue of the folk theorem, linking alternating update regret and exploitability.

**Theorem 3.2.** *Let  $\sigma^t$  be the strategy profile at some time  $t$ , and  $r_p^t$  be the regrets computed using alternating updates so that Player 1 regrets are updated using  $v^{(\sigma_1^t, \sigma_2^t)}$  and Player 2 regrets are updated using  $v^{(\sigma_1^{t+1}, \sigma_2^t)}$ . If the regrets are bounded by  $r_p^t \leq \epsilon_p$ , then the exploitability of  $(\bar{\sigma}_1^{[1,t]}, \bar{\sigma}_2^{[0,t-1]})$  is bounded by  $\epsilon_1 + \epsilon_2 - \frac{1}{t} \sum_{i=0}^{t-1} (u_1^{(\sigma_1^{i+1}, \sigma_2^i)} - u_1^{(\sigma_1^i, \sigma_2^i)})$ .*

*Proof.* Consider the sum of regrets for both players,  $r_1^t + r_2^t$

$$\begin{aligned} &= \max_{\sigma_1^*} u_1^{(\sigma_1^*, \bar{\sigma}_2^{[0,t-1]})} - \frac{1}{t} \sum_{i=0}^{t-1} u_1^{(\sigma_1^i, \sigma_2^i)} + \max_{\sigma_2^*} u_2^{(\bar{\sigma}_1^{[1,t]}, \sigma_2^*)} - \frac{1}{t} \sum_{i=0}^{t-1} u_2^{(\sigma_1^{i+1}, \sigma_2^i)} && \text{by Eq. 2.9} \\ &= \text{expl}(\bar{\sigma}_1^{[1,t]}, \bar{\sigma}_2^{[0,t-1]}) - \frac{1}{t} \sum_{i=0}^{t-1} (u_1^{(\sigma_1^i, \sigma_2^i)} + u_2^{(\sigma_1^{i+1}, \sigma_2^i)}) && \text{by Eq. 2.10} \end{aligned}$$

Given  $r_p^t \leq \epsilon_p$  for all players  $p$ , we have  $\text{expl}(\bar{\sigma}_1^{[1,t]}, \bar{\sigma}_2^{[0,t-1]})$

$$\begin{aligned} &\leq \epsilon_1 + \epsilon_2 + \frac{1}{t} \sum_{i=0}^{t-1} (u_1^{(\sigma_1^i, \sigma_2^i)} + u_2^{(\sigma_1^{i+1}, \sigma_2^i)}) \\ &= \epsilon_1 + \epsilon_2 + \frac{1}{t} \sum_{i=0}^{t-1} (u_1^{(\sigma_1^i, \sigma_2^i)} - u_1^{(\sigma_1^{i+1}, \sigma_2^i)}) && \text{by zero-sum} \\ &= \epsilon_1 + \epsilon_2 - \frac{1}{t} \sum_{i=0}^{t-1} (u_1^{(\sigma_1^{i+1}, \sigma_2^i)} - u_1^{(\sigma_1^i, \sigma_2^i)}) \end{aligned}$$

□

The gap between regret and exploitability in Observation 3.1 is now apparent as a trailing sum in Theorem 3.2. Each term in the sum measures the improvement in expected utility for Player 1 from time  $t$  to time  $t + 1$ . Motivated by this sum, we show that regret-matching, CFR, and their <sup>+</sup> variants generate new policies which are not worse than the current policy. Using these constraints, we construct an updated correctness proof for CFR<sup>+</sup>.

### 3.3.1 Regret-Matching and Regret-Matching<sup>+</sup> Properties

We will show that when using regret-matching or regret-matching<sup>+</sup>, the expected utility  $\sigma^{t+1} \cdot v^t$  is never less than  $\sigma^t \cdot v^t$ . To do this, we will need to show these algorithms have a couple of other properties. We start by showing that once there is at least one positive stored regret or regret-like value, there will always be a positive stored value.

**Lemma 3.3.** *For any  $t$ , let  $s^t$  be the stored value  $\mathbf{r}^t$  used by regret-matching or  $\mathbf{q}^t$  used by regret-matching<sup>+</sup>, and  $\sigma^t$  be the associated policy. Then for all  $t$  where  $\exists a \in A$  such that  $s_a^t > 0$ , there  $\exists b \in A$  such that  $s_b^{t+1} > 0$ .*

*Proof.* Consider any time  $t$  where  $\exists a \in A$  such that  $s_a^t > 0$ . The policy at time  $t$  is then

$$\sigma^t = \mathbf{s}^{t,+} / (\mathbf{1} \cdot \mathbf{s}^{t,+}) \quad \text{by Eqs. 3.2, 3.3, 3.5} \quad (3.7)$$

Consider the stored value  $s_a^{t+1}$ . With regret-matching  $s_a^{t+1} = r_a^{t+1} = r_a^t + v_a^t - \sigma^t \cdot v^t$  by Equation 3.1, and with regret-matching<sup>+</sup>  $s_a^{t+1} = q_a^{t+1} = (q_a^t + v_a^t - \sigma^t \cdot v^t)^+$  by Equation 3.4. For both algorithms, the value of  $s_a^{t+1}$  depends on  $v_a^t - \sigma^t \cdot v^t$ . There are two cases:

$$1. v_a^t - \sigma^t \cdot \mathbf{v}^t \geq 0$$

$$s_a^{t+1} > 0 \quad \text{by Lemma assumption, Eq. 3.1, 3.4}$$

$$2. v_a^t - \sigma^t \cdot \mathbf{v}^t < 0$$

$$\sigma_a^t (v_a^t - \sigma^t \cdot \mathbf{v}^t) < 0 \quad \text{by } s_a^t > 0, \text{ Eq. 3.7}$$

$$0 < \sigma^t \cdot \mathbf{v}^t - \sigma^t \cdot \mathbf{v}^t - \sigma_a^t (v_a^t - \sigma^t \cdot \mathbf{v}^t)$$

$$0 < \sum_{b \in A} \left( \sigma_b^t (v_b^t - \sigma^t \cdot \mathbf{v}^t) \right) - \sigma_a^t (v_a^t - \sigma^t \cdot \mathbf{v}^t) \quad \text{by } \sum_{b \in A} \sigma_b^t = 1$$

$$0 < \sum_{b \neq a} \left( \sigma_b^t (v_b^t - \sigma^t \cdot \mathbf{v}^t) \right)$$

$$\exists b \text{ s.t. } \sigma_b^t > 0, v_b^t - \sigma^t \cdot \mathbf{v}^t > 0 \quad \text{by } \sigma_{a'}^t \geq 0 \text{ for all } a' \in A$$

$$s_b^t > 0, v_b^t - \sigma^t \cdot \mathbf{v}^t > 0 \quad \text{by Eq. 3.7}$$

$$s_b^{t+1} > 0 \quad \text{by Eqs. 3.1, 3.4}$$

In both cases,  $\exists b$  such that  $s_b^{t+1} > 0$ . □

There is a corollary to Lemma 3.3, that regret-matching and regret-matching<sup>+</sup> never switch back to playing the default uniform random policy once they switch away from it.

**Corollary 3.4.** *When using regret-matching or regret-matching<sup>+</sup>, if there exists a time  $t$  such that  $\sigma^t = \mathbf{s}^{t,+} / (\mathbf{1} \cdot \mathbf{s}^{t,+})$  where  $\mathbf{s}^t$  are the stored regrets  $\mathbf{r}^t$  or regret-like values  $\mathbf{q}^t$  at time  $t$ , then  $\sigma^{t'} = \mathbf{s}^{t',+} / (\mathbf{1} \cdot \mathbf{s}^{t',+})$  for all  $t' \geq t$ .*

*Proof.* Assume that at some time  $t$ ,  $\sigma^t = \mathbf{s}^{t,+} / (\mathbf{1} \cdot \mathbf{s}^{t,+})$ . We can show by induction that  $\sigma^{t'} = \mathbf{s}^{t',+} / (\mathbf{1} \cdot \mathbf{s}^{t',+})$  for all  $t' \geq t$ . The base case  $t' = t$  of the hypothesis holds by assumption. Now, assume that  $\sigma^{t'} = \mathbf{s}^{t',+} / (\mathbf{1} \cdot \mathbf{s}^{t',+})$  for some time  $t' \geq t$ . We have

$$\exists a \in A \text{ s.t. } s_a^{t'} > 0 \quad \text{by Eq. 3.2}$$

$$\exists b \in A \text{ s.t. } s_b^{t'+1} > 0 \quad \text{by Lemma 3.3}$$

$$\sigma^{t'+1} = \mathbf{s}^{t'+1,+} / (\mathbf{1} \cdot \mathbf{s}^{t'+1,+}) \quad \text{by Eq. 3.2}$$

Therefore, by induction the hypothesis holds for all  $t' \geq t$ . □

**Lemma 3.5.** *For any  $t$ , let  $\mathbf{s}^t$  be the stored value  $\mathbf{r}^t$  used by regret-matching or  $\mathbf{q}^t$  used by regret-matching<sup>+</sup>, and  $\sigma^t$  be the associated policy. Then for all  $t$  and  $a \in A$ ,  $(s_a^{t+1,+} - s_a^{t,+})(v_a^t - \sigma^t \cdot \mathbf{v}^t) \geq 0$ .*

*Proof.* Consider whether  $v_a^t - \sigma^t \cdot \mathbf{v}^t$  is positive. There are two cases.

$$1. v_a^t - \sigma^t \cdot \mathbf{v}^t \leq 0$$

For regret-matching, where  $s_a^t = r_a^t$ , we have

$$r_a^{t+1} = r_a^t + v_a^t - \sigma^t \cdot \mathbf{v}^t \quad \text{by Eq. 3.1}$$

$$r_a^{t+1} \leq r_a^t$$

$$r_a^{t+1,+} \leq r_a^{t,+}$$

For regret-matching<sup>+</sup>, where  $s_a^t = q_a^t$ , we have

$$q_a^{t+1,+} = (q_a^t + v_a^t - \sigma^t \cdot \mathbf{v}^t)^+ \quad \text{by Eq. 3.4}$$

$$q_a^{t+1,+} = (q_a^{t,+} + v_a^t - \sigma^t \cdot \mathbf{v}^t)^+ \quad \text{by Eq. 3.4}$$

$$q_a^{t+1,+} \leq q_a^{t,+} \quad \text{by monotonicity of } (\cdot)^+$$

Therefore for both algorithms we have

$$\begin{aligned} s_a^{t+1,+} - s_a^{t,+} &\leq 0 \\ (s_a^{t+1,+} - s_a^{t,+})(v_a^t - \sigma^t \cdot \mathbf{v}^t) &\geq 0 \end{aligned}$$

2.  $v_a^t - \sigma^t \cdot \mathbf{v}^t > 0$

$$\begin{aligned} s_a^{t+1} &= s_a^t + v_a^t - \sigma^t \cdot \mathbf{v}^t && \text{by Eqs. 3.1, 3.4} \\ s_a^{t+1} &> s_a^t \\ s_a^{t+1,+} &\geq s_a^{t,+} \\ (s_a^{t+1,+} - s_a^{t,+})(v_a^t - \sigma^t \cdot \mathbf{v}^t) &\geq 0 \end{aligned}$$

In both cases, we have  $(s_a^{t+1,+} - s_a^{t,+})(v_a^t - \sigma^t \cdot \mathbf{v}^t) \geq 0$ .  $\square$

**Theorem 3.6.** *If  $\sigma^0, \sigma^1, \dots$  is the sequence of regret-matching or regret-matching<sup>+</sup> policies generated from a sequence of value functions  $\mathbf{v}^0, \mathbf{v}^1, \dots$ , then for all  $t$ ,  $\sigma^{t+1} \cdot \mathbf{v}^t \geq \sigma^t \cdot \mathbf{v}^t$ .*

*Proof.* Let  $\mathbf{s}^t$  be the stored value  $\mathbf{r}^t$  used by regret-matching or  $\mathbf{q}^t$  used by regret-matching<sup>+</sup>. Consider whether all components of  $\mathbf{s}^t$  or  $\mathbf{s}^{t+1}$  are negative. By Lemma 3.3 we know that it can not be the case that  $\exists a s_a^t > 0$  and  $\forall b s_b^{t+1} \leq 0$ . This leaves three cases.

1.  $\forall a s_a^t \leq 0$  and  $\forall a s_a^{t+1} \leq 0$

$$\begin{aligned} \sigma^t &= \sigma^{t+1} = \mathbf{1}/|A| && \text{by Eqs. 3.2, 3.3, 3.5} \\ \sigma^{t+1} \cdot \mathbf{v}^t &= \sigma^t \cdot \mathbf{v}^t \end{aligned}$$

2.  $\forall a s_a^t \leq 0$  and  $\exists a s_a^{t+1} > 0$

$$\begin{aligned} \sigma^{t+1} &= \mathbf{s}^{t+1,+} / (\mathbf{1} \cdot \mathbf{s}^{t+1,+}) && \text{by Eqs. 3.2, 3.3, 3.5} \\ \forall b, \sigma_b^{t+1} > 0 &\implies s_b^{t+1} > 0 \\ \forall b, \sigma_b^{t+1} > 0 &\implies v_b^t > \sigma^t \cdot \mathbf{v}^t && \text{by Eqs. 3.1, 3.4} \\ \sum_{b \in A} \sigma_b^{t+1} v_b^t &> \sum_b \sigma_b^{t+1} \sigma^t \cdot \mathbf{v}^t && \text{by } \sigma_b^{t+1} \geq 0 \\ \sigma^{t+1} \cdot \mathbf{v}^t &> \sigma^t \cdot \mathbf{v}^t && \text{by } \sum_{b \in A} \sigma_b^{t+1} = 1 \end{aligned}$$

3.  $\exists a s_a^t > 0$  and  $\exists b s_b^{t+1} > 0$

Let

$$\sigma(\mathbf{w}) := \mathbf{w}^+ / (\mathbf{w}^+ \cdot \mathbf{1}) \tag{3.8}$$

Then we have

$$\sigma^t = \sigma(\mathbf{s}^t), \sigma^{t+1} = \sigma(\mathbf{s}^{t+1}) \tag{3.9} \text{ by Eqs. 3.2, 3.3, 3.5}$$

Consider any ordering  $a_1, a_2, \dots, a_{|A|}$  of actions such that  $b \prec a$ . Let

$$w_j^i := \begin{cases} s_j^{t+1} & j \leq i \\ s_j^t & j > i \end{cases} \tag{3.10}$$

Note that  $\forall i < a, w_a^i = s_a^t > 0$ , and  $\forall i \geq a, w_b^i = s_b^{t+1} > 0$ , so that  $\sigma(\mathbf{w}^i)$  is always well-defined. We can show by induction that for all  $i \geq 0$

$$\sigma(\mathbf{w}^i) \cdot \mathbf{v}^t \geq \sigma^t \cdot \mathbf{v}^t \tag{3.11}$$

For the base case of  $i = 0$ , we have

$$\begin{aligned} \mathbf{w}^0 &= \mathbf{s}^t && \text{by Eq. 3.10} \\ \boldsymbol{\sigma}(\mathbf{w}^0) \cdot \mathbf{v}^t &= \boldsymbol{\sigma}(\mathbf{s}^t) \cdot \mathbf{v}^t \\ \boldsymbol{\sigma}(\mathbf{w}^0) \cdot \mathbf{v}^t &= \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eq. 3.9} \end{aligned}$$

Now assume that Equation 3.11 holds for some  $i \geq 0$ . By construction,

$$\forall j \neq i + 1, w_j^{i+1} = w_j^i \quad \text{by Eq. 3.10} \quad (3.12)$$

For notational convenience, let  $\Delta_w := w_{i+1}^{i+1,+} - w_{i+1}^{i,+} = s_{i+1}^{t+1,+} - s_{i+1}^{t,+}$ .

$$\begin{aligned} &\boldsymbol{\sigma}(\mathbf{w}^{i+1}) \cdot \mathbf{v}^t - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t \\ &= \frac{\mathbf{w}^{i+1,+} \cdot \mathbf{v}^t}{\mathbf{w}^{i+1,+} \cdot \mathbf{1}} - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eq. 3.8} \\ &= \frac{\Delta_w v_{i+1}^t + \mathbf{w}^{i,+} \cdot \mathbf{v}^t}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eqs. 3.10, 3.12} \\ &= \frac{\Delta_w v_{i+1}^t + (\mathbf{w}^{i,+} \cdot \mathbf{1}) \boldsymbol{\sigma}(\mathbf{w}^i) \cdot \mathbf{v}^t}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eq. 3.8} \\ &\geq \frac{\Delta_w v_{i+1}^t + (\mathbf{w}^{i,+} \cdot \mathbf{1}) \boldsymbol{\sigma}^t \cdot \mathbf{v}^t}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by ind. hypothesis} \\ &= \frac{\Delta_w v_{i+1}^t + (\mathbf{w}^{i,+} \cdot \mathbf{1}) \boldsymbol{\sigma}^t \cdot \mathbf{v}^t}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} - \frac{\Delta_w \boldsymbol{\sigma}^t \cdot \mathbf{v}^t + (\mathbf{w}^{i,+} \cdot \mathbf{1}) \boldsymbol{\sigma}^t \cdot \mathbf{v}^t}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} \\ &= \frac{\Delta_w (v_{i+1}^t - \boldsymbol{\sigma}^t \cdot \mathbf{v}^t)}{\Delta_w + \mathbf{w}^{i,+} \cdot \mathbf{1}} \\ &\geq 0 && \text{by Lemma 3.5} \end{aligned}$$

$\boldsymbol{\sigma}(\mathbf{w}^{i+1}) \cdot \mathbf{v}^t \geq \boldsymbol{\sigma}^t \cdot \mathbf{v}^t$ , so by induction Equation 3.11 holds for all  $i \geq 0$ . In particular, we can now say

$$\begin{aligned} \boldsymbol{\sigma}(\mathbf{w}^{|\mathcal{A}|}) \cdot \mathbf{v}^t &\geq \boldsymbol{\sigma}^t \cdot \mathbf{v}^t \\ \boldsymbol{\sigma}(\mathbf{s}^{t+1}) \cdot \mathbf{v}^t &\geq \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eq. 3.10} \\ \boldsymbol{\sigma}^{t+1} \cdot \mathbf{v}^t &\geq \boldsymbol{\sigma}^t \cdot \mathbf{v}^t && \text{by Eq. 3.9} \end{aligned}$$

In all cases, we have  $\boldsymbol{\sigma}^{t+1} \cdot \mathbf{v}^t \geq \boldsymbol{\sigma}^t \cdot \mathbf{v}^t$ . □

### 3.3.2 CFR and CFR<sup>+</sup> Properties

We now show that CFR and CFR<sup>+</sup> have properties that are similar to Theorem 3.6. After a player updates their strategy, that player's counterfactual value does not decrease for any action at any of their information sets. Similarly, the expected value of the player's new strategy does not decrease. Finally, using the property of non-decreasing value, we give an updated proof of an exploitability bound for CFR<sup>+</sup>.

**Lemma 3.7.** *Let  $p$  be the player that is about to be updated in CFR or CFR<sup>+</sup> at some time  $t$ . Let  $\sigma_p^t$  be the current strategy for  $p$ , and  $\sigma_o$  be the opponent strategy  $\sigma_{-p}^t$  or  $\sigma_{-p}^{t+1}$  used by Equation 3.6. Then  $\forall I \in \mathcal{I}_p$  and  $\forall a \in A(I)$ ,  $v^{(\sigma_p^{t+1}, \sigma_o)}(I)_a \geq v^{(\sigma_p^t, \sigma_o)}(I)_a$ .*

*Proof.* We will use some additional terminology. Let the terminal states reached from  $I$  by action  $a \in A(I)$  be

$$Z(I, a) := \bigcup_{h \in I} Z(ha) \quad (3.13)$$

and for any descendant state of  $I$ , we will call the ancestor  $h$  in  $I$

$$h^I(j) := h \in I \text{ s.t. } h \sqsubseteq j \quad (3.14)$$

Let  $D(I, a)$  be the set of information sets which are descendants of  $I$  given action  $a \in A(I)$ , and let  $C(I, a)$  be the set of immediate children:

$$\begin{aligned} D(I, a) &:= \{J \in \mathcal{I}_{p(I)} \mid \exists h \in I, j \in J \text{ s.t. } ha \sqsubseteq j\} \\ C(I, a) &:= D(I, a) \setminus \bigcup_{J \in C(I, a), b \in A(J)} D(J, b) \end{aligned} \quad (3.15)$$

Note that by perfect recall, for  $J \in C(I, a)$ ,  $\exists h \in I$  such that  $ha \sqsubseteq j$  for all  $j \in J$ : if one state in  $J$  is reached from  $I$  by action  $a$ , all states in  $J$  are reached from  $I$  by action  $a$ . Let the distance of an information set from the end of the game be

$$d(I) := \begin{cases} \max_{a \in A(I), J \in C(I, a)} (d(J) + 1) & \text{if } \exists a \text{ s.t. } C(I, a) \neq \emptyset \\ 0 & \text{if } \forall a, C(I, a) = \emptyset \end{cases} \quad (3.16)$$

Using this new terminology, we can re-write

$$\begin{aligned} v_p^\sigma(I)_a &= \sum_{h \in I} \sum_{z \in Z(h)} \pi_{-p}^\sigma(z) \pi_p^\sigma(z \mid ha) u_p(z) && \text{by Eq. 2.7} \\ &= \sum_{z \in Z(I, a)} \pi_{-p}^\sigma(z) \pi_p^\sigma(z \mid h^I(z)a) u_p(z) && \text{by Eqs. 3.13, 3.14} \end{aligned} \quad (3.17)$$

We will now show that  $\forall i \geq 0$

$$\forall I \in \mathcal{I}_p \text{ s.t. } d(I) \leq i, \forall a \in A(I), v^{(\sigma^{t+1}, \sigma_o)}(I)_a \geq v^{(\sigma_p^t, \sigma_o)}(I)_a \quad (3.18)$$

For the base case  $i = 0$ , consider any  $I \in \mathcal{I}_p$  such that  $d(I) = 0$ . Given these assumptions,

$$\begin{aligned} \forall a \in A(I), C(I, a) &= \emptyset && \text{by Eqs. 3.15, 3.16} \\ \forall \sigma, \forall a \in A(I), \forall z \in Z(I, a), \pi_p^\sigma(z \mid h^I(z)a) &= 1 && \text{by Eq. 2.4} \end{aligned} \quad (3.19)$$

Now consider  $v_p^{(\sigma_p^{t+1}, \sigma_o)}(I)_a$

$$\begin{aligned} &= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^{t+1}, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 3.17} \\ &= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 2.3} \\ &= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 3.19} \\ &= v^{(\sigma_p^t, \sigma_o)}(I)_a && \text{by Eq. 3.17} \end{aligned}$$

Assume the inductive hypothesis, Equation 3.18, holds for some  $i \geq 0$ . If  $\forall I \in \mathcal{I}_p, d(I) \leq i$ , Equation 3.18 trivially holds for  $i + 1$ . Otherwise, consider any  $I \in \mathcal{I}_p$  such that  $d(I) = i + 1$ .

Let  $T(I, a)$  be the (possibly empty) set of terminal histories in  $Z(I, a)$  that do not pass through another information set in  $\mathcal{I}_p(I)$ .

$$T(I, a) := Z(I, a) \setminus \bigcup_{J \in C(I, a), b \in A(J)} Z(J, b) \quad (3.20)$$

Because we require players to have perfect recall, terminal histories which pass through different child information sets are disjoint sets.

$$Z(J, b) \cap Z(J', b') = \emptyset \iff J = J', b = b'$$

Therefore, we can construct a partition  $\mathcal{P}$  of  $Z(I, a)$  from these disjoint sets and the terminal histories  $T(I, a)$  which do not pass through any child information set.

$$\mathcal{P} := \{Z(J, b) \mid J \in C(I, a), b \in A(J)\} \cup \{T(I, a)\} \quad (3.21)$$

Note that by the induction assumption, because  $d(I) = i + 1$

$$\begin{aligned} \forall J \in C(I, a), d(J) &\leq i && \text{by Eqs. 3.15, 3.16} \\ \forall J \in C(I, a), b \in A(J), v^{(\sigma_p^{t+1}, \sigma_o)}(J)_b &\geq v^{(\sigma_p^t, \sigma_o)}(J)_b \end{aligned} \quad (3.22)$$

Given this, we have  $v^{(\sigma_p^{t+1}, \sigma_o)}(I)_a$

$$\begin{aligned} &= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^{t+1}, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 3.17} \\ &= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 2.3} \\ &= \sum_{J \in C(I, a)} \sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) \\ &\quad + \sum_{z \in T(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eq. 3.21} \\ &= \sum_{J \in C(I, a)} \sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) \\ &\quad + \sum_{z \in T(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z \mid h^I(z)a) u_p(z) && \text{by Eqs. 2.4, 3.20} \end{aligned} \quad (3.23)$$

Looking at the terms inside  $\sum_J$  we have

$$\begin{aligned} &\sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^I(z)a) u_p(z) \\ &= \sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \sigma_p^{t+1}(J)_b \pi_p^{(\sigma_p^{t+1}, \sigma_o)}(z \mid h^J(z)b) u_p(z) && \text{by Eqs. 2.4, 3.15} \\ &= \sum_{b \in A(J)} \sigma_p^{t+1}(J)_b v^{(\sigma_p^{t+1}, \sigma_o)}(J)_b && \text{by Eq. 3.17} \\ &= \sigma_p^{t+1}(J) \cdot v^{(\sigma_p^{t+1}, \sigma_o)}(J) \\ &\geq \sigma_p^{t+1}(J) \cdot v^{(\sigma_p^t, \sigma_o)}(J) && \text{by Eq. 3.22} \\ &\geq \sigma_p^t(J) \cdot v^{(\sigma_p^t, \sigma_o)}(J) && \text{by Theorem 3.6} \\ &= \sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z \mid h^I(z)a) u_p(z) \end{aligned}$$



Substituting the terms back into Equation 3.23, we have  $v^{(\sigma_p^{t+1}, \sigma_o)}(I)_a$

$$\begin{aligned}
&\geq \sum_{J \in C(I, a)} \sum_{b \in A(J)} \sum_{z \in Z(J, b)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z | h^I(z) a) u_p(z) \\
&\quad + \sum_{z \in T(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z | h^I(z) a) u_p(z) \\
&= \sum_{z \in Z(I, a)} \pi_{-p}^{(\sigma_p^t, \sigma_o)}(z) \pi_p^{(\sigma_p^t, \sigma_o)}(z | h^I(z) a) u_p(z) && \text{by Eq. 3.21} \\
&= v^{(\sigma_p^t, \sigma_o)}(I)_a && \text{by Eq. 3.17}
\end{aligned}$$

Therefore Equation 3.18 holds for  $i + 1$ , and by induction holds for all  $i$ . In particular, it holds for  $i = \max_{I \in \mathcal{I}_p} d(I)$ , and applies to all  $I \in \mathcal{I}_p$ .  $\square$

**Theorem 3.8.** *Let  $p$  be the player that is about to be updated in CFR or CFR<sup>+</sup> at some time  $t$ . Let  $\sigma_p^t$  be the current strategy for  $p$ , and  $\sigma_o$  be the opponent strategy  $\sigma_{-p}^t$  or  $\sigma_{-p}^{t+1}$  used by the values defined in Equation 3.6. Then  $u_p^{(\sigma_p^{t+1}, \sigma_o)} \geq u_p^{(\sigma_p^t, \sigma_o)}$ .*

*Proof.* This immediately follows from Lemma 3.7 and  $u_p^\sigma = v^\sigma(I_p^\emptyset)_{a_0}$ .  $\square$

As a corollary of Theorems 3.2 and 3.8, when using alternating updates with either CFR or CFR<sup>+</sup>, the average strategy  $(\bar{\sigma}_1^{[1, t]}, \bar{\sigma}_2^{[0, t-1]})$  has  $\mathcal{O}(\sqrt{t})$  exploitability. From the original papers, both algorithms have an  $\mathcal{O}(\sqrt{t})$  regret bound, and the trailing sum in Theorem 3.2 is non-negative by Theorem 3.8. However, this only applies to a uniform average, so we need yet another theorem to bound the exploitability of the CFR<sup>+</sup> weighted average.

**Theorem 3.9.** *Let  $\sigma^t$  be the CFR<sup>+</sup> strategy profile at some time  $t$ , using alternating updates so that Player 1 regret-like values are updated using  $v^{(\sigma_1^t, \sigma_2^t)}$  and Player 2 regrets are updated using  $v^{(\sigma_1^{t+1}, \sigma_2^t)}$ . Let  $l = \max_{y, z \in Z} (u_1(y) - u_2(z))$  be the bound on terminal utilities. Then the exploitability of the weighted average strategy  $(\frac{2}{t^2+t} \sum_{i=1}^t i \sigma_1^i, \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \sigma_2^i)$  is bounded by  $2|\mathcal{I}|l\sqrt{k/t}$ , where  $k := \max_I |A(I)|$ .*

*Proof.* Consider two expanded sequences  $S^1$  and  $S^2$  of strategy profiles where the original strategy profile  $\sigma^t$  occurs  $t + 1$  times

$$\begin{aligned}
S^1 &:= \underbrace{(\sigma_1^0, \sigma_2^0)}_{1 \text{ copy}}, \underbrace{(\sigma_1^1, \sigma_2^1), (\sigma_1^1, \sigma_2^1)}_{2 \text{ copies}}, \dots, \underbrace{(\sigma_1^{t-1}, \sigma_2^{t-1}), \dots, (\sigma_1^{t-1}, \sigma_2^{t-1})}_{t \text{ copies}} \\
S^2 &:= \underbrace{(\sigma_1^1, \sigma_2^0)}_{1 \text{ copy}}, \underbrace{(\sigma_1^2, \sigma_2^1), (\sigma_1^2, \sigma_2^1)}_{2 \text{ copies}}, \dots, \underbrace{(\sigma_1^t, \sigma_2^{t-1}), \dots, (\sigma_1^t, \sigma_2^{t-1})}_{t \text{ copies}}
\end{aligned}$$

Then with respect to  $S^p$ , the total Player  $p$  regret for any information set  $I$  and action  $a$  is

$$r_p^{\frac{t^2+t}{2}}(I)_a \leq tl\sqrt{kt} \quad \text{by CFR}^+ \text{ Lemma 4 [Tammelin et al., 2015]}$$

and the average Player  $p$  regret is

$$\begin{aligned}
r_p^{\frac{t^2+t}{2}} &\leq \frac{2}{t^2+t} \sum_{I \in \mathcal{I}_p} \max_a r^{\frac{t^2+t}{2}}(I)_a && \text{by CFR Theorem 3 [Zinkevich et al., 2007]} \\
&\leq \frac{2}{t^2+t} \sum_{I \in \mathcal{I}_p} tl\sqrt{kt} \\
&\leq 2|\mathcal{I}_p|l\sqrt{k/t} && (3.24)
\end{aligned}$$

Because we have two sequences of profiles, we can not directly use Theorem 3.2, but we can follow the same form as that proof to get

$$\begin{aligned}
& r_1^{\frac{t^2+t}{2}} + r_2^{\frac{t^2+t}{2}} \\
&= \max_{\sigma_1^*} \sum_{i=0}^{\frac{t^2+t}{2}-1} \left( u_1^{(\sigma_1^*, S_{i,2}^1)} - u_1^{S_i^1} \right) \frac{2}{t^2+t} + \max_{\sigma_2^*} \sum_{i=0}^{\frac{t^2+t}{2}-1} \left( u_2^{(\sigma_2^*, S_{i,1}^2)} - u_2^{S_i^2} \right) \frac{2}{t^2+t} \\
&= \max_{\sigma_1^*} u_1^{(\sigma_1^*, \bar{S}_2^{[0, \frac{t^2+t}{2}-1]})} + \max_{\sigma_2^*} u_2^{(\bar{S}_1^{[0, \frac{t^2+t}{2}-1]}, \sigma_2^*)} - \sum_{i=0}^{\frac{t^2+t}{2}} \left( u_1^{S_i^1} + u_2^{S_i^2} \right) \frac{2}{t^2+t} \\
&= \max_{\sigma_1^*} u_1^{(\sigma_1^*, \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \sigma_2^i)} + \max_{\sigma_2^*} u_2^{(\frac{2}{t^2+t} \sum_{i=1}^t i \sigma_1^i, \sigma_2^*)} - \sum_{i=0}^{t-1} \frac{2(i+1)}{t^2+t} \left( u_1^{(\sigma_1^i, \sigma_2^i)} - u_1^{(\sigma_1^{i+1}, \sigma_2^i)} \right) \\
&= \exp \left( \frac{2}{t^2+t} \sum_{i=1}^t i \sigma_1^i, \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \sigma_2^i \right) - \sum_{i=0}^{t-1} \frac{2(i+1)}{t^2+t} \left( u_1^{(\sigma_1^i, \sigma_2^i)} - u_1^{(\sigma_1^{i+1}, \sigma_2^i)} \right)
\end{aligned}$$

Given Equation 3.24, we have  $\exp \left( \frac{2}{t^2+t} \sum_{i=1}^t i \sigma_1^i, \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \sigma_2^i \right)$

$$\leq 2|\mathcal{I}_1| l \sqrt{k/t} + 2|\mathcal{I}_2| l \sqrt{k/t} + \frac{2}{t^2+t} \sum_{i=0}^{t-1} (i+1) \left( u_1^{(\sigma_1^i, \sigma_2^i)} - u_1^{(\sigma_1^{i+1}, \sigma_2^i)} \right)$$

$$\leq 2|\mathcal{I}_1| l \sqrt{k/t} + 2|\mathcal{I}_2| l \sqrt{k/t}$$

by Theorem 3.8

$$= 2|\mathcal{I}| l \sqrt{k/t}$$

□

### 3.4 Conclusions

The original CFR<sup>+</sup> convergence proof makes unsupported use of the folk theorem linking regret to exploitability. We re-make the link between regret and exploitability for alternating updates, and provide a corrected CFR<sup>+</sup> convergence proof that recovers the original exploitability bound. The proof uses a specific property of CFR and CFR<sup>+</sup>, where for any single player update, both algorithms are guaranteed to never generate a new strategy which is worse than the current strategy.

With a corrected proof, we once again have a theoretical guarantee of correctness to fall back on, and can safely use CFR<sup>+</sup> with alternating updates, in search of its strong empirical performance without worrying that it might be worse than CFR.

The alternating update analogue of the folk theorem also provides some theoretical motivation for the empirically observed benefit of using alternating updates. Exploitability is now bounded by the regret minus the average improvement in expected values. While we proved that the improvement is guaranteed to be non-negative for CFR and CFR<sup>+</sup>, we would generally expect non-zero improvement on average, with a corresponding reduction in the bound on exploitability.

### 3.5 Author's contributions

While Neil Burch served as the primary author, I collaborated with him and Martin Schmid in developing the proof for the main theorem.

# 4. Bounding the Support Size in Extensive Form Games with Imperfect Information

This chapter is based on [Schmid et al., 2014]. In contrast to the other chapters from the thesis, results from this chapter hold also for general-sum and multi-player games.

## 4.1 Introduction

Arguably the most important solution concept in non-cooperative games is the notion of Nash equilibrium, where no player improves by deviating from this strategy profile. Support is defined as the set of actions played with non-zero probability and there are many crucial implications related to it.

Once the support is known, it is easy to compute the equilibrium in polynomial time even for general-sum games. Performance of some algorithms, namely the double-oracle algorithm for extensive form games, is tightly bound to the size of the support [Bošanský et al., 2013]. Other work shows that minimizing the support in abstracted games can lead to better strategies in the original game [Ganzfried et al., 2012]. Finally, it is advantageous to prefer strategies having a small support. Such strategies are both easier to store and play.

Extensive form games model a wide class of games with a varying levels of uncertainty. In the case of perfect information, there is an optimal strategy using only one action in any information set. In contrast, in some extensive games with imperfect information, the player can be forced to use all the possible actions to play optimally.

In this chapter, we focus on the relation between the level of uncertainty and the support size. We present an upper bound for the support size based on the uncertainty level.

Some games, such as Bayesian extensive games with observable actions or card games (such as no-limit Texas hold'em poker) have most of the information about the current state observable by all players, and therefore a low level of uncertainty. In these games, our bound guarantees the existence of Nash equilibrium having the support size considerably smaller than the number of all possible actions.

Instead of explicitly defining a level of uncertainty, we use the concept of the *public tree*. This concept provides a nice interpretation of uncertainty and *public actions*. Using the public tree, we present a new technique called the *equilibrium preserving transformation*, which transforms some equilibrium strategy profile into another. We provide an upper bound on the number of public actions used in the transformed Nash equilibrium.

Our approach also applies to games with non-observable actions, where it simply limits the number of public actions.

Applying our result to specific games, we present a new bound for the support size in these games.

For example, in no-limit Texas hold'em poker, there can be any finite number of actions available in some information sets. Our result implies the existence of an optimal strategy for which the number of actions used in every information set depends only on the number of players and the number of card combinations players can be dealt.

In Bayesian extensive games with observable actions, the bound equals to the number of different player types the chance can reveal.

Moreover, our proof is constructive. Given an extensive form game and an optimal strategy, the equilibrium preserving transformation finds another optimal strategy satisfying our bound in polynomial time.

## 4.2 Public Game Tree

The essential concept in our approach is that of a public game tree. Informally, a public game tree is a game view for an observer that knows no private information.

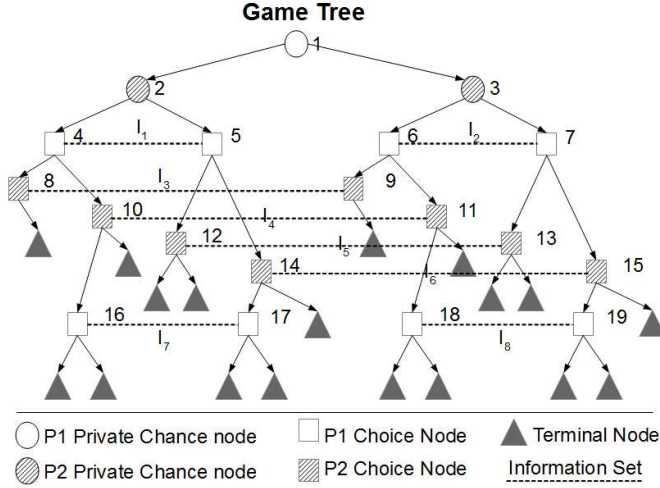


Figure 4.1: Extensive form game tree for one-card poker.

Public game tree, as introduced in [Johanson et al., 2011] is a partition of the histories.  $P$  is a **public partition** and  $\rho \in P$  is a **public state** if

- No two histories in the same information set are in different public states.
- Two histories in different public states have no descendants in the same public state.
- No public state contains both terminal and non-terminal histories (public states are either terminal or non-terminal).

The public tree offers a nice interpretation of imperfect information. For games with perfect information, the public tree is the same as the game tree. As the uncertainty grows, more and more information sets collapse into a single public state.

For the public game tree, we also define:

- The set of acting players in  $\rho \in P$  as  $p(\rho)$ .
- If the same player acts in all histories  $h \in \rho$ , then we define the acting player in  $\rho$  as:  $p(\rho) = p(h)$  for some  $h \in \rho$ .
- $C(\rho)$  to be set of child public states of  $\rho$ .
- For any public state  $\rho$  and any player  $i$ , we define  $\text{prev}(\rho, i)$  to be the set of player's last information sets he could play in before reaching  $\rho$ .  $I \in \text{prev}(\rho, i)$  if:
  - $p(I) = i$ , there are histories  $h \in \rho, h' \notin \rho, h' \in I: h' \sqsubseteq h$  and there is no history  $h'' \notin \rho: h'' \sqsubseteq h, h' \sqsubseteq h'', p(h'') = i$
- For any public state  $\rho$ , we define  $\text{last}(\rho)$  to be the last information sets the player  $p(\rho)$  plays before leaving  $\rho$ .  $I \in \text{last}(\rho)$  if  $I \in \rho$ , there is some history  $(h, a) \notin \rho$  and  $h \in \rho$ .

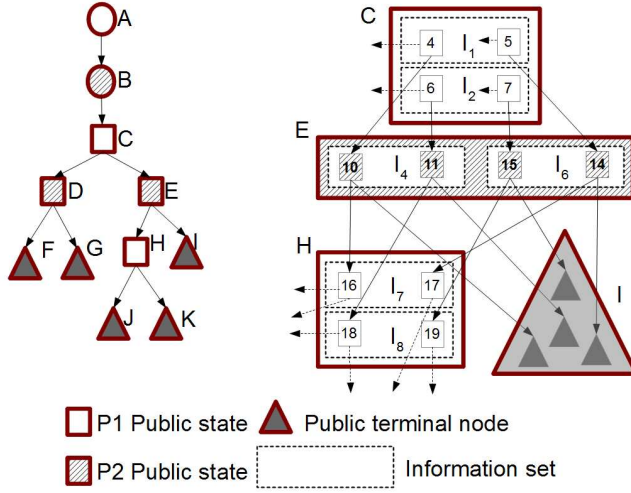


Figure 4.2: The public tree of a game in Figure (4.1). The structure of public states  $C, E, H, I$  is:

$$\begin{aligned}
prev(E, 1) &= \{I_1, I_2\}, \quad last(E) = \{I_4, I_6\} \\
C(E) &= \{H, I\}, \quad A(I_1 \rightsquigarrow E) = [4] \rightarrow [10] \\
A_p(C \rightsquigarrow E) &= \{(I_1, [4] \rightarrow [10]), (I_1, [5] \rightarrow [14]), (I_2, [6] \rightarrow [11]), (I_2, [7] \rightarrow [15])\}
\end{aligned}$$

- We call a non-terminal public state  $\rho$  **simple** if an observer knows which player acts and there are no information sets that contain actions going to a different public state as well as the actions that aren't. Formally,  $p(h) = p(h') \neq c$  for all  $h, h' \in \rho$  and no history has prefix in two different information states from  $last(\rho)$ .
- We define  $A(I \rightsquigarrow \rho)$  to be actions that a player can take in  $I \in \mathcal{I}$  in order to get to the public state  $\rho$ :

$$A(I \rightsquigarrow \rho) = \{a \in A(I) \mid h \in I, (h, a) \sqsubseteq h', h' \in \rho\}$$

- Intuitively, actions are the edges connecting any two nodes (histories) in Figure (4.1). We want **public actions** to be the edges connecting any two public states as seen on Figure (4.2). We define the public action going from public state  $\rho$  to  $\rho'$  as a set of pairs (*information set, action*):

$$A_p(\rho \rightsquigarrow \rho') = \{(I, a) \mid I \in last(\rho), a \in A(I \rightsquigarrow \rho')\}$$

See the box under the Figure (4.2) for examples of these definitions.

## 4.3 Strategies and Equilibrium

### 4.3.1 Support

The **support** of a strategy profile  $\sigma$ ,  $support^\sigma(I)$ , is the set of actions that the player players with non-zero probability in  $I$ .

We say that the public action  $A_p(\rho \rightsquigarrow \rho')$  is **supported** given the strategy profile  $\sigma$ , if for any  $(I, a) \in A_p(\rho \rightsquigarrow \rho')$ ,  $a \in support^\sigma(I)$ . Let  $support_p^\sigma(\rho)$  be the set of all supported public actions in  $\rho$ :

$$support_p^\sigma(\rho) = \{A_p(\rho \rightsquigarrow \rho') \mid A_p(\rho \rightsquigarrow \rho') \text{ is supported in } \sigma\}$$

### 4.3.2 Overall Regret

**Overall regret**  $R_i^\sigma$  is the difference between the player's utility given the strategy profile  $\sigma$  and the single strategy that would maximize his value:

$$R_i^\sigma = \max_{\sigma_i^* \in \Sigma_i} u_i(\sigma_i^*, \sigma_{-i}) - u_i(\sigma)$$

Clearly,  $R_i$  is always non-negative and there is a simple relation between the overall regret and Nash equilibrium:

$$R_i^\sigma = 0 \forall i \in N \iff \sigma \text{ is Nash equilibrium}$$

### 4.3.3 Counterfactual Values, Regret and Equilibrium

To show that some strategy profile  $\sigma$  is an equilibrium, we could show that the regret  $R_i^\sigma = 0$  for all players.

There is a way to bound this *full* regret  $R_i^\sigma$  using *partial* regrets in all information sets. These partial regrets are called counterfactual regrets.

The **Counterfactual utility**  $u_i(\sigma, I)$  is the expected utility given that information set  $I$  is reached and all players play using strategy  $\sigma$ , except that player  $i$  plays to reach  $I$  [Zinkevich et al., 2007]:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi_{-i}^\sigma(h) \pi^\sigma(h'|h) u_i(h')}{\pi_{-i}^\sigma(I)}$$

The **Counterfactual regret** [Zinkevich et al., 2007] is then defined as

$$R_i^\sigma(I) = \max_{a \in A(I)} \pi_{-i}^\sigma(I) (u_i(\sigma|_{I \rightarrow a}, I) - u_i(\sigma, I))$$

**Theorem 4.1.** [Zinkevich et al., 2007]

$$R_i^\sigma \leq \sum_{I \in \mathcal{I}_i} R_i^\sigma(I)$$

Note that in contrast to [Zinkevich et al., 2007], we are not interested in the relation between average regret and  $\epsilon$ -equilibrium (which holds only for two players, zero-sum games). We are interested only in bounding the regret of strategy profile  $\sigma$  using the Theorem (4.1). Directly from that theorem, we get the following corollary

**Corollary 4.2.** *If  $R_i^\sigma(I) = 0$  for all  $I \in \mathcal{I}_i, i \in N$ , the strategy profile  $\sigma$  forms a Nash equilibrium.*

The converse implication is not true in general. There can be an equilibrium with  $R_i^\sigma(I) > 0$  for some  $I \in \mathcal{I}_i$ . But there is always some Nash equilibrium for which  $R_i^\sigma(I) = 0$  for all  $I \in \mathcal{I}_i$ . (It is easy too see that if  $R_i^\sigma(I) > 0$  for some  $I$ , the player  $i$  plays not to reach  $I$ ,  $\pi_i^\sigma(I) = 0$ . One can make the counterfactual regret zero in these sets using backward induction.)

Finally, there is a simple way to show that the strategy profile  $\sigma'$  is a Nash equilibrium by comparing it with another Nash equilibrium.

**Lemma 4.3.** *Given a Nash equilibrium  $\sigma$ , for which  $\sum_{I \in \mathcal{I}_i} R_i^\sigma(I) = 0$ , if we find a strategy profile  $\sigma'$  such that for all  $i \in N, I \in \mathcal{I}_i$  and  $A \in A(I)$*

$$u_i(\sigma|_{I \rightarrow a}, I) = u_i(\sigma'|_{I \rightarrow a}, I)$$

and in every information set, the strategy  $\sigma'_i$  assigns a non zero probability only to actions used with non-zero probability in  $\sigma$ , the strategy profile  $\sigma'$  forms a Nash equilibrium for which  $\sum_{I \in \mathcal{I}_i} R_i^{\sigma'}(I) = 0$ .

The proof follows directly from the definition of counterfactual regret.

## 4.4 Main Theorem

Our main result shows the existence of optimal strategy with a limited number of supported public actions in simple public states.

**Theorem 4.4.** *In any finite extensive form game, there is an equilibrium strategy profile  $\sigma$  such that for every simple public state  $\rho$ , the number of supported public actions,  $|\text{support}_p^\sigma(\rho)|$ , is bounded by*

$$|\text{last}(\rho)| + \sum_{j \in N \setminus \{p(\rho)\}} \sum_{I \in \text{prev}(\rho, j)} |A(I \rightsquigarrow \rho)|$$

This bound has a nice interpretation for some specific games. For example in games with publicly observable actions, the bound depends only on the uncertainty presented by the chance.

We limit our technique only to simple public states, but it is possible to generalize the result. If there are some information sets containing actions going to a different public state as well as the actions that don't, it's easy to come up with an equivalent game where all actions are public. We are not aware of any well-known extensive form game having public states that are not simple.

## 4.5 Overview of our Approach

The core of our approach is a new technique we call **equilibrium preserving transformation**. We start with a Nash equilibrium for which  $R_i^\sigma(I) = 0$  for all  $i \in N$  and  $I \in \mathcal{I}_i$ , which is guaranteed to exist. An equilibrium preserving transformation carefully shifts probabilities locally, using the public tree. Given a public state  $\rho$ , it shifts some probabilities in information sets in  $\rho$ . The point is to keep the strategy optimal for all players, while minimizing the number of supported public actions.

Applying this transformation to a single public state  $\rho$ , we get a new equilibrium where the number of supported public actions satisfies our bound. Thus, we bound the public actions used in that information set, but we don't touch the strategies in any other public state.

Applying this transformation again to the new equilibrium, but in a different public state, we bound the number of supported public actions in that public state. Since we don't touch actions in any other public state, we do not violate the bound from the previous step.

Repeating this for all public states, we finally get a Nash equilibrium where the bound holds for all simple public states.

### 4.5.1 Optimality of the New Strategy

To show that the new strategy is an equilibrium, we leverage the concept of counterfactual values. These values are defined at the level of information sets and we can show that the

strategy is optimal by showing that these values remain unchanged thanks to Lemma (4.3). Since we change the strategy only in the information sets that are highly structured (they are in the same public state), it's relatively easy to compute the changed counterfactual values in all information sets.

## 4.6 Equilibrium Preserving Transformations

Given a Nash equilibrium  $\sigma$  where  $R_i^\sigma(I) = 0$  for all  $I \in \mathcal{I}$ , the core idea of our approach is to transform this strategy profile to another strategy profile  $\sigma'$ . We refer to this transformation as equilibrium preserving transformation or EPT and we denote the transformed strategy as  $\sigma' = EPT(\sigma)$ .

EPT shifts probabilities for a player locally, using the public tree. Given some public state  $\rho$ , we carefully change probabilities of outgoing actions. Since there's only one player acting in  $\rho$ , we change strategy only for this player  $p(\rho) = i$ .

We change the strategy only in  $last(\rho)$ , which are the last information sets in which the player acts just before reaching some  $\rho' \in C(\rho)$ .

We will continuously add some restrictions to our transformation and show what these restrictions imply for the new strategy profile  $\sigma'$ . Finally, we will see that if we transform the strategy such that all these restrictions hold,  $\sigma'$  is a Nash equilibrium.

The transformed strategy profile  $\sigma'$  differs from  $\sigma$  only in information sets  $I \in last(\rho)$

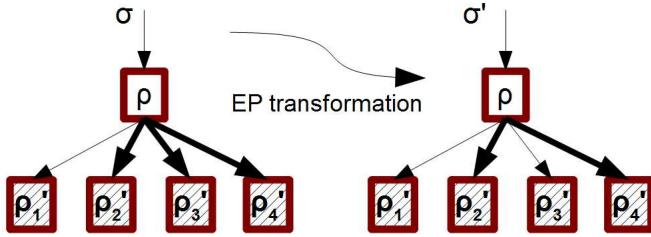


Figure 4.3: In the new strategy  $\sigma'$ , we bound the number of supported public actions (bold arrows).

The trick is to shift the probabilities in these information sets to use as few actions as possible, while keeping the strategy profile equilibrium. To do that, we keep the counterfactual values unchanged for all information sets and all actions.

To insure this, we impose two restrictions. The first one fixes counterfactual values for all information sets *after*  $\rho$ . The second one (together with the first one) fixes counterfactual values for all other information sets.

### 4.6.1 Information Sets after $\rho$

To fix the counterfactual values for the information sets after  $\rho$ , we do not shift the strategies arbitrarily. We only multiply some action probabilities with carefully chosen constants.

The last information sets the player  $p(\rho)$  acts in before leaving  $\rho$  are  $last(\rho)$ .

We consider the probability of all actions  $a \in A(I \rightsquigarrow \rho')$  for any  $I \in last(\rho)$ . Our transformation is only allowed to multiply these action probabilities using some constant, which we call  $\kappa(\rho \rightsquigarrow \rho')$ .

The probabilities of all outgoing actions from  $\rho$  to some child public state  $\rho'$  are all multiplied with some constant  $\kappa(\rho \rightsquigarrow \rho')$ . For all  $I \in last(\rho)$  and  $a \in A(I \rightsquigarrow \rho')$

$$\pi^{\sigma'}(I, a) = \kappa(\rho \rightsquigarrow \rho') \pi^\sigma(I, a) \quad (4.1)$$



We show later how to ensure that once we multiply the probabilities with corresponding  $\kappa$ , we get a valid strategy in every information set.

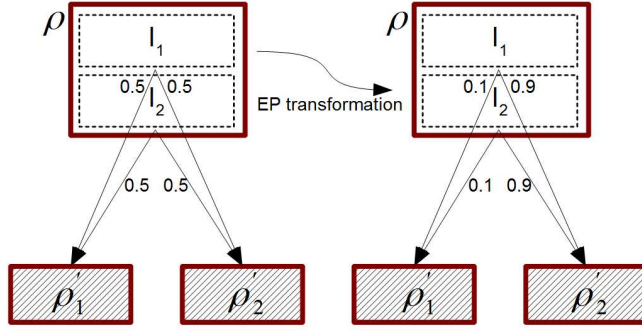


Figure 4.4: Transformation multiplies probabilities of all actions that go from  $\rho$  to some  $\rho'$  with corresponding  $\kappa$ . In this case  $\kappa(\rho \rightsquigarrow \rho'_1) = \frac{1}{5}$  and  $\kappa(\rho \rightsquigarrow \rho'_2) = \frac{9}{5}$ .

The reason why the transformation is not allowed to change the probabilities arbitrarily, but can only multiply action probabilities of actions going from  $\rho$  to  $\rho'$  with some corresponding  $\kappa(\rho \rightsquigarrow \rho')$ , is to keep the counterfactual values in all information sets after  $\rho$  unchanged.

**Lemma 4.5.** *If  $\sigma' = EPT(\sigma, \rho)$ , counterfactual values in all information sets after  $\rho$  remain unchanged.*

$$u_j(\sigma'|_{I \rightarrow a}, I) = u_j(\sigma|_{I \rightarrow a}, I) \text{ for all } j \in N \text{ and for all } I \text{ after } \rho.$$

*The proof (in Appendix A) follows directly from the definition of  $u_j(\sigma'|_{I \rightarrow a}, I)$ .*

Multiplying the strategies with corresponding  $\kappa$  is the only transformation we do. Clearly, if  $\kappa(\rho \rightsquigarrow \rho') = 0$  for some  $\rho'$ ,  $A_p(\rho \rightsquigarrow \rho')$  is not supported in the new strategy profile. In other words, we are interested in finding as many zero variables  $\kappa$  as possible.

## 4.6.2 Other Information Sets

The previous constraints keep the counterfactual values unchanged for all players and all information sets after  $\rho$ .

To ensure that the counterfactual values don't change in other information sets, we shift strategies such that the counterfactual values are unchanged in every  $I \in \text{prev}(\rho, j)$  for all players  $j \neq p(\rho)$ .

For all players  $j \neq p(\rho)$ , for all  $I \in \text{prev}(\rho, j)$  and for all  $a \in A(I \rightsquigarrow \rho)$

$$u_j(\sigma'|_{I \rightarrow a}, I) = u_j(\sigma|_{I \rightarrow a}, I) \tag{4.2}$$

The point is that if we keep counterfactual values unchanged only in these information sets, counterfactual values in all other information sets remain the same.

**Lemma 4.6.** *If  $\sigma' = EPT(\sigma, \rho)$ , counterfactual values in all information sets remain unchanged.*

*For the proof see Appendix A.*

## 4.7 System of Linear Equations to Find $\kappa$

Now we will show how to find  $\kappa(\rho \rightsquigarrow \rho')$  such that all restrictions are satisfied. We are interested only in  $\rho'$  such that  $A_p(\rho \rightsquigarrow \rho')$  is supported, all other actions have zero probability anyway. We find all  $\kappa$  using a systems of linear equations (linearity is the crucial part), with variables  $\kappa \geq 0$ .

First set of equations makes sure that  $\sigma'$  is a valid strategy. Adding another set of equations ensures that the counterfactual values remain unchanged.

Finally, using the simple property of linear equations, there must be a basic solution having limited number of non-zero variables  $\kappa$ .

### 4.7.1 First System of Equations

First, we write a simple equation for every  $I \in \text{last}(\rho)$  to make sure that we get a valid strategy after multiplying with corresponding  $\kappa$

$$\sum_{\rho' \in C(\rho)} \sum_{a \in A(I \rightsquigarrow \rho')} \pi^\sigma(I, a) \kappa(\rho \rightsquigarrow \rho') = 1 \quad (4.3)$$

Since we write down this equation for every  $I \in \text{last}(\rho)$ , there are  $|\text{last}(\rho)|$  of equations in total. Note that these equations are indeed linear in the variable  $\kappa$ .

### 4.7.2 Second System of Equations

The second system of linear equations makes sure that the restriction (3) is satisfied.

First, we compute the counterfactual values for the strategy profile  $\sigma'$  using the variables  $\kappa$  and the strategy profile  $\sigma$ .

**Lemma 4.7.** *There are some constants  $c_0(I, a) \dots c_{|C(\rho)|}(I, a)$  such that the counterfactual utility for all players  $j \neq p(\rho)$ , for all  $I \in \text{prev}(\rho, j)$  and for all  $a \in A(I \rightsquigarrow \rho)$*

$$u_j(\sigma' |_{I \rightarrow a}, I) = c_0(I, a) + \sum_{i=\{1 \dots |C(\rho)|\}} \kappa(\rho \rightsquigarrow \rho'_i) c_i(I, a)$$

*For the proof see Appendix A.*

In the proof of the above lemma, we leverage the fact that the values are unchanged in all information sets  $\rho' \in \rho$ . The history either passes through some  $\rho'$  and its probability gets multiplied with corresponding  $\kappa$ , or it doesn't and the probability remains unchanged.

Using this result, we can simply add linear equations for all players  $j \neq p(\rho)$ , for all  $I \in \text{prev}(\rho, j)$  and for all  $a \in A(I \rightsquigarrow \rho)$

$$\sum_{i=\{1 \dots |C(\rho)|\}} \kappa(\rho \rightsquigarrow \rho'_i) c_i(I, a) = u_j(\sigma |_{I \rightarrow a}, I) - c_0 \quad (4.4)$$

Again, these equations are linear in the variable  $\kappa$ .

### 4.7.3 Final System of Equations

Putting together all equations from (4.3) and (4.4), we are interested in  $\kappa \geq 0$  such that

$$\forall I \in \text{last}(\rho)$$

$$\sum_{\rho' \in C(\rho)} \sum_{a \in A(I \rightarrow \rho')} \pi^\sigma(I, a) \kappa(\rho \rightarrow \rho') = 1$$

$$\forall j \in N \setminus \{p(\rho)\}, I \in \text{prev}(\rho, j), a \in A(I)$$

$$\sum_{i=\{1 \dots |C(\rho)|\}} \kappa(\rho \rightsquigarrow \rho'_i) c_i(I, a) = u_j(\sigma|_{I \rightarrow a}, I) - c_0 \quad (4.5)$$

Combining previous results, we get a straightforward corollary

**Corollary 4.8.** *Any solution to (4.5) defines a valid equilibrium preserving transformation.*

This polyhedron is clearly bounded and non-empty ( $\kappa = 1$  is a solution to (4.5)). Finally, we use the well known property of basic solutions. There must be some basic solution where the number of non-zero variables  $\kappa$  is no larger than the number of equations [Bertsimas and Tsitsiklis, 1997]. Since number of equations is

$$|\text{last}(\rho)| + \sum_{j \in N \setminus \{p(\rho)\}} \sum_{I \in \text{prev}(\rho, j)} |A(I \rightsquigarrow \rho)| \quad (4.6)$$

our main theorem is proven. Moreover, we can find this solution efficiently in polynomial time using linear programming [Ye, 1991].

## 4.8 Example Games

In this section, we mention few existing games and show how our bound applies to these. As far as we know, these are the first bounds on the support size presented for these games.

Games where the players see the actions of all other players are called games with publicly observable actions. The only uncertainty comes from the actions of chance. In these games, all public states are simple and all information sets in any public state  $\rho$  form  $\text{last}(\rho)$ .

Because all actions are public,  $|A(I \rightsquigarrow \rho)| = 1$  for all  $\rho$  and for all  $I \in \text{prev}(\rho, j)$ . Consequently, the second term of (4.6) becomes  $\sum_{j \in N \setminus \{p(\rho)\}} |\text{prev}(\rho, j)|$ .

Finally, the bound for supported public actions implies an upper bound on the size of  $\text{support}^\sigma(I)$  for every  $I \in \rho$ .

$$|\text{support}^\sigma(I)| \leq |\text{support}_p^\sigma(\rho)|$$

### 4.8.1 Bayesian Extensive Games with Observable Actions

Bayesian extensive games with observable actions [Osborne and Rubinstein, 1994, p. 231] are games with publicly observable actions, where the only uncertainty comes from the initial move of chance. Chance selects a player **type**  $\theta \in \Theta_i$  for each player  $i$  where  $\Theta_i$  is the **set of possible types** of player  $i$ . Because chance acts at the very beginning of the game, the number of information sets grouped in every public state  $\rho$ , equals  $|\Theta_{p(\rho)}| = |\text{last}(\rho)|$ . Similarly,  $|\text{prev}(\rho, j)| = |\Theta_j|$ .

**Corollary 4.9.** *For any Bayesian extensive games with observable actions, there's a Nash equilibrium where the size of  $\text{support}^\sigma(I)$  for any  $I \in \mathcal{I}$  is bounded by*

$$\sum_{i \in N} |\Theta_i| \quad (4.7)$$

## 4.8.2 No-limit Texas Hold'em Poker

In Texas hold'em poker, players are dealt two private cards out of a deck of 52 cards. Four betting rounds follow and dealer deals some more publicly visible cards between these betting rounds.

In no-limit version, players can bet any amount of money up to their stack in every betting round. For example in the 2014 AAAI Computer Poker Competition, there are up to 20000 actions available in information sets [Bard et al., 2013].

All betting is publicly visible, and the only uncertainty is about the private cards the players were dealt.  $|prev(\rho, j)| = last(\rho) = \binom{52}{2}$  for any player  $j$  and any public state  $\rho$ .

**Corollary 4.10.** *In Texas hold'em poker, there's a Nash equilibrium where the size of support $^\sigma(I)$  for any  $I \in \mathcal{I}$  is bounded by*

$$\binom{52}{2} |N| \tag{4.8}$$

Using some isomorphisms, we can further decrease the bound in some situations. In Texas hold'em poker, there are 169 non-isomorphic [Waugh, 2013] pairs in the first round (called *preflop*).

**Corollary 4.11.** *In Texas hold'em poker, there's a Nash equilibrium where the size of support $^\sigma(I)$  for all information set in the first round (preflop) is bounded by*

$$169 |N| \tag{4.9}$$

## 4.9 Conclusion

We present a new technique called equilibrium preserving transformation, that implies the existence of Nash equilibrium having a bounded support. Our bound shows a relation between the level of uncertainty and the support size.

For Bayesian extensive games with observable actions and card games, our bound implies relatively small support. Finally, given any Nash equilibrium, EPT finds another equilibrium having the bounded support in polynomial time.

## 4.10 Author's contributions

Original observation that there is a linear formulation that ensure upper limit of support size came from Martin Schmid. I have helped find precise formulation for the constraints on the value of opponent information sets, and to formalize the proofs.

# 5. Sound Algorithms in Imperfect Information Games

This chapter is based on [Šustr et al., 2020]

## 5.1 Introduction

From the very dawn of computer game research, search was a fundamental component of many algorithms. Turing’s chess algorithm from 1950 was able to think two moves ahead [Copeland, 2004], and Shannon’s work on chess from 1950 includes an extensive section on how an evaluation function can be used within search [Shannon, 1950]. Samuel’s checkers algorithm from 1959 already combines search and learning of a value function, approximated through a self-play method and bootstrapping [Samuel, 1959]. The combination of search and learning has been a crucial component in the remarkable milestones where computers outperformed their human counterparts in challenging games: DeepBlue in Chess [Campbell et al., 2002], AlphaGo in Go [Silver et al., 2016], DeepStack and Libratus in Poker [Moravčík et al., 2017, Brown and Sandholm, 2018].

Online methods for approximating Nash equilibria in sequential imperfect information games appeared only in the last few years [Lisý et al., 2015, Brown and Sandholm, 2017, Moravčík et al., 2017, Brown and Sandholm, 2018, 2019, Brown et al., 2020]. We thus investigate what it takes for an online algorithm to be sound in imperfect information settings. While it has been known that search with imperfect information is more challenging than with perfect information [Frank and Basin, 1998, Lisý et al., 2015], the problem is more complex than previously thought. Online algorithms “live” in a fundamentally different setting, and they need to be evaluated appropriately.

Previously, a common approach to evaluate online algorithms was to compute a corresponding offline strategy by “querying” the online algorithm at each state (“tabularization” of the strategy) [Lisý et al., 2015, Šustr et al., 2019]. One would then report the exploitability of the resulting offline strategy. We show that this is not generally possible and that naive tabularization can also lead to incorrect conclusions about the online algorithm’s worst-case performance. As a consequence we show that some algorithms previously considered to be sound are not.

We first give a simple example of how an online algorithm can lose to an adversary in a repeated game setting. Previously, such an algorithm would be considered optimal based on a naive tabularization. We build on top of this example to introduce a framework for properly evaluating an online algorithm’s performance. Within this framework, we introduce the definition of a sound and  $\epsilon$ -sound algorithm. Like the exploitability of a strategy in the offline setting, the soundness of an algorithm is a measure of its performance against a worst-case adversary. Importantly, this notion collapses to the previous notion of exploitability when the algorithm follows a fixed strategy profile.

We then introduce a consistency framework, a hierarchy that formally states in what sense an online algorithm plays “consistently” with an  $\epsilon$ -equilibrium. The hierarchy allows stating multiple bounds on the algorithm’s soundness, based on the  $\epsilon$ -equilibrium and consistency type. The stronger the consistency is in our hierarchy, the stronger are the bounds. This further illustrates the discrepancy of search in perfect and imperfect information settings, as these bounds sometimes differ for perfect and imperfect information games.

The definitions of soundness and the consistency hierarchy finally provide appropriate

tools to analyze online algorithms in imperfect information games. We thus inspect some of the previous online algorithms in a new light, bringing new insights into their worst-case performance guarantees. Namely, we focus on the Online Outcome Sampling (OOS) [Lisý et al., 2015] algorithm. Consider the following statement from the OOS publication: “We show that OOS is consistent, i.e., it is guaranteed to converge to an equilibrium strategy as search time increases. To the best of our knowledge, this is not the case for any existing online game playing algorithm. . . ’ The problem is that OOS provides only the weakest of the introduced consistencies — local consistency. As the local consistency gives no guarantee for imperfect information games (in contrast to perfect information games), OOS (and potentially other locally consistent algorithms) can be highly exploited by an adversary. The experimental section then confirms this issue for OOS in two small imperfect information games.

## 5.2 Background

We present our results using the recent formalism of factored-observations stochastic games [Kovařík et al., 2021]. The entirety of this chapter trivially applies to the extensive form formalism [Osborne and Rubinstein, 1994] as well\* (as we are only relying on the notion of states and rewards). We believe this choice of formalism makes it easier to incorporate our definitions in the future online algorithms, as sound search in imperfect information critically relies on the notion of common/public information [Burch et al., 2014, Seitz et al., 2019]. Indeed, all the recently introduced online algorithms in imperfect information games rely on these notions [Moravčík et al., 2017, Brown and Sandholm, 2018, Šustr et al., 2019].

**Definition 5.1.** *A factored-observations stochastic game is a tuple*

$$\mathcal{G} = \langle \mathcal{N}, \mathcal{W}, w^0, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle,$$

where:

- $\mathcal{N} = \{1, 2\}$  is a **player set**. We use symbol  $n$  for a player and  $-n$  for its opponent.
- $\mathcal{W}$  is a set of **world states** and  $w^0 \in \mathcal{W}$  is a designated initial world state.
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$  is a space of **joint actions**. The subsets  $\mathcal{A}_n(w) \subset \mathcal{A}_n$  and  $\mathcal{A}(w) = \mathcal{A}_1(w) \times \mathcal{A}_2(w) \subset \mathcal{A}$  specify the (joint) actions legal at  $w \in \mathcal{W}$ . For  $a \in \mathcal{A}$ , we write  $a = (a_1, a_2)$ .  $\mathcal{A}_n(w)$  for  $n \in \mathcal{N}$  are either all non-empty or all empty. A world state with no legal actions is **terminal**.
- After taking a (legal) joint action  $a$  at  $w$ , the **transition function**  $\mathcal{T}$  determines the next world state  $w'$ , drawn from the probability distribution  $\mathcal{T}(w, a) \in \Delta(\mathcal{W})$ .
- $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ , and  $\mathcal{R}_n(w, a)$  is the **reward** player  $n$  receives when a joint action  $a$  is taken at  $w$ .
- $\mathcal{O} = (\mathcal{O}_{\text{priv}(1)}, \mathcal{O}_{\text{priv}(2)}, \mathcal{O}_{\text{pub}})$  is the **observation function**, where  $\mathcal{O}_{(\cdot)} : \mathcal{W} \times \mathcal{A} \times \mathcal{W} \rightarrow \mathbb{O}_{(\cdot)}$  specifies the **private observation** that player  $n$  receives, resp. the **public observation** that every player receives, upon transitioning from world state  $w$  to  $w'$  via some joint action  $a$ .

---

\*Under the assumption the games are perfect-recall and 1-timeable [Kovařík et al., 2021].

A legal **world history** (or trajectory) is a finite sequence  $h = (w^0, a^0, w^1, a^1, \dots, w^t)$ , where  $w^k \in \mathcal{W}$ ,  $a^k \in \mathcal{A}(w^k)$ , and  $w^{k+1} \in \mathcal{W}$  is in the support of  $\mathcal{T}(w^k, a^k)$ . We denote the set of all legal histories by  $\mathcal{H}$ , and the set of all sub-sequences of  $h$  that are legal histories as  $\mathcal{H}(h) \subseteq \mathcal{H}$ .

Since the last world state in each  $h \in \mathcal{H}$  is uniquely defined, the notation for  $\mathcal{W}$  can be overloaded to work with  $\mathcal{H}$  (e.g.,  $\mathcal{A}(h) := \mathcal{A}(\text{the last } w \text{ in } h)$ ,  $h$  being terminal, ...). We use  $\mathcal{Z}$  to denote the set of all terminal histories, i.e. histories where the last world state is terminal.

The **cumulative reward** of  $n$  at  $h$  is  $\sum_{k=0}^{t-1} r_n^k := \sum_{k=0}^{t-1} \mathcal{R}_n(w^k, a^k)$ . When  $h$  is a terminal history, cumulative rewards can also be called **utilities**, and denoted as  $u_n(z)$ . We assume games are **zero-sum**, so  $u_n(z) = -u_{-n}(z) \forall z \in \mathcal{Z}$ . The maximum difference of utilities is  $\Delta = |\max_{z \in \mathcal{Z}} u_1(z) - \min_{z \in \mathcal{Z}} u_1(z)|$

Player  $n$ 's **information state** or **private history** at  $h = (w^0, a^0, w^1, a^1, \dots, w^t)$  is the action-observation sequence

$s_n(h) := (O_n^0, a_n^0, O_n^1, a_n^1, \dots, O_n^t)$ , where  $O_n^k = \mathcal{O}_n(w^{k-1}, a^{k-1}, w^k)$  and  $O_n^0$  is some initial observation. The space  $\mathcal{S}_n$  of all such sequences can be viewed as the **private tree** of  $n$ .

A **strategy profile** is a pair  $\sigma = (\sigma_1, \sigma_2)$ , where each (behavioral) **strategy**  $\sigma_n : s_n \in \mathcal{S}_n \mapsto \sigma_n(s_n) \in \Delta(\mathcal{A}_n(s_n))$  specifies the probability distribution from which player  $n$  draws their next action (conditional on having information  $s_n$ ). We denote the set of all strategies of player  $n$  as  $\Sigma_n$  and the set of all strategy profiles as  $\Sigma$ .

The **reach probability** of a history  $h \in \mathcal{H}$  under  $\sigma$  is defined as  $\pi^\sigma(h) = \pi_1^\sigma(h) \pi_2^\sigma(h) \pi_c^\sigma(h)$ , where each  $\pi_n^\sigma(h)$  is a product of probabilities of the actions taken by player  $n$  between the root and  $h$ , and  $\pi_c^\sigma(h)$  is the product of stochastic transitions. The **expected utility** for player  $n$  of a strategy profile  $\sigma$  is  $u_n(\sigma) = \sum_{z \in \mathcal{Z}} \pi^\sigma(z) u_n(z)$ .

We define a **best response** of player  $n$  to the other player's strategies  $\sigma_{-n}$  as a strategy  $br(\sigma_{-n}) \in \arg \max_{\sigma'_n \in \Sigma_n} u_n(\sigma'_n, \sigma_{-n})$  and **best response value**  $brv(\sigma_{-n}) = \max_{\sigma'_n \in \Sigma_n} u_n(\sigma'_n, \sigma_{-n})$ . The profile  $\sigma$  is an  **$\epsilon$ -Nash equilibrium** if  $(\forall n \in \mathcal{N}) : u_n(\sigma) \geq \max_{\sigma'_n \in \Sigma_n} u_n(\sigma'_n, \sigma_{-n}) - \epsilon$ , and we denote the set of all  $\epsilon$ -equilibrium strategies of player  $n$  as  $\mathcal{N}\mathcal{E}_n^\epsilon$ . The **strategy exploitability** is  $\text{expl}_n(\sigma_n) := [u_n(\sigma^*) - \min_{\sigma'_{-n} \in \Sigma_{-n}} u_n(\sigma_n, \sigma'_{-n})]$  where  $\sigma^*$  is an equilibrium strategy. The **game value**  $u^* = u_1(\sigma^*)$  is the utility player 1 can achieve under a Nash equilibrium strategy profile.

## 5.3 Online Algorithm

The environment we are concerned with is that of a repeated game, consisting of a sequence of individual matches. As a match progresses, the algorithm produces a strategy for a visited information state on-line: that is, once it actually observes the state. This common framework of repeated games is particularly suitable for analysis of online algorithms, as the online algorithm can be conditioned on the past experience (e.g. by trying to adapt to the opponent or by re-using parts of the previous computation). We are then interested in the accumulated reward of the agent during the span of the repeated game. Of particular interest will be the expected reward against a worst-case adversary.

### 5.3.1 Coordinated Matching Pennies

We now introduce a small imperfect information game that will be used throughout the article – “Coordinated Matching Pennies” (CMP). It is a variation on the well-known Matching Pennies game [Osborne and Rubinstein, 1994], where players choose either Heads or Tails

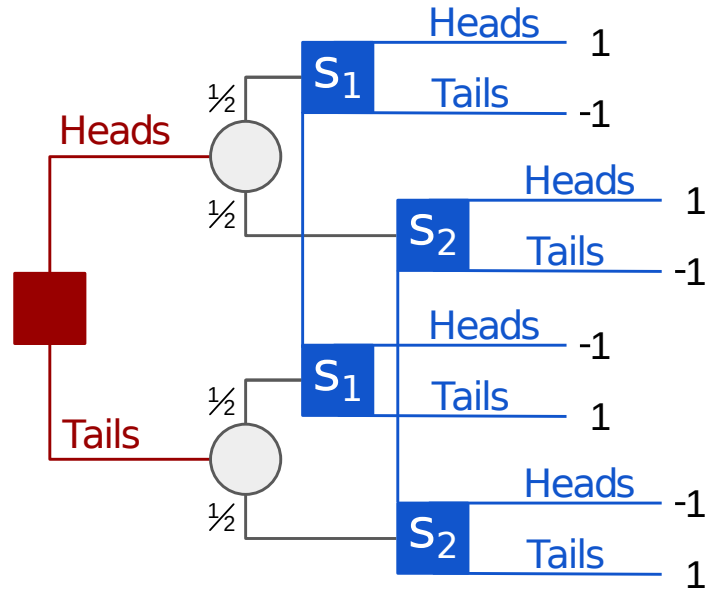


Figure 5.1: Coordinated Matching Pennies. After the first player acts, chance randomly chooses whether the second player will be playing in the information state  $s_1$  or  $s_2$ . The first player receives utility of 1 if players' actions match and  $-1$  if they mismatch.

and receive a utility of  $\pm 1$  if their actions (mis)match. For CMP, we additionally introduce a publicly observable chance event just after the first player acts. See Figure 5.1 for details.

Let  $p$  and  $q$  denote the probability of playing Heads in information states  $s_1$  and  $s_2$  respectively. An equilibrium strategy for the second player (Blue) is then any strategy where the average of  $p$  and  $q$  is  $\frac{1}{2}$ . He thus has to coordinate the actions between his two information states, while the first player has a unique uniform equilibrium strategy. Similar equilibrium coordination happens also in Kuhn Poker [Kuhn, 1950].

### 5.3.2 Naive Tabularization

We now show that if one naively tries to convert an online algorithm into a fixed strategy, the resulting exploitability is not always representative of the worst-case performance of the online algorithm. Consider the following algorithm `PlayCache` for the repeated game of CMP. `PlayCache` keeps an internal state, a cache – a mapping of information state to probability distribution over the actions, and it gradually fills the cache during the game.

Concretely, `PlayCache` plays for the second player as follows:

- Initialize algorithm's state  $\theta_0$  to an empty cache.
- Given an information state  $s$  visited during a game, there are three possible cases: i) The cache is empty: play Heads and store  $\{s, \text{Heads}\}$  into the cache. ii) The cache is non-empty and contains  $s$ : play the cached strategy for  $s$ . iii) The cache is non-empty and does not contain  $s$ : play Tails and store  $\{s, \text{Tails}\}$ .

Consider what happens if one tries to naively tabularize the `PlayCache` by querying the algorithm for all the information states. If we query the algorithm for states  $s_1, s_2$ , we get the resulting offline strategy  $s_1 : \text{Heads}, s_2 : \text{Tails}$ . Querying the algorithm for states in reverse order, i.e.  $s_2, s_1$  results in  $s_1 : \text{Tails}, s_2 : \text{Heads}$ . And while both of these offline strategies have zero exploitability, one can still exploit the algorithm during the repeated game. This follows from the fact that the very first time the `PlayCache` gets to act, it



always plays Heads. The first player can thus simply play Heads during the first match and is guaranteed to win the match. As we will show later, `PlayCache` falls within a class of algorithms that can be exploited, but where the average reward is guaranteed to converge to the game value as we repeatedly keep playing the game.

Where did this discrepancy between the exploitability of the tabularized strategy and the exploitability of the online algorithm come from? It is simply because the tabularized strategy does not properly describe the game dynamics of `PlayCache`. In fact, there is no fixed strategy that does so! We will now formalize an appropriate framework to describe the rewards and dynamics of online algorithms, which will allow us to define notions for the expected reward and the worst-case performance in the online setting.

### 5.3.3 Online Settings

The **repeated game**  $p$  consists of a finite sequence of  $k$  individual matches

$m = (z_1, z_2, \dots, z_k)$ , where each **match**  $z_i \in \mathcal{Z}$  is a sequence of world states and actions  $z_i = (w_i^0, a_i^0, w_i^1, a_i^1, \dots, a_i^{l_i-1}, w_i^{l_i})$ , ending in a terminal world state  $w_i^{l_i}$ . For each visited world state in the match, there is a corresponding information state, i.e. a player's private perspective of the game (for perfect information games, the notion of information state and world state collapse as the player gets to observe the world perfectly). An online algorithm  $\Omega$  then simply maps an information state observed during a match to a strategy, while possibly using its internal algorithm state (Def. 5.2).

Given two players that use algorithms  $\Omega_1, \Omega_2$ , we use  $P_{\Omega_1, \Omega_2}^k$  to denote the distribution over all the possible repeated games  $m$  of length  $k$  when these two players face each other. The average reward of  $m$  is  $\mathcal{R}_n(m) = 1/k \sum_{i=1}^k u_n(z_i)$  and we denote  $\mathbb{E}_{m \sim P_{\Omega_1, \Omega_2}^k} [\mathcal{R}_n(m)]$  to be the expected average reward when the players play  $k$  matches. From now on, if player  $n$  is not specified, we assume without loss of generality it is player 1. The proofs of the theorems can be found in the Appendix.

**Definition 5.2.** *Online algorithm  $\Omega$  is a function  $\mathcal{S}_n \times \Theta \mapsto \Delta(\mathcal{A}_n(s_n)) \times \Theta$ , that maps an information state  $s_n \in \mathcal{S}_n$  to the strategy  $\sigma_n(s_n) \in \Delta(\mathcal{A}_n(s_n))$ , while possibly making use of algorithm's state  $\theta \in \Theta$  and updating it. We denote the algorithm's **initial state** as  $\theta_0$ . A special case of an online algorithm is a **stateless algorithm**, where the output of the function is independent of the algorithm's state (thus independent of the previous matches). If the output depends on the algorithm's state, we say the algorithm is **stateful**.*

As the game progresses, the online algorithm produces strategies for the visited information states and updates its algorithm state. This allows it to potentially output different strategies for the same information state visited in different matches. We thus use  $\Omega^m(s_n)$  to denote the resulting strategy in the information state  $s_n$  after the algorithm has already played the matches  $m = z_1, \dots, z_k$ . Note that players can not visit the same information state twice in a single match.

**Remark 5.3.** *If we need to encode a stochastic algorithm, we can do it formally as taking the initial state to be a realization of a random variable. The initial state should be extended to encode how the algorithm should act (seemingly) randomly in any possible game-play situation beforehand.*

### 5.3.4 Soundness of Online Algorithm

We are now ready to formalize the desirable properties of an online algorithm in our settings. Exploitability, resp.  $\epsilon$ -equilibrium considers the expected utility of a fixed strategy against a

worst-case adversary in a single match. We thus define a similar concept for the settings of an online algorithm in a repeated game:  $(k, \epsilon)$ -soundness. Intuitively, an online algorithm is  $(k, \epsilon)$ -sound if and only if it is guaranteed the same reward as if it followed a fixed  $\epsilon$ -equilibrium after  $k$  matches.

**Definition 5.4.** For an  $(k, \epsilon)$ -sound online algorithm  $\Omega$ , the expected average reward against any opponent is at least as good as if it followed an  $\epsilon$ -Nash equilibrium fixed strategy  $\sigma$  for any number of matches  $k'$ :

$$\forall k' \geq k \forall \Omega_2 : \mathbb{E}_{m \sim P_{\Omega, \Omega_2}^{k'}} [\mathcal{R}(m)] \geq \mathbb{E}_{m \sim P_{\sigma, \Omega_2}^{k'}} [\mathcal{R}(m)]. \quad (5.1)$$

If algorithm  $\Omega$  is  $(k, \epsilon)$ -sound for  $\forall k \geq 1$ , we say the algorithm is  $\epsilon$ -sound.

Note that this definition guarantees that an online algorithm that simply follows a fixed  $\epsilon$ -equilibrium is  $\epsilon$ -sound. And while the online algorithm can certainly play as a fixed strategy, online algorithms are far from limited to doing so, e.g. `PlayCache` from Section 5.3. `PlayCache` is 1-sound ( $\epsilon = 1$ ) as this algorithm is highly exploitable in the first match. Additionally, an online algorithm may be sound ( $\epsilon = 0$ ), but there might not be any offline equilibrium that produces the same distribution of matches.

### 5.3.5 Response Game

To compute the expected reward  $\mathbb{E}_{m \sim P_{\Omega, \Omega_2}^{k'}} [\mathcal{R}(m)]$  as in Def. 5.4, we construct a repeated game [Osborne and Rubinstein, 1994] in the FOSG formalism, where we replace the decisions of the online algorithm with stochastic (chance) transitions. As we allow the online algorithm to be stateful and thus produce strategies depending on the game trajectory, the response game must also reflect this possibility. The resulting game  $\mathcal{G}_{\Omega}^k$  is thus exponential in size as it reflects all possible trajectories of  $k$  matches. We call this single-player game a  **$k$ -step response game**.

The  $k$ -step response game allows us to compute the best response value of a worst-case adversary in  $k$ -match game-play. We will use overloaded notation  $brv(\mathcal{G}_{\Omega}^k)$  to denote this value.

**Theorem 5.5.** If  $\forall k' \geq k \quad brv(\mathcal{G}_{\Omega}^{k'}) \leq k'\epsilon$ , then algorithm  $\Omega$  is  $(k, \epsilon)$ -sound.

*Proof.* If we used a fixed  $\epsilon$ -equilibrium strategy  $\sigma$  in each match (repetition) of a response game  $\mathcal{G}_{\sigma}^{k'}$ , then the  $brv(\mathcal{G}_{\sigma}^{k'}) = k'\epsilon$  because adversary can gain at most  $\epsilon$  in each match. Since  $\epsilon$ -sound algorithm should play at least as well as some offline  $\epsilon$ -equilibrium, it must have  $brv(\mathcal{G}_{\Omega}^k) \leq k\epsilon \forall k \geq 1$ . For a  $(k, \epsilon)$ -sound algorithm we add the condition of  $\forall k' \geq k$ .  $\square$

### 5.3.6 Tabularized Strategy

When an online algorithm produces the same strategy for an information state regardless of the previous matches, there is no need for the  $k$ -response game. Fixed strategy notion sufficiently describes the behavior of the online algorithm and thus the exploitability of the fixed strategy matches the soundness. To compute this fixed strategy, one simply queries the online algorithm for all the information states in the game.

## 5.4 Relating $(k, \epsilon)$ -Soundness and $\epsilon$ -Nash

Unfortunately, our notion of  $(k, \epsilon)$ -soundness is often infeasible to reason about, as it requires checking that the algorithm does not make strategy errors for  $\forall k' \geq k$ . In this section, we introduce the concept of consistency that allows one to formally state that the online algorithm plays “consistently” with an  $\epsilon$ -equilibrium. Our consistency notion allows us to directly bound the  $(k, \epsilon)$ -soundness of an online algorithm. We introduce three hierarchical levels of consistency, with varying restrictions and corresponding bounds. Notice that they differ mainly in the order of quantifiers.

### 5.4.1 Local Consistency

Local consistency simply guarantees that every time we query the online algorithm, there is an  $\epsilon$ -equilibrium that has the same local behavioral strategy  $\sigma(s)$  for the queried state  $s$ .

**Definition 5.6.** *Algorithm  $\Omega$  is locally consistent with  $\epsilon$ -equilibria if*

$$\forall k \forall m = (z_1, z_2, \dots, z_k) \forall h \in \mathcal{H}(z_k) \exists \sigma \in \mathcal{NE}_n^\epsilon$$

*such that  $\Omega^{(z_1, \dots, z_{k-1})}(s(h)) = \sigma(s(h))$ .*

While this suggests that the algorithm plays like some equilibrium, it is not so, and the resulting strategy can be highly exploitable. This is because one cannot combine local behavioral strategies from different  $\epsilon$ -equilibria and hope to preserve their exploitability. In another perspective, as soon as one starts to condition the selection of the strategy on private information, it risks computing strategies that can be exploited in a repeated game. This is a motivation behind introducing  $(k, \epsilon)$ -soundness, as it allows us to analyze algorithms that use such conditioning.

Consider the CMP game with two strategies  $\sigma^1 = \{(s_1, p = 1), (s_2, q = 0)\}$  and  $\sigma^2 = \{(s_1, p = 0.5), (s_2, q = 0.5)\}$ . While both strategies are equilibria, if one plays in the states  $s_1$  and  $s_2$  based on the first and second equilibrium respectively, it corresponds to an exploitable strategy  $\{(s_1, p = 1), (s_2, q = 0.5)\}$ .

An algorithm that is locally consistent with  $\epsilon$ -equilibria might not be  $(k, \epsilon)$ -sound.

Note that this can happen even in perfect information games (example in Appendix B.1). Interestingly, local consistency is sufficient if the algorithm is consistent with a subgame perfect equilibrium.

In perfect information games, an algorithm that is locally consistent with a subgame perfect equilibrium is sound.

A particularly interesting example of an algorithm that is only locally consistent is Online Outcome Sampling [Lisý et al., 2015] (OOS). See Section 5.6 for detailed discussion and experimental evaluation, where we show that this algorithm can produce highly exploitable strategies in imperfect information games.

### 5.4.2 Global Consistency

Local consistency guarantees consistency only for individual states. The problem we have then seen is that the combination of these local strategies might produce highly exploitable overall strategy. A natural extension is then to guarantee consistency with some equilibria for all the states in combination: a global consistency.

**Definition 5.7.** Algorithm  $\Omega$  is globally consistent with  $\epsilon$ -equilibria if

$$\forall k \forall m = (z_1, z_2, \dots, z_k) \exists \sigma \in \mathcal{NE}_n^\epsilon \forall h \in \mathcal{H}(z_i) \\ \text{holds that } \Omega^{(z_1, \dots, z_{i-1})}(s(h)) = \sigma(s(h)) \text{ for } \forall i \in \{1, \dots, k\}.$$

However:

An algorithm that is globally consistent with  $\epsilon$ -equilibria might not be  $\epsilon$ -sound.

*Proof.* A counter-example: The `PLAYCache` algorithm is globally consistent, but it is not sound ( $\epsilon = 0$ ), as we have seen that it is exploitable during the first match ( $k = 1$ ).  $\square$

But what if the algorithm keeps on playing the repeated game? While the global consistency with equilibria does not guarantee soundness, it guarantees that the expected average reward converges to the game value in the limit.

For an algorithm  $\Omega$  that is globally consistent with  $\epsilon$ -equilibria,

$$\forall k \forall \Omega_2 : \mathbb{E}_{m \sim P_{\Omega, \Omega_2}^k} [\mathcal{R}(m)] \geq u^* - \epsilon - \frac{|\mathcal{S}_1| \Delta}{k}. \quad (5.2)$$

**Corollary 5.8.** An algorithm  $\Omega$  that is globally consistent with  $\epsilon$ -equilibria is  $(k, \epsilon)$ -sound as  $k \rightarrow \infty$ .

### 5.4.3 Strong Global Consistency

The problem with global consistency is that it guarantees the existence of consistent equilibrium for any game-play *after* the game-play is generated. Strong global consistency additionally guarantees that the game-play *itself* is generated consistently with an equilibrium; and as in global consistency, the partial strategies for this game-play also correspond to an  $\epsilon$ -equilibrium. In other words, the online algorithm simply exactly follows a predefined equilibrium.

**Definition 5.9.** Online algorithm  $\Omega$  is strongly globally consistent with  $\epsilon$ -equilibrium if

$$\exists \sigma \in \mathcal{NE}_n^\epsilon \forall k \forall m = (z_1, z_2, \dots, z_k) \forall h \in \mathcal{H}(z_k) \\ \text{holds that } \Omega^{(z_1, \dots, z_{k-1})}(s(h)) = \sigma(s(h)).$$

Strong global consistency guarantees that the algorithm can be tabularized, and the exploitability of the tabularized strategy matches  $\epsilon$ -soundness of the online algorithm.

Online algorithm  $\Omega$  that is strongly globally consistent with  $\epsilon$ -equilibrium is  $\epsilon$ -sound.

Canonical examples of strongly globally consistent online algorithms are `DeepStack` and `Libratus`. In general, an algorithm that uses a notion of safe (continual) resolving is strongly globally consistent as it essentially re-solves some  $\epsilon$ -equilibrium (albeit an unknown one) that it follows. Another, more recent example is `ReBeL` [Brown et al., 2020], as it essentially imitates `CFR-D` iterations in conjunction with a neural network.

## Proving Strong Global Consistency

While we are not aware of an algorithm that is only globally consistent (besides the toy `PlayCache`), reasoning about global consistency can be beneficial for showing the strong global consistency. Doing so just based on its definition might not be straightforward. However, proving global consistency can be easier. If applicable, we can then use the following theorem to extend the proof to the strong global consistency.

If a globally consistent algorithm is stateless then it is also strongly globally consistent.

*Proof.* The definition of a stateless algorithm implies that for an information state  $s$  the algorithm always produces the same behavioral strategy  $\sigma(s)$  as the algorithm is deterministic (all stochasticity is encoded within the algorithm state  $\theta$ , see Remark 5.3).

This means that whatever  $\epsilon$ -equilibria the algorithm is globally consistent with is independent of the current game-play or match number. This allows us to swap the quantifiers from

$$\forall k \forall m = (z_1, z_2, \dots, z_k) \exists \sigma \in \mathcal{NE}_n^\epsilon \forall i \in \{1, \dots, k\} \forall h \in \mathcal{H}(z_i) : \\ \Omega^{(z_1, \dots, z_{i-1})}(s(h)) = \sigma(s(h))$$

to

$$\exists \sigma \in \mathcal{NE}_n^\epsilon \forall k \forall m = (z_1, z_2, \dots, z_k) \forall i \in \{1, \dots, k\} \forall h \in \mathcal{H}(z_i) : \\ \Omega^{(z_1, \dots, z_{i-1})}(s(h)) = \sigma(s(h)).$$

Using the same argument we can treat the different matches  $z_i$  as an iteration over  $k$ , leading us to strong global consistency

$$\exists \sigma \in \mathcal{NE}_n^\epsilon \forall k \forall m = (z_1, z_2, \dots, z_k) \forall h \in \mathcal{H}(z_k) : \\ \Omega^{(z_1, \dots, z_{k-1})}(s(h)) = \sigma(s(h)).$$

□

## 5.5 Relating $(k, \epsilon)$ -Soundness and Regret

Regret is an online learning concept that has triggered design of a family of powerful learning algorithms. Indeed, many algorithms that approximate Nash equilibria use regret minimization [Zinkevich et al., 2007]. There is a well-known connection between regret and the Nash equilibrium solution concept. In a zero-sum game at time  $k$ , if both players' overall regret  $R_k$  is less than  $k\epsilon$ , the average strategy profile is a  $2\epsilon$ -equilibrium [Zinkevich et al., 2007]. The use of  $k$  in  $(k, \epsilon)$ -soundness allows us to relate it with regret, and show how it is different from the consistency hierarchy.

**Corollary 5.10.** *Any regret minimizer with a regret bound of  $R_k$  is  $(k, \frac{R_k}{2k})$ -sound.*

## 5.6 Experiments

A particularly interesting example of an algorithm that is only locally consistent is Online Outcome Sampling (OOS) [Lisý et al., 2015]. We use it to demonstrate the theoretical ideas in this chapter with empirical experiments. We show that local consistency does in fact fail to result in  $\epsilon$ -soundness in the online setting. The problem we demonstrate is also not

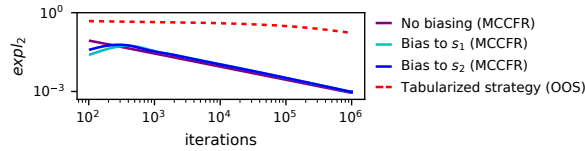


Figure 5.2: While individual MCCFR strategies have low exploitability of  $\sim 10^{-3}$ , the tabularized OOS strategy has high exploitability of 0.17 even after  $10^6$  iterations.

specific to OOS, but in general to any adaptation of an offline algorithm to the online setting where the algorithm attempts to improve its strategy during online play.

At high level, OOS runs the offline MCCFR algorithm in the full game (while also gradually building the tree), parameterized to increase the sampling probability of the current information state. The algorithm then plays based on the resulting strategy for that particular state. The problem is that these individual MCCFR runs can converge to different  $\epsilon$ -equilibria as the MCCFR is parameterized differently in each information state. In other words, the OOS algorithm exactly suffers from the fact that it is only locally consistent.

We use two games in our experiments: Coordinated Matching Pennies from Section 5.3 and Kuhn Poker [Kuhn, 1950]. We present the Coordinated Matching Pennies results here. See Appendix B.4 for the complete experimental details and a similar experiment for Kuhn Poker.

Within a single match of Coordinated Matching Pennies, the second player will act either in  $s_1$  or  $s_2$ . OOS will therefore bias MCCFR samples to whichever information state that actually occurs in the match. These two situations are distinct and result in two different strategies for the whole game (including the non-visited state), similarly to the example in Section 5.4.1. To emulate what OOS does, we parametrize MCCFR runs to bias samples into  $s_1$  and  $s_2$  respectively, and initialize the regrets in  $s_1, s_2$  so that the MCCFR is likely to produce diverse sets of strategies. As MCCFR is stochastic, we average the strategies over  $3 \cdot 10^4$  random seeds.

In Figure 5.2 we plot exploitability for the average strategies, and unbiased MCCFR for reference. The two biased variants of MCCFR actually converge at a similar rate to unbiased MCCFR, confirming that OOS is locally consistent: it quickly converges to an  $\epsilon$ -equilibria for  $s_1$  and  $s_2$  individually. However, the tabularized strategy — the strategy OOS follows online — is many orders of magnitude more exploitable even with hundreds of thousands of online iterations. The problem is that adapting its strategy online at  $s_1$  and  $s_2$  causes it to not be globally consistent with any  $\epsilon$ -equilibria.

## 5.7 Related literature

There are several known pathologies that occur in imperfect information games that are not present in the perfect information case. The pathologies that happen in the offline setting also present a problem in the online setting. In [Frank and Basin, 1998] the authors identified two problems: strategy-fusion and non-locality.

These two problems can easily arise for algorithms designed to solve only perfect-information games, such as minimax or reinforcement learning algorithms, and lead to computation of exploitable strategies. The proposed local consistency is similar in its spirit to non-locality, as composition of partial strategies (that correspond to parts of distinct equilibria) produced by an online algorithm may not be an overall equilibrium strategy. However local consistency identifies sub-optimal play also across repeated games.

In [Moravčík et al., 2017, Brown and Sandholm, 2018], they use some form of continual

re-solving, which is strongly globally consistent. This guarantees soundness of the algorithms. Continual resolving uses value functions defined over public belief spaces [Brown et al., 2020] to compute consistent strategies. Indeed, the minimal amount of information needed to properly define value functions are ranges (beliefs) over common knowledge public states [Seitz et al., 2019].

We are not aware of algorithms in the literature that are only globally consistent. This may lead to interesting future work: the algorithm may try to reduce its sub-optimal play of the first matches, while possibly not using all of the required player’s ranges.

Tabularization has been used in [Šustr et al., 2019] to compute an offline strategy and its exploitability. In [Lisý et al., 2015] they consider computing this tabularization (they refer to it as “brute-force” approach), but it is a very expensive procedure. Instead they use an “aggregate method”, which “stitches” strategy from a small number of matches and defines the strategy as uniform in non-visited information states. They do not state whether such approximation of tabularization is indeed correct.

## 5.8 Conclusion

We introduced the game of Coordinated Matching Pennies (CMP). This game illustrates the consistency issues that can arise for online algorithms in imperfect information games. We observed that exploitability is not an appropriate measure of an algorithm’s performance in online settings. This motivated us to introduce a formal framework for studying online algorithms and allowed us to define  $\epsilon$ -soundness. Just like  $\epsilon$ -exploitability, it measures the performance against the worst-case adversary. Soundness generalizes exploitability to repeated sequential games and it collapses to it when an online algorithm follows a fixed strategy. We then introduced a hierarchical consistency framework that formalizes in what sense an online algorithm can be consistent with a fixed strategy. Namely, we introduced three levels of consistency: i) local, ii) global and iii) strongly global. These connect an online algorithm’s behavior to that of a fixed strategy with increasingly tight bounds on the average expected utility against a worst-case adversary. We also stated various bounds on soundness based on the exploitability of a consistent fixed strategy. Interestingly, the implications are different in some cases for perfect and imperfect information games.

Within this framework, we saw that local consistency in imperfect information games does not guarantee correct evaluation of worst-case performance by computing exploitability. Based on this result, we argued that OOS, previously considered sound, can be exploited. This illustrates that these subtle problems with online algorithms can easily be missed and lead to wrong conclusions about their performance. Our experimental section included experiments in CMP and Kuhn Poker and showed a large discrepancy between OOS’s actual performance and the bound previously thought to hold.

## 5.9 Author’s contributions

When working on theory of search based agents, I have discovered fact that even when strategy of the agent in the whole game converges to a  $\epsilon$ -Nash equilibrium, this surprisingly does not guarantee good worst case performance. This observation motivated further research in the the theory of sound search.

# 6. Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines

This chapter is based on [Schmid et al., 2019].

## 6.1 Introduction

Policy gradient algorithms have shown remarkable success in single-agent reinforcement learning (RL) [Mnih et al., 2016, Schulman et al., 2017]. While there has been evidence of empirical success in multiagent problems [Foerster et al., 2017, Bansal et al., 2018], the assumptions made by RL methods generally do not hold in multiagent partially-observable environments. Hence, they are not guaranteed to find an optimal policy, even with tabular representations in two-player zero-sum (competitive) games [Littman, 1994]. As a result, policy iteration algorithms based on computational game theory and regret minimization have been the preferred formalism in this setting. Counterfactual regret minimization [Zinkevich et al., 2007] has been a core component of this progress in Poker AI, leading to solving Heads-Up Limit Texas Hold'em [Bowling et al., 2015] and defeating professional poker players in No-Limit [Moravčík et al., 2017, Brown and Sandholm, 2018].

The two fields of RL and computational game theory have largely grown independently. However, there has been recent work that relates approaches within these two communities. Fictitious self-play uses RL to compute approximate best responses and supervised learning to combine responses [Heinrich et al., 2015]. This idea is extended to a unified training framework that can produce more general policies by regularizing over generated response oracles [Lanctot et al., 2017]. RL-style regressors were first used to compress regrets in game theoretic algorithms [Waugh et al., 2015]. DeepStack introduced deep neural networks as generalized value-function approximators for online planning in imperfect information games [Moravčík et al., 2017]. These value functions operate on a belief-space over all possible states consistent with the players' observations.

This chapter similarly unites concepts from both fields, proposing an unbiased variance reduction technique for Monte Carlo counterfactual regret minimization using an analog of state-action baselines from actor-critic RL methods. While policy gradient methods typically involve Monte Carlo estimates, the analog in imperfect information settings is Monte Carlo Counterfactual Regret Minimization (MCCFR) [Lanctot et al., 2009]. Policy gradient estimates based on a single sample of an episode suffer significantly from variance. A common technique to decrease the variance is a state or state-action dependent baseline value that is subtracted from the observed return. These methods can drastically improve the convergence speed. However, no such methods are known for MCCFR.

MCCFR is a sample based algorithm in imperfect information settings, which approximates counterfactual regret minimization (CFR) by estimating regret quantities necessary for updating the policy. While MCCFR can offer faster short-term convergence than original CFR in large games, it suffers from high variance which leads to slower long-term convergence.

CFR+ provides significantly faster empirical performance and made solving Heads-Up



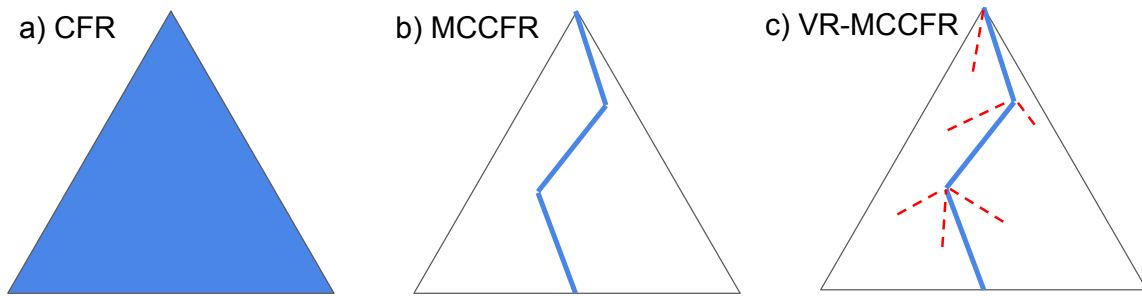


Figure 6.1: High-level overview of Variance Reduction MCCFR (VR-MCCFR) and related methods. a) CFR traverses the entire tree on every iteration. b) MCCFR samples trajectories and computes the values only for the sampled actions, while the off-trajectory actions are treated as zero-valued. While MCCFR uses importance sampling weight to ensure the values are unbiased, the sampling introduces high variance. c) VR-MCCFR follows the same sampling framework as MCCFR, but uses baseline values for both sampled actions (in blue) as well as the off-trajectory actions (in red). These baselines use control variates and send up bootstrapped estimates to decrease the per-iteration variance thus speeding up the convergence.

Limit Texas Hold'em possible [Bowling et al., 2015]. Unfortunately, CFR+ has so far did not outperform CFR in Monte Carlo settings [Burch, 2017] (also see Figure ( C.1) in the appendix for an experiment).

In this work, we reformulate the value estimates using a control variate and a state-action baseline. The new formulation includes any approximation of the counterfactual values, which allows for a range of different ways to insert domain-specific knowledge (if available) but also to design values that are learned online.

Our experiments show two orders of magnitude improvement over MCCFR. For the common testbed imperfect information game – Leduc Poker – VR-MCCFR with a state-action baseline needs 250 times fewer iterations than MCCFR to reach the same solution quality. In contrast to RL algorithms in perfect information settings, where state-action baselines bring little to no improvement over state baselines [Tucker et al., 2018], state-action baselines lead to significant improvement over state baselines in multiagent partially-observable settings. We suspect this is due to variance from the environment and different dynamics of the policies during the computation.

## 6.2 Related Work

There are standard variance reduction techniques for Monte Carlo sampling methods [Owen, 2013] and the use of control variates in these settings has a long history [Boyle, 1977]. Reducing variance is particularly important when estimating gradients from sample trajectories. Consequentially, the use of a control variates using baseline has become standard practice in policy gradient methods [Williams, 1992, Sutton and Barto, 2017]. In RL, action-dependent baselines have recently shown promise [Wu et al., 2018, Liu et al., 2018] but the degree to which variance is indeed reduced remains unclear [Tucker et al., 2018]. We show that in our setting of MCCFR in imperfect information multiplayer games, action-dependent baselines necessarily influence the variance of the estimates, and we confirm the reduction empirically. This is important because lower-variance estimates lead to better regret bounds [Gibson et al., 2012].

There have been a few uses of variance reduction techniques in multiplayer games,

within Monte Carlo tree search (MCTS). In MCTS, control variates have used to augment the reward along a trajectory using a property of the state before and after a transition [Veness et al., 2011] and to augment the outcome of a rollout from its length or some pre-determined quality of the states visited [Pepels et al., 2014].

Our baseline-improved estimates are similar to the ones used in AIVAT [Burch et al., 2018]. AIVAT defines estimates of expected values using heuristic values of states as baselines in practice. Unlike this work, AIVAT was only used for evaluation of strategies.

To the best of our knowledge, there has been two applications of variance reduction in Monte Carlo CFR: by manipulating the chance node distribution [Lanctot, 2013, Section 7.5] and by sampling (“probing”) more trajectories for more estimates of the underlying values [Gibson et al., 2012]. The variance reduction (and resulting drop in convergence rate) is modest in both cases, whereas we show more than a two order of magnitude speed-up in convergence using our method.

## 6.3 Background

In this chapter we will be leveraging definition of the extensive form games from chapter 2.

### 6.3.1 Augmented Information Sets

In addition to the standard definition of the information sets it is also often useful to consider the **augmented information sets** [Burch et al., 2014]. While an information set  $I$  groups histories  $h$  that player  $i = p(h)$  cannot distinguish, an augmented information set groups histories that player  $i$  can not distinguish, including these where  $p(h) \neq i$ . For a history  $h$ , we denote an augmented information set of player  $i$  as  $I_i(h)$ . Note that the if  $p(h) = i$  then  $I_i(h) = I(h)$  and  $I(h) = I_{p(h)}(h)$ .

### 6.3.2 Counterfactual Regret Minimization

Counterfactual Regret (CFR) Minimization is an iterative algorithm that produces a sequence of strategies  $\sigma^0, \sigma^1, \dots, \sigma^T$ , whose average strategy  $\bar{\sigma}^T$  converges to an approximate Nash equilibrium as  $T \rightarrow \infty$  in two-player zero-sum games [Zinkevich et al., 2007]. Specifically, on iteration  $t$ , for each  $I$ , it computes **counterfactual values**. Define  $Z_I = \{(h, z) \in H \times Z \mid h \in I, h \sqsubseteq z\}$ , and  $u_i^{\sigma^t}(h, z) = \pi^{\sigma^t}(h, z)u_i(z)$ . We will also sometimes use the short form  $u_i^\sigma(h) = \sum_{z \in Z, h \sqsubseteq z} u_i^\sigma(h, z)$ . A counterfactual value is:

$$v_i(\sigma^t, I) = \sum_{(h, z) \in Z_I} \pi_{-i}^{\sigma^t}(h) u_i^{\sigma^t}(h, z). \quad (6.1)$$

We also define an action-dependent counterfactual value,

$$v_i(\sigma, I, a) = \sum_{(h, z) \in Z_I} \pi_{-i}^\sigma(ha) u_i^\sigma(ha, z), \quad (6.2)$$

where  $ha$  is the sequence  $h$  followed by the action  $a$ . The values are analogous to the difference in  $Q$ -values and  $V$ -values in RL, and indeed we have  $v_i(\sigma, I) = \sum_a \sigma(I, a) v_i(\sigma, I, a)$ . CFR then computes a **counterfactual regret** for *not taking*  $a$  at  $I$ :

$$r^t(I, a) = v_i(\sigma^t, I, a) - v_i(\sigma^t, I), \quad (6.3)$$

This regret is then accumulated  $R^T(I, a) = \sum_{t=1}^T r^t(I, a)$ , which is used to update the strategies using **regret-matching** [Hart and Mas-Colell, 2000]:

$$\sigma^{T+1}(I, a) = \frac{(R^T(I, a))^+}{\sum_{a \in A(I)} (R^T(I, a))^+}, \quad (6.4)$$

where  $(x)^+ = \max(x, 0)$ , or to the uniform strategy if  $\sum_a (R^T(I, a))^+ = 0$ . CFR+ works by thresholding the quantity at each round [Tammelin et al., 2015]: define  $Q^0(I, a) = 0$  and  $Q^T(I, a) = (Q^{T-1} + r^T(I, a))^+$ ; CFR+ updates the policy by replacing  $R^T$  by  $Q^T$  in equation 6.4. In addition, it always alternates the regret updates of the players (whereas some variants of CFR update both players), and the average strategy places more (linearly increasing) weight on more recent iterations.

If for player  $i$  we denote  $u(\sigma) = u_i(\sigma_i, \sigma_{-i})$ , and run CFR for  $T$  iterations, then we can define the **overall regret** of the strategies produced as:

$$R_i^T = \max_{\sigma_i} \sum_{t=1}^T (v_i(\sigma'_i, \sigma_{-i}^t) - v_i(\sigma^t)).$$

CFR ensures that  $R_i^T/T \rightarrow 0$  as  $T \rightarrow \infty$ . When two players minimize regret, the folk theorem then guarantees a bound on the distance to a Nash equilibrium as a function of  $R_i^T/T$ .

To compute  $v_i$  precisely, each iteration requires traversing over subtrees under each  $a \in A(I)$  at each  $I$ . Next, we describe variants that allow sampling parts of the trees and using estimates of these quantities.

### 6.3.3 Monte Carlo CFR

Monte Carlo CFR (MCCFR) introduces sample estimates of the counterfactual values, by visiting and updating quantities over only part of the entire tree. MCCFR is a general family of algorithms: each instance defined by a specific sampling policy. For ease of exposition and to show the similarity to RL, we focus on **outcome sampling** [Lanctot et al., 2009]; however, our baseline-enhanced estimates can be used in all MCCFR variants. A **sampling policy**  $\xi$  is defined in the same way as a strategy (a distribution over  $A(I)$  for all  $I$ ) with a restriction that  $\xi(h, a) > 0$  for all histories and actions. Given a terminal history sampled with probability  $q(z) = \pi^\xi(z)$ , a **sampled counterfactual value**  $\tilde{v}_i(\sigma, I|z)$

$$= \tilde{v}_i(\sigma, h|z) = \frac{\pi_{-i}^\sigma(h) u_i^\sigma(h, z)}{q(z)}, \text{ for } h \in I, h \sqsubseteq z, \quad (6.5)$$

and 0 for histories that were not played,  $h \not\sqsubseteq z$ . The estimate is unbiased:  $\mathbf{E}_{z \sim \xi} [\tilde{v}_i(\sigma, I|z)] = v_i(\sigma, I)$ , by [Lanctot et al., 2009, Lemma 1]. As a result,  $\tilde{v}_i$  can be used in Equation 6.3 to accumulate estimated regrets  $\tilde{r}^t(I, a) = \tilde{v}_i(\sigma^t, I, a) - \tilde{v}_i(\sigma^t, I)$  instead. The regret bound requires an additional term  $\frac{1}{\min_{z \in \mathcal{Z}} q(z)}$ , which is exponential in the length of  $z$  and similar observations have been made in RL [Arjona-Medina et al., 2018]. The main problem with the sampling variants is that they introduce variance that can have a significant effect on long-term convergence [Gibson et al., 2012].

### 6.3.4 Control Variates

Suppose one is trying to estimate a statistic of a random variable,  $X$ , such as its mean, from samples  $\mathbf{X} = (X_1, X_2, \dots, X_n)$ . A crude Monte Carlo estimator is defined to be

$\hat{X}^{mc} = \frac{1}{n} \sum_{i=1}^n X_i$ . A **control variate** is a random variable  $Y$  with a known mean  $\mu_Y = \mathbb{E}[Y]$ , that is paired with the original variable, such that samples are instead of the form  $(\mathbf{X}, \mathbf{Y})$  [Owen, 2013]. A new random variable is then defined,  $Z_i = X_i + c(Y_i - \mu_Y)$ . An estimator  $\hat{Z}^{cv} = \frac{1}{n} \sum_{i=1}^n Z_i$ . Since  $\mathbb{E}[Z_i] = \mathbb{E}[X_i]$  for any value of  $c$ ,  $\hat{Z}^{cv}$  can be used in place of  $\hat{X}^{mc}$ . with variance  $\text{Var}[Z_i] = \text{Var}[X_i] + c^2 \text{Var}[Y_i] + 2c \text{Cov}[X_i, Y_i]$ . So when  $X$  and  $Y$  are positively correlated and  $c < 0$ , variance is reduced when  $\text{Cov}[X, Y] > \frac{c^2}{2} \text{Var}[Y]$ .

### 6.3.5 Reinforcement Learning Mapping

There are several analogies to make between Monte Carlo CFR in imperfect information games and reinforcement learning. Since our technique builds on ideas that have been widely used in RL, we end the background by providing a small discussion of the links.

First, dynamics of an imperfect information game are similar to a partially-observable episodic MDP without any cycles. Policies and strategies are identically defined, but in imperfect information games a deterministic optimal (Nash) strategy may not exist causing most of the RL methods to fail to converge. The search for a minmax-optimal strategy with several players is the main reason CFR is used instead of, for example, value iteration. However, both operate by defining values of states which are analogous (counterfactual values versus expected values) since they are both functions of the strategy/policy; therefore, can be viewed as a kind of policy iteration which computes the values and from which a policy is derived. However, the iterates  $\sigma^t$  are not guaranteed to converge to the optimal strategy, only the average strategy  $\bar{\sigma}^t$  does.

Monte Carlo CFR is an off-policy Monte Carlo analog. The value estimates are unbiased specifically because they are corrected by importance sampling. Most applications of MCCFR have operated with tabular representations, but this is mostly due to the differences in objectives. Function approximation methods have been proposed for CFR [Waugh et al., 2015] but the variance from pure Monte Carlo methods may prevent such techniques in MCCFR. The use of baselines has been widely successful in policy gradient methods, so reducing the variance could enable the practical use of function approximation in MCCFR.

## 6.4 Monte Carlo CFR with Baselines

We now introduce our technique: MCCFR with baselines. While the baselines are analogous to those from policy gradient methods (using counterfactual values), there are slight differences in their construction.

Our technique constructs value estimates using control variates. Note that MCCFR is using sampled estimates of counterfactual values  $\tilde{v}_i(\sigma, I)$  whose expected value is the counterfactual value  $v_i(\sigma, I)$ . First, we introduce an **estimated counterfactual** value  $\hat{v}_i(\sigma, I)$  to be any estimator of the counterfactual value (not necessarily  $\tilde{v}_i$  as defined above, but this is one possibility).

We now define an action-dependent **baseline**  $b_i(I, a)$  that, as in RL, serves as a basis for the sampled values. The intent is to define a baseline function to approximate or be correlated with  $\mathbb{E}[\hat{v}_i(\sigma, I, a)]$ . We also define a sampled baseline  $\hat{b}_i(I, a)$  as an estimator such that  $\mathbb{E}[\hat{b}_i(I, a)] = b_i(I, a)$ . From this, we construct a new baseline-enhanced estimate for the counterfactual values:

$$\hat{v}_i^b(\sigma, I, a) = \hat{v}_i(\sigma, I, a) - \hat{b}_i(\sigma, I, a) + b_i(\sigma, I, a) \quad (6.6)$$

First, note that  $\hat{b}_i$  is a control variate with  $c = -1$ . Therefore, it is important that  $\hat{b}_i$  be

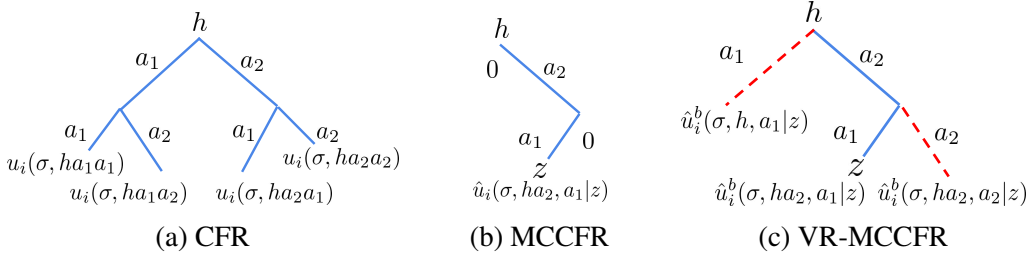


Figure 6.2: Values and updates for the discussed methods: (a) CFR updates the full tree and thus uses the exact values for all the actions, (b) MCCFR updates only a single path, and uses the sampled values for the sampled actions and zero values for the off-trajectory actions, (c) VR-MCCFR also updates only a single path, but uses the bootstrapped baseline-enhanced values for the sampled actions and baseline-enhanced values for the off-trajectory actions.

correlated with  $\hat{v}_i$ . The main idea of our technique is to replace  $\tilde{v}_i(\sigma, I, a)$  with  $\hat{v}_i^b(\sigma, I, a)$ . A key property is that by doing so, the expectation remains unchanged.

**Lemma 6.1.** *For any  $i \in N - \{c\}, \sigma_i, I \in \mathcal{I}, a \in A(I)$ , if  $\mathbb{E}[\hat{b}_i(I, a)] = b_i(I, a)$  and  $\mathbb{E}[\hat{v}_i(\sigma, I, a)] = v_i(\sigma, I, a)$ , then  $\mathbb{E}[\hat{v}_i^b(\sigma, I, a)] = v_i(\sigma, I, a)$ .*

The proof is in the appendix. As a result, any baseline whose expectation is known can be used and the baseline-enhanced estimates are consistent. However, not all baselines will decrease variance. For example, if  $\text{Cov}[\hat{v}_i, \hat{b}_i]$  is too low, then the  $\text{Var}[\hat{b}_i]$  term could dominate and actually increase the variance.

## 6.4.1 Recursive Bootstrapping

Consider the individual computation (6.1) for all the information sets on the path to a sampled terminal history  $z$ . Given that the counterfactual values up the tree can be computed from the counterfactual values down the tree, it is natural to consider propagating the already baseline-enhanced counterfactual values (6.6) rather than the original noisy sampled values - thus propagating the benefits up the tree. The Lemma (6.2) then shows that by doing so, the updates remain unbiased. Our experimental section shows that such bootstrapping a crucial component for the proper performance of the method.

To properly formalize this bootstrapping computation, we must first recursively define the **expected value**:

$$\hat{u}_i(\sigma, h, a|z) = \begin{cases} \hat{u}_i(\sigma, ha|z)/\xi(h, a) & \text{if } ha \sqsubseteq z \\ 0 & \text{otherwise} \end{cases}, \quad (6.7)$$

and

$$\hat{u}_i(\sigma, h|z) = \begin{cases} u_i(h) & \text{if } h = z \\ \sum_a \sigma(h, a) \hat{u}_i(\sigma, h, a|z) & \text{if } h \sqsubset z \\ 0 & \text{otherwise} \end{cases}. \quad (6.8)$$

Next, we define a baseline-enhanced version of the expected value. Note that the baseline  $b_i(I, a)$  can be arbitrary, but we discuss a particular choice and update of the baseline in the later section. For every action, given a specific sampled trajectory  $z$ , then  $\hat{u}_i^b(\sigma, h, a|z) =$

$$\begin{cases} b_i(I_i(h), a) + \frac{\hat{u}_i^b(\sigma, ha|z) - b_i(I_i(h), a)}{\xi(h, a)} & \text{if } ha \sqsubseteq z \\ b_i(I_i(h), a) & \text{if } h \sqsubset z, ha \not\sqsubseteq z \\ 0 & \text{otherwise} \end{cases} \quad (6.9)$$

and

$$\hat{u}_i^b(\sigma, h|z) = \begin{cases} u_i(h) & \text{if } h = z \\ \sum_a \sigma(h, a) \hat{u}_i^b(\sigma, h, a|z) & \text{if } h \sqsubset z \\ 0 & \text{otherwise} \end{cases} . \quad (6.10)$$

These are the values that are bootstrapped. We estimate counterfactual values needed for the regret updates using these values as:

$$\hat{v}_i^b(\sigma, I(h), a|z) = \hat{v}_i^b(\sigma, h, a|z) = \frac{\pi_i^\sigma(h)}{q(h)} \hat{u}_i^b(\sigma, h, a|z). \quad (6.11)$$

We can now formally state that the bootstrapping keeps the counterfactual values unbiased:

**Lemma 6.2.** *Let  $\hat{v}_i^b$  be defined as in Equation 6.11. Then, for any  $i \in N - \{c\}$ ,  $\sigma_i, I \in \mathcal{I}$ ,  $a \in A(I)$ , it holds that  $\mathbb{E}_z[\hat{v}_i^b(\sigma, I, a|z)] = v_i(\sigma, I, a)$ .*

The proof is in the appendix. Since each estimate builds on other estimates, the benefit of the reduction in variance can be propagated up through the tree.

Another key result is that there exists a perfect baseline that leads to zero-variance estimates at the updated information sets.

**Lemma 6.3.** *There exists a perfect baseline  $b^*$  and optimal unbiased estimator  $\hat{v}_i^*(\sigma, h, a)$  such that under a specific update scheme:  $\forall ar_{h,z \sim \xi, h \in I, h \sqsubseteq z}[\hat{v}_i^*(\sigma, h, a|z)] = 0$ .*

The proof and description of the update scheme are in the appendix. We will refer to  $b^*$  as the **oracle baseline**. Note that even when using the oracle baseline, the convergence rate of MCCFR is still not identical to CFR because each iteration applies regret updates to a portion of the tree, whereas CFR updates the entire tree.

Finally, using unbiased estimates to tabulate regrets  $\hat{r}(I, a)$  for each  $I$  and  $a$  leads to a probabilistic regret bound:

**Theorem 6.4.** *[Gibson et al., 2012, Theorem 2] For some unbiased estimator of the counterfactual values  $\hat{v}_i$  and a bound on the difference in its value  $\hat{\Delta}_i = |\hat{v}_i(\sigma, I, a) - \hat{v}_i(\sigma, I, a')|$ , with probability  $1-p$ ,  $\frac{R_i^T}{T}$*

$$\leq \left( \hat{\Delta}_i + \frac{\sqrt{\max_{t, I, a} \text{Var}[r_i^t(I, a) - \hat{r}_i^t(I, a)]}}{\sqrt{p}} \right) \frac{|\mathcal{I}_i| |A_i|}{\sqrt{T}}.$$

## 6.4.2 Choice of Baselines

How does one choose a baseline, given that we want these to be good estimates of the individual counterfactual values? A common choice of the baseline in policy gradient algorithms is the mean value of the state, which is learned online [Mnih et al., 2016].

Inspired by this, we choose a similar quantity: the average expected value  $\bar{u}_i(I_i, a)$ . That is, in addition to accumulating regret for each  $I$ , average expected values are also tracked.

While a direct average can be tracked, we found that an exponentially-decaying average that places heavier weight on more recent samples to be more effective in practice. On the  $k^{\text{th}}$  visit to  $I$  at iteration  $t$ ,

$$\bar{u}_i^k(I_i, a) = \begin{cases} 0 & \text{if } k = 0 \\ (1 - \alpha)\bar{u}_i^{k-1}(I_i, a) + \alpha\hat{u}_i^b(\sigma^t, I_i, a) & \text{if } k > 0 \end{cases}$$

We then define the baseline  $b_i(I_i, a) = \bar{u}_i(I_i, a)$ , and

$$\hat{b}_i(I_i, a|z) = \begin{cases} b_i(I_i, a)/\xi(I_i, a) & \text{if } ha \sqsubseteq z, h \in I_i \\ 0 & \text{otherwise.} \end{cases}$$

The baseline can therefore be thought as *local* to  $I_i$  since it depends only on quantities defined and tracked at  $I_i$ . Note that  $\mathbb{E}_{a \sim \xi(I_i)}[\hat{b}_i(I_i, a|z)] = b_i(I_i, a)$  as required.

### 6.4.3 Summary of the Full Algorithm

We now summarize the technique developed above. One iteration of the algorithm consists of:

1. Repeat the steps below for each  $i \in N - \{c\}$ .
2. Sample a trajectory  $z \sim \xi$ .
3. For each history  $h \sqsubseteq z$  in reverse order (longest first):
  - (a) If  $h$  is terminal, simply return  $u_i(h)$
  - (b) Obtain current strategy  $\sigma(I)$  from Eq. 6.4 using cumulative regrets  $R(I, a)$  where  $h \in I$ .
  - (c) Use the child value  $\hat{u}_i^b(\sigma, ha)$  to compute  $\hat{u}_i^b(\sigma, h)$  as in Eq. 6.9.
  - (d) If  $p(h) = i$  then for  $a \in A(I)$ , compute  $\hat{v}_i^b(\sigma, I, a) = \frac{\pi_{-i}(h)}{q(h)}\hat{u}_i^b(\sigma, ha)$  and accumulate regrets  $R(I, a) \leftarrow R(I, a) + \hat{v}_i^b(\sigma, I, a) - \hat{v}_i^b(\sigma, I)$ .
  - (e) Update  $\bar{u}_i(\sigma, I_i, a)$ .
  - (f) Finally, return  $\hat{u}_i^b(\sigma, h)$ .

Note that the original outcome sampling is an instance of this algorithm. Specifically, when  $b_i(I_i, a) = 0$ , then  $\hat{v}_i^b(\sigma, I, a) = \tilde{v}_i(\sigma, I, a)$ . Step by step example of the computation is in the appendix.

## 6.5 Experimental Results

We evaluate the performance of our method on **Leduc poker** [Southey et al., 2005], a commonly used benchmark poker game. Players have an unlimited number of chips, and the deck has six cards, divided into two suits of three identically-ranked cards. There are two rounds of betting; after the first round a single public card is revealed from the deck. Each player antes 1 chip to play, receiving one private card. There are at most two bet or raise actions per round, with a fixed size of 2 chips in the first round, and 4 chips in the second round.

For the experiments, we use a vectorized form of CFR that applies regret updates to each information set consistent with the public information. The first vector variants were introduced in [Johanson et al., 2012], and have been used in DeepStack and Libratus [Moravčík et al., 2017, Brown and Sandholm, 2018]. See the appendix for more detail on the implementation. Baseline average values  $\hat{u}_i^b(I, a)$  used a decay factor of  $\alpha = 0.5$ . We used a uniform sampling in all our experiments,  $\xi(I, a) = \frac{1}{|A(I)|}$ .

We also consider the best case performance of our algorithm by using the oracle baseline. It uses baseline values of the true counterfactual values. We also experiment with and without CFR+, demonstrating that our technique allows the CFR+ to be for the first time efficiently used with sampling.

### 6.5.1 Convergence

We compared MCCFR, MCCFR+, VR-MCCFR, VR-MCCFR+, and VR-MCCFR+ with the oracle baseline, see Fig. 6.3. The variance-reduced VR-MCCFR and VR-MCCFR+ variants converge significantly faster than plain MCCFR. Moreover, the speedup grows as the baseline improves during the computation. A similar trend is shown by both VR-MCCFR and VR-MCCFR+, see Fig. 6.4. MCCFR needs hundreds of millions of iterations to reach the same exploitability as VR-MCCFR+ achieves in one million iterations: a 250-times speedup. VR-MCCFR+ with the oracle baseline significantly outperforms VR-MCCFR+ at the start of the computation, but as time progresses and the learned baseline improves, the difference shrinks. After one million iterations, exploitability of VR-MCCFR+ with a learned baseline approaches the exploitability of VR-MCCFR+ with the oracle baseline. This oracle baseline result gives a bound on the gains we can get by constructing better learned baselines.

### 6.5.2 Observed Variance

To verify that the observed speedup of the technique is due to variance reduction, we experimentally observed variance of counterfactual value estimates for MCCFR+ and MCCFR, see Fig. 6.5. We did that by sampling 1000 alternative trajectories for all visited information sets, with each trajectory sampling a different estimate of the counterfactual value. While the variance of value estimates in the plain algorithm seems to be more or less constant, the variance of VR-MCCFR and VR-MCCFR+ value estimates is lower, and continues to decrease as more iterations are run. This confirms that the combination of baseline and bootstrapping is reducing variance, which implies better performance given the connection between variance and MCCFR’s performance (Theorem 6.4).

### 6.5.3 Evaluation of Bootstrapping and Baseline Dependence on Actions

Recent work that evaluates action-dependent baselines in RL [Tucker et al., 2018], shows that there is often no real advantage compared to baselines that depend just on the state. It is also not common to bootstrap the value estimates in RL. Since VR-MCCFR uses both of these techniques it is natural to explore the contribution of each idea. We compared four VR-MCCFR+ variants: with or without bootstrapping and with baseline that is state or state-action dependant, see Fig. 6.6. The conclusion is that the improvement in the performance is very small unless we use both bootstrapping and an action-dependant baseline.



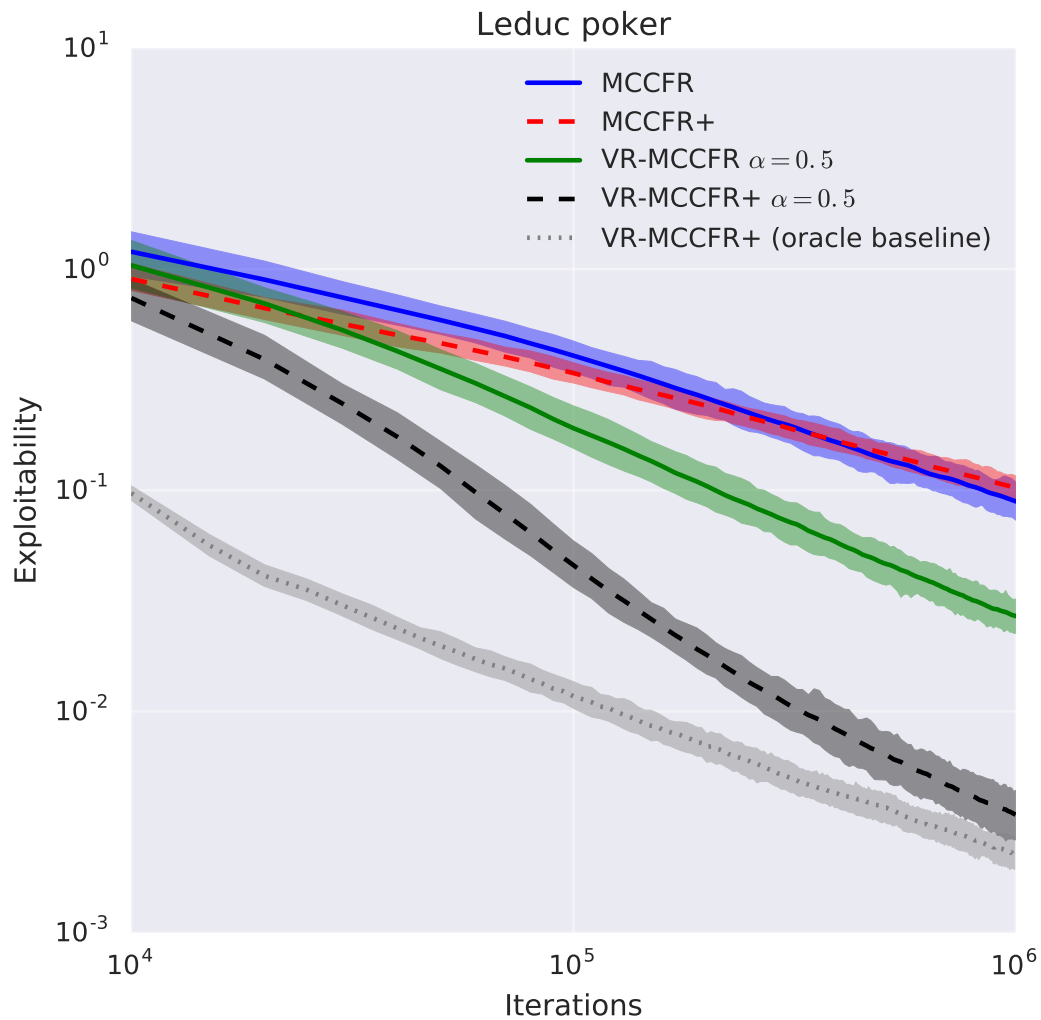


Figure 6.3: Convergence of exploitability for different MCCFR variants on logarithmic scale. VR-MCCFR converges substantially faster than plain MCCFR. VR-MCCFR+ bring roughly two orders of magnitude speedup. VR-MCCFR+ with oracle baseline (actual true values are used as baselines) is used as a bound for VR-MCCFR’s performance to show possible room for improvement. When run for  $10^6$  iterations VR-MCCFR+ approaches performance of the oracle version. The ribbons show 5th and 95th percentile over 100 runs.

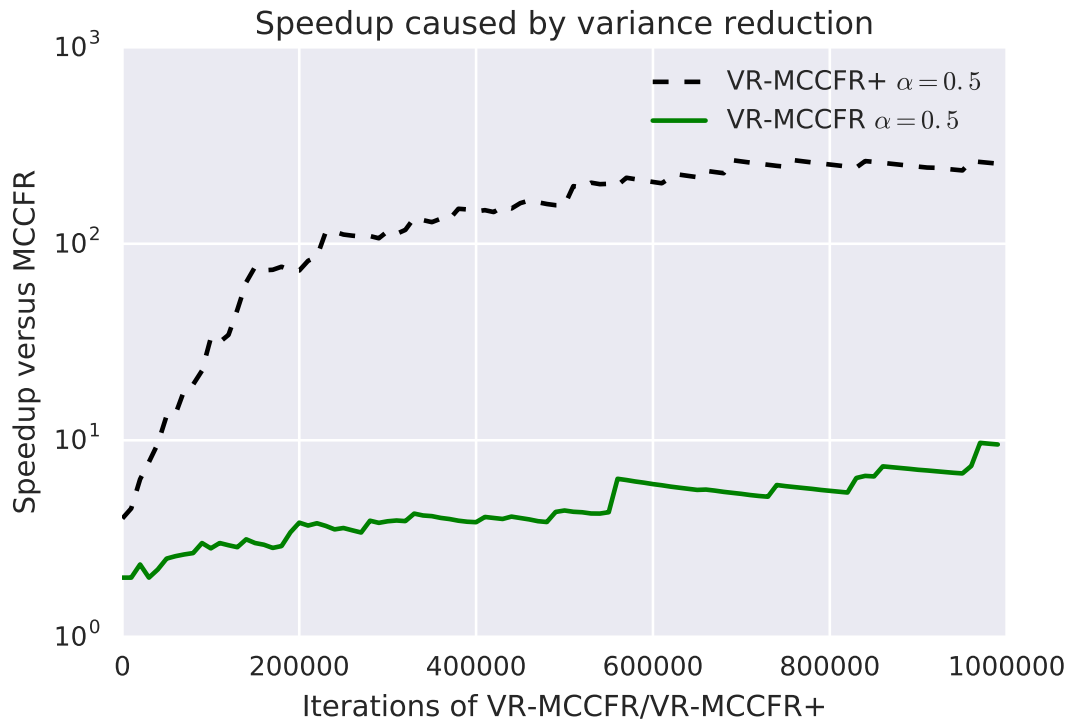


Figure 6.4: Speedup of VR-MCCFR and VR-MCCFR+ compared to plain MCCFR. Y-axis show how many times more iterations are required by MCCFR to reach the same exploitability as VR-MCCFR or VR-MCCFR+.

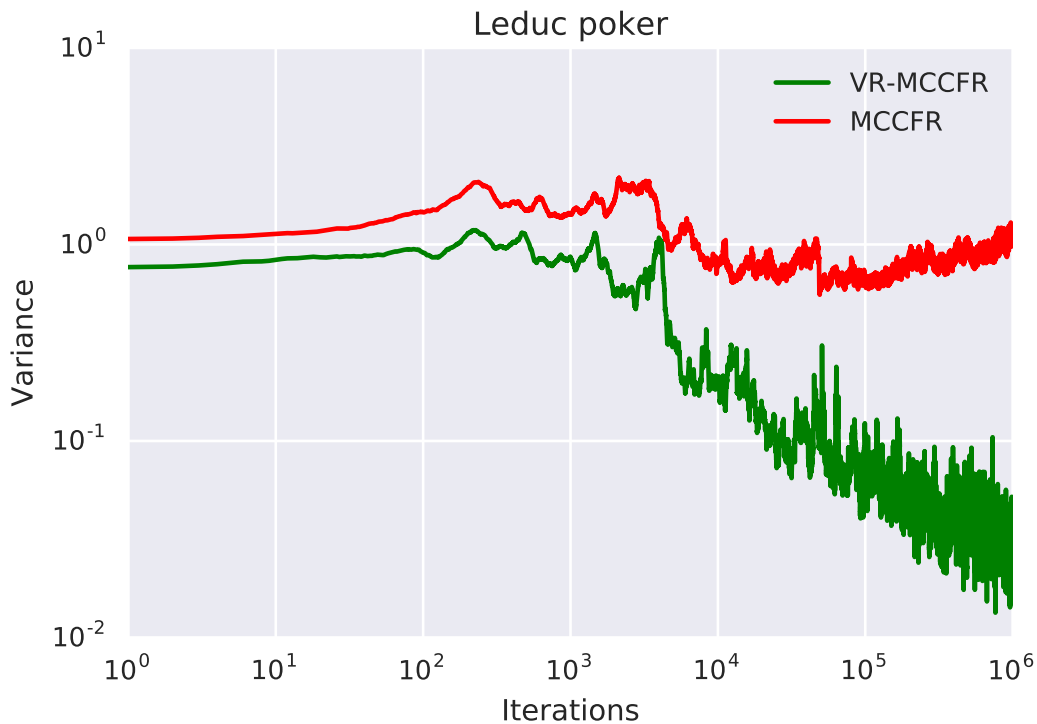


Figure 6.5: Variance of counterfactual values in VR-MCCFR and plain MCCFR with both regret matching and regret matching+. The curves were smoothed by computing moving average over a sliding window of 100 iterations.

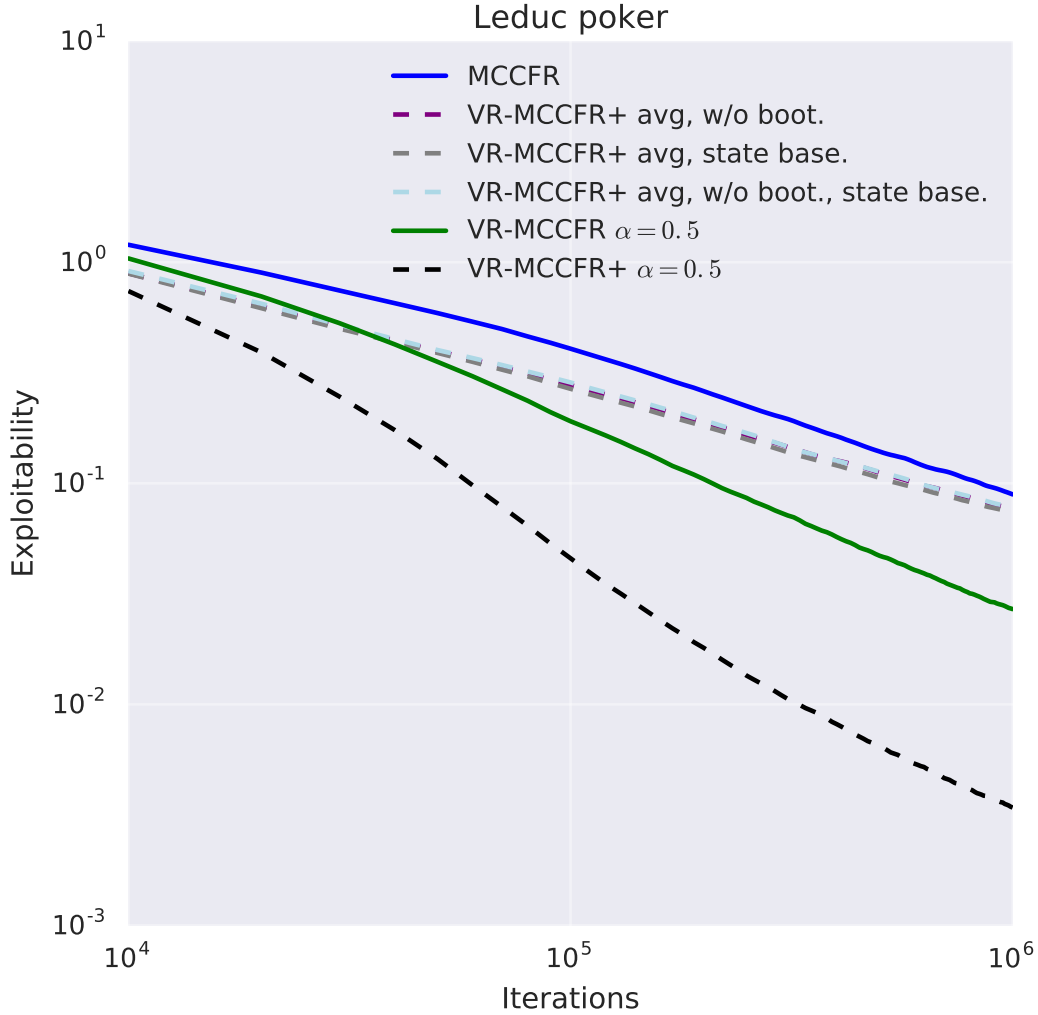


Figure 6.6: Detailed comparison of different VR-MCCFR variants on logarithmic scale. The curves for MCCFR, VR-MCCFR and VR-MCCFR+ are the same as in the previous plot, the other lines show how the algorithm performs when using state baselines instead of state-action baselines, and without bootstrapping. All of these reduced variants perform better than plain MCCFR, however they are worse than full VR-MCCFR. This ablation study shows that the combination of all VR-MCCFR features is important for final performance.

## 6.6 Conclusions

We have presented a new technique for variance reduction for Monte Carlo counterfactual regret minimization. This technique has close connections to existing RL methods of state and state-action baselines. In contrast to RL environments, our experiments in imperfect information games suggest that state-action baselines are superior to state baselines. Using this technique, we show that empirical variance is indeed reduced, speeding up the convergence by an order of magnitude. The decreased variance allows for the first time CFR+ to be used with sampling, bringing the speedup to two orders of magnitude.

## **6.7 Author's contributions**

I significantly contributed to the formation of the theory, the implementation of the algorithm and experiments.

# 7. Refining Subgames in Large Imperfect Information Games

This chapter is based on [Moravčík et al., 2016]

## 7.1 Introduction

Extensive form games are a powerful model capturing a wide class of real-world problems. The games can be either perfect information (Chess) or imperfect information (poker). Applications of imperfect information games range from security problems [Pita et al., 2009] to card games [Bowling et al., 2015]

The largest imperfect information game to be (essentially) solved today is the limit version of two-player Texas Hold'em poker [Bowling et al., 2015], with approximately  $10^{17}$  nodes [Johanson, 2013]. Unfortunately, many games remain that are much too large to be solved with current techniques. For example, the more popular “No-Limit” variant of two-player Texas Hold'em poker has approximately  $10^{165}$  nodes [Johanson, 2013].

The leading approach to solving imperfect information games of this magnitude is to create a simplified abstraction of the game, compute an  $\epsilon$ -equilibrium in the abstract game, and finally use the strategy from the abstracted game to play the original, unabstracted game [Billings et al., 2003a] [Sandholm, 2010] [Johanson et al., 2013] [Gibson, 2014]. The amount of simplification needed to produce the abstracted game is determined by the maximum size of the game tree that we are able to learn with the computing resources available. While abstraction pathologies mean that larger abstractions are not guaranteed to produce better strategies [Waugh et al., 2009], empirical results have shown that finer-grained abstractions are generally better [Johanson et al., 2013]

An appealing compromise is to pre-calculate the largest possible abstraction we can handle for the entire game and then improve this in real-time with refinements. The original strategy is used to play the early parts of the game (the trunk) and once the remaining portion of the game tree (the subgame) becomes tractable, we can refine the strategy for the subgame in real-time using even finer-grained abstraction. Figure 7.1 illustrates the approach.

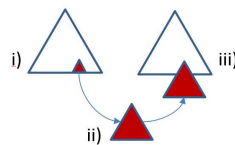


Figure 7.1: Subgame refinement framework. (i) the strategy for the game is pre-computed using coarse-grained abstraction (ii) during the play, once we reach a node defining a sufficiently small subgame, we refine the strategy for that subgame (iii) this together with the original strategy for the trunk creates a combined strategy. The point is to produce improved combined strategy

Note that not only can we enlarge the size of the abstraction in the subgame, we can also reduce the “off the tree problem”. When an opponent takes an action that is not found in the abstraction, it needs to be mapped onto a (similar) one in the abstraction. This mapping can destroy relevant game information. To reduce this effect, we can construct the subgame so that it starts in the exact state of the game so far [Ganzfried and Sandholm, 2015].

Subgame refinement has been successfully used in perfect information games to improve the strategies [Müller and Gasser, 1996] [Müller, 2002]. Unfortunately, the nature of imperfect information games means that it is difficult to isolate subgames. Current attempts to apply subgame refinement to imperfect information games have led to marginal gains or potentially result in a more exploitable final solution. The reason for this is that if we change our strategy in the subgame then this gives our opponent the opportunity to exploit our combined strategy by altering their behavior in the trunk of the game. See [Burch et al., 2014] or [Ganzfried and Sandholm, 2015] for details and several nice examples of this flaw.

The first approach, “endgame solving”, does not guarantee a decrease in exploitability, and can instead produce a strategy that is drastically more exploitable. [Ganzfried and Sandholm, 2015]. The second approach, re-solving, was originally designed for subgame strategy re-solving. In other words, it aims to reproduce the original strategy from a compact representation. The resulting strategy is guaranteed to be no more exploitable than the original one. Although this technique can be used to refine the subgame strategy, there is no explicit construction that forces the refined strategy to be any better than the original, even if much stronger strategies exist. [Burch et al., 2014]

In this chapter, we present a new technique, max-margin subgame refinement, that is tailor-made to reduce exploitability in imperfect information games. We introduce the notion of subgame margin, a simple value with appealing properties, which motivates subgame refinements that result in large positive margins.

We regard the problem of safe subgame refinement as a linear optimization problem. This perspective demonstrates the drawbacks and connections between the two previous approaches, and ultimately introduce linear optimization to maximize the subgame margin. Subsequently, we describe an imperfect information game construction that can be used to find such a strategy (rather than solving the resulting linear optimization problem). This allows us to solve larger subgames using recently introduced techniques, namely the CFR+ [Tammelin et al., 2015] and domain-specific speedup tricks [Johanson et al., 2012].

Finally, we experimentally evaluate all the approaches - endgame solving, re-solving and max-margin subgame refinement. For the first time, we evaluate these techniques on the safe-refinement task as part of a large-scale game by using one of the top participating agents in AAI-14 Computer Poker Competition as the baseline strategy to be refined in subgames.

## 7.2 Previous Work

Despite the lack of theoretical guarantees, variants of subgame refinement have been used in imperfect information games for some time. The poker agent GS1-G4 [Gilpin and Sandholm, 2006] [Gilpin et al., 2007] and its successor Tartanian [Ganzfried and Sandholm, 2013] [Ganzfried and Sandholm, 2015] used various techniques to either refine or solve the endgame. The authors call their newest version of their approach “endgame solving”, and report both positive practical performance results as well as potentially negative impacts on the exploitability of the combined strategy [Ganzfried and Sandholm, 2015]. This is a property shared by all of these variants - the resulting strategy can be substantially more exploitable than the original strategy started with.

We are aware of only one prior subgame refinement technique that is guaranteed to produce a combined strategy that is no-more exploitable than the original strategy, re-solving [Burch et al., 2014] The technique works by computing the best response values for the opponent and using these values to construct a gadget game. Unfortunately, there is no explicit mechanism to cause the refined strategy to be any better than the original one, even

if much stronger strategies are possible. By formulating this technique as an optimization problem, we can easily see this property.

## 7.3 Background and Notation

Notation of this chapter is based on the definition of extensive form game from chapter 2

### 7.3.1 Counterfactual Best Response

A **counterfactual best response**  $CBR_p(\sigma)$  is a strategy where

$\sigma_p(I, a) > 0$  iff  $v_p^{\sigma|_{I \rightarrow a}}(I) = \max_{a'} v_p^{\sigma|_{I \rightarrow a'}}(I)$ . It maximizes counterfactual value at every information set.  $CBR_p$  is always a best response but best response may not be contractual best response since it can choose an arbitrary action in information sets where  $\pi_p(I) = 0$ .

The well-known recursive tree walk algorithm for best response computation produces a counterfactual best response.

To simplify the notation we define a **counterfactual best response value**  $CBV_p^\sigma(I)$ . It is very similar to standard definition of counterfactual value, with exception that player  $p$  plays according to  $CBR_p(\sigma)$  instead of  $\sigma$ . Formally  $CBV_p^\sigma(I) = v_p^{(\sigma_{-p}, CBR_p(\sigma))}(I)$

### 7.3.2 Subgame

In a perfect information game, a subgame is a subtree of the original game tree rooted at any node. This definition is problematic for imperfect information games, since such subtree could include one part of an information set and exclude another. To define a subgame for an imperfect information game, a generalized concept of information set is used. Information set  $I(h)$  groups histories that the acting player  $p = P(h)$  cannot distinguish. **Augmented information set** set adds also histories that any of the remaining players cannot distinguish [Burch et al., 2014]. Using this notion, one can define subgame.

**Definition 7.1.** *An imperfect information subgame [Burch et al., 2014] is a forest of trees, closed under both the descendant relation and membership within augmented information sets for any player.*

Note that root of the subgame, denoted  $R(S)$ , will not typically be a single (augmented) information set because different players typically have different information available to them, thus grouping of histories to augmented information sets will be different. We denote the set of all information sets of the player  $p$  at the root of the subgame as  $\mathcal{I}_p^{R(S)}$ .

### 7.3.3 Formulating Subgame Refinement using Optimization

In this section, we briefly describe the two current techniques - (i) endgame solving [Ganzfried and Sandholm, 2015] and (ii) re-solving [Burch et al., 2014] We also reformulate both of them as equivalent optimization problems. Regarding these techniques as optimizations helps us to see the underlying properties of these two techniques. Subsequently, we use these insights to motivate our new, max-margin technique. We will assume, without loss of generality, that we are refining the strategy for player 1 ( $p_1$ ) for the rest of this chapter.

### 7.3.4 Endgame Solving

We start by constructing a fine-grained subgame abstraction. The original strategies for the subgame are discarded and only the strategies prior to the subgame (trunk) are needed. The strategies in the trunk are used to compute the joint distribution (belief) over the states at the beginning of the subgame. Finally, we add a chance node just before the fine-grained subgame. The node leads to the states at the root of the subgame. The chance node plays according to the computed belief. Adding the chance node roots the subgame, thus making it well-defined game. See Figure 7.2.

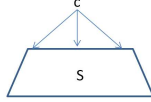


Figure 7.2: Endgame solving construction - Gadget 1. The (c)hance plays according to the belief computed using the trunk’s strategy. The finer-grained (S)ubgame follows.

The following is a formulation of the linear optimization problem corresponding to the game construction.  $LP1$  is the standard sequence form LP for the Gadget 1.

$$\begin{aligned} \max_{v,x} \quad & f^\top v \\ & Ex = e \\ & F^\top v - A_1^\top x \leq 0 \\ & x \geq 0 \end{aligned}$$

*LP1 - optimization problem corresponding to endgame solving.  $A_1$  is the sequence form payoff matrix,  $x$  is the vector of  $p_1$  strategies,  $v$  is the vector of (negative) counterfactual best response values for  $p_2$ ,  $E$  and  $F$  are sequence constraint matrices and  $e$  is sequence constraint vector [Nisan et al., 2007] [Čermák et al., 2014]*

The flaw in this technique stems from the fact that even if the trunk strategy (and thus the starting distribution) is optimal, the combined strategy can become drastically more exploitable. [Ganzfried and Sandholm, 2015] [Burch et al., 2014]

### 7.3.5 Re-solving

Again, we start by creating a fine-grained abstraction for the subgame. The original strategy for the subgame (from the coarse abstraction) is then translated into the fine-grained abstraction as  $\sigma_1^S$ . The translated strategy is now used to compute  $CBV_2^{\sigma_1^S}(I)$  for every information set  $I$  at the root of the subgame. These values will be useful for the gadget construction to guarantee the safety of the resulting strategy.

To construct the gadget, we add one chance node at the root of the game, followed by additional nodes for  $p_2$  - one for every state at the root of the subgame. At each of these nodes,  $p_2$  may either accept the corresponding counterfactual best response value calculated earlier or play the subgame (to get to the corresponding state at the root of the subgame). The chance player distributes the  $p_2$  into these states using the (normalized)  $\pi_{-2}^\sigma$  (how likely is the state given that  $p_2$  plays to reach it). Since the game is zero sum, this forces  $p_1$  to play the subgame well enough that the opponent’s value is no greater than the original  $CBV$ . See Figure 7.3 for a sketch of the construction. For more details see [Burch et al., 2014].



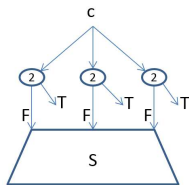


Figure 7.3: re-solving gadget construction - Gadget 2. The opponent chooses in every state prior to the endgame to either (F)ollow the action into the endgame or to (T)erminate. His utility after the (T)erminal action is set to his counterfactual best response in that state.

Next, we formulate a linear optimization problem corresponding to the gadget construction. This time, the presented LP is not a straightforward sequential-form representation of the construction. Although such a representation would be possible, it would not help provide the insight we are seeking. Instead, we formulate a LP that solves the same game (for the  $p_1$ ) while demonstrating the underlying properties of the re-solving approach. The formulation uses the fact that any strategy for which the opponent's current counterfactual best response is no greater than the original one, is a solution to the game (this follows from the construction of Gadget 2).

$$\begin{aligned}
 & \max_{v,x} \mathbf{0} \\
 & v_I \geq CBV_2^\sigma(I), \quad I \in \mathcal{I}_2^{R(S)} \\
 & Ex = e \\
 & F^\top v - A_2^\top x \leq 0 \\
 & x \geq 0
 \end{aligned}$$

$LP2 - \mathcal{I}_2^{R(S)}$  denotes the root information sets,  $CBV_2^\sigma(I)$  is the original counterfactual best response value of  $p_2$  in the information set  $I$ . The sequence payoff matrices  $A_1$  and  $A_2$  are slightly different to reflect different strategy of the chance player in Gadget 1 and Gadget 2.

It is worth noting three critical points here.

1.  $LP2$  is not maximizing any value, but rather finding a feasible solution (though theoretically equivalent, it is semantically different for the strategy in this case).
2. The original, unrefined strategy is a solution to  $LP2$
3. Although 1) and 2) suggest that the strategy might not improve, empirical evaluations show that if one uses a  $CFR$  algorithm to solve the corresponding game (Gadget 2), the refined strategy's performance improves upon the original[Burch et al., 2014]. Our experiments further confirm this.

### 7.3.6 Discussion

Looking at the  $LP1$  and  $LP2$ , it's easy to see the properties of existing approaches. The  $LP1$  (endgame solving) lacks the constraints ( $v_I \geq CBV_2^\sigma(I)$ ) that bound the exploitability, possibly producing strategy drastically more exploitable than the original one.  $LP2$  (re-solving) bounds the exploitability, but lacks maximization factor, possibly producing strategies no better than the original one. As we will see, our approach both bounds the exploitability while maximizing some well-motivated function.

## 7.4 Our Technique

The outline of this section is following: 1. we list the steps used by our technique 2. we use the problem of refining imperfect information subgames to motivate a value to maximize 3. we formalize this value as the subgame margin 4. we discuss and formalize its properties 5. we formulate an LP optimizing the subgame margin 6. we describe a corresponding extensive form game construction - Gadget 3

Our technique follows the steps of the subgame refinement framework: (i) Create an abstraction for the game. (ii) Compute an equilibrium approximation within the abstraction. (iii) Play according to this strategy. (iv) When the play reaches final stage of the game, create a fine-grained abstraction for the endgame. (v) Refine the strategy in the fine-grained abstraction. (vi) Use the resulting strategy in that subgame (creating a combined strategy).

Since all the steps except of the step five are identical to already described techniques, we describe only this steps in details.

### 7.4.1 Subgame Margin

To address the potential increase in exploitability caused by an opponent altering his behavior in the trunk, we ensure that there is no distribution of starting states that would allow him to increase his  $CBV$  when confronted by subgame refinement. The simplest way to ensure this is to decrease his  $CBV$  in all possible starting states. We can put a lower bound on this improvement by measuring the state with the smallest decrease in  $CBV$ . Our goal is to maximize this lower bound. We refer to this values as the **subgame margin**.

**Definition 7.2.** *Subgame Margin*

Let  $\sigma_1, \sigma'_1$  be a pair of  $p_1$  strategies for subgame  $S$ . Then a subgame margin

$$SM_1(\sigma_1, \sigma'_1, S) = \min_{I_2 \in \mathcal{I}_2^{R(S)}} CBV_2^{\sigma_1}(I_2) - CBV_2^{\sigma'_1}(I_2)$$

Subgame margin has several useful properties. The exploitability is strongly related to the value of the margin. If it is non-negative, the new combined strategy is guaranteed to be no more exploitable than original one. Furthermore, given that the opponent's best response reaches the subgame with non-zero probability, the exploitability of our combined strategy is reduced. This improvement is at least proportional to the subgame margin (and may be greater).

**Theorem 7.3.** *Given a strategy  $\sigma_1$ , a subgame  $S$  and a refined subgame strategy  $\sigma_1^S$ , let  $\sigma'_1 = \sigma_1[S \leftarrow \sigma_1^S]$  be a combined strategy of  $\sigma_1$  and  $\sigma_1^S$ . Let the subgame margin  $SM_1(\sigma_1, \sigma'_1, S)$  be non-negative. Then  $u_1(\sigma'_1, CBR(\sigma'_1)) - u_1(\sigma_1, CBR(\sigma_1)) \geq 0$ . Furthermore, if there is a best response strategy  $\sigma_2^* = BR(\sigma'_1)$  such that  $\pi^{(\sigma'_1, \sigma_2^*)}(I_2) > 0$  for some  $I_2 \in \mathcal{I}_2^{R(S)}$ , then  $u_1(\sigma'_1, CBR(\sigma'_1)) - u_1(\sigma_1, CBR(\sigma_1)) \geq \pi_{-2}^{\sigma'_1}(I_2) SM_1(\sigma_1, \sigma'_1, S)$ .*

*This theorem is generalization of the Theorem 1 in [Burch et al., 2014]. Intuitively, it follows from the way one computes a best response using the bottom-up algorithm. For the formal proof, see appendix A or the authors' homepage.*

Though this lower bound might seem artificial at first, it has promising properties for subgame refinement. Since we refine the strategy once we reach the subgame, we are either facing  $p_2$ 's best response that reaches  $S$  or he has made a mistake earlier in the game. Furthermore, the probability of reaching a subgame is proportional to  $\pi_{-2}^{\sigma'_1}(I_2)$ . As this term (and by extension, the bound) increases, the probability of reaching that subgame grows. Thus, we are more likely to reach a subgame with larger bound.

## 7.4.2 Optimization Formulation

To find a strategy that maximizes the subgame margin, we can easily modify the  $LP2$ .

$$\begin{aligned}
 & \max_{v,x} m \\
 & v_I - m \geq CBV_2^\sigma(I), \quad I \in \mathcal{I}_2^{R(S)} \\
 & Ex = e \\
 & F^\top v - A_2^\top x \leq 0 \\
 & x \geq 0
 \end{aligned}$$

$LP3$  - maximizing the subgame margin,  $m$  is scalar corresponding to the subgame margin that we aim to maximize.

The similarities between  $LP3$  and  $LP2$  make it easier to see that where the  $LP2$  optimization guarantees non-negative margin, we maximize it. While the optimization formulation is almost identical to the re-solving, our gadget construction is different.

## 7.4.3 Gadget Game

One way to find the refined strategy is to solve the corresponding linear program. However, algorithms that are tailor-made for extensive form games often outperform the optimization approach [Bošanský, 2013]. These algorithms often permit the use of domain-specific tricks to provide further performance gains [Johanson et al., 2012]. Thus, formulating our optimization problem  $LP3$  as an extensive form game will mean that we can compute larger subgame abstractions using the available computing resources. Essentially, the construction of a Gadget 3 corresponding to the  $LP3$  will allow us to compute larger subgames than would be possible if we simply used  $LP3$ . We now provide the construction of such a gadget game.

## 7.4.4 Gadget Game Construction

All states in the original subgame are directly copied into the resulting gadget game. We create the gadget game by making two alterations to the original subgame. (i) we shift  $p_2$ 's utilities using the  $CBV_2$  (To initialize all  $p_2$  values to zero) and (ii) we add a  $p_2$  node followed by chance nodes at the top of the subgame (to allow the opponent to pick any starting state, relating the game values to margin) We will distinguish the states, strategies, utilities, etc. for the gadget game by adding a tilde to corresponding notation. The following is a description of the steps (see also Figure 7.4 that visualizes the constructed Gadget 3)

1. We establish a common baseline. To compare the changes in the performance of each of  $p_2$ 's root information sets, it is necessary to give them a common baseline. We use the original strategy  $\sigma_1^S$  as the starting point. For every  $I \in \mathcal{I}_2^{R(S)}$ , we subtract the opponent's original counterfactual best response value, setting the utility at each terminal node  $z \in Z(I)$  to  $\tilde{u}_2(z) = u_2(z) - CBV_2^{\sigma_1^S}(I)$  (we also update  $\tilde{u}_1(\tilde{z}) = -\tilde{u}_2(\tilde{z})$  since we need the game to remain zero-sum). This shifting gives all of our opponent's starting states a value of zero if we do not deviate from our original strategy  $\sigma_1^S$ .
2.  $p_2$  is permitted to choose his belief at the start of the subgame, while  $p_1$  retains his belief from the original strategy at the point where the subgame begins. Since  $p_2$

is aiming to maximize  $\tilde{u}_2$ , he will always select the information set with the lowest margin. The minimax nature of the zero-sum game forces  $p_1$  to find a strategy that maximizes this value. We add additional decision node  $\tilde{d}$  for  $p_2$ . Each action corresponds to choosing an information set  $I$  to start with, but we do not connect this action directly to this state. Instead, each action leads to a new chance node  $s_{\tilde{I}}$ , where the chance player chooses the histories  $h \in \tilde{I}$  based on the probability  $\pi_{-2}^\sigma(h)$ .

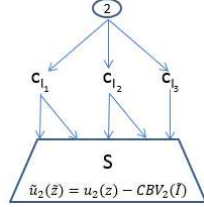


Figure 7.4: Max margin gadget - Gadget 3. Notice that given the original strategy of  $p_1$ , opponent’s best response utility is zero (thanks to the offset of terminal utilities).

**Lemma 7.4.** *Strategy for the Gadget 3 is Nash Equilibrium if and only if it’s a solution to the LP3*

*Follows from the construction of the Gadget 3.*

## 7.5 Experiments

In this section, we evaluate endgame solving, re-solving and max-margin subgame refinement on the safe-refinement task for a large-scale game. We use an improved version of the Nyx agent, the second strongest participant at the 2014 Annual Computer Poker Competition (heads-up no-limit Texas Hold’em Total Bankroll) as the baseline strategy to be refined in subgames.

All three of the subgame refinement techniques tested here used the same abstractions and trunk strategy. Following [Ganzfried and Sandholm, 2015], we begin the subgame at start of the last round (the river). While we used card abstraction to compute the original (trunk) strategy (specifically [Schmid et al., 2015] and [Johanson et al., 2013]), the fine-grained abstraction for the endgame is calculated without the need for card abstraction. This is an improvement over the original implementation [Ganzfried and Sandholm, 2015], where both the trunk strategy and the refined subgame used card abstraction. This is a result of the improved efficiency of the CFR+ algorithm (and the domain-specific speedups it enables), whereas the endgame solving in [Ganzfried and Sandholm, 2015] used linear programming to compute the strategy.

The original strategy uses action abstraction with up to 16 actions in an information set. While this number is relatively large compared to other participating agents, it is still distinctly smaller compared to the best-known upper-bound on the size of the support of an optimal strategy [Schmid et al., 2014]. In contrast to the action abstraction used for the original Nyx strategy that uses imperfect recall for the action abstraction, the refined subgame uses perfect recall. We use the same actions in the refined subgame as in the original strategy.

We refine only the subgames that (after creating the fine-grained abstraction) are smaller than 1,000 betting sequences - this is simply to speed up the experiments. The original agent strategy is used for both  $p_1$  and  $p_2$  in the trunk of the game. Once gameplay reaches

the subgame (river), we refine the  $P1$  strategy using each of the three techniques. We ran 10,000 iterations of the CFR+ algorithm in the corresponding gadget games. Exponential weighting is used to update the average strategies [Tammelin et al., 2015]. Each technique was used to refine around 2,000 subgames. Figure 7.5 visualizes the average margins for the evaluated techniques.

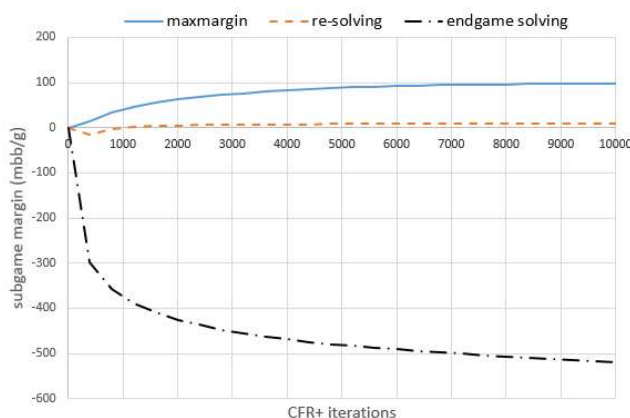


Figure 7.5: Subgame margins of the refined strategies. One big blind corresponds to 100 chips. The max-margin technique produces the optimal value. We see that the optimal value is much greater than the one produced by either re-solving or endgame solving (which produces even negative margins). The 95% confidence intervals for the results (after 10,000 iterations) are: maxmargin  $101.49 \pm 7.09$ , re-solving  $8.79 \pm 2.45$ , endgame solving  $-518.5 \pm 49.19$

**Endgame Solving** The largely negative margin values for the endgame solving suggest that the produced strategy may indeed be much more exploitable.

**Re-solving** The positive margin for re-solving shows that, although there’s no explicit construction that forces the margin to be greater than zero, it does increase in practice. Notice, however, that the margin is far below the optimal level.

**Max-margin Refinement** This technique produces a much larger subgame margin than the previous techniques. The size of the margin suggests that the original strategy is potentially quite exploitable, and our technique can substantially decrease the exploitability - see Theorem 7.3.

## 7.6 Conclusion

We have introduced max-margin subgame refinement, a new technique for subgame refinement of large imperfect information games. The subgame margin is a well-motivated value with appealing properties for endgame solving, namely regarding the resulting exploitability. We formalized and proved these properties in Theorem 1. As the name of the our technique suggests, the technique aims to maximize this well-motivated value. We also formulated our approach using both linear optimization and extensive form game (gadget) construction. Experimental results have confirmed that our gadget game successfully finds refined strategies with substantially larger margins than previous approaches. The rather large values of the margin that the technique provided suggest that even though we evaluated the technique using a state-of-the-art strategy, such strategies still contain tremendous space for improvement in such large games.

## **7.7 Author's contributions**

I significantly contributed to the formulation of the original algorithm idea, which included LP formulation and gadget game design, as well as to the implementation of the algorithm.

# 8. AIVAT: A New Variance Reduction Technique for Agent Evaluation in Imperfect Information Games

This chapter is based on [Burch et al., 2018]

## 8.1 Introduction

Evaluating an agent’s performance in stochastic settings can be hard. Non-zero variance in outcomes means the game must be played multiple times to compute a confidence interval that likely contains the true expected value. Regardless of whether the variance arises from player actions or from chance events, we might need to observe many samples before we get a narrow enough interval to draw desirable conclusions. In many situations, it is simply not feasible (e.g., when the evaluation involves human participation) to simply observe more samples, so we must turn to statistical techniques that use additional information to help narrow the confidence interval.

This agent evaluation problem is commonly encountered in games, where the goal is to estimate the expected performance difference between players. For example, consider poker games. Poker is not only a long-standing challenge problem for AI [von Neumann, 1928, Koller and Pfeffer, 1997, Billings et al., 2002] with annual competitions [Zinkevich and Littman, 2006, Bard et al., 2013], but also a very popular game played by an estimated 150 million players worldwide [Economist, 2007]. Heads-up no-limit Texas hold’em (HUNL) is a particular variant of the game that has received considerable attention in the AI community in recent years, including a “Brains vs. AI” event pitting Claudico [cmu, 2015], a top HUNL computer program, against professional poker players. That match involved 80,000 hands of poker, played over seven days, involving four poker players, playing dozens of hours each. Despite Claudico losing by over 9 big blinds per 100 hands (a margin that is considered huge by poker professionals) [Wood, 2015], the result is only on the edge of statistical significance, making it hard to draw a conclusion from this large investment of human time.

Previous techniques for variance reduction to achieve stronger statistical conclusions in this setting have used two broad classes of statistical techniques. Techniques like MIVAT [White and Bowling, 2009] use the method of control variates with heuristic value estimates to reduce the variance caused by chance events. The technique of importance sampling over imaginary observations [Bowling et al., 2008] takes a different approach, using knowledge of a player strategy to evaluate multiple states given a single observation. Imaginary observations can be used to reduce the variance caused by privately observed chance events, as well as the player’s randomly chosen choice of whether to make any actions which would immediately end the game.

Techniques from the two classes can be combined, but are not specifically designed to work together for the greatest reduction in variance, and none of the techniques deal with the variance caused by non-terminal action selection. Because good play in imperfect information games generally requires randomised action selection, ignoring action variance is an important shortcoming. We introduce the action-informed value assessment tool (AIVAT), an unbiased low-variance estimator for imperfect information games which extends the use of control variates to player actions, and makes explicit use of imaginary observations to exploit knowledge of the game structure and player strategies.

## 8.2 Value Estimation

When talking about estimating the value for players in a game, we are trying to find the expected value  $\mathbb{E}_z[v_p(z)] = \sum_{z \in Z} \pi(z)v_p(z)$ . An estimator  $e(z)$  is said to be unbiased if the expected value  $\mathbb{E}_z[e(z)] = \mathbb{E}_z[v_p(z)]$ . Having an estimator be provably unbiased is important because it is in some sense truthful: a player can not appear to do better by changing their play to take advantage of the estimation method.

## 8.3 MIVAT and Imaginary Observations

AIVAT is an extension of two earlier techniques, MIVAT and importance sampling over imaginary observations. MIVAT [White and Bowling, 2009] and its precursor DIVAT [Zinkevich et al., 2006] use value functions for a control variate that estimates the expected utility given observed chance events. Conceptually, the techniques subtract the expected chance utility to get a lower variance value which mostly depends on the player actions. For example, in poker, it is likely that good hands end in positive outcomes and bad hands end in negative outcomes. Starting with the observed outcome, we could subtract some value for good hands and add a value for bad hands, and we would expect the corrected value to have lower variance. If the expected value of the correction terms is zero, we can use the lower variance corrected value as an unbiased estimator of player value.

DIVAT requires a strategy for all players to generate value estimates for states through self-play, which MIVAT generalised by allowing for arbitrary value functions defined after chance events. MIVAT adds a correction term for each chance event in an observed state. In order to remain unbiased despite using an arbitrary value estimation function  $u(a)$ , MIVAT uses a correction term of the form  $\mathbb{E}_a[u(a)] - u(o)$  for an observation with outcome  $o$ . Computing this expectation requires us to know the probability distribution that  $o$  was drawn from, which is true in the case of chance events as  $\sigma_{pc}$  is public knowledge. These terms are guaranteed to have an expected value of zero, making the MIVAT value (observed value plus correction terms) an unbiased estimate of player value. In a game like poker, MIVAT will account for the dealer giving a player favourable or unfavourable cards, but not for lucky player actions selected from a randomised strategy.

Imaginary observations with importance sampling [Bowling et al., 2008] use knowledge of a player's strategy to compute an expected value of multiple states given an observation of a single state. Due to imperfect information, there may be many states which are all guaranteed to have the same probability of the opponent making their actions. If we consider importance sampling over these imaginary observations, the opponent's probability of reaching the state cancels out so we do not need the opponent's strategy. By taking an expectation over a set of states for every observation, we get a lower variance value.

There are two kinds of situations where we can use imaginary observations. First, for any states  $h$  where player  $p$  could have made an action  $a$  which ends the game, we can add the imaginary observation of the terminal state  $h \cdot a$ . For example, in poker this lets us consider player  $p$  folding to a bet they called or raised, or calling a bet we folded to in the final round. Second, because of the information partitions in imperfect information games, there may be other states that have identical opponent probabilities. In poker, this lets us consider all the states where the public player actions are the same, the opponent private cards and public board cards are the same, but player  $p$  has different private cards. Imaginary observations do not let us reduce the variance caused by choosing non-terminal actions or the outcomes of publicly visible chance events.

MIVAT and imaginary observations consider different information and can be combined



to get a value estimate with lower variance than either technique used individually. Instead of using the terminal value  $v(z)$  for an imaginary observation  $z$ , we could use the MIVAT value estimate given  $z$ . However, because neither technique has terms which address the effect of non-terminal actions, we would never expect this combination of techniques to produce a zero variance value estimate. Even with a “perfect” value function that correctly estimates the expected value of a state and action for the players, there would still be some variance in the value estimate due to the random action selection by players.

## 8.4 AIVAT

Conceptually, AIVAT combines the chance correction terms of MIVAT with imaginary observations across private information, along with new MIVAT-like correction terms for player actions. The AIVAT estimator is the sum of a base value using imaginary observations, plus imaginary observation correction terms for both player actions and chance events. Roughly speaking, moving backwards through the choices in an observed game, the AIVAT correction terms are constructed in a fashion that shifts an estimate of the expected value after a choice was made towards an estimate of the expected value before the choice.

Because imaginary observations with importance sampling provides an unbiased estimate of the expected value of the players, and the MIVAT-like terms have an expected value of zero, AIVAT is also an unbiased estimator of the expected player value. Furthermore, with well-structured games, “perfect” value functions, and knowledge of all player strategies, we could see zero variance: the imaginary observation values and the correction terms would sum to the expected player value, regardless of the observed game.

Figure 8.1 gives a high level overview of MIVAT, imaginary observations, and AIVAT. In this example, we are interested in the expected value for player 1, and know player 1’s strategy. We use an observation of one hand of Leduc hold’em poker, a small synthetic game constructed for artificial intelligence research [Southey et al., 2005]. Leduc hold’em is a two round game with one private card for each player, and one publicly visible board card that is revealed after the first round of player actions. In the example, player 1 is dealt  $\mathbf{Q}\spadesuit$  and player 2 is dealt  $\mathbf{K}\spadesuit$ . Player 1 makes the **check** action followed by a player 2 **check** action. The public board card is revealed to be  $\mathbf{J}\heartsuit$ . After the round two actions **check**, **raise**, **call**, player 1 loses 5 chips.

### 8.4.1 AIVAT Correction Terms

We start by describing the correction terms added for chance events and actions. Given information about a player’s strategy, we can treat that player’s choice events as chance events and construct MIVAT-like correction terms for them. The player strategy also allows imaginary observations considering alternative histories with identical opponent probabilities, so we can compute an expectation over a set of compatible histories rather than using the single observed outcome.

The correction term at a decision point will be the expectation across all compatible histories of the expected value before a choice, minus the value after the observed choice. As with MIVAT, the values are estimated using an arbitrary fixed value function to estimate the value after every decision. Value estimates which more closely approximate the true expected value will result in greater variance reduction.

To consider imaginary observations, we need at least one player for which we know the strategy. Let  $P_a$  be a non-empty set of players, including  $p_c$ , such that  $\forall p \in P_a$  we know  $\sigma_p$ , and  $P_o = P \setminus P_a$  be the set of opponent players for which we do not know the

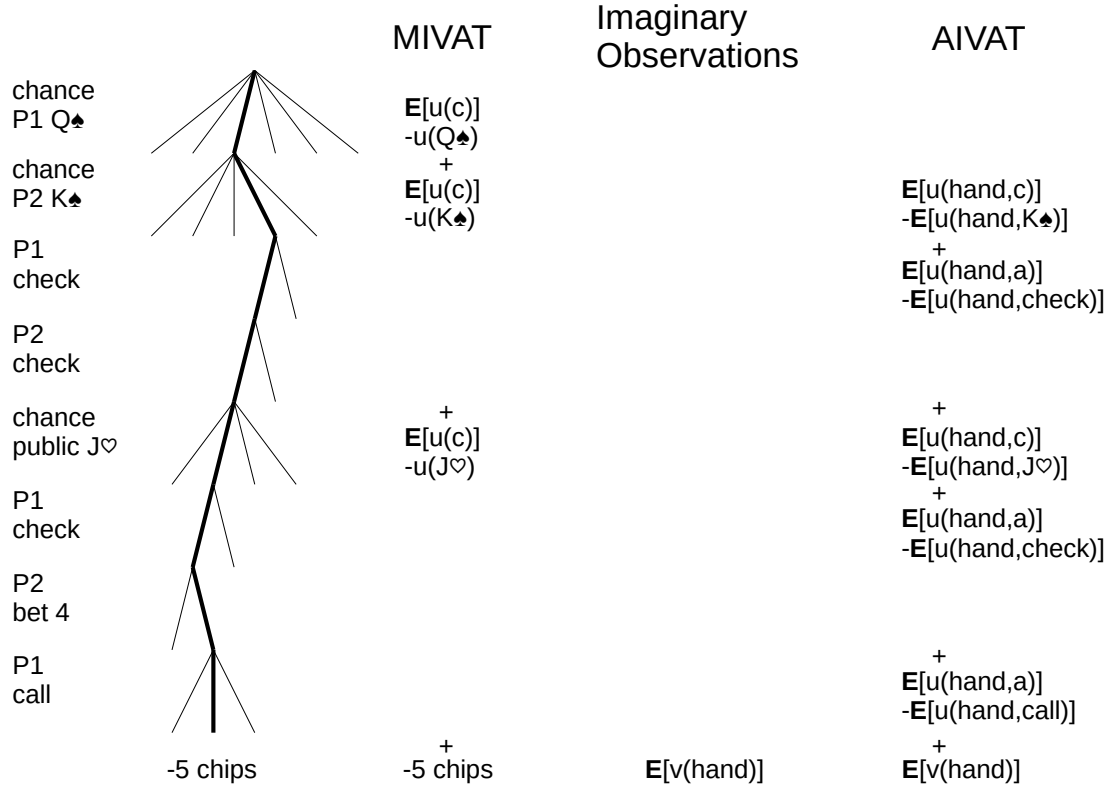


Figure 8.1: Comparison of MIVAT, imaginary observations, and AIVAT. On the left, there are all the actions that occurred in the game and the corresponding trajectory in the game tree. Next, for each variance reduction technique, we show the correction terms corresponding to these actions. Note that for MIVAT, corrections are non-zero only for the chance actions and for imaginary observations there there is only one correction correction at the game end. In contrast AIVAT leverages combination of correction terms corresponding to each action (except the first) and to the game end. This allows for a significantly larger variance reduction than previous techniques.

strategy. If  $P_a = \{p_c\}$  then AIVAT would be identical to MIVAT. We must also partition the states into the sets we can evaluate given an observation of a completed game. Let  $\mathcal{H}$  be a partition of states  $\{h|p(h) \in P_a\}$  such that  $\forall H \in \mathcal{H}$  and  $\forall h, h' \in H$ ,

1.  $\forall p \in P_o \forall \sigma_p \pi_p(h) = \pi_p(h')$ . For example, this can be enforced by requiring  $h$  and  $h'$  to pass through the same sequence of player  $p$  information sets and make the same actions at those information sets.
2.  $h \not\sqsubset h'$ . This implies a uniqueness property, where for any terminal  $z$ ,  $\{h''|h'' \sqsubset z, h'' \in H\}$  is either empty or a singleton.
3. We will extend the actions so that  $A'(h) = \bigcup_{h'' \in H} A(h'')$  and let  $\sigma(h, a) = 0 \forall a \in A'(h) \setminus A(h)$ . Because  $A'(h) = A'(h')$  we will say  $A(H) = A'(h)$ .

Similar to MIVAT, we need value functions that give an estimate of the expected value after an action. Let there be arbitrary functions  $u_h(a) : A'(h) \mapsto \mathbf{R}$  for each state  $h$  where  $p(h) \in P_a$ . Say we have seen a terminal state  $z$ . Consider a part  $H \in \mathcal{H}$ . If  $\nexists h \in H$  such that  $h \sqsubset z$ , then the correction term  $k_H(z) = 0$ . Otherwise, property 2 of  $\mathcal{H}$  implies there

is a unique observed action  $a_O$  such that  $h \cdot a_O \sqsubseteq z$ ,  $h \in H$ ,  $a_O \in A(h)$ , and the correction term is

$$k_H(z) = \frac{\sum_{a \in A(H)} \sum_{h \in H} \pi_{P_a}(h \cdot a) u_h(a)}{\sum_{h \in H} \pi_{P_a}(h)} - \frac{\sum_{h \in H} \pi_{P_a}(h \cdot a_O) u_h(a_O)}{\sum_{h \in H} \pi_{P_a}(h \cdot a_O)}$$

AIVAT uses the sum of  $k_H(z)$  across all  $H \in \mathcal{H}$ .

## 8.4.2 AIVAT Base Value

The AIVAT correction terms have an expected value of zero, and are not a value estimate by themselves. They must be combined with an unbiased estimate of player value. For improved variance reduction, the form of the correction terms must match the choice of base value estimate.

To see how the terms match, consider a simplified version of AIVAT where the final correction term for a terminal state  $h \cdot o$  has the form  $\mathbb{E}_a[u_h(a)] - u_h(o)$ . Ideally, we would like the value estimate for  $h \cdot a$  to be  $u_h(a)$ . The value estimate plus the correction term will then have the same value  $\mathbb{E}_a[u_h(a)]$  for all actions at  $h$ , resulting in zero variance.

For the AIVAT correction terms, the correct choice is to use imaginary observations of all possible private information for players in  $P_a$ , as in “Example 3: Private Information” of the paper by Bowling *et al.* [Bowling et al., 2008]. In poker, it corresponds to evaluating the game with all possible private cards, weighted by the likelihood of holding the cards given the observed game. For completeness, we formally describe the particular instance of this existing estimator using the notation of this chapter.

Given the correction term partition  $\mathcal{H}$  of player  $P_a$  states, we construct a matching partition  $\mathcal{W}$  of terminal states such that  $\forall W \in \mathcal{W}$  and  $\forall z, z' \in W$ ,

- $\forall p \in P_o \forall \sigma_p \pi_p(z) = \pi_p(z')$ .
- a player in  $P_a$  made an action in  $z \iff$  a player in  $P_a$  made an action in  $z'$ .
- if a player in  $P_a$  made an action in  $z$ , then for the longest prefix  $h \sqsubseteq z$  and  $h' \sqsubseteq z'$  such that  $p(h) \in P_a$  and  $p(h') \in P_a$ , both  $h$  and  $h'$  are in the same part of  $\mathcal{H}$ .

The last two conditions on  $\mathcal{W}$  ensure that the imaginary observation estimate does not include terminal states that the correction terms will also account for. This rules out a form of double counting which would not produce a biased estimator, but would increase the variance when using high quality estimates in the correction terms.

If we observe a terminal state  $z$ , let  $W \in \mathcal{W}$  be the part such that  $z \in W$ . The base estimated value for player  $p$  is

$$\frac{\sum_{z' \in W} \pi_{P_a}(z') v_p(z')}{\sum_{z' \in W} \pi_{P_a}(z')}$$

## 8.4.3 AIVAT Value Estimate

The AIVAT estimator gives an unbiased estimate of the expected value  $\mathbb{E}_z[v_p(z)]$ . If we use partitions  $\mathcal{H}$  and  $\mathcal{W}$  as described above, and are given an observation of a terminal state  $z \in W \in \mathcal{W}$ , the value estimate is

$$\text{AIVAT}(z) = \frac{\sum_{z' \in W} \pi_{P_a}(z') v_p(z')}{\sum_{z' \in W} \pi_{P_a}(z')} + \sum_{H \in \mathcal{H}} k_H(z) \quad (8.1)$$

Note that there is a subtle difference between AIVAT and a simple combination of imaginary observations and an extended MIVAT framework using player strategy information to add control variates for actions. Using an extended MIVAT plus imaginary observations, we would consider the expected MIVAT value estimate across all terminal histories compatible with the observed terminal state. In AIVAT, for each correction term we would consider all histories compatible with the state at that decision point.

As a concrete example of the difference, consider the game used in Figure 8.1. MIVAT with imaginary observations would only consider private cards for player 1 that do not conflict with the opponent's  $\mathbf{K}\spadesuit$  or the public card  $\mathbf{J}\heartsuit$ , even when computing the  $\mathbb{E}[u(c)] - u(\mathbf{J}\heartsuit)$  control variate term for the public card. In contrast, AIVAT considers  $\mathbf{J}\heartsuit$  as a possible player card for the term.

## 8.5 Unbiased Value Estimate

It is desirable to have an unbiased value estimate for games, so that players can not improve their estimated value by changing their strategy to fit the estimation technique. We prove that AIVAT is unbiased. The value estimate  $\text{AIVAT}(z)$  in Equation 8.1 is a sum of two parts. The fraction in the first part is an unbiased estimator based on imaginary observations [Bowling et al., 2008], so we only need to show that the sum of all  $k_H$  terms has an expected value of 0.

**Lemma 8.1.**  $\forall H \in \mathcal{H} \mathbb{E}_{z \in Z} [k_H(z)] = 0$

*Proof.* Consider an arbitrary  $H \in \mathcal{H}$ . Let  $Z(H) = \{z \in Z \mid \exists h \in H, h \sqsubset z\}$  be the set of terminal states passing through  $H$ . Expanding definitions, using property 1 of  $\mathcal{H}$  and multiplying by  $\pi_{P_o}(H)/\pi_{P_o}(H) = 1$  we get

$$\begin{aligned} \mathbb{E}_{z \in Z} [k_H(z)] &= \sum_{z \in Z} \pi(z) k_H(z) = \sum_{z \in Z(H)} \pi(z) k_H(z) \\ &= \sum_{z \in Z(H)} \pi(z) \frac{\pi_{P_o}(H)}{\pi_{P_o}(H)} \frac{\sum_{a \in A(H)} \sum_{h \in H} \pi_{P_a}(h \cdot a) u_h(a)}{\sum_{h \in H} \pi_{P_a}(h)} \\ &\quad - \sum_{z \in Z(H)} \pi(z) \frac{\pi_{P_o}(H)}{\pi_{P_o}(H)} \frac{\sum_{h \in H} \pi_{P_a}(h \cdot a_O) u_h(a_O)}{\sum_{h \in H} \pi_{P_a}(h \cdot a_O)} \end{aligned}$$

Using  $\pi_{P_o}(h) \pi_{P_a}(h) = \pi(h)$

$$\begin{aligned} &= \sum_{z \in Z(H)} \pi(z) \frac{\sum_{a \in A(H)} \sum_{h \in H} \pi(h \cdot a) u_h(a)}{\sum_{h \in H} \pi(h)} \\ &\quad - \sum_{z \in Z(H)} \pi(z) \frac{\sum_{h \in H} \pi(h \cdot a_O) u_h(a_O)}{\sum_{h \in H} \pi(h \cdot a_O)} \end{aligned}$$

Using  $\sum_{z, h \sqsubset z} \pi(z) = \pi(h)$  and  $\sum_{z, h \cdot a \sqsubset z} \pi(z) = \pi(h \cdot a)$

$$\begin{aligned} &= \sum_{h' \in H} \pi(h') \frac{\sum_{a \in A(H)} \sum_{h \in H} \pi(h \cdot a) u_h(a)}{\sum_{h \in H} \pi(h)} \\ &\quad - \sum_{h' \in H} \sum_{a \in A(h')} \pi(h' \cdot a) \frac{\sum_{h \in H} \pi(h \cdot a) u_h(a)}{\sum_{h \in H} \pi(h \cdot a)} \end{aligned}$$

Using property 3 of  $\mathcal{H}$

$$\begin{aligned}
&= \sum_{h' \in H} \pi(h') \frac{\sum_{a \in A(H)} \sum_{h \in H} \pi(h \cdot a) u_h(a)}{\sum_{h \in H} \pi(h)} \\
&\quad - \sum_{a \in A(H)} \sum_{h' \in H} \pi(h' \cdot a) \frac{\sum_{h \in H} \pi(h \cdot a) u_h(a)}{\sum_{h \in H} \pi(h \cdot a)} \\
&= \sum_{a \in A(H)} \sum_{h \in H} \pi(h \cdot a) u_h(a) - \sum_{a \in A(H)} \sum_{h \in H} \pi(h \cdot a) u_h(a) \\
&= 0
\end{aligned}$$

Because the expected value is 0 for an arbitrary  $H$ , the expected value is 0 for the sum of all  $H \in \mathcal{H}$ .  $\square$

**Theorem 8.2.**  $\mathbb{E}_{z \in Z} [\sum_{H \in \mathcal{H}} k_H(z)] = 0$

*Proof.* This immediately follows from Lemma 8.1, as the expected value of a sum of terms is the sum of the expected values of the terms, which are all 0.  $\square$

## 8.6 Experimental Results

We demonstrate the effectiveness of AIVAT in two poker games, Leduc hold'em and heads-up no-limit Texas hold'em (HUNL). Both Leduc hold'em and HUNL have a convenient structure where all actions are public, and there is a mix of chance events in the form of completely public board cards and completely private hole cards. The uncomplicated structure leads to a clear choice for the partition  $\mathcal{H}$ . Each  $H \in \mathcal{H}$  has states with identical betting, public board cards, and private hole cards for any players in  $P_o$ .

In all experiments the value functions  $u_h(a)$  are self-play values, generated by solving the game to find a Nash equilibrium strategy using a variant of the Monte Carlo CFR algorithm [Lanctot et al., 2009]. For each player  $p_x$  and partition  $H$ , we save the average observed values for opponent  $p_y$  across all iterations, giving us a value  $w_H(a) \approx \sum_{h \in H} \pi_{p_x}(h \cdot a) \mathbb{E}[v_{p_y}(h)] / \sum_{h \in H} \pi_{p_x}(h \cdot a)$ .  $w_H(a)$  is an expected self-play value for  $p_y$  at  $H$ , given the probability distribution of hands for  $p_x$  that reach  $H$  and play  $a$ . Because we are playing a zero-sum game and  $v_{p_x}(h) = -v_{p_y}(h)$ , we can use  $u_h(a) = -w_H(a) \forall h \in H$ . In HUNL, which is too large to solve directly, we solve a very small abstraction of the game [Billings et al., 2003a, Ganzfried and Sandholm, 2014] with only 8 million information sets, which gives us a rough estimate of  $w_H(a)$  that is identical across many partitions of HUNL states.

Poker is played in an alternating fashion, where agents take turns playing in different positions. Let us say we have two agents,  $x$  and  $y$ . In poker, in odd-numbered games (starting at game 1) we would have  $x$  as player 1 and  $y$  as player 2, and in even-numbered games we would have  $y$  as player 1 and  $x$  as player 2. For the experiments, we model this as an extended game where there is an initial 50/50 chance event that assigns a position to the agent, along with a AIVAT correction term for the position.

All experiments will compare AIVAT value estimates with the unmodified game values from counting chips, the MIVAT value estimate, and the combination of MIVAT and imaginary observations using the strategy for agent  $x$  (MIVAT+IO $_x$ ). Because poker is a zero-sum game, it is sufficient to present results from the point of view of agent  $x$ .

### 8.6.1 Leduc Hold'em

The small size of Leduc hold'em lets us test both the case where  $P_a$  only contains one non-chance player, as well as the full-knowledge case where  $P_a = P$ . AIVAT and chip count results are generated from observations of 100,000 games. All of the numbers are in units of chips, where Leduc hold'em has a 1 chip ante, and 2 chip and 4 chip bets in the first and second rounds, respectively.

Figure 8.2 looks at self-play, where both  $x$  and  $y$  play the same Nash equilibrium that was used to generate  $u_h(a)$ . The true expected value for player  $x$  is 0. Because we are using value functions computed from their self-play, this experiment represents a best-case situation. With knowledge of both player's strategies, the only remaining variance comes from noise in the  $u_h(a)$  value function that arises from the sampling and averaging used in the MCCFR computation.

Estimator	$\bar{v}_x$	$SD(v_x)$
chips	0.01374	3.513
MIVAT	0.00448	2.327
MIVAT+IO <sub><i>x</i></sub>	0.00987	1.928
$P_a = \{p_c, x\}$	-0.00009	0.00643
$P_a = \{p_c, x, y\}$	-0.00001	0.00377

Figure 8.2: Value estimates for self-play in Leduc hold'em

With knowledge of both player's strategies, we reduce the per-game standard deviation of the estimated player value by a little less than 99.9%. This situation might be unlikely in practice, but does demonstrate that the AIVAT computation correctly shifts every observed outcome to the expected player value, given full correct information. Surprisingly, the one-sided evaluation where we use only one player's strategy still reduces the standard deviation by 99.8%. Using MIVAT or MIVAT+IO<sub>*x*</sub>, we only see a 33.8% and 45.1% reduction, respectively.

Moving away from the best-case situation, Figure 8.3 looks at games where  $x$  is the same Nash equilibrium from above, and  $y$  is an agent that randomly calls or raises. Given these strategies, the true expected value for player  $x$  is 0.69358.

Estimator	$\bar{v}_x$	$SD(v_x)$
chips	0.71673	5.761
MIVAT	0.68932	4.412
MIVAT+IO <sub><i>x</i></sub>	0.69968	4.295
$P_a = \{p_c, x\}$	0.69050	1.437
$P_a = \{p_c, x, y\}$	0.68698	1.782
$P_a = \{p_c, y\}$	0.69614	2.983

Figure 8.3: Value estimates for dissimilar strategies in Leduc hold'em

Using the call/raise strategy for  $y$  demonstrates that the amount of variance reduction does depend on how well the value functions estimates the true expected value of a situation. We used value functions which encode self-play values for  $x$ , and while  $y$  is sufficiently similar to  $x$  that the true values are still positively correlated with the estimated values for both players, they are no longer an almost-perfect match. Despite the strategic mismatch, using AIVAT we see a reduction in the standard deviation of 48% to 75% compared to the basic chip-count estimate. All of the AIVAT estimators outperform the 25% reduction using MIVAT plus imaginary observations.

## 8.6.2 No-limit Texas Hold'em

The game of HUNL better represents a potential real-world application. The game is commonly played, it is too large to easily compute exact expected values directly even when the strategy of both agents is known, average win rate is a statistic of interest to players and observers, and the high per-game variance of outcomes obscures the win rate even after hundreds of thousands of hands.

The variant of HUNL that we use has a small blind of 1 chip and big blind of 2 chips, and each player has 200 chips (*that is*, 100 big blinds.) Due to the large branching factor of chance events, we can only present results for AIVAT analysis using the strategy of one agent. All results are generated from observations of 1 million games.

We start by looking at self-play, using a low-quality Nash equilibrium approximation for both players  $x$  and  $y$ . The value functions  $u_h(a)$  are generated using this same weak approximation. Figure 8.4 gives the results for the different estimation methods. The true expected value for  $x$  is 0.

Estimator	$\bar{v}_x$	$SD(v_x)$
chips	0.03871	25.962
MIVAT	0.02038	21.293
MIVAT+IO $_x$	0.02596	16.073
$P_a = \{p_c, x\}$	0.00186	8.095

Figure 8.4: Value estimates for self-play in HUNL

In Figure 8.5 we look at games where  $x$  uses the same low-quality approximation of a Nash equilibrium, and  $y$  is a much stronger agent using a high-quality approximation of a Nash equilibrium. The value functions  $u_h(a)$  are still generated using the low-quality approximation. The true expected value for player  $x$  is not known.

Estimator	$\bar{v}_x$	$SD(v_x)$
chips	-0.10017	26.308
MIVAT	-0.11565	21.546
MIVAT+IO $_x$	-0.11297	16.051
$P_a = \{p_c, x\}$	-0.10971	8.301

Figure 8.5: Value estimates for dissimilar strategies in HUNL

In both experiments, we see a 39% reduction in the standard deviation when using MIVAT with imaginary observations, and a bit more than a 68% reduction using AIVAT. It must be noted that our value function could be improved, as the 18% reduction for MIVAT in this experiment does not match the 23% improvement previously demonstrated using values learned from data [White and Bowling, 2009]. The small abstract game used to generate the value functions does not do a good job of understanding the consequences of cards being dealt, as it can not distinguish most card situations. Despite this handicap, the full AIVAT estimator still significantly improves on the state of the art for low-variance value estimators for imperfect information games.

## 8.7 Conclusions

We introduce a technique for value estimation in imperfect information games that extends and combines existing techniques. AIVAT uses heuristic value functions, knowledge of

game structure, and knowledge about player strategies to both add a control variate term for chance and player decisions, and to average over multiple possible outcomes given a single observation. We prove AIVAT is unbiased, and demonstrate that with (almost) perfect value functions we see (almost) complete elimination of variance. Even with imprecise value functions, we show variance reduction in a real-world game that significantly exceeds existing techniques. AIVAT's three times reduction in standard deviation allows us to achieve the same statistical significance with ten times less data. A factor of ten is substantial: for problems with limited data, like human play against bots, ten times as many games could be the distinction between practical and impractical.

## **8.8 Author's contributions**

I contributed significantly to the conception of the variance reduction algorithm, its initial implementation and the evaluation of DeepStack.



# 9. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker

This chapter is based on [Moravčík et al., 2017]

## 9.1 Introduction

Games have long served as benchmarks and marked milestones of progress in artificial intelligence (AI). In the last two decades, computer programs have reached a performance that exceeds expert human players in many games, e.g., backgammon [Tesauro, 1995], checkers [Schaeffer et al., 1996], chess [Campbell et al., 2002], Jeopardy! [Ferrucci, 2012], Atari video games [Mnih et al., 2015], and go [Silver et al., 2016]. These successes all involve games with information symmetry, where all players have identical information about the current state of the game. This property of perfect information is also at the heart of the algorithms that enabled these successes, e.g., local search during play [Samuel, 1959, Kocsis and Szepesvári, 2006].

The founder of modern game theory and computing pioneer, von Neumann, envisioned reasoning in games without perfect information. “Real life is not like that. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory.” [Bronowski, 1973] One game that fascinated von Neumann was poker, where players are dealt private cards and take turns making bets or bluffing on holding the strongest hand, calling opponents’ bets, or folding and giving up on the hand and the bets already added to the pot. Poker is a game of imperfect information, where players’ private cards give them asymmetric information about the state of game.

Heads-up no-limit Texas hold’em (HUNL) is a two-player version of poker in which two cards are initially dealt face-down to each player, and additional cards are dealt face-up in three subsequent rounds. No limit is placed on the size of the bets although there is an overall limit to the total amount wagered in each game. AI techniques have previously shown success in the simpler game of heads-up limit Texas hold’em, where all bets are of a fixed size resulting in just under  $10^{14}$  decision points [Bowling et al., 2009, 2015]. By comparison, computers have exceeded expert human performance in go [Silver et al., 2016], a perfect information game with approximately  $10^{170}$  decision points [Allis, 1994]. The imperfect information game HUNL is comparable in size to go, with the number of decision points exceeding  $10^{160}$  [Johanson, 2013].

Imperfect information games require more complex reasoning than similarly sized perfect information games. The correct decision at a particular moment depends upon the probability distribution over private information that the opponent holds, which is revealed through their past actions. However, how our opponent’s actions reveal that information depends upon their knowledge of our private information and how our actions reveal it. This kind of recursive reasoning is why one cannot easily reason about game situations in isolation, which is at the heart of heuristic search methods for perfect information games. Competitive AI approaches in imperfect information games typically reason about the entire game and produce a complete strategy prior to play [Zinkevich et al., 2007, Gilpin et al., 2007]. Counterfactual regret minimization (CFR) [Zinkevich et al., 2007, Burch et al., 2014, Bowling et al., 2015] is one such technique that uses self-play to do recursive reasoning through adapting its strategy against itself over successive iterations. If the game is too

large to be solved directly, the common response is to solve a smaller, abstracted game. To play the original game, one translates situations and actions from the original game to the abstract game.

Although this approach makes it feasible for programs to reason in a game like HUNL, it does so by squeezing HUNL’s  $10^{160}$  situations down to the order of  $10^{14}$  abstract situations. Likely as a result of this loss of information, such programs are behind expert human play. In 2015, the computer program Claudico lost to a team of professional poker players by a margin of 91 mbb/g, which is a “huge margin of victory” [Wood, 2015]. Furthermore, it has been recently shown that abstraction-based programs from the Annual Computer Poker Competition have massive flaws [Lisý and Bowling, 2017a]. Four such programs (including top programs from the 2016 competition) were evaluated using a local best-response technique that produces an approximate lower-bound on how much a strategy can lose. All four abstraction-based programs are beatable by over 3,000 mbb/g, which is four times as large as simply folding each game.

DeepStack takes a fundamentally different approach. It continues to use the recursive reasoning of CFR to handle information asymmetry. However, it does not compute and store a complete strategy prior to play and so has no need for explicit abstraction. Instead it considers each particular situation as it arises during play, but not in isolation. It avoids reasoning about the entire remainder of the game by substituting the computation beyond a certain depth with a fast approximate estimate. This estimate can be thought of as DeepStack’s intuition: a gut feeling of the value of holding any possible private cards in any possible poker situation. Finally, DeepStack’s intuition, much like human intuition, needs to be trained. We train it with deep learning using examples generated from random poker situations. We show that DeepStack is theoretically sound, produces strategies substantially more difficult to exploit than abstraction-based techniques, and defeats professional poker players at HUNL with statistical significance.

## 9.2 DeepStack

DeepStack is a general-purpose algorithm for a large class of sequential imperfect information games. For clarity, we will describe its operation in the game of HUNL. The state of a poker game can be split into the players’ private information, hands of two cards dealt face down, and the public state, consisting of the cards laying face up on the table and the sequence of betting actions made by the players. Possible sequences of public states in the game form a public tree with every public state having an associated public subtree (Fig. 9.1).

A player’s strategy defines a probability distribution over valid actions for each decision point, where a decision point is the combination of the public state and the hand for the acting player. Given a player’s strategy, for any public state one can compute the player’s range, which is the probability distribution over the player’s possible hands given that the public state is reached.

Fixing both players’ strategies, the utility for a particular player at a terminal public state, where the game has ended, is a bilinear function of both players’ ranges using a payoff matrix determined by the rules of the game. The expected utility for a player at any other public state, including the initial state, is the expected utility over reachable terminal states given the players’ fixed strategies. A best-response strategy is one that maximizes a player’s expected utility against an opponent strategy. In two-player zero-sum games, like HUNL, a solution or Nash equilibrium strategy [Nash, 1950] maximizes the expected utility when playing against a best-response opponent strategy. The exploitability of a strategy is the

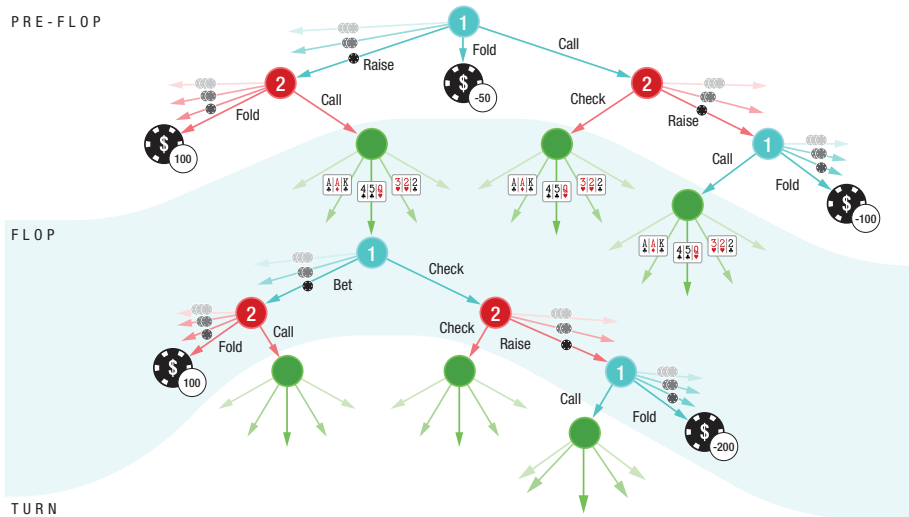


Figure 9.1: A portion of the public tree in HUNL. Nodes represent public states, whereas edges represent actions: red and turquoise showing player betting actions, and green representing public cards revealed by chance. The game ends at terminal nodes, shown as a chip with an associated value. For terminal nodes where no player folded, the player whose private cards form a stronger poker hand receives the value of the state.

difference in expected utility against its best-response opponent and the expected utility under a Nash equilibrium.

The DeepStack algorithm seeks to compute and play a low-exploitability strategy for the game, i.e., solve for an approximate Nash equilibrium. DeepStack computes this strategy during play only for the states of the public tree that actually arise. Although computed during play, DeepStack’s strategy is static, albeit stochastic, because it is the result of a deterministic computation that produces a probability distribution over the available actions.

The DeepStack algorithm (Fig. 9.2) is composed of three ingredients: a sound local strategy computation for the current public state, depth-limited lookahead using a learned value function to avoid reasoning to the end of the game, and a restricted set of lookahead actions. At a conceptual level these three ingredients describe heuristic search, which is responsible for many of AI’s successes in perfect information games. Until DeepStack, no theoretically sound application of heuristic search was known in imperfect information games. The heart of heuristic search methods is the idea of “continual re-searching”, where a sound local search procedure is invoked whenever the agent must act without retaining any memory of how or why it acted to reach the current state. At the heart of DeepStack is continual re-solving, a sound local strategy computation which only needs minimal memory of how and why it acted to reach the current public state.

### 9.2.1 Continual re-solving

Suppose we have taken actions according to a particular solution strategy but then in some public state forget this strategy. Can we reconstruct a solution strategy for the subtree without having to solve the entire game again? We can, through the process of re-solving [Burch et al., 2014]. We need to know both our range at the public state and a vector of expected values achieved by the opponent under the previous solution for each opponent hand. With these values, we can reconstruct a strategy for only the remainder of the game, which does not increase our overall exploitability. Each value in the opponent’s vector is a counterfactual

value, a conditional “what-if” value that gives the expected value if the opponent reaches the public state with a particular hand. The CFR algorithm also uses counterfactual values, and if we use CFR as our solver, it is easy to compute the vector of opponent counterfactual values at any public state.

Re-solving, however, begins with a strategy, whereas our goal is to avoid ever maintaining a strategy for the entire game. We get around this by doing continual re-solving: reconstructing a strategy by re-solving every time we need to act; never using the strategy beyond our next action. To be able to re-solve at any public state, we need only keep track of our own range and a suitable vector of opponent counterfactual values. These values must be an upper bound on the value the opponent can achieve with each hand in the current public state, while being no larger than the value the opponent could achieve had they deviated from reaching the public state. This is an important relaxation of the counterfactual values typically used in re-solving, with a proof of sufficiency included in our proof of Theorem 9.1 below.

At the start of the game, our range is uniform and the opponent counterfactual values are initialized to the value of being dealt each private hand. When it is our turn to act we re-solve the subtree at the current public state using the stored range and opponent values, and act according to the computed strategy, discarding the strategy before we act again. After each action, either by a player or chance dealing cards, we update our range and opponent counterfactual values according to the following rules: (i) Own action: replace the opponent counterfactual values with those computed in the re-solved strategy for our chosen action. Update our own range using the computed strategy and Bayes’ rule. (ii) Chance action: replace the opponent counterfactual values with those computed for this chance action from the last re-solve. Update our own range by zeroing hands in the range that are impossible given new public cards. (iii) Opponent action: no change to our range or the opponent values are required.

These updates ensure the opponent counterfactual values satisfy our sufficient conditions, and the whole procedure produces arbitrarily close approximations of a Nash equilibrium (see Theorem 9.1). Notice that continual re-solving never keeps track of the opponent’s range, instead only keeping track of their counterfactual values. Furthermore, it never requires knowledge of the opponent’s action to update these values, which is an important difference from traditional re-solving. Both will prove key to making this algorithm efficient and avoiding any need for the translation step required with action abstraction methods [Gilpin et al., 2008, Schnizlein et al., 2009].

Continual re-solving is theoretically sound, but by itself impractical. While it does not ever maintain a complete strategy, re-solving itself is intractable except near the end of the game. In order to make continual re-solving practical, we need to limit the depth and breadth of the re-solved subtree.

## 9.2.2 Limited depth lookahead via intuition

As in heuristic search for perfect information games, we would like to limit the depth of the subtree we have to reason about when re-solving. However, in imperfect information games we cannot simply replace a subtree with a heuristic or precomputed value. The counterfactual values at a public state are not fixed, but depend on how players play to reach the public state, i.e., the players’ ranges [Burch et al., 2014]. When using an iterative algorithm, such as CFR, to re-solve, these ranges change on each iteration of the solver.

DeepStack overcomes this challenge by replacing subtrees beyond a certain depth with a learned counterfactual value function that approximates the resulting values if that public state were to be solved with the current iteration’s ranges. The inputs to this function are the

ranges for both players, as well as the pot size and public cards, which are sufficient to specify the public state. The outputs are a vector for each player containing the counterfactual values of holding each hand in that situation. In other words, the input is itself a description of a poker game: the probability distribution of being dealt individual private hands, the stakes of the game, and any public cards revealed; the output is an estimate of how valuable holding certain cards would be in such a game. The value function is a sort of intuition, a fast estimate of the value of finding oneself in an arbitrary poker situation. With a depth limit of four actions, this approach reduces the size of the game for re-solving from  $10^{160}$  decision points at the start of the game down to no more than  $10^{17}$  decision points. DeepStack uses a deep neural network as its learned value function, which we describe later.

### 9.2.3 Sound reasoning

DeepStack’s depth-limited continual re-solving is sound. If DeepStack’s intuition is “good” and “enough” computation is used in each re-solving step, then DeepStack plays an arbitrarily close approximation to a Nash equilibrium.

**Theorem 9.1.** *If the values returned by the value function used when the depth limit is reached have error less than  $\epsilon$ , and  $T$  iterations of CFR are used to re-solve, then the resulting strategy’s exploitability is less than  $k_1\epsilon + k_2/\sqrt{T}$ , where  $k_1$  and  $k_2$  are game-specific constants. For the proof, see supplementary material.*

### 9.2.4 Sparse lookahead trees

The final ingredient in DeepStack is the reduction in the number of actions considered so as to construct a sparse lookahead tree. DeepStack builds the lookahead tree using only the actions fold (if valid), call, 2 or 3 bet actions, and all-in. This step voids the soundness property of Theorem 9.1, but it allows DeepStack to play at conventional human speeds. With sparse and depth-limited lookahead trees, the re-solved games have approximately  $10^7$  decision points, and are solved in under five seconds using a single NVIDIA GeForce GTX 1080 graphics card. We also use the sparse and depth-limited lookahead solver from the start of the game to compute the opponent counterfactual values used to initialize DeepStack’s continual re-solving.

### 9.2.5 Relationship to heuristic search in perfect information games

There are three key challenges that DeepStack overcomes to incorporate heuristic search ideas in imperfect information games. First, sound re-solving of public states cannot be done without knowledge of how and why the players acted to reach the public state. Instead, two additional vectors, the agent’s range and opponent counterfactual values, must be maintained to be used in re-solving. Second, re-solving is an iterative process that traverses the lookahead tree multiple times instead of just once. Each iteration requires querying the evaluation function again with different ranges for every public state beyond the depth limit. Third, the evaluation function needed when the depth limit is reached is conceptually more complicated than in the perfect information setting. Rather than returning a single value given a single state in the game, the counterfactual value function needs to return a vector of values given the public state and the players’ ranges. Because of this complexity, to learn such a value function we use deep learning, which has also been successful at learning complex evaluation functions in perfect information games [Silver et al., 2016].

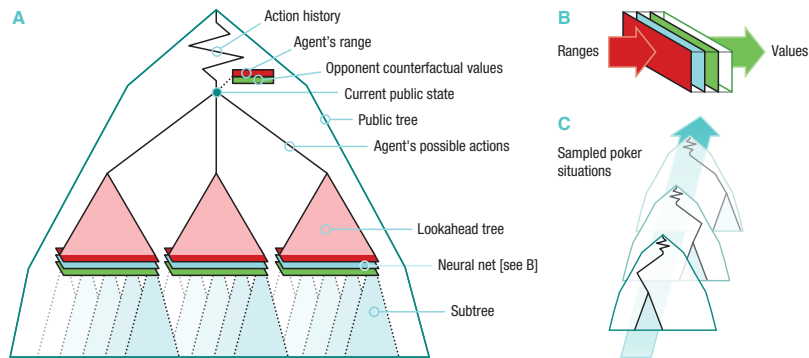


Figure 9.2: **DeepStack overview** (A) DeepStack reasons in the public tree always producing action probabilities for all cards it can hold in a public state. It maintains two vectors while it plays: its own range and its opponent’s counterfactual values. As the game proceeds, its own range is updated via Bayes’ rule using its computed action probabilities after it takes an action. Opponent counterfactual values are updated as discussed under “Continual re-solving”. To compute action probabilities when it must act, it performs a re-solve using its range and the opponent counterfactual values. To make the re-solve tractable it restricts the available actions of the players and lookahead is limited to the end of the round. During the re-solve, counterfactual values for public states beyond its lookahead are approximated using DeepStack’s learned evaluation function. (B) The evaluation function is represented with a neural network that takes the public state and ranges from the current iteration as input and outputs counterfactual values for both players (Fig. 9.3). (C) The neural network is trained prior to play by generating random poker situations (pot size, board cards, and ranges) and solving them to produce training examples. Complete pseudocode can be found in supplementary material.

### 9.2.6 Relationship to abstraction-based approaches

Although DeepStack uses ideas from abstraction, it is fundamentally different from abstraction-based approaches. DeepStack restricts the number of actions in its lookahead trees, much like action abstraction [Gilpin et al., 2008, Schnizlein et al., 2009]. However, each re-solve in DeepStack starts from the actual public state and so it always perfectly understands the current situation. The algorithm also never needs to use the opponent’s actual action to obtain correct ranges or opponent counterfactual values, thereby avoiding translation of opponent bets. We used hand clustering as inputs to our counterfactual value functions, much like explicit card abstraction approaches [Gilpin et al., 2007, Johanson et al., 2013]. However, our clustering is used to estimate counterfactual values at the end of a lookahead tree rather than limiting what information the player has about their cards when acting. We later show that these differences result in a strategy substantially more difficult to exploit.

## 9.3 Deep Counterfactual Value Networks

Deep neural networks have proven to be powerful models and are responsible for major advances in image and speech recognition [Krizhevsky et al., 2012, Hinton et al., 2012], automated generation of music [Oord et al., 2016], and game-playing [Mnih et al., 2015, Silver et al., 2016]. DeepStack uses deep neural networks with a tailor-made architecture, as the value function for its depth-limited lookahead (Fig. 9.3). Two separate networks are trained: one estimates the counterfactual values after the first three public cards are dealt (flop network), the other after dealing the fourth public card (turn network). An auxiliary

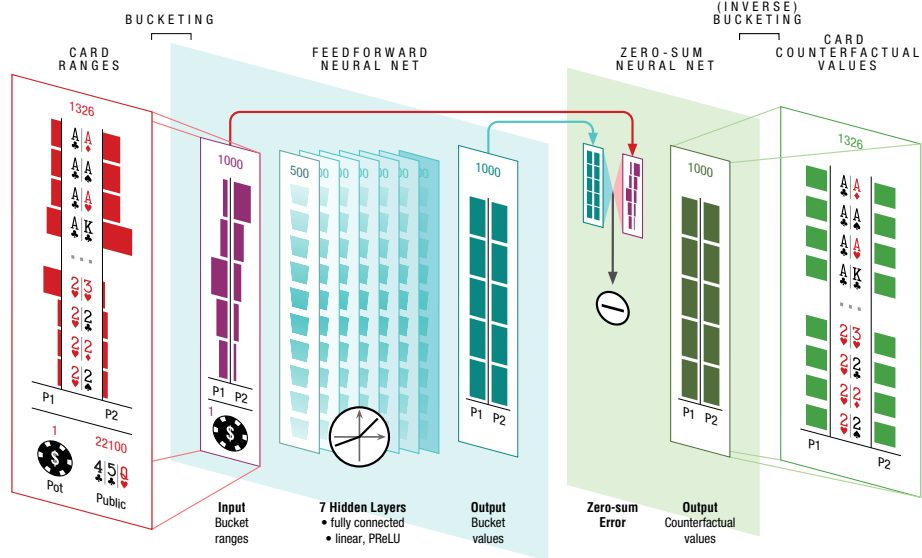


Figure 9.3: **Deep counterfactual value network** The inputs to the network are the pot size, public cards, and the player ranges, which are first processed into hand clusters. The output from the seven fully connected hidden layers is post-processed to guarantee the values satisfy the zero-sum constraint, and then mapped back into a vector of counterfactual values.

network for values before any public cards are dealt is used to speed up the re-solving for early actions.

### 9.3.1 Architecture

DeepStack uses a standard feedforward network with seven fully connected hidden layers each with 500 nodes and parametric rectified linear units [He et al., 2015] for the output. This architecture is embedded in an outer network that forces the counterfactual values to satisfy the zero-sum property. The outer computation takes the estimated counterfactual values, and computes a weighted sum using the two players' input ranges resulting in separate estimates of the game value. These two values should sum to zero, but may not. Half the actual sum is then subtracted from the two players' estimated counterfactual values. This entire computation is differentiable and can be trained with gradient descent. The network's inputs are the pot size as a fraction of the players' total stacks and an encoding of the players' ranges as a function of the public cards. The ranges are encoded by clustering hands into 1,000 buckets, as in traditional abstraction methods [Shi and Littman, 2001, Gilpin et al., 2007, Johanson et al., 2013], and input as a vector of probabilities over the buckets. The output of the network are vectors of counterfactual values for each player and hand, interpreted as fractions of the pot size.

### 9.3.2 Training

The turn network was trained by solving 10 million randomly generated poker turn games. These turn games used randomly generated ranges, public cards, and a random pot size. The target counterfactual values for each training game were generated by solving the game with players' actions restricted to fold, call, a pot-sized bet, and an all-in bet, but no card abstraction. The flop network was trained similarly with 1 million randomly generated flop games. However, the target counterfactual values were computed using our depth-limited solving procedure and our trained turn network. The networks were trained using the Adam

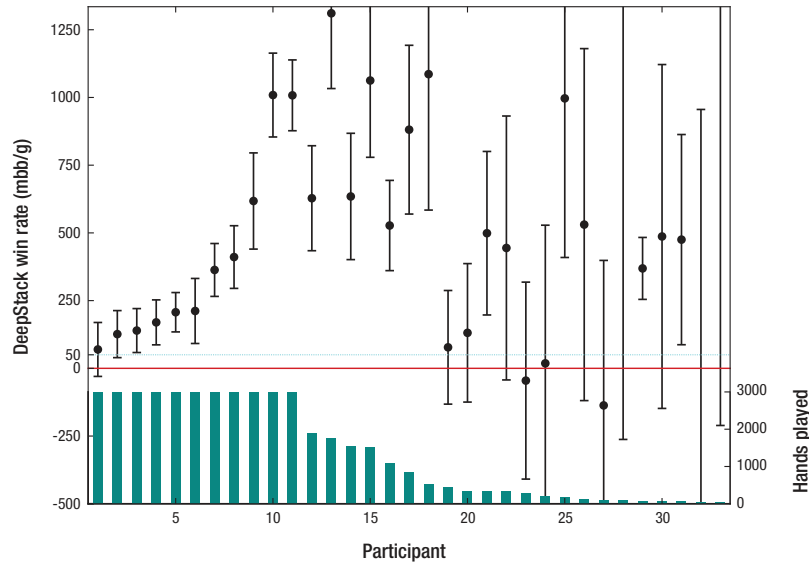


Figure 9.4: **Performance of professional poker players against DeepStack** Performance estimated with AIVAT along with a 95% confidence interval. The solid bars at the bottom show the number of games the participant completed.

gradient descent optimization procedure [Kingma and Ba, 2014] with a Huber loss [Huber, 1964].

## 9.4 Evaluating DeepStack

We evaluated DeepStack by playing it against a pool of professional poker players recruited by the International Federation of Poker. Thirty-three players from 17 countries were recruited. Each was asked to complete a 3,000 game match over a period of four weeks between November 7th and December 12th, 2016. Cash incentives were given to the top three performers (\$5,000, \$2,500, and \$1,250 CAD).

Evaluating performance in HUNL is challenging because of the large variance in per-game outcomes owing to randomly dealt cards and stochastic choices made by the players. The better player may lose in a short match simply because they were dealt weaker hands or their rare bluffs were made at inopportune times. As seen in the Claudico match [Wood, 2015], even 80,000 games may not be enough to statistically significantly separate players whose skill differs by a considerable margin. We evaluate performance using AIVAT [Burch et al., 2018], a provably unbiased low-variance technique for evaluating performance in imperfect information games based on carefully constructed control variates. AIVAT requires an estimated value of holding each hand in each public state, and then uses the expected value changes that occur due to chance actions and actions of players with known strategies (i.e., DeepStack) to compute the control variate. DeepStack’s own value function estimate is perfectly suited for AIVAT. Indeed, when used with AIVAT we get an unbiased performance estimate with an impressive 85% reduction in standard deviation. Thanks to this technique, we can show statistical significance in matches with as few as 3,000 games.

In total 44,852 games were played by the thirty-three players with 11 players completing the requested 3,000 games. Over all games played, DeepStack won 492 mbb/g. This is over 4 standard deviations away from zero, and so, highly significant. Note that professional poker players consider 50 mbb/g a sizable margin. Using AIVAT to evaluate performance, we see DeepStack was overall a bit lucky, with its estimated performance actually 486



Table 9.1: **Exploitability bounds from Local Best Response.** For all listed programs, the value reported is the largest estimated exploitability when applying LBR using a variety of different action sets. ‡: LBR was unable to identify a positive lower bound for DeepStack’s exploitability.

Program	LBR (mbb/g)
Hyperborean (2014)	4675
Slumbot (2016)	4020
Act1 (2016)	3302
Always Fold	750
<b>DeepStack</b>	<b>0 ‡</b>

mbb/g. However, as a lower variance estimate, this margin is over 20 standard deviations from zero.

The performance of individual participants measured with AIVAT is summarized in Figure 9.4. Amongst those players that completed the requested 3,000 games, DeepStack is estimated to be winning by 394 mbb/g, and individually beating 10 out of 11 such players by a statistically significant margin. Only for the best performing player, still estimated to be losing by 70 mbb/g, is the result not statistically significant. More details on the participants and their results are presented in supplementary material.

### 9.4.1 Exploitability

The main goal of DeepStack is to approximate Nash equilibrium play, i.e., minimize exploitability. While the exact exploitability of a HUNL poker strategy is intractable to compute, the recent local best-response technique (LBR) can provide a lower bound on a strategy’s exploitability [Lisý and Bowling, 2017a] given full access to its action probabilities. LBR uses the action probabilities to compute the strategy’s range at any public state. Using this range it chooses its response action from a fixed set using the assumption that no more bets will be placed for the remainder of the game. Thus it best-responds locally to the opponent’s actions, providing a lower bound on their overall exploitability. As already noted, abstraction-based programs from the Annual Computer Poker Competition are highly exploitable by LBR: four times more exploitable than folding every game (Table 9.1). However, even under a variety of settings, LBR fails to exploit DeepStack at all — itself losing by over 350 mbb/g to DeepStack. Either a more sophisticated lookahead is required to identify DeepStack’s weaknesses or it is substantially less exploitable.

## 9.5 Discussion

DeepStack defeated professional poker players at HUNL with statistical significance, a game that is similarly sized to go, but with the added complexity of imperfect information. It achieves this goal with little domain knowledge and no training from expert human games. The implications go beyond being a milestone for artificial intelligence. DeepStack represents a paradigm shift in approximating solutions to large, sequential imperfect information games. Abstraction and offline computation of complete strategies has been the dominant approach for almost 20 years [Shi and Littman, 2001, Billings et al., 2003b, Sandholm, 2010]. DeepStack allows computation to be focused on specific situations that arise when making decisions and the use of automatically trained value functions. These are two

of the core principles that have powered successes in perfect information games, albeit conceptually simpler to implement in those settings. As a result, the gap between the largest perfect and imperfect information games to have been mastered is mostly closed.

With many real world problems involving information asymmetry, DeepStack also has implications for seeing powerful AI applied more in settings that do not fit the perfect information assumption. The abstraction paradigm for handling imperfect information has shown promise in applications like defending strategic resources [Lisý et al., 2016] and robust decision making as needed for medical treatment recommendations [Chen and Bowling, 2012]. DeepStack’s continual re-solving paradigm will hopefully open up many more possibilities.

## **9.6 Author’s contributions**

I contributed significantly to formulation of the original idea, implementation of the algorithm, experiments, and the statistical evaluation of the matches against human professionals.

# 10. Player of Games

This chapter is based on [Schmid et al., 2021]

## 10.1 Introduction

In the 1950s, Arthur L. Samuel developed a Checkers-playing program that employed what is now called minimax search (with alpha-beta pruning) and “rote learning” to improve its evaluation function via self-play [Samuel, 1959]. This investigation inspired many others, and ultimately Samuel co-founded the field of artificial intelligence [Russell and Norvig, 2003] and popularized the term “machine learning”. A few years ago, the world witnessed a computer program defeat a long-standing professional at the game of Go [Silver et al., 2016]. AlphaGo also combined learning and search. Many similar achievements happened in between such as the race for super-human chess leading to DeepBlue [Hsu, 2006] and TD-Gammon teaching itself to play master-level performance in Backgammon through self-play [Tesauro, 1994], continuing the tradition of using games as canonical markers of mainstream progress across the field.

Throughout the stream of successes, there is an important common element: the focus on a single game. Indeed, DeepBlue could not play Go, and Samuel’s program could not play chess. Likewise, AlphaGo could not play chess; however its successor AlphaZero [Silver et al., 2018] could, and did. AlphaZero demonstrated that a single algorithm could master three different perfect information games using a simplification of AlphaGo’s approach, and with minimal human knowledge. Despite this success, AlphaZero could not play poker, and the extension to imperfect information games was unclear.

Meanwhile, approaches taken to achieve super-human poker AI were significantly different. Strong poker play has relied on *game-theoretic reasoning* to ensure that private information is concealed effectively. Initially, super-human poker agents were based primarily on computing approximate Nash equilibria offline [Johanson, 2016]. Search was then added and proved to be a crucial ingredient to achieve super-human success in no-limit variants [Moravčík et al., 2017, Brown and Sandholm, 2018, 2019]. Training for other large games have also been inspired by game-theoretic reasoning and search, such as Hanabi [Bard et al., 2020, Lerer et al., 2020], The Resistance [Serrino et al., 2019], Bridge [Lockhart et al., 2020], AlphaStar [Vinyals et al., 2019], and (no-press) Diplomacy [Anthony et al., 2020, Gray et al., 2020, Bakhtin et al., 2021]. Here again, however, despite remarkable success: each advance was still on a single game, with some clear uses of domain-specific knowledge and structure to reach strong performance.

In this chapter, we introduce Player of Games (POG), a new algorithm that generalizes the class of games in which strong performance can be achieved using self-play learning, search, and game-theoretic reasoning. POG uses growing-tree counterfactual regret minimization (GT-CFR): an anytime local search that builds subgames non-uniformly, expanding the tree toward the most relevant future states while iteratively refining values and policies. In addition, POG employs sound self-play: a learning procedure that trains value-and-policy networks using both game outcomes *and* recursive sub-searches applied to situations that arose in previous searches.

Player of Games is the first algorithm to achieve strong performance in challenge domains with both perfect *and* imperfect information — an important step towards truly general algorithms that can learn in arbitrary environments. Applications of traditional search suffer well-known problems in imperfect information games [Russell and Norvig,

2003]. Evaluation has remained focused on single domains (e.g. poker) despite recent progress toward sound search in imperfect information games [Moravčík et al., 2017, Brown and Sandholm, 2017, Šustr et al., 2020]. Player of Games fills this gap, using a single algorithm with minimal domain-specific knowledge. Its search is sound [Šustr et al., 2020] across these fundamentally different game types: it is guaranteed to find an approximate Nash equilibrium by re-solving subgames to remain consistent during online play, and yielding low exploitability in practice in small games where exploitability is computable. POG demonstrates strong performance across four different games: two perfect information (chess and Go) and two imperfect information (poker and Scotland Yard). Finally, unlike poker, Scotland Yard has significantly longer search horizons and game lengths, requiring long-term planning.

## 10.2 Background and Terminology

We start with necessary background and notation to describe the main algorithm and results. We relate our algorithm to other approaches in Section 10.5. Here, we give a concise introduction to necessary concepts, which are based on the Factored-Observation Stochastic Games (FOSG) formalism. For further details on the formalism, see [Kovařík et al., 2021, Schmid, 2021].

A game between two players starts in a specific **world state**  $w^{init}$  and proceeds to the successor world states  $w \in \mathcal{W}$  as a result of players choosing actions  $a \in A$  until the game is over when a terminal state is reached. At any world state  $w$ , we will use the notation  $A(w) \subseteq A$  to refer to those actions that are available, or legal, in world state  $w$ . Sequences of actions taken along the course of the game are called **histories** and denoted  $h \in H$ , with  $h' \sqsubseteq h$  denoting a prefix history (subsequence). At terminal histories,  $z \subset H$ , each player  $i$  receives a utility  $u_i(z)$ .

An **information state** (private state) is a state with respect to one player’s information. Specifically,  $s_i \in \mathcal{S}_i$  for player  $i$  is a set of histories that are indistinguishable due to missing information. A simple example is a specific decision point in poker where player  $i$  does not know the opponent’s private cards; the histories in the information state are different only

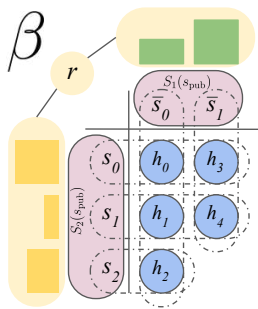


Figure 10.1: An example structure of public belief state  $\beta = (s_{\text{pub}}, r)$ .  $s_{\text{pub}}$  translates to two sets of information states, one for player 1,  $\mathcal{S}_1(s_{\text{pub}}) = \{\bar{s}_0, \bar{s}_1\}$ , and one for player 2,  $\mathcal{S}_2(s_{\text{pub}}) = \{s_0, s_1, s_2\}$ . Each information state includes different partitions of possible histories. Finally  $r$  contains reach probabilities for information states for both players.

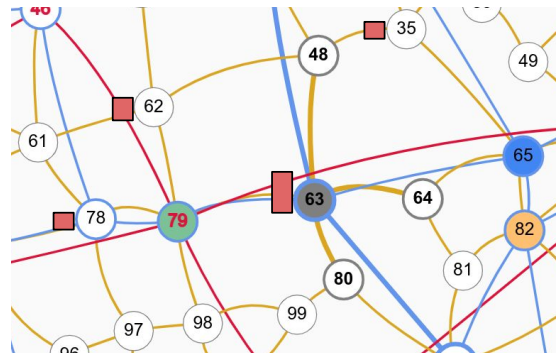


Figure 10.2: A depiction of a public belief state in Scotland Yard: circles are locations, edges are transportation links, and red bars indicate beliefs of the private (unrevealed) information state of Mr. X’s location.

in the chance event outcomes that determine the opponent’s private cards, since everything else is public knowledge. A player  $i$  plays a **policy**  $\pi_i : \mathcal{S}_i \rightarrow \Delta(A)$ , where  $\Delta(A)$  denotes the set of probability distributions over actions  $A$ . The goal of each player is to find a policy that maximizes their own expected utility.

Every time a player takes an action, each player gets a **private observation**

$\mathcal{O}_{\text{priv}(i)}(w, a, w')$  and a **public observation**  $\mathcal{O}_{\text{pub}}(w, a, w')$  as a result of applying action  $a$ , changing the game’s state from  $w$  to  $w'$ . A **public state**  $s_{\text{pub}} = s_{\text{pub}}(h) \in \mathcal{S}_{\text{pub}}$  is the sequence of public observations encountered along the history  $h$ . For example, a public state in Texas hold’em poker is represented by initial public information (stack sizes and antes), the betting history, and any publicly revealed board cards. Let  $\mathcal{S}_i(s_{\text{pub}})$  be the set of possible information states for player  $i$  given  $s_{\text{pub}}$ : each information state  $s_i \in \mathcal{S}_i(s_{\text{pub}})$  is consistent with public observations in  $s_{\text{pub}}$  but has different sequences of private observations. For example, in poker the information states would contain the private cards of player  $i$ . A full example of a FOSG is shown in Appendix F.1.

A **public belief state**  $\beta = (s_{\text{pub}}, r)$ , where the **range** (or beliefs)  $r \in \Delta(\mathcal{S}_1(s_{\text{pub}})) \times \Delta(\mathcal{S}_2(s_{\text{pub}}))$  is a pair of distributions over possible information states for both players representing the beliefs over information states in  $s_{\text{pub}}$ . A basic depiction of the various components of a public belief state is depicted in Figure 10.1. In the game of Scotland Yard, for instance, the information states correspond to the location of the evader (Mr. X). A specific example is shown in Figure 10.2. The true location of the evader is hidden but can be one of four different locations, and detectives have a stronger suspicion that the evader is on location 63 than 35, 62, or 78. In Scotland Yard, one of the distributions in  $r$  is a point mass because the detectives do not have any private information hidden from Mr. X.

Suppose players use joint policy  $\pi = (\pi_1, \pi_2)$ . Denote the expected utility to player to player  $i$  as  $u_i(\pi_1, \pi_2)$  and  $-i$  as the opponent of player  $i$ . Recall that **best response** policy is any policy  $\pi_i^b$  that achieves maximal utility against some  $\pi_{-i}$ :  $u_i(\pi_i^b, \pi_{-i}) = \max_{\pi'_i} u_i(\pi'_i, \pi_{-i})$ . A joint policy  $\pi$  is a **Nash equilibrium** if and only if  $\pi_1$  is a best response to  $\pi_2$  and  $\pi_2$  is a best response to  $\pi_1$ . There are also approximate equilibria:  $\pi$  is an  **$\epsilon$ -Nash equilibrium** if and only if  $u_i(\pi_i^b, \pi_{-i}) - u_i(\pi_i, \pi_{-i}) \leq \epsilon$  for all players  $i$ . In two-player zero-sum games, Nash equilibria are optimal because they maximize worst-case utility guarantees for both players. Also, equilibrium strategies are *interchangeable*: if  $\pi^A$  and  $\pi^B$  are Nash equilibria, then  $(\pi_1^A, \pi_2^B)$  and  $(\pi_1^B, \pi_2^A)$  are also both equilibria. Hence, the agent’s goal is to compute one such optimal (or approximately optimal) equilibrium strategy.

### 10.2.1 Tree Search and Machine Learning

The first major milestones in the field of AI were obtained by efficient search techniques inspired by the minimax theorem [Samuel, 1959, Hsu, 2006]. In a two-player zero-sum game with perfect information, the approach uses depth-limited search starting from the current world state  $w_t$ , along with a heuristic evaluation function to estimate value of the states beyond the depth limit,  $h(w_{t+d})$  and game-theoretic reasoning to back up values [Knuth and Moore, 1975]. Researchers developed significant search enhancements [Marsland and Campbell, 1981, Schaeffer and Plaat, 1996] which greatly improved performance, leading to IBM’s super-human DeepBlue chess program [Hsu, 2006].

This classical approach was, however, unable to achieve super-human performance in Go, which has significantly larger branching factor and state space complexity than chess. Prompted by the challenge of Go [Gelly et al., 2012], researchers proposed Monte Carlo tree search (MCTS) [Kocsis and Szepesvári, 2006, Coulom, 2007]. Unlike minimax search, MCTS builds trees via simulations, starting with an empty tree rooted by  $w_t$  and expanding the tree by adding the first state encountered in simulated trajectories not currently in the tree,

and finally estimating values from rollouts to the end of the game. MCTS led to significantly stronger play in Go and other games [Browne et al., 2012], attaining 6 dan amateur level in Go. However, heuristics in the form of domain knowledge were still necessary to achieve these milestones.

In AlphaGo [Silver et al., 2016], value functions and policies are incorporated, learned initially from human expert data, and then improved via self-play. A deep network approximates the value function and a prior policy helps guide the selection of actions during the tree search. The approach was the first to achieve super-human level play in Go [Silver et al., 2016]. AlphaGo Zero removed the initial training from human data and Go-specific features [Silver et al., 2017b]. AlphaZero reached state-of-the-art performance in Chess and Shogi as well as Go, using minimal domain knowledge [Silver et al., 2018].

POG, like AlphaZero, combines search and learning from self-play, using minimal domain knowledge. Unlike MCTS, however, which is not sound for imperfect information games, POG’s search algorithm is based on counterfactual regret minimization and is sound for both perfect and imperfect information games.

## 10.2.2 Game-Theoretic Reasoning and Counterfactual Regret Minimization

In imperfect information games, the choice of strategies that arise from hidden information can be crucial to determining each player’s expected rewards. Simply playing too predictably can be problematic: in the classic example game of Rock, Paper, Scissors, the only thing a player does not know is the choice of the opponent’s action, however this information fully determines their achievable reward. A player choosing to always play one action (e.g. rock) can be easily beaten by another playing the best response (e.g. paper). The Nash equilibrium plays each action with equal probability, which minimizes the benefit of any particular counter-strategy. Similarly, in poker, knowing the opponent’s cards or their strategy could yield significantly higher expected reward, and in Scotland Yard, players have a higher chance of catching the evader if their current location is known. In these examples, players can exploit any knowledge of hidden information to play the counter-strategy resulting in higher reward. Hence, to avoid being exploited, players must act in a way that does not reveal their own private information. We call this general behavior **game-theoretic reasoning** because it emerges as the result of computing (approximate) minimax-optimal strategies. Game-theoretic reasoning has been paramount to the success of competitive poker AI over the last 20 years.

One algorithm for computing approximate optimal strategies is counterfactual regret minimization (CFR) [Zinkevich et al., 2007]. CFR is a self-play algorithm that produces policy iterates  $\pi_i^t(s)$  for each player  $i$  at each of their information states  $s$  in a way that minimizes long-term average regret. As a result, the average policy over  $T$  iterations,  $\bar{\pi}^T$ , employed by CFR in self-play converges to an  $\epsilon$ -Nash equilibrium, at a rate of  $O(1/\sqrt{T})$ . At each iteration,  $t$ , counterfactual values  $v_i(s, a)$  are computed for each action  $a \in A(s)$  and immediate regrets for not playing  $a$ ,  $r(s, a) = v_i(s, a) - \sum_{a \in A(s)} \pi(s, a)v_i(s, a)$ , are computed and tabulated in a cumulative regret table storing  $R^T(s, a) = \sum_{t=1}^T r^t(s, a)$ . A new policy is computed using regret-matching [Hart and Mas-Colell, 2000]:  $\pi^{t+1}(s) = \frac{[R^t(s, a)]^+}{\sum_a [R^t(s, a)]^+}$ , where  $[x]^+ = \max(x, 0)$ , and reset to uniform if all the regrets are non-positive.

CFR<sup>+</sup> [Tammelin et al., 2015] is a successor of CFR that played a key role in fully solving the game of heads-up limit hold’em poker, the largest imperfect information game to be solved to date [Bowling et al., 2015]. A main component of CFR<sup>+</sup> is a different policy update mechanism, regret-matching<sup>+</sup>, which defines cumulative values slightly differently:

$Q^t(s, a) = (Q^{t-1}(s, a) + r^t(s, a))^+$ , and  $\pi^{t+1}(s, a) = Q^t(s, a) / \sum_b Q^t(s, b)$ .

A common form of CFR (or CFR<sup>+</sup>) is one that traverses the **public tree**, rather than the classical extensive-form game tree. Quantities required to compute counterfactual values, such as each player’s probabilities of reaching each information state under their policy (called their **range**) are maintained as beliefs. Finally, leaf nodes can be evaluated directly using the ranges, chance probabilities, and utilities (often more efficiently [Johanson et al., 2012]).

### 10.2.3 Imperfect Information Search, Decomposition, and Re-Solving

Solution concepts like Nash equilibria and minimax are defined over joint *policies*. The policy is fixed during play. Search could instead be described as a process, which might return different action distributions at subsequent visits to the same state. In search-based decision-making, new solutions are computed at the decision-time. Each state could depend on the past game-play, time limits, and samples of stochastic (chance) events, which introduces important subtleties such as solution compatibility across different searches [Šustr et al., 2020].

CFR has been traditionally used as a solver, computing entire policies via self-play. Each iteration traverses the entire game tree, recursively computing values for information states from other values at subgames deeper in the tree. Suppose one wanted a policy for a part of the game up to some depth  $d > 0$ . If there was an oracle to compute the values at depth  $d$ , then each iteration of CFR could be run to depth  $d$  and query the oracle to return the values. As a result, the policies would not be available at depths  $d' > d$ . Summarizing the policies below depth  $d$  by a set of values which can be used to reconstruct policies at depth  $d$  and beyond is the basis of **decomposition** in imperfect information games [Burch et al., 2014]. A **subgame** in an imperfect information game is a game rooted at a public state  $s_{\text{pub}}$ . In order for a subgame to be a proper game, it is paired with a belief distribution  $r$  over initial information states,  $s \in \mathcal{S}_i(s_{\text{pub}})$ . This is a strict generalization of subgames in perfect information games, where every public state has exactly one information state (which is, in fact, no longer private as a result) and a single belief with probability 1.

Subgame decomposition has been a crucial component of most recent developments of poker AI that scale to large games such as no-limit Texas hold’em [Moravčík et al., 2017, Brown and Sandholm, 2018, Brown et al., 2020]. Subgame decomposition enables local search to refine the policy during play analogously to the classical search algorithms in perfect information games and traditional Bellman-style bootstrapping to learn value functions [Moravčík et al., 2017, Serrino et al., 2019, Zarick et al., 2020, Brown et al., 2020]. Specifically, a **counterfactual value network** (CVN) represented by parameters  $\theta$  encodes the value function  $\mathbf{v}_\theta(\beta) = \{v_i(s_i)\}_{s_i \in \mathcal{S}_i(s_{\text{pub}}), i \in \{1, 2\}}$ , where  $\beta$  includes player’s beliefs over information states for the public information at  $s_{\text{pub}}$ . The function  $\mathbf{v}_\theta$  can then be used in place of the oracles mentioned above to summarize values of the subtrees below  $s_{\text{pub}}$ . An example of depth-limited CFR solver using decomposition is shown in Figure 10.3.

**Safe re-solving** is a technique that generates subgame policies from only summary information of a previous (approximate) solution: a player’s range and their opponent’s counterfactual values. This is done by constructing an auxiliary game with specific constraints. The subgame policies in the auxiliary game are generated in a way that preserves the exploitability guarantees of the original solve, so they can replace the original policies in the subgame. Thorough examples of the auxiliary game construction are found in [Burch et al., 2014] and [Brown and Sandholm, 2017, Section 4.1].

**Continual re-solving** is an analogue of classical game search, adapted to imperfect information games, that uses repeated applications of safe re-solving to play an episode of a

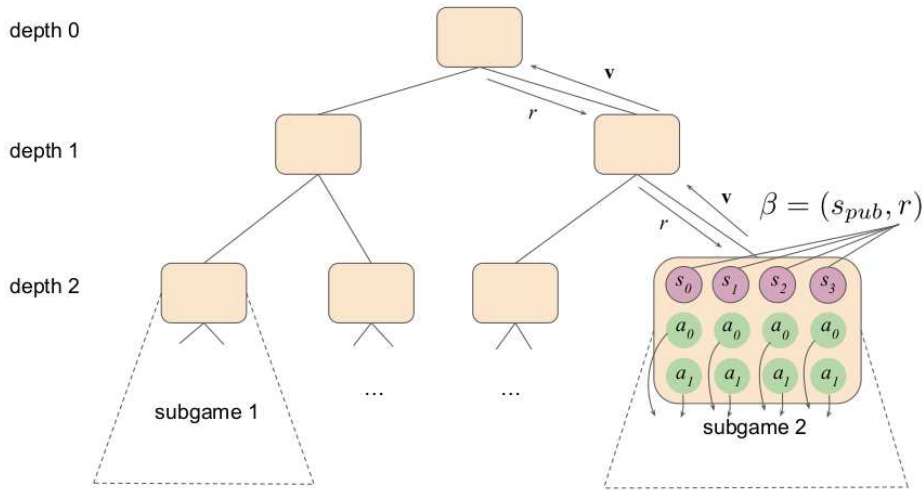


Figure 10.3: An example game with two specific subgames shown. Standard CFR would require traversing all the subgames. Depth-limited CFR decomposes the solve into running down to depth  $d = 2$  and using  $\mathbf{v} = \mathbf{v}_\theta(\beta)$  to represent the second subgame’s values. On the downward pass, ranges  $r$  are formed from policy reach probabilities. Values are passed back up to tabulate accumulating regrets. Re-solving a subgame would require construction of an auxiliary game [Burch et al., 2014] (not shown).

game [Moravčík et al., 2017]. It starts by solving a depth-limited game tree rooted at the beginning of the game, and search is a re-solving step. As the game progresses, for every subsequent decision at some information state  $s_i$ , continual re-solving will refine the current strategy by re-solving at  $s_i$ . Like other search methods, it is using additional computation to more thoroughly explore a specific situation encountered by the player. The continual re-solving method of [Moravčík et al., 2017] uses a few properties of poker which are not found in other games like Scotland Yard, so we use a more general re-solving method which can be applied to a broader class of games. We discuss the details of this more general re-solving variant in Appendix F.2.1.

## 10.3 Player of Games

We now describe our main algorithm. As POG has several components, we describe them each individually first, and then describe how they are all combined toward the end of the section.

For clarity, many of the details (including full pseudocode) are presented in Appendix F.2.

### 10.3.1 Counterfactual Value-and-Policy Networks

The first major component of POG is a **counterfactual value-and-policy network** (CVPN) with parameters  $\theta$ , depicted in Figure 10.4. These parameters represent a function  $f_\theta(\beta) = (\mathbf{v}, \mathbf{p})$ , where outputs  $\mathbf{v}$  are counterfactual values (one per information state per player), and *prior policies*  $\mathbf{p}$ , one per information state for the acting player, in the public state  $s_{\text{pub}}(h)$  at some history of play  $h$ .

In our experiments, we use standard feed-forward networks and residual networks. The details of the architecture are described in Section F.2.3.



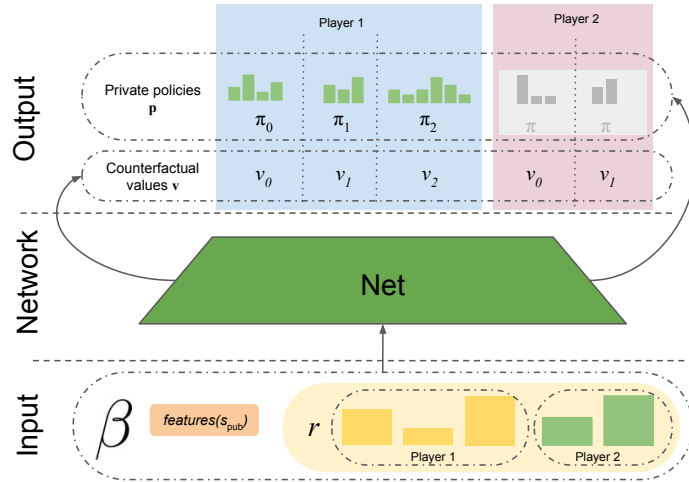


Figure 10.4: A counterfactual value-and-policy network (CVPN). Each query,  $\beta$ , to the network includes beliefs  $r$  and an encoding of  $s_{\text{pub}}$  to get the counterfactual values  $\mathbf{v}$  for both players and policies  $\mathbf{p}$  for the acting player in each information state  $s_i \in s_{\text{pub}}(h)$ , producing outputs  $f_{\theta}$ . Since players may have different actions spaces (as in *for example*, Scotland Yard) there are two sets of policy outputs: one for each player, and  $\mathbf{p}$  refers to the one for the acting player at  $s_{\text{pub}}$  only (depicted as player 1 in this diagram by greying out player 2’s policy output).

### 10.3.2 Search via Growing-Tree CFR

Growing-tree CFR (GT-CFR) is a new algorithm that runs a CFR variant on a public game tree that is incrementally grown over time. GT-CFR starts with an initial tree,  $\mathcal{L}^0$ , containing  $\beta$  and all of its child public states. Then each iteration,  $t$ , of GT-CFR consists of two phases:

1. The **regret update phase** (described in detail in subsection 10.3.2) runs several public tree CFR updates on the current tree  $\mathcal{L}^t$ .
2. The **expansion phase** (described in detail in subsection 10.3.2) expands  $\mathcal{L}^t$  by adding new public states via simulation-based expansion trajectories, producing a new larger tree  $\mathcal{L}^{t+1}$ .

When reporting the results we use the notation  $\text{POG}(s, c)$  for POG running GT-CFR with  $s$  total expansion simulations, and  $c$  expansion simulations per regret update phase, so the total number of GT-CFR iterations is then  $\lceil \frac{s}{c} \rceil$ . For example,  $\text{POG}(8000, 10)$  refers to 8000 expansion simulations at 10 expansions per regret update (800 GT-CFR iterations). The  $c$  can be fractional, so *for example*, 0.1 indicates a new node every 10 regret update phases. Figure 10.5 depicts the whole GT-CFR cycle. We chose this specific notation to directly compare total expansion simulations,  $s$ , to AlphaZero.

#### The Regret Update Phase of Growing-Tree CFR

The regret update phase runs  $\lceil \frac{1}{c} \rceil$  updates (iterations) of public tree CFR on  $\mathcal{L}^t$  using simultaneous updates, regret-matching<sup>+</sup>, and linearly-weighted policy averaging [Tammelin et al., 2015]. At public tree leaf nodes, a **query** is made to the CVPN at belief state  $\beta'$ , whose values  $f_{\theta}(\beta') = (\mathbf{v}, \mathbf{p})$  are used as estimates of counterfactual values for the public subgame rooted at  $\beta'$ .

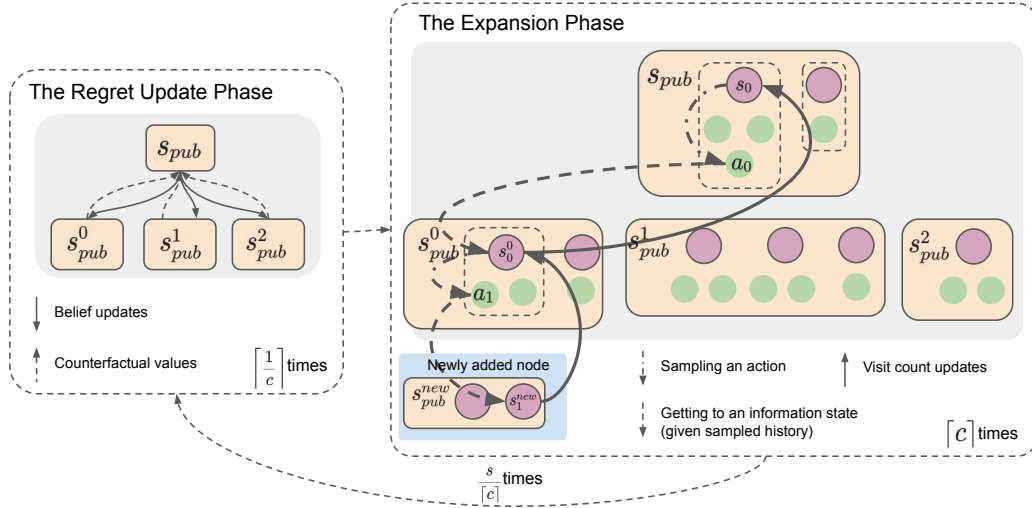


Figure 10.5: Overview of phases in one iteration of Growing-Tree CFR. The regret update phase propagates beliefs down the tree, obtains counterfactual values from the CPVN at leaf nodes (or from the environment at terminals), and passes back counterfactual values to apply the CFR update. The expansion phase simulates a trajectory from the root to leaf, adding public states to the tree. In this case the trajectory starts in the public belief state  $s_{pub}$  by sampling the information state  $s_0$ , after that the sampled action  $a_0$  leads to the information state  $s_0^0$  in public state  $s_{pub}^0$  finally the action  $a_1$  leads to a new public state that is added to the tree.

### The Expansion Phase of Growing-Tree CFR

In the expansion phase, new public tree nodes are added to  $\mathcal{L}$ . Search statistics, initially empty, are maintained over information states  $s_i$ , accumulated over all expansion phases within the same search. At the start of each simulation, a information state  $s_i$  is sampled from the beliefs in  $\beta_{root}$ . Then, a world state  $w_{root}$  is sampled from  $s_i$ , with associated history  $h_{root}$ . Actions are selected according to a mixed policy that takes into account learned values (via  $\pi_{PUCT}(s_i(h))$ ) as well as the currently active policy ( $\pi_{CFR}(s_i(h))$ ) from search:  $\pi_{select}(s_i(h)) = \frac{1}{2}\pi_{PUCT}(s_i(h)) + \frac{1}{2}\pi_{CFR}(s_i(h))$ . The first policy is determined by PUCT [Silver et al., 2016] using counterfactual values  $v_i(s_i, a)$  normalized by the sum of the opponent’s reach probability at  $s_i$  to resemble state-conditional action values, and the prior policy  $\mathbf{p}$  obtained from the queries. The second is simply CFR’s current policy at  $s_i(h)$ . As soon as the simulation encounters a information state  $s_i \in s_{pub}$  such that  $s_{pub} \notin \mathcal{L}$ , the simulation ends,  $s_{pub}$  is added to  $\mathcal{L}$ , and visit counts are updated along nodes visited during the trajectory. Similarly to AlphaZero [Silver et al., 2018], virtual losses [Segal, 2010] are added to the PUCT statistics when doing  $\lceil c \rceil$  simulations inside one GT-CFR iteration.

AlphaZero always expands a single action/node at the end of the iteration (the action with the highest UCB score). Optimal policies in perfect information games can be deterministic and thus expanding a single action/node is sufficient. In imperfect information games, optimal policies might be stochastic, having non-zero probability over multiple actions (the number of such actions is then referred to as the support size). There is a direct link between the level of uncertainty in the game (in terms of information states per public state) and the support size [Schmid et al., 2014]. In other words, the number of actions potentially required by an optimal policy is a function of the number of information states per public state, and we refer to this number as the minimum support size  $k$ . Rather than expanding a single action, POG thus expands the top  $k$  actions as ranked by the prior. All the public states corresponding to the expanded actions are then added to the tree (together with all the

actions leading to those public states). When POG expands a previously expanded node, only a single action is additionally expanded as the minimum support size requirement is already satisfied. Note that for perfect information games, the expansion acts the same as AlphaZero as the support size  $k = 1$  and thus a single action having the highest prior is added.

### Convergence Guarantees

Growing the tree in GT-CFR allows the search to selectively focus on parts of the space that are important for local decisions. Starting with a small tree and adding nodes over time does not have an additional cost in terms of convergence:

**Theorem 10.1.** *Let  $\mathcal{L}^t$  be the public tree at time  $t$ . Assume public states are never removed from the lookahead tree, so  $\mathcal{L}^t \subseteq \mathcal{L}^{t+1}$ . For any given tree  $\mathcal{L}$ , let  $\mathcal{N}(\mathcal{L})$  be the interior of the tree: all non-leaf, non-terminal public states where GT-CFR generates a policy. Let  $\mathcal{F}(\mathcal{L})$  be the frontier of  $\mathcal{L}$ , containing the non-terminal leaves where GT-CFR uses  $\epsilon$ -noisy estimates of counterfactual values. Let  $U$  be the maximum difference in counterfactual value between any two strategies, at any information state, and  $A$  be the maximum number of actions at any information state. Then, the regret at iteration  $T$  for player  $i$  is bounded:*

$$R_i^{T,\text{full}} \leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)| \epsilon + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}^T)} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{AT}$$

The regret  $R_i^{T,\text{full}}$  in Theorem 10.1 is the gap in performance between GT-CFR iterations and any possible strategy. Theorem 10.1 shows that the average policy returned by GT-CFR converges towards a Nash equilibrium at a rate of  $1/\sqrt{T}$ , but with some minimum exploitability due to  $\epsilon$ -error in the value function. There is also no additional cost when using GT-CFR as the game-solving algorithm for each re-solving search step in continual re-solving:

**Theorem 10.2.** *Assume we have played a game using continual re-solving, with one initial solve and  $D$  re-solving steps. Each solving or re-solving step finds an approximate Nash equilibrium through  $T$  iterations of GT-CFR using an  $\epsilon$ -noisy value function, public states are never removed from the lookahead tree, the maximum interior size  $\sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}^T)} |\mathcal{S}_i(s_{\text{pub}})|$  of all lookahead trees is bounded by  $N$ , the sum of frontier sizes across all lookahead trees is bounded by  $F$ , the maximum number of actions at any information sets is  $A$ , and the maximum difference in values between any two strategies is  $U$ . The exploitability of the final strategy is then bounded by  $(5D + 2) \left( F\epsilon + NU\sqrt{\frac{A}{T}} \right)$ .*

Theorem 10.2 is similar to Theorem 1 of [Moravčík et al., 2017], adapted to GT-CFR and using a more detailed error model which can more accurately describe value functions trained on approximate equilibrium strategies. It shows that continual re-solving with GT-CFR has the general properties we might desire: exploitability decreases with more computation time and decreasing value function error, and does not grow uncontrollably with game length. Proofs of the theorems are presented in Appendix F.5.

### 10.3.3 Data Generation via Sound Self-play

Player of Games generates episodes of data in self-play by running searches at each decision point. Each episode starts at the initial history  $h_0$  corresponding to the start of the game, and produces a sequence of histories  $(h_0, h_1, \dots)$ . At time  $t$ , the agent runs a local search

and then selects an action  $a_t$ , and the next history  $h_{t+1}$  is obtained from the environment by taking action  $a_t$  at  $h_t$ . Data for training the CVPN is collected via resulting trajectories *and* the individual searches.

When generating data for training the CVPN, it is important that searches performed at different public states be consistent with both the CVPN represented by  $\theta$  and with searches made at previous public states along the same trajectory (*for example*, two searches should not be computing parts of two different optimal policies). This is a critical requirement for sound search [Burch et al., 2014, Moravčík et al., 2017, Šustr et al., 2020], and we refer to the process of a sound search algorithm generating data in self-play as **sound self-play**.

To achieve sound self-play, searches performed during data generation run GT-CFR on the standard safe resolving auxiliary game (as described in Section 10.2.3). The auxiliary game includes an option at the initial decision for the opponent to decide to enter the subgame, or take the alternative values returned by  $f_\theta(\beta)$ . For a detailed construction of this resolving process, see Section F.2.1.

### 10.3.4 Training Process

The quality of the policies produced by GT-CFR and data generated by sound self-play depends critically on the values returned by the CVPN. Hence, it is important for the estimates to be accurate in order to produce high-performance searches and generate high-quality data. In this subsection, we describe the procedure we use to train the CVPN. The process is summarized in Figure 10.6.

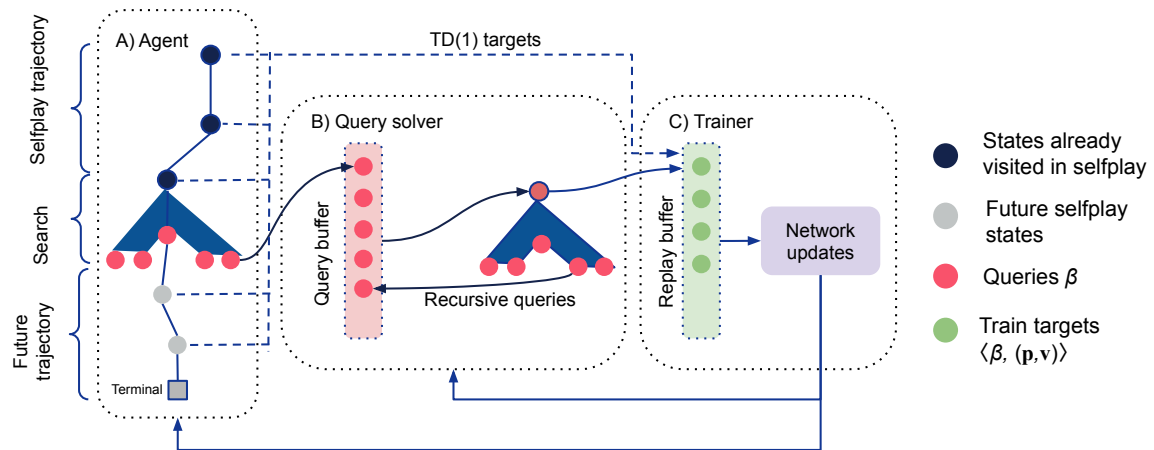


Figure 10.6: POG Training Process. Actors collect data via sound self-play and trainers run separately over a distributed network. (A) Each search produces a number of CVPN **queries** with input  $\beta$ . (B) Queries are added to a query buffer and subsequently solved by a **solver** that studies the situation more closely via another invocation of GT-CFR. During solving, new recursive queries might be added back to the query buffer; separately the network is (C) trained on minibatches sampled from the replay buffer to predict values and policy targets computed by the solver.

#### Query Collection

As described in sections 10.3.2 and 10.3.3, episodes are generated by each player running searches of GT-CFR from the current public state. Each search produces a number of network queries from public tree leaf nodes  $\beta$  (depicted as pink nodes in Figure 10.6).

The training process improves the CVPN via supervised learning. Values are trained using Huber loss [Huber, 1964] based on value targets and the policy loss is cross entropy with respect to a target policy. Value and policy targets are added to a sliding window data set of training data that is used to train the CVPN concurrently. The CVPN is updated asynchronously on the actors during training.

### Computing Training Targets

Policy targets are assembled from the searches started at public states along the main line of episodes (the histories reached in self-play) generated by sound self-play described in 10.3.3. Specifically, they are the output policies for all information states within the root public state, computed in the regret update phase of GT-CFR.

Value targets are obtained in two different ways. Firstly, the outcome of the game is used as a (TD(1)) value target for states along the main line of episodes generated by sound self-play. Secondly, value targets are also obtained by bootstrapping: running an instance of GT-CFR from subgames rooted at input queries. In principle, any solver could be used because any subgame rooted at  $\beta$  has well-defined values. Thus, this step acts much like a policy improvement operator via decomposition described in Section 10.2.3. Specifically, the value targets are the final counterfactual values after  $T$  iterations of GT-CFR for all the information states within the public state that initiated the search. The specific way that the different value targets are assigned is described by the pseudocode in Section F.2.5 and determined by a hyperparameter described in Section F.2.4.

### Recursive Queries

While the solver is computing targets for a query, it is also generating more queries itself by running GT-CFR. Some of these **recursive queries** are also added to the buffer for future solving. As a result, at any given time the buffer may include queries generated by search in the main self-play game or by solver-generated queries off the main line. To ensure that the buffer is not dominated by recursive queries, we set the probability of adding a new recursive query to less than 1 (in our experiments, the value is typically 0.1 or 0.2; see Section F.2.4 for the exact values).

### Consistency of Training Process

One natural question is whether, or under what circumstances, the training process could ensure convergence to the optimal values? The answer is positive: the training process converges to the optimal values, asymptotically, as  $T \rightarrow \infty$  and with very large (exponential) memory.

Informally, imagine an oracle function  $f(\beta)$  that can simply memorize the values and policy for the particular  $\beta$  similar to a tabular value or policy iteration algorithm except with continuous keys. For any subgame rooted at some  $\beta$  with a depth of 1 (every action leads to terminal states), the values and policies can be computed and stored for  $\beta$  after  $T$  iterations of the solver. This can then be applied inductively: since CFR is deterministic, for any subgame on the first iteration of GT-CFR, a finite number of queries will be generated. Each of these queries will be solved using GT-CFR. Eventually, the query will be a specific one that is one step from the terminal state whose values can be computed exactly and stored in  $f(\beta)$ . As this value was generated in self-play or by a query solver, and CFR is deterministic, it will produce another self-play game with the identical query, except it will load the solved value from  $f(\beta)$ , and inductively the values will get propagated from the bottom up. Since

CFR is deterministic and  $T$  is finite, these ensure that the memory requirement is not infinite despite the continuous-valued keys.

Practically, the success of the training process will depend on the representational capacity and training efficacy of the function approximation (*that is*, neural network architecture).

### 10.3.5 Bringing it all Together: Full Algorithm Overview

The POG algorithm learns via sound self-play: each player, when faced with a decision to make, employs a sound growing-tree CFR search equipped with a counterfactual value-and-policy network to generate a policy, which is then used to sample an action to take. This process generates two types of training data: search queries, which are then solved separately (sometimes recursively generating new queries), and full-game trajectories. The training process uses the data in different ways: outcomes of the games and solved queries to train the value head of the network, and policies output from search along the main line to train the policy head of the network. In practice, the self-play data generation and training happen in parallel: actors generate the self-play data (and solve queries) while trainers learn new networks and periodically update the actors.

For a fully-detailed description of the algorithm, including hyperparameter values and specific descriptions of each process described above, see Appendix F.2.

## 10.4 Evaluation

We evaluate POG on four games: chess, Go, heads-up no-limit Texas hold'em poker, and Scotland Yard. We also evaluate POG on a smaller benchmark poker game Leduc hold'em and a custom Scotland Yard map, where the approximation quality compared to the optimal policy can be computed exactly.

Chess and Go are well-known classic games, both seen as grand challenges of AI [Hsu, 2006, Gelly et al., 2012] which have driven progress in artificial intelligence since its inception. The achievement of DeepBlue beating Kasparov in 1997 is widely regarded to be the first big milestone of AI. Today, computer-playing programs remain consistently super-human, and one of the strongest and most widely-used programs is Stockfish [The Stockfish Development Team, 2021]. Go emerged as the favorite new challenge domain, which was particularly difficult for classical search techniques [Gelly et al., 2012]. Monte Carlo tree search [Coulom, 2007, Kocsis and Szepesvári, 2006, Browne et al., 2012] emerged as the dominant search technique in Go. The best of these programs, Crazy Stone and Zen, were able to reach the level of 6 dan amateur [Silver et al., 2016]. It was not until 2016 that AlphaGo defeated the first human professional Lee Sedol in the historical 2016 match, and also defeated the top human Ke Jie in 2017.

Heads-up no-limit Texas hold'em is the most common two-player version of poker played by humans, also played by DeepStack and Libratus [Moravčík et al., 2017, Brown and Sandholm, 2018]. Human-level poker has been the standard challenge domain among imperfect information games, inspiring the field of game theory itself. no-limit Texas hold'em adds the complexity of stochastic events (card draws), imperfect information (private cards), and a very large state space [Johanson, 2013]. We use blinds of 100 and 50 chips, and stack sizes of 200 big blinds (20,000 chips).

Scotland Yard is a compelling board game of imperfect information, receiving Spiel des Jahres award in 1983 as well as being the “The most popular game '83” by SpielBox [spiel-des-jahres, 2020]. The game is played on a map of London, where locations are connected by edges representing different modes of transportation. One player plays as “Mr. X” (the

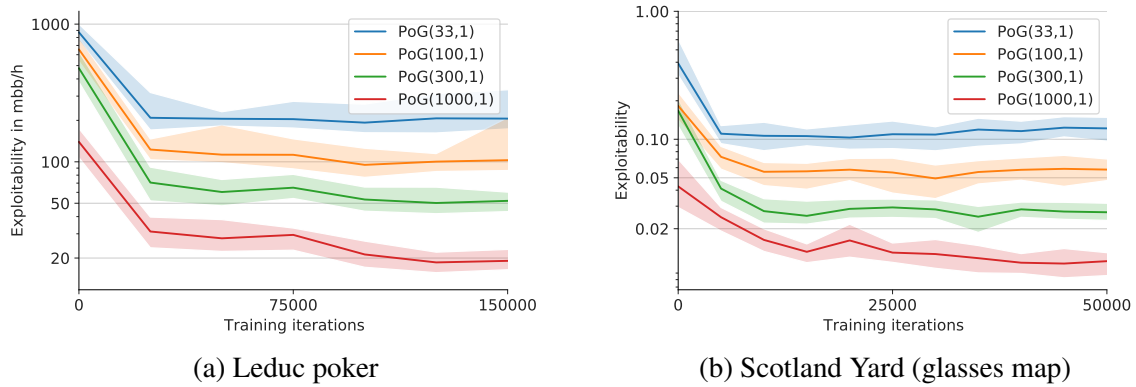


Figure 10.7: Exploitability of POG as a function of the number of training steps under different number of simulations of GT-CFR in (a) Leduc poker, and (b) Scotland Yard (glasses map). All networks were trained using a single training run of POG(100, 1), and the x-values correspond to a network trained for the corresponding number of steps. Each line corresponds to a different evaluation condition, *for example*, POG( $s, c$ ) used at evaluation time. The ribbon shows minimum and maximum exploitability out of 50 seeded runs for each setup. The units of the y-axis in Leduc poker are milli big blinds per hand (where a milli big blind is corresponds to one thousandth of a chip in Leduc), in Scotland Yard the reward is either -1 (loss) or +1 (win).

evader) and other controls detectives (pursuers). Mr. X is not visible for most of the game, but detectives get to see the mode of transportation Mr. X uses (e.g. taxi, bus, subway). In order to win, detectives need to catch Mr. X within 24 rounds. Scotland Yard is a perfect example of a game that combines search and imperfect information: agents have to search future position while correctly reasoning about the likelihood of Mr. X’s current location. Also, Scotland Yard has partially-observable *actions*, so private information is coupled with the effects of the choices in a way that is not present in Texas hold’em poker. More detailed explanation of Scotland Yard is in Appendix F.4.

This suite of games covers the classic challenge domains across game types (perfect information and imperfect information, some with stochastic elements and others not), as well as a new challenging imperfect information game with significantly longer sequences of actions and a fundamentally different type of uncertainty over hidden actions.

When reporting the results we use POG( $s, c$ ) notation from Section 10.3.2. As a reminder,  $s$  is the total number of expansion trajectories,  $c$  is the number of expansion trajectories per regret update phase, and  $\lceil \frac{s}{c} \rceil$  is the total number of GT-CFR iterations.

#### 10.4.1 Exploitability in Leduc Poker and Small Scotland Yard Map

We evaluate POG in Leduc poker [Southey et al., 2005], a commonly used benchmark poker game, and Scotland Yard on a custom map. The full description of Leduc poker and map are presented in Appendix F.3.

**Exploitability** is a standard metric to represent empirical convergence rates: it represents the average amount a player can gain by deviating to a best response, which is zero at equilibrium. For a given joint policy in a two-player zero-sum game  $\pi = (\pi_1, \pi_2)$ ,  $\text{EXPLOITABILITY}(\pi) = (\max_{\pi'_1} u_1(\pi'_1, \pi_2) + \max_{\pi'_2} u_2(\pi_1, \pi'_2))/2$ , where  $u_i(\pi)$  is the expected utility to player  $i$  under joint policy  $\pi$ . Exploitability is a function of a specific (fixed) joint policy. However, for a search algorithm like POG, previous searches may affect policies computed at later points within the same game, as explained in Section 10.2.3. Hence, we

construct multiple samples of the POG policy by choosing a random seed, running the search algorithm at every public state in a breadth-first manner such that every search is conditioned on previous searches at predecessor states, and composing together the policies obtained from each search. We then show the minimum, average, and maximum exploitabilities over policies constructed in this way from 50 different choices of seeds. If the minimum and maximum exploitability values are tight, then they represent an accurate estimate of overall exploitability.

Figure 10.7 shows the convergence rates of POG in Leduc poker and the glasses map of Scotland Yard, as a function of the number of CVPN training steps. For these graphs, we evaluate multiple networks (each trained for a different number of steps) generated by a single training run of POG(100, 1). Each data point corresponds to a specific network (determined by number of steps trained) being evaluated under different settings during play. For each specific  $x$ -value, a single network was used to obtain each exploitability value of POG using the network under different evaluation conditions. We observe that exploitability drops down fairly quickly as the training steps increase. Also, even using only 1 CFR update per simulation, there is significant difference in exploitability when more simulations are used.

As Theorem 10.1 suggests, more training (by reducing  $\epsilon$ ) and more search (by increasing  $T$ ) reduces the exploitability of POG. Standard RL algorithms in self-play are not guaranteed to reduce exploitability with continued training in this setting. We show this lack of convergence in practice in Section F.3.4.

## 10.4.2 Results in Challenge Domains

We now present our main results: the performance of POG in chess, Go, heads-Up no-limit Texas hold'em, and Scotland Yard.

We trained a version of AlphaZero using its original settings in Chess and Go using 3500 concurrent actors using one TPUv4 each, for a total of 800k training steps. POG was trained using a similar amount of concurrent resources.

In chess, we evaluated POG against Stockfish 8, level 20 [The Stockfish Development Team, 2021] and AlphaZero.

POG(800, 1) was run in training for 3M training steps. During evaluation, Stockfish uses various search controls: number of threads, and time per search. We evaluate AlphaZero and POG up to 60000 simulations. A tournament between all of the agents was played at 200 games per pair of agents (100 games as white, 100 games as black). Table 10.1a shows the relative Elo comparison obtained by this tournament, where a baseline of 0 is chosen for Stockfish(threads=1, time=0.1s).

In Go, we evaluate POG(60000, 10) using a similar tournament as in Chess, against two previous Go programs: GnuGo (at its highest level, 10) [Team, 2009] and Pachi v7.0.0 [Baudis et al., 2016] with 10k and 100k simulations, as well as AlphaZero [Silver et al., 2018]. The POG network was trained for 1M training steps. Table 10.1b shows the relative Elo comparison for a subset of the agents that played in this tournament, where a baseline of 0 is chosen for GnuGo. The full results are presented in Appendix F.3.

For chess and Go, we also present direct Elo comparisons when increasing the number of neural network evaluations in Figure 10.8. Note that while the neural networks evaluations account for the majority of the run time, the complexity of the regret update phase is linear in the size of the tree. The run time is thus quadratic in the number of GT-CFR iterations. The absolute time cost could be reduced by an implementation that runs the regret update and expansion phase in parallel. For a more detailed analysis of POG's complexity, see Section F.2.2.



Agent	Rel. Elo	Agent	Rel. Elo
AlphaZero(sims=60k)	+592	AlphaZero(s=16k, t=800k)	+3139
Stockfish(threads=16, time=4s)	+530	AlphaZero(s=8k, t=800k)	+2875
AlphaZero(sims=8k)	+455	<b>PoG(s=16k, c=10)</b>	<b>+1970</b>
<b>PoG(s=60k, c=10)</b>	<b>+420</b>	<b>PoG(s=8k, c=10)</b>	<b>+1902</b>
Stockfish(threads=4, time=1s)	+382	Pachi(s=100k)	+869
<b>PoG(s=8k, c=10)</b>	<b>+268</b>	Pachi(s=10k)	+231
Stockfish(threads=1, time=0.1s)	0	GnuGo(l=10)	0

(a) Chess results. Elo of Stockfish with a single thread and 100ms thinking time was set to be 0. The other values are relative to that.

(b) Go results. Elo of GnuGo was set to be 0. AlphaZero(s=16k, t=800k) refers to 16000 search simulations. For full results, see Appendix F.3.

Table 10.1: A table with relative Elo of different agents. Each agent played 200 matches (100 as white and 100 as black) against every other agent in the tournament.

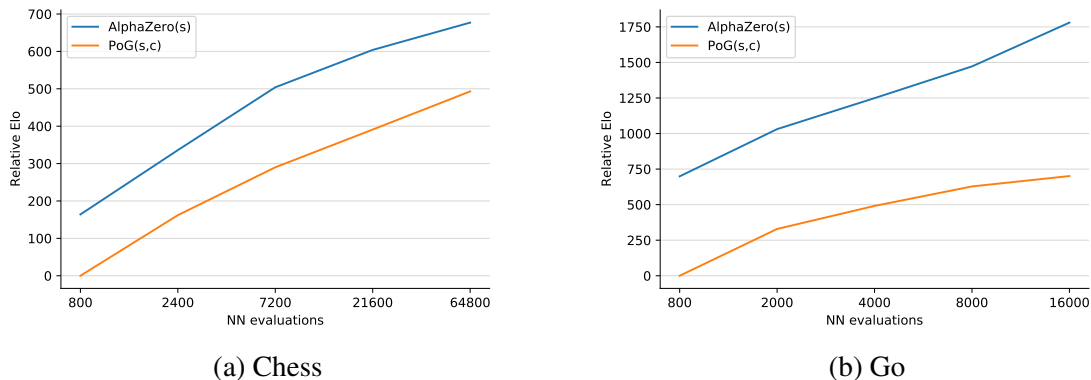


Figure 10.8: Scalability of PoG with increasing number of neural network evaluations compared to AlphaZero measured on relative Elo scale. The x-axis corresponds to the number of simulations in AlphaZero and  $s$  in  $\text{PoG}(s, c)$ .

We notice in both chess and Go that PoG reaches strong performance. In Chess,  $\text{PoG}(60000, 10)$  is stronger than Stockfish using 4 threads and one second of search time. In Go,  $\text{PoG}(16000, 10)$  is more than 1100 Elo stronger than Pachi with 100,000 simulations. Also,  $\text{PoG}(16000, 10)$  wins 0.5% (2/400) of its games against the strongest AlphaZero(s=8000, t=800k). As a result, PoG appears to be performing at the level of top human amateur, possibly even professional level. In both cases, PoG is weaker than AlphaZero, with the gap being smaller in Chess. We hypothesize that this difference is the result of MCTS being more efficient than CFR on perfect information games, as the price of PoG’s generality.

In heads-up no-limit Texas hold’em, we evaluate PoG against Slumbot2019 [Jackson, 2013, unspecified], the best openly-available computer poker player. When training poker, PoG uses randomized betting abstractions described in Section F.2.7 to reduce the number of actions from 20,000 to 4 or 5.  $\text{PoG}(10, 0.1)$  is trained for up to 1.1M training steps and then evaluated. Since poker has particularly high variance, we use AIVAT [Burch et al., 2018] to compute a more accurate estimate of performance. Head-to-head results are shown in Table 10.9.  $\text{PoG}(10, 0.01)$  wins on average  $7 \pm 3$  milli big blinds (0.7 chips) per hand, with 95% confidence intervals (3.1M matches). We also evaluate PoG against a local best-response (LBR) player that can use only fold and call actions with a poker-specific

Agent Name	Slumbot	LBR [Lisý and Bowling, 2017b]
Slumbot (2016)	-	
ARMAC [Gruslys et al., 2020]	-	
DeepStack [Moravčík et al., 2017]	-	
Modicum [Brown et al., 2018b]	$11 \pm 5$	
ReBeL [Brown et al., 2020]	$45 \pm 5$	
Supremus [Zarick et al., 2020]	$176 \pm 44$	
PoG(10, 0.01)	$7 \pm 3$	

Figure 10.9: Head-to-head results showing expected winnings of PoG (mbb/h) against Slumbot and LBR together with results of other recently published agents. The LBR agent use either fold or call (FC) actions in the all four rounds. The  $\pm$  shows one standard deviation. LBR results for Slumbot are from [Lisý and Bowling, 2017b]. The other results are from the papers describing the agents.

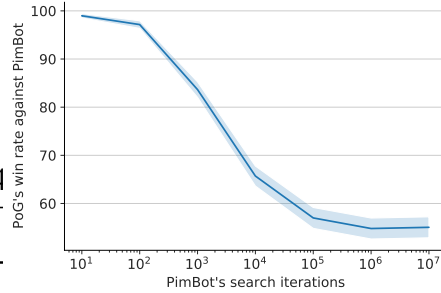


Figure 10.10: Win rate of PoG(400, 1) against PimBot with varying simulations. 2000 matches were played for each data point, with roles swapped for half of the matches. Note that the x-axis has logarithmic scale. The ribbon shows 95% confidence interval.

heuristic which has shown to find exploits in previous poker agents [Lisý and Bowling, 2017b]. LBR fails to find an exploit of PoG’s strategy, and PoG wins on average by  $434 \pm 9$  milli big blinds per hand. Table 10.9 summarizes the results of PoG along with other recent poker agents.

In Scotland Yard, the current state-of-the-art agent in this game is based on MCTS with game-specific heuristic enhancements [Nijssen and Winands, 2012]. We call this agent “PimBot” based on its main author, Joseph Antonius Maria (“Pim”) Nijssen. PimBot implements a variant of MCTS that uses determinization and heuristic evaluations and playout policies [Nijssen and Winands, 2012, Nijssen, 2013]. PimBot won 34 out of 50 manually played games against the Nintendo DS Scotland Yard AI.

In our experiment PoG is trained up to 17M steps. In evaluation we play a head-to-head match with PoG(400, 1) against PimBot at different number of simulations per search. The results are shown in Figure 10.10. These results show that PoG is winning significantly even against PimBot with 10M search simulations (55% win rate), compared to PoG searching a tiny fraction of the game. Interestingly PimBot doesn’t seem to play stronger with more search at this point, as both the 1M and 10M iteration versions have the same performance against PoG.

As in chess and Go, PoG also demonstrates strong performance in these complex imperfect information games. In the case of poker, in addition to beating Slumbot it also beats the local best-response agent which was not possible for some previous agents (including Slumbot). Finally, PoG significantly beats the state-of-the-art agent in Scotland Yard, an imperfect information game with longer episodes and fundamentally different kind of imperfect information than in poker. Together, these results indicate that PoG is capable of strong performance across four games, two fundamentally different game types, and can act as a truly unified algorithm combining search, learning, and game-theoretic reasoning for competitive games.

## 10.5 Related Work

Player of Games builds upon several components that have been developed in previous work. In this section, we describe the most relevant of these past works and how they relate to POG.

POG combines many elements that were originally proposed in AlphaZero and its predecessors, as well as DeepStack [Silver et al., 2016, 2017b, 2018, Moravčík et al., 2017]. Specifically, POG uses the combined search and learning using deep neural networks from AlphaGo and DeepStack, along with game-theoretic reasoning and search in imperfect information games from DeepStack. The use of public belief states and decomposition in imperfect information games has been a critical component of success in no-limit Texas Hold'em poker [Burch et al., 2014, Brown and Sandholm, 2017, Moravčík et al., 2017, Brown and Sandholm, 2018, Brown et al., 2018b, Brown and Sandholm, 2019, Brown et al., 2020]. The main difference from AlphaZero is that the search and self-play training in POG are also sound for imperfect information games, and evaluation across game types. The main difference from DeepStack is the use of significantly less domain knowledge: the use of self-play (rather than poker-specific heuristics) to generate training data and a single network for all stages of the game. The most closely related algorithm is Recurrent Belief-based Learning (ReBeL) [Brown et al., 2020]. Like POG, ReBeL combines search, learning, and game-theoretic reasoning via self-play. The main difference is that POG is based on (safe) continual resolving and sound self-play. To achieve ReBeL's guarantees, its test-time search must be conducted with the same algorithm as in training, whereas POG can use any belief-based value-and-policy network of the form described in Section 10.3.1 (similarly to e.g. AlphaZero, which trains using 800 simulations but then can use substantially larger simulation limits at test-time). POG is also validated empirically across many different challenge games of different game types.

There has been considerable work in search for imperfect information games. One method that has been quite successful in practice is determinization: at decision-time, a set of candidate world states are sampled, and some form of search is performed [Cowling et al., 2012, Long et al., 2010]. In fact, PimBot [Nijssen and Winands, 2012, Nijssen, 2013] is based on these methods and achieved state-of-the-art results in Scotland Yard. However, these methods are not guaranteed to converge to an optimal strategy over time. We demonstrate this lack of convergence in practice over common search algorithms and standard RL benchmarks in Section F.3.4. In contrast, the search in POG is based on game-theoretic reasoning. Other algorithms have proposed adding game-theoretic reasoning to search: Smooth UCT [Heinrich and Silver, 2015] combines UCT [Kocsis and Szepesvári, 2006] with fictitious play, however its convergence properties are not known. Online Outcome Sampling [Lisý et al., 2015] derives an MCTS variant of Monte Carlo CFR [Lanctot et al., 2009]; however, OOS is only guaranteed to approach an approximate equilibrium at a single information state (local consistency) and has not been evaluated in large games. GT-CFR used by POG makes use of sound search based on decomposition and is globally consistent [Burch et al., 2014, Šustr et al., 2020].

There have been a number of RL algorithms that have been proposed for two-player zero-sum games: Fictitious Self-Play [Heinrich et al., 2015, Heinrich and Silver, 2016], Policy-Space Response Oracles (PSRO) [Lanctot et al., 2017, Muller et al., 2020, McAleer et al., 2020], Double Neural CFR [Li et al., 2019], Deep CFR and DREAM [Brown et al., 2018a, Steinberger et al., 2020], Regret Policy Gradients [Srinivasan et al., 2018], Exploitability Descent [Lockhart et al., 2019], Neural Replicator Dynamics (NeuRD) [Hennes et al., 2020], Advantage Regret-Matching Actor Critic [Gruslys et al., 2020], Friction FoReL [Perolat et al., 2021], MAIO [Munos et al., 2020], Extensive-form Double Oracle (XDO) [McAleer

et al., 2021], and Neural Auto-curricula (NAC) [Feng et al., 2021]. These methods adapt classical algorithms for computing (approximate) Nash equilibria to the RL setting with sampled experience and general function approximation. As such, they combine game-theoretic reasoning and learning. Several of these methods have shown promise to scale: Pipeline PSRO defeated the best openly available agent in Stratego Barrage; Deep CFR, DREAM, and ARMAC showed promising results on large poker games. Combined with human data, AlphaStar was able to use game-theoretic reasoning to create master-level real-time strategy policy [Vinyals et al., 2019]. However, none of them can use search at test-time to refine their policy.

Lastly, there have been works that use some combination of search, learning, and/or game-theoretic reasoning applied to specific domains. Neural networks have been trained via Q-learning to learn to play Scotland Yard [Dash et al., 2018]; however, the overall play strength of the resulting policy was not directly compared to any other known Scotland Yard agent. In poker, Supremus proposed a number of improvements to DeepStack and demonstrated that they make a big difference when playing human experts [Zarick et al., 2020]. Another work used a method inspired by DeepStack applied to The Resistance [Serrino et al., 2019]. In the cooperative setting, several works have made use of belief-based learning (and search) using public subgame decomposition [Foerster et al., 2019, Lerer et al., 2020, Sokota et al., 2021], applied to Hanabi [Bard et al., 2020]. Search and reinforcement learning were combined to produce a bridge bidding player that cooperated with a state-of-the-art bot (WBridge5) and with humans [Lockhart et al., 2020]. Learning and game-theoretic reasoning were also recently combined to produce agents that play well with humans without human data on the collaborative game Overcooked [Strouse et al., 2021]. Of considerable note is the game of (no-press) Diplomacy. Game-theoretic reasoning was combined with learning in Best Response Policy Iteration [Anthony et al., 2020]. Game-theoretic search and supervised learning were employed in [Gray et al., 2020] reaching human-level performance on the two-player variant. Recently, all three were combined in DORA [Bakhtin et al., 2021] which learned to play Diplomacy without human data, and also reached human-level performance on the two-player variant. The main difference between POG and these works is that they focus on specific games and exploit domain-specific knowledge to attain strong performance.

## 10.6 Conclusion

In this chapter, we describe Player of Games (POG) a unified algorithm that combines search, learning, and game-theoretic reasoning. POG is comprised of two main components: a novel growing-tree counterfactual regret minimization (GT-CFR), and sound self-play whichs learn counterfactual value-and-policy networks via self-play. Most notably, POG is a sound algorithm for both perfect *and* imperfect information games: as computational resources increase, POG is guaranteed to produce better approximation of minimax-optimal strategies. This finding is also verified empirically in Leduc poker, where additional search leads to test-time approximation refinement, unlike any pure reinforcement learning algorithms that do not use search.

POG is the first sound algorithm in this class to demonstrate strong performance on challenge domains, using minimal domain knowledge. In the perfect information games of chess and Go, POG performs at the level of human experts or professionals, but can be significantly weaker than specialized algorithms for this class of games, like AlphaZero, when given the same resources. In the imperfect information game no-limit Texas hold'em poker, POG beats Slumbot, the best openly available poker agent, and is shown not to be

exploited by a local best-response agent using poker-specific heuristics. In Scotland Yard, POG defeats the state-of-the-art agent.

There are some limitations of POG that are worth investigating in future work. First, the use of betting abstractions in poker could be removed in favor of a general action-reduction policy for large action spaces. Second, POG currently requires enumerating the information states per public state which can be prohibitively expensive in some games; this might be approximated by a generative model that samples world states and operates on the sampled subset. Finally, significant computational resources are used to attain strong play in challenge domains; an interesting question is whether this level of play is achievable with less computational resources.

## **10.7 Author's contributions**

Player of Games was culmination of an extensive research endeavor involving multiple authors spanning several years. I have significantly contributed to various aspects of the research, including the theoretical and experimental work related to growing search trees, the self-play process, the application of deep learning techniques, and the appropriate evaluation of the agent.

# 11. Conclusion

Combination of the decision-time search with a heuristic value function allowed AI agents to outperform the best human players in games such as Backgammon, Chess, Go, and Arimaa [Tesauro, 1995, Campbell et al., 2002, Silver et al., 2016, Wu, 2015]. More recently, universal agents that learned through self-play and can master multiple games using "zero" prior knowledge have emerged [Silver et al., 2017a, Schrittwieser et al., 2020].

On the other hand, traditional techniques used in imperfect information games worked very differently. They created a small, abstract version of a game and solved this abstraction in one go. This process was game-specific and had to be manually redone for each new game. While this approach was successful when used in smaller games, it resulted in severe weaknesses in play when applied to larger games, as shown by local best response [Lisý and Bowling, 2017a].

Techniques discussed in this thesis help to bridge the gap between perfect and imperfect information.

DeepStack introduced generalization of the search with the learned value function to imperfect-information settings. This has led to the first AI victory over human professional players in no-limit poker completing a long-standing AI challenge.

Similarly, the generalization of self-play combined with a growing search tree introduced by the Player of Games resulted in a universal algorithm that can master both perfect and imperfect information games starting from scratch.

## 11.1 Potential Applications

The concepts introduced in this thesis hold potential for new and exciting applications, as many real-world problems lack perfect information.

Sound search from DeepStack is used by GTO Wizard [GTO Wizzard Development Team, 2023], software leveraged by top professional poker players to analyze and improve their play.

A lot of previous work on AI for large imperfect information games was focused on poker. One specificity of poker is that all actions of a player can be observed by their opponents. The game of Scotland Yard tackled by Player of Games is more general and it resembles patrolling games used for real world problems like airport security [Pita et al., 2009] and wildlife protection [Fang et al., 2015].

## 11.2 Future Work

The main limitation of the sound search used by DeepStack and Player of Games is the need to enumerate all possible information states contained in a public state. This prohibits straightforward use of these algorithms in games with large belief spaces such as full Stratego. This could be an interesting area of future research; potential solutions could involve Monte Carlo subsampling of information sets or learned implicit representations of belief states.

A significant recent milestone in perfect information games was MuZero [Schrittwieser et al., 2020]. It is not only able to learn to play a game without prior knowledge, it is also capable of learning the rules of the game itself just from interaction with the environment. Extending this capability to imperfect information games would produce even more general agents capable of mastering environments with unknown dynamics.

# Bibliography

- Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.
- V. L. Allis. *Searching for solutions in games and artificial intelligence*. PhD thesis, University of Limburg, 1994.
- Thomas W. Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian M. Gemp, Thomas C. Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, Satinder Singh, Thore Graepel, and Yoram Bachrach. Learning to play no-press Diplomacy with best response policy iteration. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *CoRR*, abs/1806.07857, 2018.
- Anton Bakhtin, David Wu, Adam Lerer, and Noam Brown. No-press Diplomacy from scratch. In *Proceedings of the Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *Proceedings of the Sixth International Conference on Learning Representations*, 2018.
- Nolan Bard, John Hawkin, Jonathan Rubin, and Martin Zinkevich. The annual computer poker competition. *AI Magazine*, 34(2):112–112, 2013.
- Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280, 2020.
- Petr Baudis, Jean loup Gailly, and Lemonsqueeze. Pachi: Software for the board game of Go/Weiqi/Baduk. <https://pachi.or.cz/>, 2016.
- Dimitris Bertsimas and John N Tsitsiklis. Introduction to linear optimization. 1997.
- Darse Billings, Aaron Davidson, Jonathen Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1–2):201–240, 2002.
- 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 *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, 2003a.
- 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 *IJCAI*, volume 3, page 661, 2003b.

- Branislav Bošanský. Solving extensive-form games with double-oracle methods. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems*, pages 1423–1424, 2013.
- Michael Bowling, Michael Johanson, Neil Burch, and Duane Szafron. Strategy evaluation in extensive games with importance sampling. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*, pages 72–79, 2008.
- Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.
- Michael H Bowling, Nicholas Abou Risk, Nolan Bard, Darse Billings, Neil Burch, Joshua Davidson, John Alexander Hawkin, Robert Holte, Michael Johanson, Morgan Kan, et al. A demonstration of the Polaris poker system. In *AAMAS (2)*, pages 1391–1392. Citeseer, 2009.
- Phelim P Boyle. Options: A Monte Carlo approach. *Journal of financial economics*, 4(3): 323–338, 1977.
- Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, Jiri Cermak, and Michal Pechoucek. Double-oracle algorithm for computing an exact Nash equilibrium in zero-sum extensive-form games. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 335–342. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
- J. Bronowski. The ascent of man, 1973. Documentary (1973). Episode 13.
- Noam Brown and Tuomas Sandholm. Strategy-based warm starting for regret minimization in games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.
- Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.
- Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.
- Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep Counterfactual Regret Minimization. *CoRR*, abs/1811.00164, 2018a.
- Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Proceedings of the Thirty-second Conference on Neural Information Processing Systems*, 2018b.
- Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. In *Thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.



- Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
- Neil Burch. *Time and Space: Why Imperfect Information Games are Hard*. PhD thesis, University of Alberta, 2017.
- Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *AAAI*, pages 602–608, 2014.
- Neil Burch, Martin Schmid, Matej Moravčík, Dustin Morill, and Michael Bowling. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- Neil Burch, Matej Moravčík, and Martin Schmid. Revisiting CFR+ and alternating updates. *Journal of Artificial Intelligence Research*, 64:429–443, 2019.
- Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- Jiří Čermák, B Bošanský, and V Lisý. Practical performance of refinements of Nash equilibria in extensive-form zero-sum games. In *Proceedings of the European Conference on Artificial Intelligence*, 2014.
- Katherine Chen and Michael Bowling. Tractable objectives for robust policy optimization. volume 25, 2012.
- cmu. Brains Vs. AI. <http://www.cs.cmu.edu/brains-vs-ai>, 2015.
- Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS workshop*, number CONF, 2011.
- B Jack Copeland. *The essential turing*. Clarendon Press, 2004.
- Rémi Coulom. *Efficient selectivity and backup operators in Monte-Carlo tree search*, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
- Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4: 120–143, 2012.
- George B. Dantzig. *A proof of the equivalence of the programming problem and the game problem*, page 330–335. Wiley, New York, NY, 1951.
- Tirtharaj Dash, Sahith N Dambekodi, Preetham N Reddy, and Ajith Abraham. Adversarial neural networks for playing hide-and-search board game Scotland Yard. *Neural Computing and Applications*, 32(8):3149–3164, 2018.
- Economist. Poker: A big deal. *The Economist*, pages 31–38, 2007.
- Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

- Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Online convex optimization for sequential decision processes and extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1917–1925, 2019.
- Xidong Feng, Oliver Slumbers, Ziyu Wan, Bo Liu, Stephen Marcus McAleer, Ying Wen, Jun Wang, and Yaodong Yang. Neural auto-curricula in two-player zero-sum games. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- David A Ferrucci. Introduction to “this is watson”. *IBM Journal of Research and Development*, 56(3.4):1–1, 2012.
- Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017.
- Jakob N. 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. 2019.
- Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.
- Ian Frank, David A Basin, and Hitoshi Matsubara. Finding optimal strategies for imperfect information games. In *AAAI/IAAI*, pages 500–507, 1998.
- Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36: 193–202, 1980.
- Sam Ganzfried and Tuomas Sandholm. Improving performance in imperfect-information games with large state and action spaces by solving endgames. In *Computer Poker and Imperfect Information Workshop at the National Conference on Artificial Intelligence*, 2013.
- Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover’s distance in imperfect-information games. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 682–690, 2014.
- Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 37–45, 2015.
- Sam Ganzfried, Tuomas Sandholm, and Kevin Waugh. Strategy purification and thresholding: Effective non-equilibrium approaches for playing large games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 871–878. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, March 2012.

- Richard Gibson. Regret minimization in games and the development of champion multi-player computer poker-playing agents. *Ph.D. Dissertation, University of Alberta*, 2014.
- Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, and Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1355–1361, 2012.
- 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*, volume 21, page 1007, 2006.
- Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 50. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press, 2007.
- 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 *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 911–918. Citeseer, 2008.
- Jonathan Gray, Adam Lerer, Anton Bakhtin, and Noam Brown. Human-level performance in no-press Diplomacy via equilibrium search. In *In Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.
- Audrūnas Gruslys, Marc Lanctot, Rémi Munos, Finbarr Timbers, Martin Schmid, Julien Perolat, Dustin Morrill, Vinicius Zambaldi, Jean-Baptiste Lespiau, John Schultz, Mohammad Gheshlaghi Azar, Michael Bowling, and Karl Tuyls. The advantage regret-matching actor-critic. 2020.
- GTO Wizzard Development Team. GTO Wizzard, 2023. URL <https://gtowizard.com/en/>.
- Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Finding equilibria in games of no chance. In *International Computing and Combinatorics Conference*, pages 274–284. Springer, 2007.
- S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- J. Heinrich and D. Silver. Smooth UCT search in computer poker. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
- Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.
- Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, 2015.

- Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Remi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duenez-Guzman, and Karl Tuyls. Neural replicator dynamics. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020.
- Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.
- Bret Hoehn, Finnegan Southey, Robert C Holte, and Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. In *AAAI*, volume 5, pages 783–788, 2005.
- Feng-Hsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Championship*. Princeton University Press, 2006.
- Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.
- Eric Jackson. Slumbot NL: Solving large games with counterfactual regret minimization using sampling and distributed processing. In *Proceedings of the Computer Poker and Imperfect Information: Papers from the AAI 2013 Workshop*, 2013. <https://github.com/ericgjackson/slumbot2019>.
- Eric Jackson. Slumbot github repository, unspecified. URL <https://github.com/ericgjackson/slumbot2017>.
- M. Johanson. Measuring the size of large no-limit poker games. Technical Report TR13-01, Department of Computing Science, University of Alberta, 2013.
- Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 258–265. AAAI Press, 2011.
- Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling. Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.
- Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. volume 1, page 271–278, 05 2013.
- Michael Bradley Johanson. *Robust Strategies and Counter-Strategies: From Superhuman to Optimal Play*. PhD thesis, University of Alberta, 2016. URL <http://johanson.ca/publications/theses/2016-johanson-phd-thesis/2016-johanson-phd-thesis.pdf>.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.
- Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
- Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

- Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94:167–215, 1997.
- Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. Rethinking formal models of partially observable multiagent decision making. *AIJ*, 2021. to appear.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. volume 25, 2012.
- Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1: 97–103, 1950.
- 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.
- Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. *Advances in neural information processing systems*, 22, 2009.
- Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.
- Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.
- Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- Hui Li, Kailiang Hu, Shaohua Zhang, Yuan Qi, and Le Song. Double neural counterfactual regret minimization. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR)*, 2019.
- Viliam Lisý, Marc Lanctot, and Michael Bowling. Online Monte Carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 27–36. International Foundation for Autonomous Agents and Multiagent Systems, 2015.
- V. Lisý and M. Bowling. Equilibrium approximation quality of current no-limit poker bots. In *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games*, 2017a. <https://arxiv.org/abs/1612.07547>.
- Viliam Lisý and Michael Bowling. Equilibrium approximation quality of current no-limit poker bots. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017b.

- Viliam Lisý, Trevor Davis, and Michael Bowling. Counterfactual regret minimization in sequential security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.
- Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-dependent control variates for policy optimization via stein identity. 2018.
- Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.
- Edward Lockhart, Neil Burch, Nolan Bard, Sebastian Borgeaud, Tom Eccles, Lucas Smaira, and Ray Smith. Human-agent cooperation in bridge bidding. In *Proceedings of the Cooperative AI Workshop at 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.
- Jeffrey Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information Monte Carlo sampling in game tree search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, page 134–140. AAAI Press, 2010.
- T. A. Marsland and M. Campbell. A survey of enhancements to the alpha-beta algorithm. In *Proceedings of the ACM '81 Conference*, page 109–114, New York, NY, USA, 1981. Association for Computing Machinery.
- Stephen McAleer, John Lanier, Roy Fox, and Pierre Baldi. Pipeline PSRO: A scalable approach for finding approximate Nash equilibria in large games. In *Proceedings of the Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- Stephen McAleer, John Lanier, Pierre Baldi, and Roy Fox. Xdo: A double oracle algorithm for extensive-form games. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.
- Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, 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.
- Matej Moravčík, Martin Schmid, Karel Ha, Milan Hladik, and Stephen J Gaukrodger. Refining subgames in large imperfect information games. In *AAAI*, pages 572–578, 2016.



- Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton university press, 1953.
- D. Morrill. Annual computer poker competition poker GUI client, 2012. URL [https://github.com/dmorrill10/acpc\\_poker\\_gui\\_client/tree/v1.2](https://github.com/dmorrill10/acpc_poker_gui_client/tree/v1.2).
- Martin Müller. Computer Go. *Artificial Intelligence*, 134(1):145–179, 2002.
- Martin Müller and Ralph Gasser. Experiments in computer Go endgames. *Games of No Chance*, pages 273–284, 1996.
- Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. A generalized training approach for multiagent learning. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR)*, 2020.
- Remi Munos, Julien Perolat, Jean-Baptiste Lespiau, Mark Rowland, Bart De Vylder, Marc Lanctot, Finbarr Timbers, Daniel Hennes, Shayegan Omidshafiei, Audrunas Gruslys, Mohammad Gheshlaghi Azar, Edward Lockhart, and Karl Tuyls. Fast computation of Nash equilibria in imperfect information games. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.
- J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.
- J v Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.
- J.A.M. Nijssen. *Monte-Carlo Tree Search for Multi-Player Games*. PhD thesis, Maastricht University, 2013.
- J.A.M. Nijssen and M.H.M. Winands. Monte-Carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4): 282–294, 2012.
- Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge university press, 2007.
- Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.
- Art B. Owen. *Monte Carlo theory, methods and examples*. 2013.
- Tom Pepels, Mandy J.W. Tak, Marc Lanctot, and Mark H.M. Winands. Quality-based rewards for Monte-Carlo tree search simulations. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 2014.
- Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. From Poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *Proceedings of the The Thirty-eighth International Conference on Machine Learning (ICML)*, 2021.

- Michele Piccione and Ariel Rubinstein. On the interpretation of decision problems with imperfect recall. *Games and Economic Behavior*, 20(1):3–24, 1997.
- Michele Piccione, Ariel Rubinstein, et al. *The absent minded driver’s paradox: Synthesis and responses*. Sackler Institute for Economic Studies, 1996.
- piel-des jahres. Game of the year 1983: Scotland Yard, 2020. URL <https://www.spiel-des-jahres.de/spiel-des-jahres-1983-scotland-yard/>. Accessed: 2020-Jan-30.
- James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Using game theory for Los Angeles airport security. *AI Magazine*, 30(1):43, 2009.
- Kuhn poker. Kuhn poker — wikipedia, the free encyclopedia, 2018. [Online; accessed 28-August-2018].
- Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2nd edition, 2003.
- Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44:206–226, 1959.
- Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *Ai Magazine*, 31(4):13–32, 2010.
- Jonathan Schaeffer and Aske Plaat. New advances in alpha-beta searching. In *CSC ’96*, 1996.
- Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook the world man-machine checkers champion. *AI magazine*, 17(1):21–21, 1996.
- Martin Schmid. *Search in Imperfect Information Games*. PhD thesis, 2021. URL <https://arxiv.org/abs/2111.05884>.
- Martin Schmid, Matej Moravčík, and Milan Hladik. Bounding the support size in extensive form games with imperfect information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.
- Martin Schmid, Matej Moravčík, Milan Hladik, and Stephen J Gaukroder. Automatic public state space abstraction in imperfect information games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravčík, Rudolf Kadlec, and Michael Bowling. Variance reduction in Monte Carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines. In *Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.
- Martin Schmid, Matej Moravčík, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, et al. Player of games. *arXiv preprint arXiv:2112.03178*, 2021.
- David Schnizlein, Michael Bowling, and Duane Szafron. Probabilistic state translation in extensive games with large action sets. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.



- 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. *Nature*, 588(7839):604–609, 2020.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Richard B. Segal. On the scalability of parallel UCT. In *CG'10: Proceedings of the 7th international conference on Computers and games*, pages 36–47, 2010.
- Dominik Seitz, Vojtěch Kovařík, Viliam Lisý, 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.
- Jack Serrino, Max Kleiman-Weiner, David C. Parkes, and Joshua B. Tenenbaum. Finding friend and foe in multi-agent games. In *Proceedings of the Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- Claude E Shannon. XXII. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.
- Jiefu Shi and Michael L Littman. Abstraction methods for game theoretic poker. In *Computers and Games: Second International Conference, CG 2000 Hamamatsu, Japan, October 26–28, 2000 Revised Papers 2*, pages 333–345. Springer, 2001.
- 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.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.
- David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017b.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 632(6419):1140–1144, 2018.
- Samuel Sokota, Edward Lockhart, Finbarr Timbers, Elnaz Davoodi, Ryan D’Orazio, Neil Burch, Martin Schmid, Michael Bowling, and Marc Lanctot. Solving common-payoff games with approximate policy iteration. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.
- Finnegan Southey, Michael H. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and D. Chris Rayner. Bayes’ bluff: Opponent modelling in poker. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, pages 550–558, 2005.

- 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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- Eric Steinberger, Adam Lerer, and Noam Brown. DREAM: Deep regret minimization with advantage baselines and model-free learning, 2020.
- DJ Strouse, Kevin R. McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems*, 2021.
- Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. Monte Carlo continual resolving for online strategy computation in imperfect information games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 224–232, 2019.
- Michal Šustr, Martin Schmid, Matej Moravčík, Neil Burch, Marc Lanctot, and Michael Bowling. Sound algorithms in imperfect information games. *arXiv preprint arXiv:2006.08740*, 2020.
- R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2017. Draft, in progress.
- O. Tammelin. Solving large imperfect information games using CFR+. *CoRR*, abs/1407.5042, 2014.
- Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.
- The GnuGo Development Team. Gnugo, 2009. URL <https://www.gnu.org/software/gnugo/>.
- Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, March 1994.
- Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- The Stockfish Development Team. Stockfish: Open source chess engine, 2021. URL <https://stockfishchess.org/>.
- George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. *arXiv preprint arXiv:1802.10031*, 2018.
- Joel Veness, Marc Lanctot, and Michael Bowling. Variance reduction in Monte-Carlo tree search. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1836–1844, 2011.
- Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor

- Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1): 295–320, 1928.
- J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.
- Michal Šustr, Martin Schmid, Matej Moravčík, Neil Burch, Marc Lanctot, and Michael Bowling. Sound search in imperfect information games. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020.
- Kevin Waugh. A fast and optimal hand isomorphism algorithm. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.
- Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 781–788, 2009.
- Kevin Waugh, Dustin Morrill, J. Andrew Bagnell, and Michael Bowling. Solving games with functional regret estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.
- Martha White and Michael H. Bowling. Learning a value analysis tool for agent evaluation. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1976–1981, 2009.
- Daniel Whitehouse. *Monte Carlo tree search for games with hidden information and uncertainty*. PhD thesis, University of York, 2014.
- R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229—256, 1992.
- J. Wood. Doug Polk and team beat Claudico to win \$100,000 from Microsoft & The Rivers Casino. <http://pokerfuse.com/news/media-and-software/26854-doug-polk-and-team-beat-claudico-win-100000-microsoft/>, 2015.
- Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. *CoRR*, 2018. abs/1803.07246.
- David J Wu. Designing a winning Arimaa program. *ICGA Journal*, 38(1):19–40, 2015.
- Yinyu Ye. An  $O(n^3 \log n)$  potential reduction algorithm for linear programming. *Mathematical programming*, 50(1-3):239–258, 1991.
- Ryan Zarick, Bryan Pellegrino, Noam Brown, and Caleb Banister. Unlocking the potential of deep counterfactual value networks. *CoRR*, abs/2007.10442, 2020.

M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 905–912, 2007.

Martin Zinkevich and Michael Littman. The AAAI computer poker competition. *Journal of the International Computer Games Association*, 29, 2006. News item.

Martin Zinkevich, Michael H. Bowling, Nolan Bard, Morgan Kan, and Darse Billings. Optimal unbiased estimators for evaluating agent performance. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 573–579, 2006.

# List of Figures

4.1	Extensive form game tree for one-card poker. . . . .	33
4.2	The public tree of a game . . . . .	34
4.3	In the new strategy $\sigma'$ , we bound the number of supported public actions (bold arrows). . . . .	37
4.4	Equilibrium Preserving Transformation . . . . .	38
5.1	Coordinated Matching Pennies . . . . .	45
5.2	Exploitability of Online Outcome Sampling . . . . .	51
6.1	High-level overview of VR-MCCFR and related methods . . . . .	54
6.2	Values and updates for the discussed methods . . . . .	58
6.3	Convergence of exploitability for different MCCFR variants . . . . .	62
6.4	Speedup of VR-MCCFR . . . . .	63
6.5	Variance of counterfactual values in VR-MCCFR . . . . .	63
6.6	Detailed comparison of different VR-MCCFR variants . . . . .	64
7.1	Subgame refinement framework . . . . .	66
7.2	Endgame solving construction - Gadget 1 . . . . .	69
7.3	re-solving gadget construction - Gadget 2 . . . . .	70
7.4	Max margin gadget - Gadget 3 . . . . .	73
7.5	Subgame margins of the refined strategies . . . . .	74
8.1	Comparison of MIVAT, imaginary observations, and AIVAT . . . . .	79
8.2	Value estimates for self-play in Leduc hold'em . . . . .	83
8.3	Value estimates for dissimilar strategies in Leduc hold'em . . . . .	83
8.4	Value estimates for self-play in HUNL . . . . .	84
8.5	Value estimates for dissimilar strategies in HUNL . . . . .	84
9.1	A portion of the public tree in HUNL . . . . .	88
9.2	DeepStack overview . . . . .	91
9.3	Deep counterfactual value network . . . . .	92
9.4	Performance of professional poker players against DeepStack . . . . .	93
10.1	An example structure of public belief state . . . . .	97
10.2	A depiction of a public belief state in Scotland Yard . . . . .	97
10.3	An example game with two specific subgames . . . . .	101
10.4	A counterfactual value-and-policy network . . . . .	102
10.5	Overview of phases in one iteration of Growing-Tree CFR . . . . .	103
10.6	POG Training Process . . . . .	105
10.7	Exploitability of POG as a function of the number of training steps . . . . .	108
10.8	Scalability of POG with increasing number of neural network evaluations compared to AlphaZero . . . . .	110
10.9	Head-to-head results and LBR of PoG and Slumbot . . . . .	111
10.10	Win rate of POG(400, 1) against PimBot . . . . .	111
B.1	Coordinated Matching Pennies . . . . .	144
B.2	Kuhn Poker . . . . .	144
C.1	Convergence of MCCFR and MCCFR+ on logarithmic scale . . . . .	146

E.1	Huber loss with different numbers of hidden layers in the neural network . .	164
E.2	DeepStack’s exploitability within a particular public state at the start of the river as a function of the number of re-solving iterations. . . . .	170
F.1	An example of Factored-Observation Stochastic Game . . . . .	173
F.2	An example of a public tree . . . . .	174
F.3	Finding the root of the search tree . . . . .	175
F.4	Initial situation on the glasses map for Scotland Yard . . . . .	181
F.5	Comparing performance of DQN, A2C, tabular Q-learning and uniform random policy in Kuhn poker and Leduc poker . . . . .	184
F.6	An example situation in Scotland Yard . . . . .	185

# List of Tables

9.1	Exploitability bounds from Local Best Response . . . . .	94
10.1	A table with relative Elo of different agents . . . . .	110
C.1	Detailed example of updates computed for player 1 in Kuhn poker during forward pass of the algorithm . . . . .	151
C.2	Detailed example of updates for the backward pass . . . . .	152
E.1	Results against professional poker players estimated with AIVAT . . . . .	157
E.2	Exploitability lower bound of different programs using local best response . . . . .	158
E.3	Lookahead re-solving specifics by round . . . . .	159
E.4	Absolute ( $L_1$ ), Euclidean ( $L_2$ ), and maximum absolute ( $L_\infty$ ) errors . . . . .	160
E.5	Performance of LBR exploitation of DeepStack with different actions allowed . . . . .	161
E.6	Thinking times for both humans and DeepStack . . . . .	162
F.1	A neural network architecture and features used for each game. . . . .	177
F.2	Hyperparameters for each game. . . . .	178
F.3	Full Go results . . . . .	182
F.4	Average exploitability (in mbb/h) over five policy constructions obtained by independent searches of IS-MCTS runs at each information state in Leduc Poker . . . . .	183
F.5	Hyper parameters swept over in each RL algorithm. . . . .	184

# List of Publications

## Journal Papers

Moravcik Matej\*, Martin Schmid\*, Neil Burch, Viliam Lisý, 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, no. 6337 (2017): 508-513.

Burch, N., Moravcik, M. and Schmid, M., 2019. Revisiting cfr+ and alternating updates. *Journal of Artificial Intelligence Research*, 64, pp.429-443.

## Conference Papers

Sustr, M., Schmid, M., Moravcik, M., Burch, N., Lanctot, M., & Bowling, M. (2020). "Sound search in imperfect information games" 20th International Conference on Autonomous Agents and Multiagent Systems

Schmid, Martin, et al. "Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. No. 01. 2019.

Burch, N., Schmid, M., Moravcik, M., Morill, D. and Bowling, M., 2018, April. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Moravcik, M., Schmid, M., Ha, K., Hladik, M. and Gaukrodger, S.J., 2016, February. Refining subgames in large imperfect information games. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Schmid, M., Moravcik, M. and Hladik, M., 2014, June. Bounding the support size in extensive form games with imperfect information. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

## Workshop Papers

Schmid, Martin, et al. "Automatic public state space abstraction in imperfect information games." *AAAI Workshop: Computer Poker and Imperfect Information*. 2015.

## ArXiv

Schmid, M., Moravcik, M., Burch, N., Kadlec, R., Davidson, J., Waugh, K., Bard, N., Timbers, F., Lanctot, M., Holland, Z., Davoodi, E., Christianson, A., Bowling, M. "Player of Games." *arXiv preprint arXiv:2112.03178* (2021).

---

\*Equal contribution, alphabetical order.



# A. Attachments for Chapter 4

## A.1 Proof of Lemma 4.5

*Proof.* To prove the lemma, we look at definition of  $u_i$

$$u_i(\sigma|_{I \rightarrow a}, I) = \frac{\sum_{h \in I, h' \in Z} \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \pi^{\sigma|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma|_{I \rightarrow a}}(I)}$$

Because  $I$  is in some descendant of  $\rho$  and all action probabilities after  $I$  remain unchanged,  $\pi^{\sigma'|_{I \rightarrow a}}(h'|h) = \pi^{\sigma|_{I \rightarrow a}}(h'|h)$ .

If  $i = p(\rho)$ , clearly  $\pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) = \pi_{-i}^{\sigma|_{I \rightarrow a}}(h)$ ,  $\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I) = \pi_{-i}^{\sigma|_{I \rightarrow a}}(I)$  and the lemma holds.

If  $i \neq p(\rho)$ , let  $\rho'$  be the first public node after  $\rho$  on the path from  $\rho$  to  $I$ . For each  $h \in I$ , we decompose  $h = (h_1, b, h_2)$ , where  $h_1 \in \rho$  and  $(h_1, b) \in \rho'$ . Only the probability of action  $b$  could be changed (multiplied with  $\kappa$ ), therefore

$$\begin{aligned} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) &= \kappa(\rho \rightsquigarrow \rho') \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \\ \pi_{-i}^{\sigma'|_{I \rightarrow a}}(I) &= \kappa(\rho \rightsquigarrow \rho') \pi_{-i}^{\sigma|_{I \rightarrow a}}(I) \end{aligned}$$

After substitution to the definition of  $u_i$ , we get  $u_i(\sigma'|_{I \rightarrow a}, I) = u_i(\sigma|_{I \rightarrow a}, I)$ .  $\square$

## A.2 Proof of Lemma 4.6

*Proof.* From the previous lemma, we know the property holds for any information set  $I$  after  $p$ . We look at all other information sets. There are two cases we consider, based on the acting player in  $I$ ,  $p(I)$ .

**Information sets for all players except of the player  $p(\rho)$**

$p(I) \neq p(\rho)$  and since the public state  $\rho$  is simple,  $I \notin p(\rho)$ .

For  $I \in \text{prev}(\rho, i)$ , lemma holds directly from the restrictions on EPT. In these information sets, also  $u_i(\sigma', I) = u_i(\sigma, I)$ .

For  $I \notin \text{prev}(\rho, i)$ , player  $i = p(I)$  and public set  $\rho$ , we define  $O(I, \rho)$  as the set of nearest descendants in  $\text{prev}(\rho, i)$ .

$$O(I, \rho) = \{I' \in \text{prev}(\rho, i); | h' \in I' \implies \exists h \in I, h \sqsubseteq h' \text{ and there is no other } I'' \in \text{prev}(\rho, i), h'' \in I'', h'' \sqsubseteq h'\}$$

Because the game satisfies perfect recall, any history  $h \in \rho$  having a prefix in  $I$  has also a prefix in  $O(I, \rho)$ .

Next step is to divide the set of terminal histories  $Z$  to two disjoint subsets

$$\begin{aligned} Z_1 &= \text{terminal histories with prefix in } O(I, \rho). \\ Z_2 &= Z \setminus Z_1. \end{aligned}$$

Since these sets are disjoint, we can compute  $u_i$  as

$$u_i(\sigma'|_{I \rightarrow a}, I) = \frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} + \quad (\text{A.1})$$

$$\frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} \quad (\text{A.2})$$

We compute the value of (A.1) as

$$(A.1) = \frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h') u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (\text{A.3})$$

$$\frac{\sum_{I^* \in O(I, \rho)} \sum_{h \in I, h' \in Z, h^* \in I^*} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi_i^{\sigma'|_{I \rightarrow a}}(h^*|h) \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h^*|h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h^*) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (\text{A.4})$$

$$\frac{\sum_{I^* \in O(I, \rho)} \pi_i^{\sigma'|_{I \rightarrow a}}(I^*|I) \sum_{h \in I, h' \in Z, h^* \in I^*} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h^*|h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h^*) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (\text{A.5})$$

$$\frac{\sum_{I^* \in O(I, \rho)} \pi_i^{\sigma'|_{I \rightarrow a}}(I^*|I) \sum_{h' \in Z, h^* \in I^*} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h^*) \pi^{\sigma'|_{I \rightarrow a}}(h'|h^*) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (\text{A.6})$$

$$\frac{\sum_{I^* \in O(I, \rho)} \pi_i^{\sigma'|_{I \rightarrow a}}(I^*|I) \pi_{-i}^{\sigma'|_{I \rightarrow a}}(I^*) u_i(\sigma'|_{I \rightarrow a}, I^*)}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (\text{A.7})$$

$$\sum_{I^* \in O(I, \rho)} u_i(\sigma'|_{I \rightarrow a}, I^*) \pi^{\sigma'|_{I \rightarrow a}}(I^*|I) = \quad (\text{A.8})$$

$$\sum_{I^* \in O(I, \rho)} u_i(\sigma|_{I \rightarrow a}, I^*) \pi^{\sigma|_{I \rightarrow a}}(I^*|I) = \quad (\text{A.9})$$

$$\frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \pi^{\sigma|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma|_{I \rightarrow a}}(I)} \quad (\text{A.10})$$

(A.5) follows from the fact that  $\pi_i^{\sigma'|_{I \rightarrow a}}(h^*|h) = \pi_i^{\sigma'|_{I \rightarrow a}}(I^*|I) \forall h^* \in I^*$

(A.7) follows from definition of  $u_i(\sigma'|_{I \rightarrow a}, I')$

(A.9) from the properties of equilibrium transformation.

Computing (A.2) is easy, because  $\sigma'|_{I \rightarrow a}$  differs from  $\sigma|_{I \rightarrow a}$  only in  $\rho$

$$(A.2) = \frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \pi^{\sigma|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma|_{I \rightarrow a}}(I)}$$

This finalizes the lemma for all information set where  $p(I) \neq p(\rho)$ .

**Information sets for the player acting in  $\rho$ .**

If  $I \in \text{last}(\rho)$ ,  $u_i(\sigma'|_{I \rightarrow a}, I) = u_i(\sigma|_{I \rightarrow a}, I)$  for each  $a \in A(I)$  because equilibrium preserving transformation can't doesn't change the probabilities in any information set after  $\rho$ .

If  $I \notin \text{last}(\rho)$ , we again divide the set of terminal histories  $Z$  to two disjoint subsets, and use them to compute  $u_i$

$Z_1 =$  histories with prefix in  $\text{last}(\rho)$ .

$Z_2 = Z \setminus Z_1$ .

$$u_i(\sigma'|_{I \rightarrow a}, I) = \frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} + \quad (\text{A.11})$$

$$\frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} \quad (\text{A.12})$$

$\sigma'|_{I \rightarrow a}$  can differ form  $\sigma|_{I \rightarrow a}$  only in  $\rho$  therefore

$$(\text{A.12}) = \frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \pi^{\sigma|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma|_{I \rightarrow a}}(I)}$$

Analogically to the case when  $p(I) \neq p(\rho)$ , we can get value of (A.11) as

$$(\text{A.11}) = \sum_{I^* \in \text{last}(\rho)} u_i(\sigma'|_{I \rightarrow a}, I^*) \pi^{\sigma'|_{I \rightarrow a}}(I^*|I) \quad (\text{A.13})$$

Because any information set has zero regret in strategy  $\sigma$ , for any  $I \in \text{last}(\rho)$

$$u_i(\sigma, I) = \max_{a^* \in A(I)} (u_i(\sigma|_{I \rightarrow a^*}, I)) = u_i(\sigma', I) \quad (\text{A.14})$$

First equality in (A.14) follows from the definition of regret, second equality form the fact that if some some action in  $\sigma'$  have nonzero probability, this action must have nonzero probability in  $\sigma$ .

This gives us

$$\sum_{I^* \in \text{last}(\rho)} u_i(\sigma'|_{I \rightarrow a}, I^*) \pi^{\sigma'|_{I \rightarrow a}}(I^*|I) = \sum_{I^* \in \text{last}(\rho)} u_i(\sigma|_{I \rightarrow a}, I^*) \pi^{\sigma|_{I \rightarrow a}}(I^*|I) \quad (\text{A.15})$$

Which finalizes the lemma. □

### A.3 Proof o Lemma 4.7

*Proof.* We divide set of terminal histories  $Z$  to two subsets

$Z_1$  - histories with prefix in  $(\rho)$ .

$Z_2 = Z - Z_1$ .

Now we can compute  $u_i$  as

$$u_i(\sigma'|_{I \rightarrow a}, I) = \frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} + \quad (\text{A.16})$$

$$\frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} \quad (\text{A.17})$$

$\sigma'|_{I \rightarrow a}$  can differ from  $\sigma|_{I \rightarrow a}$  only in  $\rho$  therefore

$$(A.17) = \frac{\sum_{h \in I, h' \in Z_2} \pi_{-i}^{\sigma|_{I \rightarrow a}}(h) \pi^{\sigma|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma|_{I \rightarrow a}}(I)}$$

This value is constant, and corresponds to  $c_0$  from the lemma. To get other constants, we split  $Z_1$  to disjoint sets. Define  $Z_{1,i}$  as the set of terminal histories with prefix in  $i$ -th child of  $\rho$ .  $Z_1 = \bigcup_{i=\{1 \dots |C(\rho)|\}} Z_{1,i}$  Now we compute (A.16) as

$$(A.16) = \frac{\sum_{h \in I, h' \in Z_1} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (A.18)$$

$$\frac{\sum_{i=\{1 \dots |C(\rho)|\}} \sum_{h \in I, h' \in Z_{1,i}} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (A.19)$$

$$\frac{\sum_{i=\{1 \dots |C(\rho)|\}} \sum_{h \in I, h' \in Z_{1,i}} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \kappa(\rho \rightsquigarrow \rho'_i) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} = \quad (A.20)$$

$$\sum_{i=\{1 \dots |C(\rho)|\}} \kappa(\rho \rightsquigarrow \rho'_i) \frac{\sum_{h \in I, h' \in Z_{1,i}} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} \quad (A.21)$$

Last equation gives us a value of  $c_i$  (see that this expression is constant)

$$c_i = \frac{\sum_{h \in I, h' \in Z_{1,i}} \pi_{-i}^{\sigma'|_{I \rightarrow a}}(h) \pi^{\sigma'|_{I \rightarrow a}}(h'|h) u_i(h')}{\pi_{-i}^{\sigma'|_{I \rightarrow a}}(I)} \quad (A.22)$$

□

# B. Attachments for Chapter 5

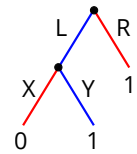
## B.1 Consistency Examples

**Example B.1.** *An online algorithm may be sound ( $\epsilon = 0$ ), but there might not be any offline equilibrium that produces the same distribution of matches.*

Suppose we have a game where each player acts once, chooses from actions  $\{A, B, C\}$  and receives zero utility (i.e. a normal-form game with  $3 \times 3$  zero payoff matrix). All strategies are equilibria. If we play  $k = 3$  matches and the players play pure strategies  $A, B$  and  $C$  in each match, we get a distribution of matches  $m = (z_1, z_2, z_3)$  that cannot be achieved with fixed offline equilibrium. In this case, the distribution is  $((w^0, (A, A)), (w^0, (B, B)), (w^0, (C, C)))$  with probability one, and all other terminal histories with probability zero.

**Example B.2.** *An algorithm that is locally consistent with equilibria can be exploited in a perfect information game.*

Suppose we have a single player game as in figure on the right. Both blue  $L, Y$  and red  $R, X$  pure strategies are equilibria. However, if the top node is locally consistent with the blue strategy, and the bottom node with the red strategy, the resulting strategy the algorithm actually plays is  $L, X$ , which is sub-optimal.



## B.2 Tabularization

We can consider two ways how the online setting can be realized, with respect to how players' state changes between  $k$  matches in the repeated game: i) no-memory, where the players take turns in a match, and their memory is reset when each match is over (players are allowed to retain memory within the individual matches), or ii) with-memory, where the players are allowed to retain memory between the matches.

As exploitability of tabularized strategy is guaranteed to reflect  $\epsilon$ -soundness only for strongly globally consistent algorithms, we assume their use only. The with-memory case then collapses to the no-memory case: strongly globally consistent algorithm simply plays as some predefined (offline) equilibrium. The following text then simply defines how to compute the offline equilibrium by querying the algorithm in all states.

**Definition B.3** (Partial strategy). *For a terminal history*

$$z = (w^0, a^0, w^1, a^1, \dots, w^l, a^l, w^{l+1})$$

*player  $n$  has a corresponding sequence of information states\**

$$s = (s^0, s^1, \dots, s^l, s^{l+1}).$$

*We say a partial strategy  $\sigma_n^{\Omega, \theta_0}(z)$  for player  $n$  who uses search  $\Omega$  and starts with state  $\theta_0$ , is an expected behavioral strategy defined only for the visited information states:*

$$\sigma_n^{\Omega, \theta_0}(z) = \{(s^t, \mu_t) \mid (\mu_t, \theta_{t+1}) = \mathbb{E}_{\theta_t} [\Omega(s^t, \theta_t) \mid s^t] \ \forall t \in \{0, \dots, l\}\}.$$

\*We omit the index  $n$  for information state  $s_n$  for clarity.

Note that when we compute the strategy  $\mu_t$  from  $\mathbb{E}_{\theta_t} [\Omega(s^t, \theta_t) | s^t]$ , we must compute it as a weighted average to respect the structure of the private tree. The weights are reach probabilities of the information state  $s^t$ : cumulative product of the player’s strategy over the sequence of information states  $s^0, \dots, s^{t-1}$ , leading to target information state  $s^t$ . See [Zinkevich et al., 2007, Eq. 4] for more details.

**Definition B.4.** *A composition of partial strategies for terminals  $\mathcal{Z}$  is a tabularized strategy*

$$\sigma_n^{\Omega, \theta_0}(\mathcal{Z}) = \bigcup_{z_i \in \mathcal{Z}} \sigma_n^{\Omega, \theta_0}(z_i).$$

### B.3 Proofs of Theorems

In perfect information games, an algorithm that is locally consistent with a subgame perfect equilibrium is sound.

*Proof.* In perfect information games the notions of an information state and a history blend together, as there is a one-to-one correspondence between them. Expected utility of a history is the same for all subgame perfect equilibria. It corresponds to the best achievable value against worst-case adversary, given that the history occurred. This property implies that the worst-case expected utility of a history remains optimal, if the player plays only actions that are in the support of any subgame perfect equilibrium in the consequent states. A formal proof can be constructed by induction on the maximal distance from a terminal history.

Notice that this exactly happens if the player plays according to an algorithm locally consistent with subgame perfect equilibria. The expected worst-case utility for any history will be optimal, and the worst-case expected utility of a match will correspond to the worst-case expected utility of the history at the beginning of the game. Therefore the worst-case expected utility of each match is also optimal and the algorithm is sound.  $\square$

We will now prepare the ground to prove Thm. 5.4.2.

When an algorithm that is globally consistent with an  $\epsilon$ -equilibrium is queried in some information states in a match in the repeated game, it will always keep playing the same behavioral strategy in these situations in subsequent matches. We call this as “filling in” strategy. Once the algorithm fills the strategy in all player’s information states, we are guaranteed to get match reward of  $u^\epsilon = u^* - \epsilon$  on average against a worst-case adversary.

Informally speaking, the bound in Thm. 5.4.2 can be easily seen to be true for a game like Coordinated Matching Pennies, or some generalization which will have a larger number of information states that need to be coordinated (think of “Coordinated Rock-Paper-Scissors”). At every match, we can incur a loss of at most  $\Delta$  when reaching an unfilled history. This is a rather pessimistic lower bound on the value, but it lets us ignore the algorithm state: we are either playing at filled information states, or achieving the worst possible value. The problematic part is making sure the bound holds also when we (repeatedly) visit previously filled information states. For each of the possible future subgames, there are two cases. In both cases, the number of  $\Delta$ -sized losses in utility plus the number of unfilled information states does not increase, so we can use induction on the length of the game to prove the claim. Along branches where the opponent had an opportunity to exploit the algorithm by playing into an unfilled information state, the algorithm loses at most  $\Delta$  utility compared to the equilibrium, but must fill in at least one information state to do so. Along branches where the agent played through filled information states, the algorithm is playing identically to the equilibrium strategy and thus achieves the same value.

To prove Thm. 5.4.2 we will need to establish a Lemma B.5, a bound of difference of utilities a player can gain if he plays according to a partially filled  $\epsilon$ -equilibrium strategy compared to  $u^\epsilon$  within an arbitrary match. The idea of the proof for Thm. 5.4.2 is then to bound this difference for any number of non-visited information states and any number of remaining matches within the response game using induction.

An online algorithm can fill in the strategy only into information states found on the trajectory to a terminal history, as it will be queried only in these situations. So after playing through a match  $z = (w^0, a^0, w^1, a^1, \dots, w^l, a^l, w^{l+1})$ , the algorithm's response at  $s_n(w^i)$  will be fixed as  $\sigma(s_n(w^i))$  for all visited worlds  $w^i$  on the trajectory  $z$ .

To talk about possible filled strategies within a single match, we will partition  $\mathcal{Z}$  into two non-empty sets of terminal histories  $\mathcal{Z}_\bullet$  (pronounced “filled”) and  $\mathcal{Z}_\circ$  (pronounced “empty” or “unfilled”). The partition has a special property of “being possible to realize in online setting”: all information states on the trajectory to terminals  $\mathcal{Z}_\bullet$  are filled, and all terminals that can be reached just through these filled information states are also in  $\mathcal{Z}_\bullet$  (we are not taking into consideration the opponent's information states, i.e. we operate only on the online player's private tree).

**Lemma B.5.** *For a probability of reaching a filled terminal  $P(\bullet) = \sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet)$ , an expected received utility for filled terminals  $u(\bullet) = \frac{\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet) u_1(z_\bullet)}{\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet)}$  and an utility of playing outside of filled histories  $u(\times)$ , it holds that*

$$P(\bullet)(u(\bullet) - u^\epsilon) \geq -(1 - P(\bullet))(u(\times) - u^\epsilon + \Delta), \quad (\text{B.1})$$

assuming  $0 < P(\bullet) < 1$ .

*Proof.* For any strategy profile  $\sigma = (\sigma_1^\epsilon, \sigma_2)$  with an  $\epsilon$ -equilibrium strategy  $\sigma_1^\epsilon$  and arbitrary opponent strategy  $\sigma_2$  it holds that

$$\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet) u_1(z_\bullet) + \sum_{z_\circ \in \mathcal{Z}_\circ} \pi^\sigma(z_\circ) u_1(z_\circ) \geq u^\epsilon. \quad (\text{B.2})$$

The terms can be simplified and rewritten as factorization of product of probabilities and (weighted) utilities as

$$P(\bullet)u(\bullet) = \underbrace{\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet)}_{P(\bullet)} \cdot \underbrace{\frac{\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet) u_1(z_\bullet)}{\sum_{z_\bullet \in \mathcal{Z}_\bullet} \pi^\sigma(z_\bullet)}}_{u(\bullet)},$$

and similarly for the “ $\circ$ ” partition. It also holds that  $P(\bullet) + P(\circ) = 1$ , as the probability of reaching a terminal history within a match is equal to one.

We can restate (B.2) as

$$P(\bullet)(u(\bullet) - u^\epsilon) + (1 - P(\bullet))(u(\circ) - u^\epsilon) \geq 0. \quad (\text{B.3})$$

Suppose that for the partition “ $\circ$ ” we didn't use an equilibrium strategy for player 1, but arbitrary strategy profile  $\sigma'$  satisfying  $P^{\sigma'}(\bullet) + P^{\sigma'}(\circ) = 1$ . We will denote its utility

$$u(\times) = \frac{\sum_{z_\circ \in \mathcal{Z}_\circ} \pi^{\sigma'}(z_\circ) u_1(z_\circ)}{\sum_{z_\circ \in \mathcal{Z}_\circ} \pi^{\sigma'}(z_\circ)}. \quad (\text{B.4})$$

The value of any two strategies cannot differ by more than the maximum difference of utilities in the game:

$$u(\circ) \leq u(\times) + \Delta. \quad (\text{B.5})$$

Putting (B.5) back to (B.3), we get the lemma that lower bounds the difference of filled partition and  $u^\epsilon$  for an arbitrary match:

$$P(\bullet)(u(\bullet) - u^\epsilon) \geq -(1 - P(\bullet))(u(\times) - u^\epsilon + \Delta). \quad (\text{B.6})$$

□

For an algorithm  $\Omega$  that is globally consistent with  $\epsilon$ -equilibria,

$$\forall k \forall \Omega_2 : \mathbb{E}_{m \sim P_{\Omega, \Omega_2}^k} [\mathcal{R}(m)] \geq u^* - \epsilon - \frac{|\mathcal{S}_1| \Delta}{k}. \quad (\text{5.2})$$

*Proof.* Let us rewrite the theorem slightly:

$$\forall k \forall \Omega_2 : k \mathbb{E}_{m \sim P_{\Omega, \Omega_2}^k} [\mathcal{R}(m)] - k u^\epsilon \geq -|\mathcal{S}_1| \Delta.$$

Since  $\mathcal{R}(m)$  is average reward, multiplying by  $k$  we get cumulative utilities in the game-play  $m = (z_1, z_2, \dots, z_k)$ :

$$\forall k \forall \Omega_2 : \mathbb{E}_{m \sim P_{\Omega, \Omega_2}^k} \left[ \sum_{i=1}^k u_1(z_i) \right] - k u^\epsilon \geq -|\mathcal{S}_1| \Delta. \quad (\text{B.7})$$

So on the left side of the inequality we have a difference of cumulative (expected) utilities and of cumulative  $u^\epsilon$ . We use cumulative values because we are now in the setting of a  $k$ -repeated game.

Let  $v$  be the number of non-visited information states of player 1 (resp. the number of unfilled information states) in a match, i.e.  $0 \leq v \leq |\mathcal{S}_1|$ , and let  $l$  be the number of next matches (including the current one), i.e.  $1 \leq l \leq k$ . We will use  $a_{v,l}$  to denote the difference between expected cumulative rewards and cumulative  $u^\epsilon$  from the current match (inclusively) until the end of the game, if we are playing against worst-case adversary. The left side of (B.7) corresponds to a value equal or greater than  $a_{|\mathcal{S}_1|,k}$ , so we need to prove that  $a_{|\mathcal{S}_1|,k} \geq -|\mathcal{S}_1| \Delta$ . It is sufficient to consider only the worst-case adversary, as the bound on  $a_{v,l}$  will hold for any other opponent as well.

We will prove the theorem by induction on  $a_{v,l}$  using  $v$  and  $l$  simultaneously. Let us characterize the *base case*. If we have visited all information states ( $v = 0$ ), we filled  $\epsilon$ -equilibrium strategy everywhere. So at each visit of such a match we receive a reward of  $u^\epsilon$ , and the difference between expected cumulative rewards and cumulative  $u^\epsilon$  is zero:

$$a_{0,l} = 0 \quad \forall l. \quad (\text{B.8})$$

The *induction hypothesis* is

$$a_{x,y} \geq -x \Delta \quad \forall x \leq v \quad \forall y < l. \quad (\text{B.9})$$

There are two possibilities of what can happen in a match. We either “hit” the filled information states, receive some (expected) reward  $u(v, l)$  and possibly continue into next



match where we receive  $a_{v,l-1}$  (if the current match is not the last one, i.e.  $l > 1$ ). Or we “miss” the filled information states, meaning we visit arbitrary number of new information states previously not visited. This will also change  $v$  to be smaller for all subsequent matches.

We state this with an abuse of notation as

$$\begin{aligned}
a_{v,l} &= P(v,l)(u(v,l) - u^\epsilon + a_{v,l-1}) \\
&\quad + P(v-1,l)(u(v-1,l) - u^\epsilon + a_{v-1,l-1}) \\
&\quad + P(v-2,l)(u(v-2,l) - u^\epsilon + a_{v-2,l-1}) \\
&\quad + \dots \\
&\quad + P(v-v,l)(u(v-v,l) - u^\epsilon + a_{v-v,l-1}),
\end{aligned} \tag{B.10}$$

where the terms  $P(v-i,l)$  and  $u(v-i,l)$  are defined similarly to how we defined them for  $P(\bullet)$  and  $u(\bullet)$ . They correspond to the probability and utilities received when we visit  $i$  new (previously unfilled) information states with  $l$  remaining matches (including current one). It holds that  $P(v,l) + P(v-1,l) + P(v-2,l) + \dots + P(v-v,l) = 1$  as the probability of reaching a terminal history within a match is equal to one.

By using the induction hypothesis (B.9) on terms  $a_{x,l-1} \forall x < v$  we get a lower bound  $a_{x,l-1} \geq -x\Delta$  on all of  $x$ . By comparing these bounds we can deduce that  $a_{v-1,l-1}$  lower bounds all of  $a_{x,l-1}$  with

$$a_{v-1,l-1} \geq -(v-1)\Delta. \tag{B.11}$$

Using this bound in (B.10) we get

$$\begin{aligned}
a_{v,l} &\geq P(v,l)(u(v,l) - u^\epsilon + a_{v,l-1}) \\
&\quad + P(v-1,l)(u(v-1,l) - u^\epsilon - (v-1)\Delta) \\
&\quad + P(v-2,l)(u(v-2,l) - u^\epsilon - (v-1)\Delta) \\
&\quad + \dots \\
&\quad + P(v-v,l)(u(v-v,l) - u^\epsilon - (v-1)\Delta).
\end{aligned} \tag{B.12}$$

We can factor it out as

$$\begin{aligned}
a_{v,l} &\geq P(v,l)(u(v,l) - u^\epsilon + a_{v,l-1}) \\
&\quad + (1 - P(v,l))(-(v-1)\Delta - u^\epsilon) \\
&\quad + P(v-1,l)u(v-1,l) + P(v-2,l)u(v-2,l) \\
&\quad + \dots + P(v-v,l)u(v-v,l).
\end{aligned} \tag{B.13}$$

We replace the utilities  $u(v-1), u(v-2), \dots, u(v-v)$  by  $u(\times)$  from (B.4):

$$a_{v,l} \geq P(v,l)(u(v,l) - u^\epsilon + a_{v,l-1}) \tag{B.14}$$

$$+ (1 - P(v,l))(u(\times) - u^\epsilon - (v-1)\Delta). \tag{B.15}$$

By using the induction hypothesis (B.9) we get

$$a_{v,l} \geq P(v,l)(u(v,l) - u^\epsilon - v\Delta) \tag{B.16}$$

$$+ (1 - P(v,l))(u(\times) - u^\epsilon - (v-1)\Delta). \tag{B.17}$$

Expanding the terms

$$\begin{aligned}
a_{v,l} &\geq P(v,l)(-v\Delta) \\
&\quad + (1 - P(v,l))(-(v-1)\Delta) \\
&\quad + P(v,l)(u(v,l) - u^\epsilon) \\
&\quad + (1 - P(v,l))(u(\times) - u^\epsilon)
\end{aligned} \tag{B.18}$$

and using Lemma B.5 with  $\bullet = v, l$  we have

$$\begin{aligned} a_{v,l} &\geq P(v, l)(-v\Delta) + (1 - P(v, l))(-(v - 1)\Delta) \\ &\quad - (1 - P(v, l))(u(\times) - u^\epsilon + \Delta) \\ &\quad + (1 - P(v, l))(u(\times) - u^\epsilon). \end{aligned} \tag{B.19}$$

Simplifying, we get a bound on  $a_{v,l}$ :

$$a_{v,l} \geq -v\Delta. \tag{B.20}$$

Note that this bound holds also if  $P(v, l) = 1$  or  $P(v, l) = 0$ :

- $P(v, l) = 0$ : Then (B.16) becomes

$$a_{v,l} \geq u(\times) - u^\epsilon - (v - 1)\Delta.$$

Using the same argument as in (B.5),

$$a_{v,l} \geq -\Delta - (v - 1)\Delta = -v\Delta.$$

- $P(v, l) = 1$ : We use (B.3), which becomes  $u(v, l) - u^\epsilon \geq 0$ . Then (B.16) becomes

$$a_{v,l} \geq u(v, l) - u^\epsilon - v\Delta \geq -v\Delta.$$

Since at the beginning of the game-play there are  $|\mathcal{S}_1|$  unfilled information states, we arrive at the original theorem

$$a_{|\mathcal{S}_1|,k} \geq -|\mathcal{S}_1|\Delta.$$

□

## B.4 Experiment details

As OOS runs MCCFR samples biased to particular information states, individual MCCFR runs can converge to different  $\epsilon$ -equilibria, as the MCCFR is parametrized differently in each information state. Additionally OOS runs in an online setting, where the algorithm is given a time budget for computing the strategy, and it may make different numbers of samples in each targeted information state.

We emulate this experimentally by slightly modifying initial regrets to produce distinct convergence trajectories. We show it is possible to highly exploit the online algorithm: in fact, it is possible to exploit the algorithm more than the worst of any individual biased strategies it produces, not just the expected strategies. This modification is sound: the initial regrets will “vanish” over longer sampling and the strategies will converge to an equilibrium in the limit. This is justified by the MCCFR regret bound [Lanctot et al., 2009, Theorem 5].

We use two games: Coordinated Matching Pennies (CMP) from Section 5.3 and Kuhn Poker [Kuhn, 1950]. We use the no-memory online setting. Nash equilibria in both games are parametrized with a single parameter  $\alpha \in \langle 0, 1 \rangle$  for one player, while the opponent has only a single unique equilibrium<sup>†</sup>. In both games, equilibria require the strategies to be

<sup>†</sup>In CMP,  $p = \alpha$  (playing Heads in  $s_1$ ) and  $q = 1 - \alpha$  (playing Heads in  $s_2$ ). In Kuhn Poker, constructing equilibrium strategy based on  $\alpha$  is more complicated and we refer the reader to [Kuhn, 1950] or [Hoehn et al., 2005] for more details.

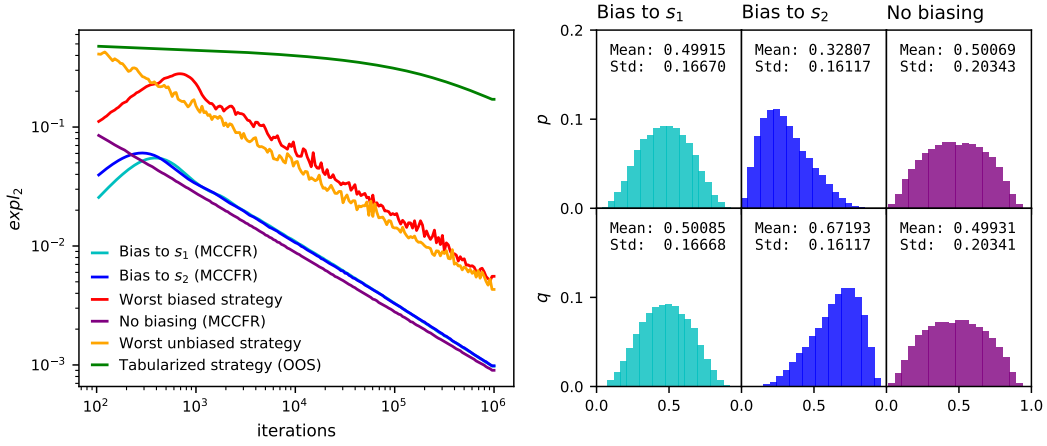


Figure B.1: Coordinated Matching Pennies. Left: While individual MCCFR strategies have low exploitability of  $\sim 10^{-3}$ , the tabularized OOS strategy has high exploitability of 0.17 after  $10^6$  iterations. Right: Normalized histograms of probabilities of playing Heads in  $s_1$  -  $p$  and  $s_2$  -  $q$  after  $10^6$  iterations. The histograms within columns are correlated, as they approximately satisfy equilibrium condition  $p + q = 1$ . Tabularized strategy, combination of  $(p, s_1)$  and  $(q, s_2)$  violates this constraint, resulting in high exploitability.

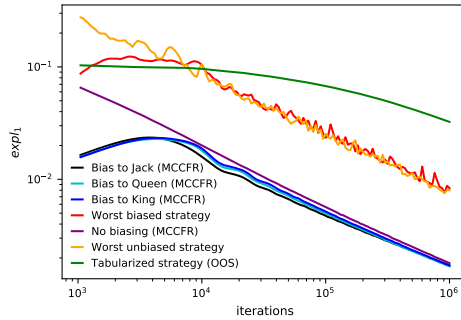


Figure B.2: Kuhn Poker

appropriately balanced, an effect of non-locality problem [Frank and Basin, 1998] present only in imperfect information games. When we compose the final strategy from partial online strategies, this balance can be lost, resulting in high exploitability of the composed strategy.

We modify the initial regrets with following procedure:

- Choose a distinct value of  $\alpha$ , one for each of the player's top-most information states in the game. Compute an equilibrium strategy according to  $\alpha$ .
- Directly copy the behavioral strategy into regret accumulators, and multiply them by a constant  $\mu$ .

This simple procedure effectively kick-starts the algorithm to produce distinct trajectories based on  $\alpha$ .

In Figure B.1 and in Figure B.2, we show that individual biased strategies converge to Nash equilibria, but the tabularized strategy has higher exploitability even than the worst individual strategy. In CMP, we bias the second player to play in information state  $s_1$  ( $\alpha = 0.5$ ) or  $s_2$  ( $\alpha = 1$ ) information states. In Poker, we bias the first player to play Jack ( $\alpha = 0$ ), Queen ( $\alpha = 1/2$ ) or King ( $\alpha = 1$ ) card. For both experiments, exploration was set to 0.6, biasing to 0.1, and  $\mu = 500$ , a small regret that can be accumulated after less than 500 samples. Within our online framework, the state  $\theta$  consists of regrets and average strategy

accumulators for all information states, and from the state of the pseudo-random number generator, which has distinct initial seeds for each match. The expected strategies are estimated as an average over  $3 \cdot 10^4$  seeds. We plot the worst strategy from these individual biased strategies over all the seeds for all iterations. We plot also MCCFR strategy for reference, to see the influence of biasing and regret initialization.

# C. Attachments for Chapter 6

## C.1 MCCFR and MCCFR+ comparison

While it is known in that MCCFR+ is outperformed by MCCFR [Burch, 2017], we are not aware on any explicit comparison of these two algorithms in literature. Fig. C.1 shows experimental evaluation of these two techniques on Leduc poker.

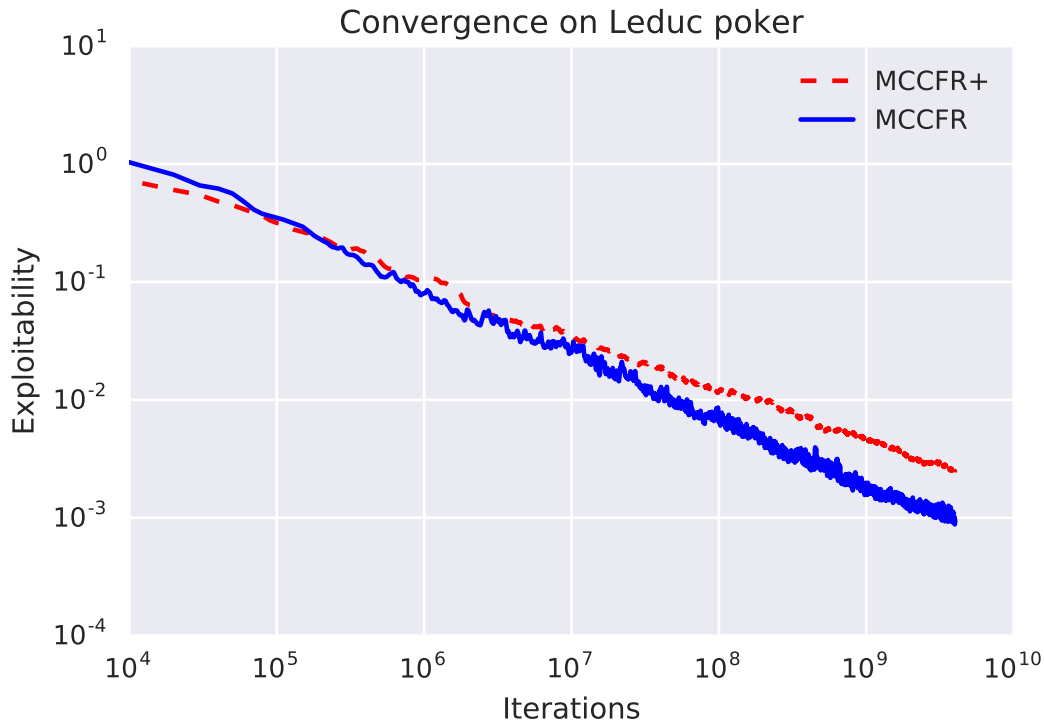


Figure C.1: Convergence of MCCFR and MCCFR+ on logarithmic scale. For the first  $10^6$  iterations, MCCFR+ performs simillary to the MCCFR. After approximately  $10^7$  iterations, the difference in favor of MCCFR starts to be visible and the gap in exploitability widens as the number of iterations grows.

## C.2 Vector Form of CFR

The first appearance of the vector form was presented in [Johanson et al., 2011]. In our work, the best response computation, needed to compute exploitability, was sped-up by re-defining the computation using the notion of a public tree. At the heart of a public tree is the notion of a **public state** which contains a set of information sets whose histories are consistent with the public information revealed so far [Johanson et al., 2011, Definition 2]. This allowed the method to compute quantities for all information sets consistent with a public state at once (stored in vectors) and operations to compute them could be vectorized during a traversal of the public tree. There are also game-specific optimizations that could be applied at leaf nodes to asymptotically reduce the total computation necessary.

A similar construction was used in several sampling variants introduced in [Johanson et al., 2012]. Here, instead of computing necessary for best response, counterfactual values were vectorized and stored instead. The paper describes several ways to sample at various types of chance nodes (ones which reveal public information, or private information to each

player), but the concept of a vectorized form of CFR was general. In fact, a vector form of vanilla CFR is possible in any game: when traversing down the tree, these vectors store the probability of reaching each information set (called a *range* in [Moravčík et al., 2017]) and return vectors of counterfactual values. Both DeepStack and Libratus used vector forms of CFR and CFR+ in No-Limit poker.

For the MCCFR variants in our work, the idea is the same as the previous sample variants. For any sequence of public actions, we concurrently maintain and update all information sets consistent with the sequence of public actions. For example in Leduc poker, six trajectories per player are maintained which all share the same sequence of public actions.

The main difference in our implementation is that baselines are kept as vectors at each public state, each representing a baseline for the information sets corresponding to the public state. Also, the average values tracked are counterfactual and normalized by the range. So, for example in Leduc, for five information sets in some public state,  $(I_1, I_2, \dots, I_5)$ , quantity tracked by the baseline at this public state for action  $a$  is:

$$\frac{\hat{v}_i^b(\sigma, I_k, a)}{\sum_{k'} \pi_{opp}^\sigma(I_{k'}^{opp})},$$

where  $\pi_{opp}^\sigma$  is the reach probability of the opponent only (excluding chance), and  $I^{opp}$  refers to the augmented information set belonging to the opponent at  $I$ . Then, when using the baseline values to compute the modified counterfactual values, we need to multiply them by the current  $\sum_{k'} \pi_{opp}^\sigma(I_{k'}^{opp})$  to get the baseline values under the current strategy  $\sigma$ .

## C.3 Proofs

### C.3.1 Proof of Lemma 6.1

$$\begin{aligned} \mathbb{E}[\hat{v}^b(\sigma, I, a)] &= \mathbb{E}[\hat{v}_i(\sigma, I, a)] - \mathbb{E}[\hat{b}_i(I, a)] + \mathbb{E}[b_i(I, a)] \\ &= v_i(\sigma, I, a) - b_i(I, a) + b_i(I, a) \\ &= v_i(\sigma, I, a). \end{aligned} \quad \square$$

### C.3.2 Proof of Lemma 6.2

We begin by proving a few supporting lemmas regarding local expectations over actions at specific histories:

**Lemma C.1.** *Given some  $h \in H$ , for any  $z \in Z$  generated by sampling  $\xi : H \mapsto A$  and all actions  $a$ ,  $\mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)] = \sum_{z, ha \sqsubseteq z} q(z) \hat{u}_i^b(\sigma, ha|z) / \xi(h, a)$ :*

*Proof.*  $\hat{u}_i^b(\sigma, h, a|z)$  has three cases, from which we get  $\mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)]$

$$\begin{aligned}
&= \sum_{z, ha \sqsubseteq z} q(z) \\
&\quad \left( b_i(I_i(h), a) + \frac{-b_i(I_i(h), a) + \hat{u}_i^b(\sigma, ha|z)}{\xi(h, a)} \right) \\
&+ \sum_{z, h \sqsubset z, ha \not\sqsubseteq z} q(z)(b_i(I_i(h), a)) \\
&+ \sum_{h \not\sqsubseteq z} 0 \\
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i^b(\sigma, ha|z)/\xi(h, a) \\
&+ (q(ha) - q(ha)/\xi(h, a))b_i(I_i(h), a) \\
&+ q(h)(1 - \xi(h, a))b_i(I_i(h), a) \\
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i^b(\sigma, ha|z)/\xi(h, a)
\end{aligned}$$

□

**Lemma C.2.** *Given some  $h \in H$ , for any  $z \in Z$  generated by sampling  $\xi : H \mapsto A$ , the local baseline-enhanced estimate is an unbiased estimate of expected values for all actions  $a$ :*

$$\mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)] = \mathbb{E}_{z \sim \xi}[\hat{u}_i(\sigma, h, a|z)].$$

*Proof.* We prove this by induction on the maximum distance from  $ha$  to any terminal. The base case is  $ha \in Z$ .  $\mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)]$

$$\begin{aligned}
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i^b(\sigma, ha|z)/\xi(h, a) \quad \text{by Lemma C.1} \\
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i(\sigma, ha|z)/\xi(h, a) \quad \text{by Eq. 6.10} \\
&= \mathbb{E}_{z \sim \xi}[\hat{u}_i(\sigma, h, a|z)] \quad \text{by Eq. 6.7, 6.8}
\end{aligned}$$

Now assume for  $i \geq 0$  that the lemma property holds for all  $h'a'$  that are at most  $j \leq i$  steps from a terminal. Consider history  $ha$  being  $i + 1$  steps from some terminal, which implies that  $ha \notin Z$ . We have  $\mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)]$

$$\begin{aligned}
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i^b(\sigma, ha|z)/\xi(h, a) \quad \text{by Lemma C.1} \\
&= \sum_{z, ha \sqsubseteq z} q(z) \sum_{a'} \sigma(ha, a')\hat{u}_i^b(\sigma, ha, a'|z)/\xi(h, a) \\
&\quad \text{by Eq. 6.10} \\
&= \sum_{z, ha \sqsubseteq z} q(z) \sum_{a'} \sigma(ha, a')\hat{u}_i(\sigma, ha, a'|z)/\xi(h, a) \\
&\quad \text{by assumption} \\
&= \sum_{z, ha \sqsubseteq z} q(z)\hat{u}_i(\sigma, ha|z)/\xi(h, a) \quad \text{by Eq. 6.8} \\
&= \mathbb{E}_{z \sim \xi}[\hat{u}_i(\sigma, h, a|z)] \quad \text{by Eq. 6.7}
\end{aligned}$$

The lemma property holds for distance  $i + 1$ , and so by induction the property holds for all  $h$  and  $a$ . □

**Lemma C.3.** Given some  $h \in H$ , for any  $z \in Z$  generated by sampling  $\xi : H \mapsto A$  and for all actions  $a$ , the local baseline-enhanced estimate is an unbiased estimate of the original sampled counterfactual value:  $\mathbb{E}_{z \sim \xi}[\hat{v}_i^b(\sigma, I_i(h), a|z)] = \mathbb{E}_{z \sim \xi}[\tilde{v}_i(\sigma, I_i(h), a|z)]$ .

*Proof.* First,  $\mathbb{E}_{z \sim \xi}[\hat{v}_i^b(\sigma, I_i(h), a|z)]$

$$\begin{aligned}
&= \mathbb{E}_{z \sim \xi} \left[ \frac{\pi_{-i}^\sigma(h)}{q(h)} \hat{u}_i^b(\sigma, h, a|z) \right] \quad \text{by Eq. 6.11} \\
&= \frac{\pi_{-i}^\sigma(h)}{q(h)} \mathbb{E}_{z \sim \xi}[\hat{u}_i^b(\sigma, h, a|z)] \\
&= \frac{\pi_{-i}^\sigma(h)}{q(h)} \mathbb{E}_{z \sim \xi}[\hat{u}_i(\sigma, h, a|z)] \quad \text{by Lemma C.2} \\
&= \mathbb{E}_{z \sim \xi}[\tilde{v}_i(\sigma, I_i(h), a|z)] \quad \text{by Eq. 6.5, 6.7.}
\end{aligned}$$

□

*Proof of Lemma 2.* The proof now follows directly:

$$\mathbb{E}_{z \sim \xi}[\hat{v}_i^b(\sigma, I, a|z)]$$

$$\begin{aligned}
&= \mathbb{E}_{z \sim \xi}[\tilde{v}_i(\sigma, I, a|z)] \quad \text{by Lemma C.3} \\
&= v_i(\sigma, I, a) \quad \text{by [Lanctot et al., 2009, Lemma 1].}
\end{aligned}$$

□

### C.3.3 Proof of Lemma 6.3

We start by proving that given an oracle baseline, the baseline-enhanced expected value is always equal to the true expected value, and therefore has zero variance.

**Lemma C.4.** Using an oracle baseline defined over histories,  $b_i^*(h, a) = u_i^\sigma(ha)$ , then for all  $z$  such that  $h \sqsubseteq z$ ,  $\hat{u}_i^{b^*}(\sigma, h, a|z) = u_i^\sigma(ha)$ .

*Proof.* Similar to above, we prove this by induction on the maximum distance from  $ha$  to  $z$ . The base case is  $ha \in Z$ . By assumption  $h \sqsubseteq z$  so we have  $\hat{u}_i^{b^*}(\sigma, h, a|z)$

$$\begin{aligned}
&= \begin{cases} b_i^*(h, a) + \frac{\hat{u}_i^{b^*}(\sigma, ha|z) - b_i^*(h, a)}{\xi(h, a)} & \text{if } ha = z \\ b_i^*(h, a) & \text{otherwise} \end{cases} \\
&\quad \text{by Eq. 6.9} \\
&= \begin{cases} u_i^\sigma(ha) + \frac{u_i^\sigma(ha) - u_i^\sigma(ha)}{\xi(h, a)} & \text{if } ha = z \\ u_i^\sigma(ha) & \text{otherwise} \end{cases} \\
&\quad \text{by Eq. 6.10 and definition of } b_i^*(h, a) \\
&= u_i^\sigma(ha)
\end{aligned}$$

Now assume for  $i \geq 0$  that the lemma property holds for all  $h'a'$  that are at most  $j \leq i$  steps from a terminal. Consider history  $ha$  being  $i + 1$  steps from some terminal, which implies  $ha \notin Z$ . We have

$$\hat{u}_i^{b^*}(\sigma, ha|z) = u_i^\sigma(ha) \tag{C.1}$$



because  $\hat{u}_i^{b^*}(\sigma, ha|z)$

$$\begin{aligned}
&= \sum_{a'} \sigma(ha, a') \hat{u}_i^{b^*}(\sigma, ha, a'|z) \quad \text{by Eq. 6.10} \\
&= \sum_{a'} \sigma(ha, a') u_i^\sigma(haa') \quad \text{by assumption} \\
&= u_i^\sigma(ha) \quad \text{by definition of } u_i^\sigma
\end{aligned}$$

We now look at  $\hat{u}_i^{b^*}(\sigma, h, a|z)$

$$\begin{aligned}
&= \begin{cases} u_i^\sigma(ha) + \frac{\hat{u}_i^{b^*}(\sigma, ha|z) - u_i^\sigma(ha)}{\xi(h, a)} & \text{if } ha \sqsubseteq z \\ u_i^\sigma(ha) & \text{otherwise} \end{cases} \\
&\quad \text{by Eq. 6.9 and definition of } b_i^*(h, a) \\
&= \begin{cases} u_i^\sigma(ha) + \frac{u_i^\sigma(ha) - u_i^\sigma(ha)}{\xi(h, a)} & \text{if } ha \sqsubseteq z \\ u_i^\sigma(ha) & \text{otherwise} \end{cases} \\
&\quad \text{by Eq. C.1} \\
&= u_i^\sigma(ha)
\end{aligned}$$

The lemma property holds for distance  $i + 1$ , and so by induction the property holds for all  $h$  and  $a$ .  $\square$

*Proof of Lemma 6.3.* Given  $z$  such that  $h \sqsubseteq z$ , we have  $\hat{v}_i^*(\sigma, h, a|z)$

$$\begin{aligned}
&= \frac{\pi_{-i}^\sigma(h)}{q(h)} \hat{u}_i^{b^*}(\sigma, h, a|z) \quad \text{by Eq. 6.11} \\
&= \frac{\pi_{-i}^\sigma(h)}{q(h)} u_i^\sigma(ha) \quad \text{by Lemma C.4}
\end{aligned}$$

None of the terms above depend on  $z$ , and so we have  $\text{Var}_{h, z \sim \xi, h \in I, h \sqsubseteq z}[\hat{v}_i^*(\sigma, h, a|z)] = 0$ . Note as well that  $\pi_{-i}^\sigma(h) u_i^\sigma(ha)$  corresponds to the terms in the summation of Equation 6.2, so abusing notation, we have  $\hat{v}_i^*(\sigma, h, a|z) = v_i(\sigma, h, a)/q(h)$ : the counterfactual value of taking action  $a$  at  $h$ , with an importance sampling weight to correct for the likelihood of reaching  $h$ .  $\square$

In MCCFR, the optimal baseline  $b^*$  is not known, as it would require traversing the entire tree, taking away any advantages of sampling. However,  $b^*$  can be approximated (learned online), which motivates the choice for tracking its average value presented in the main chapter.

## C.4 Kuhn Example

In this section, we present a step-by-step example of one iteration of the algorithm on Kuhn poker [poker, 2018]. Kuhn poker is a simplified version of poker with three cards and is therefore suitable for demonstration purposes. Table C.1 show forward pass of VR-MCCFR algorithm, Table C.2 shows backward pass.


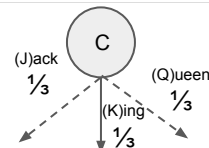
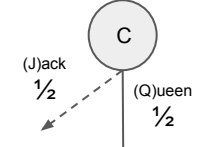
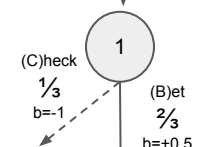
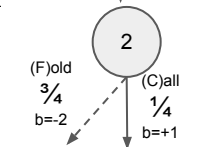
		Forward pass				
$h$		Game tree trajectory	$\pi_{-1}^\sigma(h)$	$q(h)$	$I_1 = I_1(h)$	$I_2 = I_2(h)$
History			Reach prob.	Sampling prob.	Infoset for P11	Infoset for P12
	$\emptyset$		1	1	$\emptyset$	$\emptyset$
	K		$\frac{1}{3}$	$\frac{1}{3}$	K	?
	KQ		$\frac{1}{6}$	$\frac{1}{6}$	K?	?Q
	KQB		$\frac{1}{6}$	$\frac{1}{12}$	K?B	?QB
	KQBC	+2	$\frac{1}{24}$	$\frac{1}{24}$	K?BC	?QBC

Table C.1: Detailed example of updates computed for player 1 in Kuhn poker during forward pass of the algorithm. Backward pass that uses these values is shown in Table C.2. In our representation history  $h$  is a concatenation of all public and private actions. The game tree trajectory column shows the path in the game tree that was sampled. Solid arrows denote sampled actions while dashed arrows show other available actions, all actions have their probability under current strategy  $\sigma$  next to them. The sampled history in this case is: chance deals (K)ing to player 1, chance deals (Q)ueen to player 2, player 1 (B)ets, player 2 (C)alls. We will use shorter notation  $KQBC$  to refer to this history. For each history  $h$  reach probability  $\pi_{-1}^\sigma(h)$  shows how likely the history is reached when player 1 plays in a way to get to this history. The sampling probabilities  $q(h)$  are computed following sampling policy  $\xi$  which is uniform in this case, i.e. for each history all available actions have the same probability that they will be sampled. The last two columns show augmented information sets for each player in each history. For example for player 1 history KQB is represented by information set K?B since he does not know what card was dealt to PLAYER 2. Light gray background marks cells where the values are well defined however they are not used in our example update for player 1.

		Backward pass			
	$h$ History	Game tree trajectory	$\hat{u}_1^b(\sigma, h, a z)$ Sampled corrected history-action utility	$\hat{u}_1^b(\sigma, h z)$ Sampled corrected history utility	$\hat{v}_1^b(\sigma, I_1, a z)$ Sampled corrected cf-value
Def.			Eq. 6.9	Eq. 6.10	Eq. 6.11
↑ Backward pass	$\emptyset$				
	K				
	KQ		$\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h}, \mathbf{B} z) = \frac{\hat{u}_1^b(\sigma, h, B z) - b(I_1, B)}{\xi(h, B)} + b(I_1, B)$ $= \frac{-\frac{3}{4} - 0.5}{\frac{1}{2}} + 0.5$ $= -2$ $\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h}, \mathbf{C} z) = b(I_1, C)$ $= -1$	$\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h} z) = \sum_a \sigma(h, a) \hat{u}_1^b(\sigma, h, a z)$ $= \frac{1}{3} * (-1) + \frac{2}{3} * (-2)$ $= -\frac{5}{3}$	$\hat{v}_1^b(\sigma, \mathbf{I}_1, \mathbf{B} z) = \frac{\pi_{-1}^{\sigma}(h)}{q(h)} \hat{u}_1^b(\sigma, h, B z)$ $= \frac{1}{\frac{1}{6}} * (-2)$ $= -2$ $\hat{v}_1^b(\sigma, \mathbf{I}_1, \mathbf{C} z) = \frac{\pi_{-1}^{\sigma}(h)}{q(h)} \hat{u}_1^b(\sigma, h, C z)$ $= \frac{1}{\frac{1}{6}} * (-1)$ $= -1$
	KQB		$\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h}, \mathbf{C} z) = \frac{\hat{u}_1^b(\sigma, h, C z) - b(I_1, C)}{\xi(h, C)} + b(I_1, C)$ $= \frac{2-1}{\frac{1}{2}} + 1$ $= 3$ $\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h}, \mathbf{F} z) = b(I_1, F)$ $= -2$	$\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h} z) = \sum_a \sigma(h, a) \hat{u}_1^b(\sigma, h, a z)$ $= \frac{3}{4} * (-2) + \frac{1}{4} * 3$ $= -\frac{3}{4}$	
	KQBC				$\hat{\mathbf{u}}_1^b(\sigma, \mathbf{h},  z) = u_1(h)$ $= 2$

Table C.2: The backward pass starts by evaluating utility of the terminal history:  $\hat{u}_1^b(\sigma, KQBC|KQBC) = +2$  since player 1 has (K)ing which is better card than opponent's (Q)ueen. In the next step computation updates values for history  $KQB$ . Expected baseline corrected history-action value  $\hat{u}_1^b(\sigma, KQB, Call|KQBC)$  is computed based on current sample and then used together with  $\hat{u}_1^b(\sigma, KQB, Fold|KQBC)$  to compute  $\hat{u}_1^b(\sigma, KQB|KQBC)$ . When updating values for history  $KQ$  baseline corrected sampled counterfactual values are computed based on just updated  $\hat{u}_1^b(\sigma, KQ, Bet|KQBC)$  for the sampled Bet action and on a baseline value  $\hat{u}_1^b(\sigma, KQ, Check|KQBC)$  for Check action that was not sampled. Reach probability  $\pi_{-1}^{\sigma}(KQ)$  and sampling probability  $q(KQ)$  that are also needed to compute counterfactual-values  $\hat{v}_1^b(\sigma, K?, a|KQBC)$  were already computed in the forward pass. The counterfactual values are then used to compute actions' regrets (Eq. 6.3) which is not shown in the table. Values in cell with light gray background are not used in computation of  $\hat{v}_1^b(\sigma, K?, a|KQBC)$ .

# D. Attachments for Chapter 7

**Theorem D.1.** *Given a strategy  $\sigma_1$ , a subgame  $S$  and a refined subgame strategy  $\sigma_1^S$ , let  $\sigma_1' = \sigma_1[S \leftarrow \sigma_1^S]$  be a combined strategy of  $\sigma_1$  and  $\sigma_1^S$ . Let the subgame margin  $SM_1(\sigma_1, \sigma_1', S)$  be non-negative. Then  $u_1(\sigma_1', CBR(\sigma_1')) - u_1(\sigma_1, CBR(\sigma_1)) \geq 0$ . Furthermore, if there's a best response strategy  $\sigma_2^* = BR(\sigma_1')$  such that  $\pi^{(\sigma_1', \sigma_2^*)}(I_2) > 0$  for some  $I_2 \in \mathcal{I}_2^{R(S)}$ , then  $u_1(\sigma_1', CBR(\sigma_1')) - u_1(\sigma_1, CBR(\sigma_1)) \geq \pi_{-2}^{\sigma_1'}(I_2) SM_1(\sigma_1, \sigma_1', S)$ .*

See [Burch et al., 2014] for proof of the following inequity:  $u_1(\sigma_1', CBR(\sigma_1')) - u_1(\sigma_1, CBR(\sigma_1)) \geq 0$ . Directly from the proof also, we get that  $CBV_2^{\sigma_1'}(I) \leq CBV_2^{\sigma_1}(I)$  for any information set  $I \not\subseteq S \setminus R(S)$

Now, we will prove that lower bound for improvement holds in case when there exists a  $\sigma_2^* = BR(\sigma_1')$  such that  $\pi^{(\sigma_1', \sigma_2^*)}(I_2) > 0$  for some  $I_2 \in \mathcal{I}_2^{R(S)}$ . Without loss of generality we can assume that  $\sigma_2^* = CBR_2(\sigma_1')$  (since we can just change strategy in information sets  $I'$  where  $\pi_2^{\sigma_2^*}(I') = 0$ ) and that  $\sigma_2^*(I_2) = 1$  (since we can choose any action from best response support with probability 1).

First, we show that if  $p(I) = 2$  and  $I$  lays on a path from  $\emptyset$  (root of the whole game) to  $I_2$ , then  $CBV_2^{\sigma_1'}(I) \leq CBV_2^{\sigma_1}(I) - \pi_{-2}^{\sigma_1'}(I \rightarrow I_2) SM_1(\sigma_1, \sigma_1', S)$ .

We will use induction on length (measured in count of  $p_2$  information sets) of the path ( $I \rightarrow I_2$ ).

We can see (directly from the definition of the subgame margin) that the claim holds for  $I_2$ .

Lets take  $I \neq I_2$  form the path and denote  $I'$  next  $p_2$  information set on the path. Then, for an action  $a$  leading to the  $I_2$  we have  $\pi^{\sigma_2^*}(I, a) = 1$ ,  $\pi_2^{(\sigma_1', \sigma_2^*)}(I \rightarrow I') = 1$  and we can express counterfactual best response value in  $I$  as:

$$\begin{aligned} CBV_2^{\sigma_1'}(I) &= v_2^{(\sigma_1', \sigma_2^*)}(I, a) \leq \\ &v_2^{(\sigma_1, CBR_2(\sigma_1))}(I, a) - \pi^{\sigma_1', \sigma_2^*}(I \rightarrow I') \pi_{-2}^{\sigma_1'}(I' \rightarrow I_2) SM_1(\sigma_1, \sigma_1', S) = \\ &v_2^{(\sigma_1, CBR_2(\sigma_1))}(I, a) - \pi_{-2}^{\sigma_1'}(I \rightarrow I') \pi_{-2}^{\sigma_1'}(I' \rightarrow I_2) SM_1(\sigma_1, \sigma_1', S) = \\ &v_2^{(\sigma_1, CBR_2(\sigma_1))}(I, a) - \pi_{-2}^{\sigma_1'}(I \rightarrow I_2) SM_1(\sigma_1, \sigma_1', S) \leq \max_{a'} v_2^{(\sigma_1, CBR_2(\sigma_1))}(I, a') - \pi_{-2}^{\sigma_1'}(I \rightarrow \\ &I_2) SM_1(\sigma_1, \sigma_1', S) = \\ &CBV_2(\sigma_1)(I) - \pi_{-2}^{\sigma_1'}(I \rightarrow I_2) SM_1(\sigma_1, \sigma_1', S) \end{aligned}$$

If  $P(\emptyset) = 2$  then  $u_1(\sigma_1', CBR(\sigma_1')) = -CBV_2(\sigma_1')(\emptyset)$ ,  $u_1(\sigma_1, CBR(\sigma_1)) = -CBV_2(\sigma_1)(\emptyset)$  and the theorem holds. If this is not the case then we can simply add a new player's 2 information set at the beginning of the game where the player 2 has only one action leading to  $\emptyset$ .

# E. Attachments for Chapter 9

## E.1 Game of Heads-Up No-Limit Texas Hold'em

Heads-up no-limit Texas hold'em (HUNL) is a two-player poker game. It is a repeated game, in which the two players play a match of individual games, usually called hands, while alternating who is the dealer. In each of the individual games, one player will win some number of chips from the other player, and the goal is to win as many chips as possible over the course of the match.

Each individual game begins with both players placing a number of chips in the pot: the player in the dealer position puts in the small blind, and the other player puts in the big blind, which is twice the small blind amount. During a game, a player can only wager and win up to a fixed amount known as their stack. In the particular format of HUNL used in the Annual Computer Poker Competition [Zinkevich and Littman, 2006] and this article, the big blind is 100 chips and the stack is 20,000 chips or 200 big blinds. Resetting the stacks after each game is called “Doyle’s Game”, named for the professional poker player Doyle Brunson who publicized this variant [Gilpin et al., 2008]. It is used in the Annual Computer Poker Competitions because it allows for each game to be an independent sample of the same game.

A game of HUNL progresses through four rounds: the pre-flop, flop, turn, and river. Each round consists of cards being dealt followed by player actions in the form of wagers as to who will hold the strongest hand at the end of the game. In the pre-flop, each player is given two private cards, unobserved by their opponent. In the later rounds, cards are dealt face-up in the center of the table, called public cards. A total of five public cards are revealed over the four rounds: three on the flop, one on the turn, and one on the river.

After the cards for the round are dealt, players alternate taking actions of three types: fold, call, or raise. A player folds by declining to match the last opponent wager, thus forfeiting to the opponent all chips in the pot and ending the game with no player revealing their private cards. A player calls by adding chips into the pot to match the last opponent wager, which causes the next round to begin. A player raises by adding chips into the pot to match the last wager followed by adding additional chips to make a wager of their own. At the beginning of a round when there is no opponent wager yet to match, the raise action is called bet, and the call action is called check, which only ends the round if both players check. An all-in wager is one involving all of the chips remaining the player’s stack. If the wager is called, there is no further wagering in later rounds. The size of any other wager can be any whole number of chips remaining in the player’s stack, as long as it is not smaller than the last wager in the current round or the big blind.

The dealer acts first in the pre-flop round and must decide whether to fold, call, or raise the opponent’s big blind bet. In all subsequent rounds, the non-dealer acts first. If the river round ends with no player previously folding to end the game, the outcome is determined by a showdown. Each player reveals their two private cards and the player that can form the strongest five-card poker hand (see “List of poker hand categories” on Wikipedia; accessed January 1, 2017) wins all the chips in the pot. To form their hand each player may use any cards from their two private cards and the five public cards. At the end of the game, whether ended by fold or showdown, the players will swap who is the dealer and begin the next game.

Since the game can be played for different stakes, such as a big blind being worth \$0.01 or \$1 or \$1000, players commonly measure their performance over a match as their

average number of big blinds won per game. Researchers have standardized on the unit milli-big-blinds per game, or mbb/g, where one milli-big-blind is one thousandth of one big blind. A player that always folds will lose 750 mbb/g (by losing 1000 mbb as the big blind and 500 as the small blind). A human rule-of-thumb is that a professional should aim to win at least 50 mbb/g from their opponents. Milli-big-blinds per game is also used as a unit of exploitability, when it is computed as the expected loss per game against a worst-case opponent. In the poker community, it is common to use big blinds per one hundred games (bb/100) to measure win rates, where 10 mbb/g equals 1 bb/100.

## E.2 Poker Glossary

**all-in** A wager of the remainder of a player's stack. The opponent's only response can be call or fold.

**bet** The first wager in a round; putting more chips into the pot.

**big blind** Initial wager made by the non-dealer before any cards are dealt. The big blind is twice the size of the small blind.

**call** Putting enough chips into the pot to match the current wager; ends the round.

**check** Declining to wager any chips when not facing a bet.

**chip** Marker representing value used for wagers; all wagers must be a whole numbers of chips.

**dealer** The player who puts the small blind into the pot. Acts first on round 1, and second on the later rounds. Traditionally, they would distribute public and private cards from the deck.

**flop** The second round; can refer to either the 3 revealed public cards, or the betting round after these cards are revealed.

**fold** Give up on the current game, forfeiting all wagers placed in the pot. Ends a player's participation in the game.

**hand** Many different meanings: the combination of the best 5 cards from the public cards and private cards, just the private cards themselves, or a single game of poker (for clarity, we avoid this final meaning).

**milli-big-blinds per game (mbb/g)** Average winning rate over a number of games, measured in thousandths of big blinds.

**pot** The collected chips from all wagers.

**pre-flop** The first round; can refer to either the hole cards, or the betting round after these cards are distributed.

**private cards** Cards dealt face down, visible only to one player. Used in combination with public cards to create a hand. Also called hole cards.

**public cards** Cards dealt face up, visible to all players. Used in combination with private cards to create a hand. Also called community cards.

**raise** Increasing the size of a wager in a round, putting more chips into the pot than is required to call the current bet.

**river** The fourth and final round; can refer to either the 1 revealed public card, or the betting round after this card is revealed.

**showdown** After the river, players who have not folded show their private cards to determine the player with the best hand. The player with the best hand takes all of the chips in the pot.

**small blind** Initial wager made by the dealer before any cards are dealt. The small blind is half the size of the big blind.

**stack** The maximum amount of chips a player can wager or win in a single game.

**turn** The third round; can refer to either the 1 revealed public card, or the betting round after this card is revealed.

### **E.3 Performance Against Professional Players**

To assess DeepStack relative to expert humans, players were recruited with assistance from the International Federation of Poker to identify and recruit professional poker players through their member nation organizations. We only selected participants from those who self-identified as a “professional poker player” during registration. Players were given four weeks to complete a 3,000 game match. To incentivize players, monetary prizes of \$5,000, \$2,500, and \$1,250 (CAD) were awarded to the top three players (measured by AIVAT) that completed their match. The participants were informed of all of these details when they registered to participate. Matches were played between November 7th and December 12th, 2016, and run using an online user interface [Morrill, 2012] where players had the option to play up to four games simultaneously as is common in online poker sites. A total of 33 players from 17 countries played against DeepStack. DeepStack’s performance against each individual is presented in Table E.1, with complete game histories available as part of the supplementary online materials.

### **E.4 Local Best Response of DeepStack**

The goal of DeepStack, and much of the work on AI in poker, is to approximate a Nash equilibrium, i.e., produce a strategy with low exploitability. The size of HUNL makes an explicit best-response computation intractable and so exact exploitability cannot be measured. A common alternative is to play two strategies against each other. However, head-to-head performance in imperfect information games has repeatedly been shown to be a poor estimation of equilibrium approximation quality. For example, consider an exact Nash equilibrium strategy in the game of Rock-Paper-Scissors playing against a strategy that almost always plays “rock”. The results are a tie, but their playing strengths in terms of exploitability are vastly different. This same issue has been seen in heads-up limit Texas hold’em as well (Johanson, IJCAI 2011), where the relationship between head-to-head play and exploitability, which is tractable in that game, is indiscernible. The introduction of local best response (LBR) as a technique for finding a lower-bound on a strategy’s exploitability gives evidence of the same issue existing in HUNL. Act1 and Slumbot (second and third place in the previous ACPC) were statistically indistinguishable in head-to-head play (within

Table E.1: Results against professional poker players estimated with AIVAT (Luck Adjusted Win Rate) and chips won (Unadjusted Win Rate), both measured in mbb/g. Recall 10mbb/g equals 1bb/100. Each estimate is followed by a 95% confidence interval. ‡ marks a participant who completed the 3000 games after their allotted four week period.

Player		Rank	Hands	Luck Adjusted Win Rate	Unadjusted Win Rate
Martin Sturc		1	3000	70 ± 119	-515 ± 575
Stanislav Voloshin		2	3000	126 ± 103	-65 ± 648
Prakshat Shrimankar		3	3000	139 ± 97	174 ± 667
Ivan Shabalin		4	3000	170 ± 99	153 ± 633
Lucas Schaumann		5	3000	207 ± 87	160 ± 576
Phil Laak		6	3000	212 ± 143	774 ± 677
Kaishi Sun		7	3000	363 ± 116	5 ± 729
Dmitry Lesnoy		8	3000	411 ± 138	-87 ± 753
Antonio Parlavecchio		9	3000	618 ± 212	1096 ± 962
Muskan Sethi		10	3000	1009 ± 184	2144 ± 1019
Pol Dmit <sup>‡</sup>		-	3000	1008 ± 156	883 ± 793
Tsuneaki Takeda		-	1901	628 ± 231	-332 ± 1228
Youwei Qin		-	1759	1311 ± 331	1958 ± 1799
Fintan Gavin		-	1555	635 ± 278	-26 ± 1647
Giedrius Talacka		-	1514	1063 ± 338	459 ± 1707
Juergen Bachmann		-	1088	527 ± 198	1769 ± 1662
Sergey Indenok		-	852	881 ± 371	253 ± 2507
Sebastian Schwab		-	516	1086 ± 598	1800 ± 2162
Dara O’Kearney		-	456	78 ± 250	223 ± 1688
Roman Shaposhnikov		-	330	131 ± 305	-898 ± 2153
Shai Zurr		-	330	499 ± 360	1154 ± 2206
Luca Moschitta		-	328	444 ± 580	1438 ± 2388
Stas Tishekevich		-	295	-45 ± 433	-346 ± 2264
Eyal Eshkar		-	191	18 ± 608	715 ± 4227
Jefri Islam		-	176	997 ± 700	3822 ± 4834
Fan Sun		-	122	531 ± 774	-1291 ± 5456
Igor Naumenko		-	102	-137 ± 638	851 ± 1536
Silvio Pizzarello		-	90	1500 ± 2100	5134 ± 6766
Gaia Freire		-	76	369 ± 136	138 ± 694
Alexander Bös		-	74	487 ± 756	1 ± 2628
Victor Santos		-	58	475 ± 462	-1759 ± 2571
Mike Phan		-	32	-1019 ± 2352	-11223 ± 18235
Juan Manuel Pastor		-	7	2744 ± 3521	7286 ± 9856
Human Professionals			44852	486 ± 40	492 ± 220



Table E.2: Exploitability lower bound of different programs using local best response (LBR). LBR evaluates only the listed actions in each round as shown in each row. F, C, P, A, refer to fold, call, a pot-sized bet, and all-in, respectively. 56bets includes the actions fold, call and 56 equidistant pot fractions as defined in the original LBR paper [Lisý and Bowling, 2017a]. ‡: Always Fold checks when not facing a bet, and so it cannot be maximally exploited without a betting action.

		Local best response performance (mbb/g)			
LBR Settings	Pre-flop	F, C	C	C	C
	Flop	F, C	C	C	56bets
	Turn	F, C	F, C, P, A	56bets	F, C
	River	F, C	F, C, P, A	56bets	F, C
Hyperborean (2014)		721 ± 56	3852 ± 141	4675 ± 152	983 ± 95
Slumbot (2016)		522 ± 50	4020 ± 115	3763 ± 104	1227 ± 79
Act1 (2016)		407 ± 47	2597 ± 140	3302 ± 122	847 ± 78
Always Fold		‡250 ± 0	750 ± 0	750 ± 0	750 ± 0
Full Cards [100 BB]		-424 ± 37	-536 ± 87	2403 ± 87	1008 ± 68
DeepStack		-428 ± 87	-383 ± 219	-775 ± 255	-602 ± 214

20 mbb/g), but Act1 is 1300mbb/g less exploitable as measured by LBR. This is why we use LBR to evaluate DeepStack.

LBR is a simple, yet powerful, technique to produce a lower bound on a strategy’s exploitability in HUNL [Lisý and Bowling, 2017a]. It explores a fixed set of options to find a “locally” good action against the strategy. While it seems natural that more options would be better, this is not always true. More options may cause it to find a locally good action that misses out on a future opportunity to exploit an even larger flaw in the opponent. In fact, LBR sometimes results in larger lower bounds when not considering any bets in the early rounds, so as to increase the size of the pot and thus the magnitude of a strategy’s future mistakes. LBR was recently used to show that abstraction-based agents are significantly exploitable (see Table E.2). The first three strategies are submissions from recent Annual Computer Poker Competitions. They all use both card and action abstraction and were found to be even more exploitable than simply folding every game in all tested cases. The strategy “Full Cards” does not use any card abstraction, but uses only the sparse fold, call, pot-sized bet, all-in betting abstraction using hard translation [Schnizlein et al., 2009]. Due to computation and memory requirements, we computed this strategy only for a smaller stack of 100 big blinds. Still, this strategy takes almost 2TB of memory and required approximately 14 CPU years to solve. Naturally, it cannot be exploited by LBR within the betting abstraction, but it is heavily exploitable in settings using other betting actions that require it to translate its opponent’s actions, again losing more than if it folded every game.

As for DeepStack, under all tested settings of LBR’s available actions, it fails to find any exploitable flaw. In fact, it is losing 350 mbb/g or more to DeepStack. Of particular interest is the final column aimed to exploit DeepStack’s flop strategy. The flop is where DeepStack is most dependent on its counterfactual value networks to provide it estimates through the end of the game. While these experiments do not prove that DeepStack is flawless, it does suggest its flaws require a more sophisticated search procedure than what is needed to exploit abstraction-based programs.

Table E.3: Lookahead re-solving specifics by round. The abbreviations of F, C,  $\frac{1}{2}P$ , P, 2P, and A refer to fold, call, half of a pot-sized bet, a pot-sized bet, twice a pot-sized bet, and all in, respectively. The final column specifies which neural network was used when the depth limit was exceeded: the flop, turn, or the auxiliary network.

Round	CFR Iterations	Omitted Iterations	First Action	Second Action	Remaining Actions	NN Eval
Pre-flop	1000	980	F, C, $\frac{1}{2}P$ , P, A	F, C, $\frac{1}{2}P$ , P, 2P, A	F, C, P, A	Aux/Flop
Flop	1000	500	F, C, $\frac{1}{2}P$ , P, A	F, C, P, A	F, C, P, A	Turn
Turn	1000	500	F, C, $\frac{1}{2}P$ , P, A	F, C, P, A	F, C, P, A	—
River	2000	1000	F, C, $\frac{1}{2}P$ , P, 2P, A	F, C, $\frac{1}{2}P$ , P, 2P, A	F, C, P, A	—

## E.5 DeepStack Implementation Details

Here we describe the specifics for how DeepStack employs continual re-solving and how its deep counterfactual value networks were trained.

### E.5.1 Continual Re-Solving

As with traditional re-solving, the re-solving step of the DeepStack algorithm solves an augmented game. The augmented game is designed to produce a strategy for the player such that the bounds for the opponent’s counterfactual values are satisfied. DeepStack uses a modification of the original CFR-D gadget [Burch et al., 2014] for its augmented game, as discussed below. While the max-margin gadget [Moravčík et al., 2016] is designed to improve the performance of poor strategies for abstracted agents near the end of the game, the CFR-D gadget performed better in early testing.

The algorithm DeepStack uses to solve the augmented game is a hybrid of vanilla CFR [Zinkevich et al., 2007] and CFR<sup>+</sup> [Tammelin et al., 2015], which uses regret matching<sup>+</sup> like CFR<sup>+</sup>, but does uniform weighting and simultaneous updates like vanilla CFR. When computing the final average strategy and average counterfactual values, we omit the early iterations of CFR in the averages.

A major design goal for DeepStack’s implementation was to typically play at least as fast as a human would using commodity hardware and a single GPU. The degree of lookahead tree sparsity and the number of re-solving iterations are the principle decisions that we tuned to achieve this goal. These properties were chosen separately for each round to achieve a consistent speed on each round. Note that DeepStack has no fixed requirement on the density of its lookahead tree besides those imposed by hardware limitations and speed constraints.

The lookahead trees vary in the actions available to the player acting, the actions available for the opponent’s response, and the actions available to either player for the remainder of the round. We use the end of the round as our depth limit, except on the turn when the remainder of the game is solved. On the pre-flop and flop, we use trained counterfactual value networks to return values after the flop or turn card(s) are revealed. Only applying our value function to public states at the start of a round is particularly convenient in that that we don’t need to include the bet faced as an input to the function. Table E.3 specifies lookahead tree properties for each round.

The pre-flop round is particularly expensive as it requires enumerating all 22,100 possible public cards on the flop and evaluating each with the flop network. To speed up pre-flop play, we trained an additional auxiliary neural network to estimate the expected value of the flop network over all possible flops. However, we only used this network during the initial

Table E.4: Absolute ( $L_1$ ), Euclidean ( $L_2$ ), and maximum absolute ( $L_\infty$ ) errors, in mbb/g, of counterfactual values computed with 1,000 iterations of CFR on sparse trees, averaged over 100 random river situations. The ground truth values were estimated by solving the game with 9 betting options and 4,000 iterations (first row).

Betting	Size	$L_1$	$L_2$	$L_\infty$
F, C, Min, $\frac{1}{4}$ P, $\frac{1}{2}$ P, $\frac{3}{4}$ P, P, 2P, 3P, 10P, A [4,000 iterations]	555k	0.0	0.0	0.0
F, C, Min, $\frac{1}{4}$ P, $\frac{1}{2}$ P, $\frac{3}{4}$ P, P, 2P, 3P, 10P, A	555k	18.06	0.891	0.2724
F, C, 2P, A	48k	64.79	2.672	0.3445
F, C, $\frac{1}{2}$ P, A	100k	58.24	3.426	0.7376
F, C, P, A	61k	25.51	1.272	0.3372
F, C, $\frac{1}{2}$ P, P, A	126k	41.42	1.541	0.2955
F, C, P, 2P, A	204k	27.69	1.390	0.2543
F, C, $\frac{1}{2}$ P, P, 2P, A	360k	20.96	1.059	0.2653

omitted iterations of CFR. During the final iterations used to compute the average strategy and counterfactual values, we did the expensive enumeration and flop network evaluations. Additionally, we cache the re-solving result for every observed pre-flop situation. When the same betting sequence occurs again, we simply reuse the cached results rather than recomputing. For the turn round, we did not use a neural network after the final river card, but instead solved to the end of the game. However, we used a bucketed abstraction for all actions on the river. For acting on the river, the re-solving includes the remainder of the game and so no counterfactual value network was used.

**Actions in Sparse Lookahead Trees.** DeepStack’s sparse lookahead trees use only a small subset of the game’s possible actions. The first layer of actions immediately after the current public state defines the options considered for DeepStack’s next action. The only purpose of the remainder of the tree is to estimate counterfactual values for the first layer during the CFR algorithm. Table E.4 presents how well counterfactual values can be estimated using sparse lookahead trees with various action subsets.

The results show that the F, C, P, A, actions provide an excellent tradeoff between computational requirements via the size of the solved lookahead tree and approximation quality. Using more actions quickly increases the size of the lookahead tree, but does not substantially improve errors. Alternatively, using a single betting action that is not one pot has a small effect on the size of the tree, but causes a substantial error increase.

To further investigate the effect of different betting options, Table E.5 presents the results of evaluating DeepStack with different action sets using LBR. We used setting of LBR that proved most effective against the default set of DeepStack actions (see Table E.3). While the extent of the variance in the 10,000 hand evaluation shown in Table E.5 prevents us from declaring a best set of actions with certainty, the crucial point is that LBR is significantly losing to each of them, and that we can produce play that is difficult to exploit even choosing from a small number of actions. Furthermore, the improvement of a small number of additional actions is not dramatic.

**Opponent Ranges in Re-Solving.** Continual re-solving does not require keeping track of the opponent’s range. The re-solving step essentially reconstructs a suitable range using the bounded counterfactual values. In particular, the CFR-D gadget does this by giving the opponent the option, after being dealt a uniform random hand, of terminating the game (T)

Table E.5: Performance of LBR exploitation of DeepStack with different actions allowed on the first level of its lookahead tree using the best LBR configuration against the default version of DeepStack. LBR cannot exploit DeepStack regardless of its available actions.

First level actions	LBR performance
F, C, P, A	$-479 \pm 216$
Default	$-383 \pm 219$
F, C, $\frac{1}{2}$ P, P, $1\frac{1}{2}$ P, 2P, A	$-406 \pm 218$

instead of following through with the game (F), allowing them to simply earn that hand’s bound on its counterfactual value. Only hands which are valuable to bring into the subgame will then be observed by the re-solving player. However, this process of the opponent learning which hands to follow through with can make re-solving require many iterations. An estimate of the opponent’s range can be used to effectively warm-start the choice of opponent ranges, and help speed up the re-solving.

One conservative option is to replace the uniform random deal of opponent hands with any distribution over hands as long as it assigns non-zero probability to every hand. For example, we could linearly combine an estimated range of the opponent from the previous re-solve (with weight  $b$ ) and a uniform range (with weight  $1 - b$ ). This approach still has the same theoretical guarantees as re-solving, but can reach better approximate solutions in fewer iterations. Another option is more aggressive and sacrifices the re-solving guarantees when the opponent’s range estimate is wrong. It forces the opponent with probability  $b$  to follow through into the game with a hand sampled from the estimated opponent range. With probability  $1 - b$  they are given a uniform random hand and can choose to terminate or follow through. This could prevent the opponent’s strategy from reconstructing a correct range, but can speed up re-solving further when we have a good opponent range estimate.

DeepStack uses an estimated opponent range during re-solving only for the first action of a round, as this is the largest lookahead tree to re-solve. The range estimate comes from the last re-solve in the previous round. When DeepStack is second to act in the round, the opponent has already acted, biasing their range, so we use the conservative approach. When DeepStack is first to act, though, the opponent could only have checked or called since our last re-solve. Thus, the lookahead has an estimated range following their action. So in this case, we use the aggressive approach. In both cases, we set  $b = 0.9$ .

**Speed of Play.** The re-solving computation and neural network evaluations are both implemented in Torch7 [Collobert et al., 2011] and run on a single NVIDIA GeForce GTX 1080 graphics card. This makes it possible to do fast batched calls to the counterfactual value networks for multiple public subtrees at once, which is key to making DeepStack fast.

Table E.6 reports the average times between the end of the previous (opponent or chance) action and submitting the next action by both humans and DeepStack in our study. DeepStack, on average, acted considerably faster than our human players. It should be noted that some human players were playing up to four games simultaneously (although few players did more than two), and so the human players may have been focused on another game when it became their turn to act.

Table E.6: Thinking times for both humans and DeepStack. DeepStack’s extremely fast pre-flop speed shows that pre-flop situations often resulted in cache hits.

Round	Thinking Time (s)			
	Humans		DeepStack	
	Median	Mean	Median	Mean
Pre-flop	10.3	16.2	0.04	0.2
Flop	9.1	14.6	5.9	5.9
Turn	8.0	14.0	5.4	5.5
River	9.5	16.2	2.2	2.1
Per Action	9.6	15.4	2.3	3.0
Per Hand	22.0	37.4	5.7	7.2

## E.5.2 Deep Counterfactual Value Networks

DeepStack uses two counterfactual value networks, one for the flop and one for the turn, as well as an auxiliary network that gives counterfactual values at the end of the pre-flop. In order to train the networks, we generated random poker situations at the start of the flop and turn. Each poker situation is defined by the pot size, ranges for both players, and dealt public cards. The complete betting history is not necessary as the pot and ranges are a sufficient representation. The output of the network are vectors of counterfactual values, one for each player. The output values are interpreted as fractions of the pot size to improve generalization across poker situations.

The training situations were generated by first sampling a pot size from a fixed distribution which was designed to approximate observed pot sizes from older HUNL programs.\* The player ranges for the training situations need to cover the space of possible ranges that CFR might encounter during re-solving, not just ranges that are likely part of a solution. So we generated pseudo-random ranges that attempt to cover the space of possible ranges. We used a recursive procedure  $R(S, p)$ , that assigns probabilities to the hands in the set  $S$  that sum to probability  $p$ , according to the following procedure.

1. If  $|S| = 1$ , then  $\Pr(s) = p$ .
2. Otherwise,
  - (a) Choose  $p_1$  uniformly at random from the interval  $(0, p)$ , and let  $p_2 = p - p_1$ .
  - (b) Let  $S_1 \subset S$  and  $S_2 = S \setminus S_1$  such that  $|S_1| = \lfloor |S|/2 \rfloor$  and all of the hands in  $S_1$  have a hand strength no greater than hands in  $S_2$ . Hand strength is the probability of a hand beating a uniformly selected random hand from the current public state.
  - (c) Use  $R(S_1, p_1)$  and  $R(S_2, p_2)$  to assign probabilities to hands in  $S = S_1 \cup S_2$ .

Generating a range involves invoking  $R(\text{all hands}, 1)$ . To obtain the target counterfactual values for the generated poker situations for the main networks, the situations were approximately solved using 1,000 iterations of CFR<sup>+</sup> with only betting actions fold, call, a pot-sized bet, and all-in. For the turn network, ten million poker turn situations (from after

\*The fixed distribution selects an interval from the set of intervals  $\{[100, 100), [200, 400), [400, 2000), [2000, 6000), [6000, 19950]\}$  with uniform probability, followed by uniformly selecting an integer from within the chosen interval.

the turn card is dealt) were generated and solved with 6,144 CPU cores of the Calcul Québec MP2 research cluster, using over 175 core years of computation time. For the flop network, one million poker flop situations (from after the flop cards are dealt) were generated and solved. These situations were solved using DeepStack’s depth limited solver with the turn network used for the counterfactual values at public states immediately after the turn card. We used a cluster of 20 GPUS and one-half of a GPU year of computation time. For the auxiliary network, ten million situations were generated and the target values were obtained by enumerating all 22,100 possible flops and averaging the counterfactual values from the flop network’s output.

**Neural Network Training.** All networks were trained using built-in Torch7 libraries, with the Adam stochastic gradient descent procedure [Kingma and Ba, 2014] minimizing the average of the Huber losses [Huber, 1964] over the counterfactual value errors. Training used a mini-batch size of 1,000, and a learning rate 0.001, which was decreased to 0.0001 after the first 200 epochs. Networks were trained for approximately 350 epochs over two days on a single GPU, and the epoch with the lowest validation loss was chosen.

**Neural Network Range Representation.** In order to improve generalization over input player ranges, we map the distribution of individual hands (combinations of public and private cards) into distributions of buckets. The buckets were generated using a clustering-based abstraction technique, which cluster strategically similar hands using  $k$ -means clustering with earth mover’s distance over hand-strength-like features [Johanson et al., 2013, Ganzfried and Sandholm, 2014]. For both the turn and flop networks we used 1,000 clusters and map the original ranges into distributions over these clusters as the first layer of the neural network (see Figure 3 of the main article). This bucketing step was not used on the auxiliary network as there are only 169 strategically distinct hands pre-flop, making it feasible to input the distribution over distinct hands directly.

**Neural Network Accuracies.** The turn network achieved an average Huber loss of 0.016 of the pot size on the training set and 0.026 of the pot size on the validation set. The flop network, with a much smaller training set, achieved an average Huber loss of 0.008 of the pot size on the training set, but 0.034 of the pot size on the validation set. Finally, the auxiliary network had average Huber losses of 0.000053 and 0.000055 on the training and validation set, respectively. Note that there are, in fact, multiple Nash equilibrium solutions to these poker situations, with each giving rise to different counterfactual value vectors. So, these losses may overestimate the true loss as the network may accurately model a different equilibrium strategy.

**Number of Hidden Layers.** We observed in early experiments that the neural network had a lower validation loss with an increasing number of hidden layers. From these experiments, we chose to use seven hidden layers in an attempt to tradeoff accuracy, speed of execution, and the available memory on the GPU. The result of a more thorough experiment examining the turn network accuracy as a function of the number of hidden layers is in Figure E.1. It appears that seven hidden layers is more than strictly necessary as the validation error does not improve much beyond five. However, all of these architectures were trained using the same ten million turn situations. With more training data it would not be surprising to see the larger networks see a further reduction in loss due to their richer representation power.

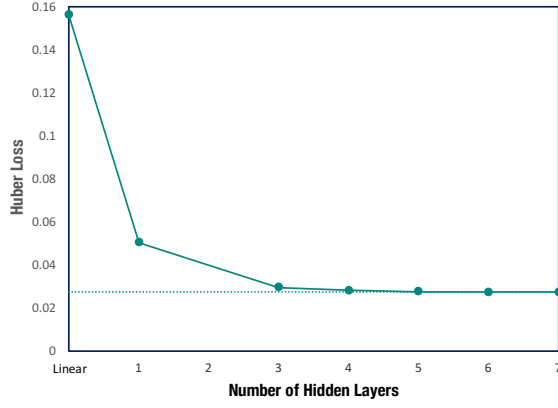


Figure E.1: Huber loss with different numbers of hidden layers in the neural network.

## E.6 Proof of Theorem 9.1

The formal proof of Theorem 9.1, which establishes the soundness of DeepStack’s depth-limited continual re-solving, is conceptually easy to follow. It requires three parts. First, we establish that the exploitability introduced in a re-solving step has two linear components; one due to approximately solving the subgame, and one due to the error in DeepStack’s counterfactual value network (see Lemmas 1 through 5). Second, we enable estimates of subgame counterfactual values that do not arise from actual subgame strategies (see Lemma 6). Together, parts one and two enable us to use DeepStack’s counterfactual value network for a single re-solve.<sup>†</sup> Finally, we show that using the opponent’s values from the best action, rather than the observed action, does not increase overall exploitability (see Lemma 7). This allows us to carry forward estimates of the opponent’s counterfactual value, enabling continual re-solving. Put together, these three parts bound the error after any finite number of continual re-solving steps, concluding the proof. We now formalize each step.

There are a number of concepts we use throughout this section. We use the notation from Burch et al. [Burch et al., 2014] without any introduction here. We assume player 1 is performing the continual re-solving. We call player 2 the opponent. We only consider the re-solve player’s strategy  $\sigma$ , as the opponent is always using a best response to  $\sigma$ . All values are considered with respect to the opponent, unless specifically stated. We say  $\sigma$  is  $\epsilon$ -exploitable if the opponent’s best response value against  $\sigma$  is no more than  $\epsilon$  away from the game value for the opponent.

A public state  $S$  corresponds to the root of an imperfect information subgame. We write  $\mathcal{I}_2^S$  for the collection of player 2 information sets in  $S$ . Let  $G\langle S, \sigma, w \rangle$  be the subtree gadget game (the re-solving game of Burch et al. [Burch et al., 2014]), where  $S$  is some public state,  $\sigma$  is used to get player 1 reach probabilities  $\pi_{-2}^\sigma(h)$  for each  $h \in S$ , and  $w$  is a vector where  $w_I$  gives the value of player 2 taking the terminate action (T) from information set  $I \in \mathcal{I}_2^S$ . Let

$$\text{BV}_I(\sigma) = \max_{\sigma_2^*} \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma, \sigma_2^*}(h) / \pi_{-2}^\sigma(I),$$

be the counterfactual value for  $I$  given we play  $\sigma$  and our opponent is playing a best response. For a subtree strategy  $\sigma^S$ , we write  $\sigma \rightarrow \sigma^S$  for the strategy that plays according to  $\sigma^S$  for any state in the subtree and according to  $\sigma$  otherwise. For the gadget game  $G\langle S, \sigma, w \rangle$ , the

<sup>†</sup>The first part is a generalization and improvement on the re-solving exploitability bound given by Theorem 3 in Burch et al. [Burch et al., 2014], and the second part generalizes the bound on decomposition regret given by Theorem 2 of the same work.

gadget value of a subtree strategy  $\sigma^S$  is defined to be:

$$\text{GV}_{w,\sigma}^S(\sigma^S) = \sum_{I \in \mathcal{I}_2^S} \max(w_I, \text{BV}_I(\sigma \rightarrow \sigma^S)),$$

and the underestimation error is defined to be:

$$\text{U}_{w,\sigma}^S = \min_{\sigma^S} \text{GV}_{w,\sigma}^S(\sigma^S) - \sum_{I \in \mathcal{I}_2^S} w_I.$$

**Lemma E.1.** *The game value of a gadget game  $G\langle S, \sigma, w \rangle$  is*

$$\sum_{I \in \mathcal{I}_2^S} w_I + \text{U}_{w,\sigma}^S.$$

*Proof.* Let  $\tilde{\sigma}_2^S$  be a gadget game strategy for player 2 which must choose from the F and T actions at starting information set  $I$ . Let  $\tilde{u}$  be the utility function for the gadget game.

$$\begin{aligned} \min_{\sigma_1^S} \max_{\tilde{\sigma}_2^S} \tilde{u}(\sigma_1^S, \tilde{\sigma}_2^S) &= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \frac{\pi_{-2}^\sigma(I)}{\sum_{I' \in \mathcal{I}_2^S} \pi_{-2}^\sigma(I')} \max_{a \in \{F, T\}} \tilde{u}^{\sigma^S}(I, a) \\ &= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma^S}(h)) \end{aligned}$$

A best response can maximize utility at each information set independently:

$$\begin{aligned} &= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \max_{\sigma_2^S} \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma^S}(h)) \\ &= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \text{BV}_I(\sigma \rightarrow \sigma_1^S)) \\ &= \text{U}_{w,\sigma}^S + \sum_{I \in \mathcal{I}_2^S} w_I \end{aligned}$$

□

**Lemma E.2.** *If our strategy  $\sigma^S$  is  $\epsilon$ -exploitable in the gadget game  $G\langle S, \sigma, w \rangle$ , then  $\text{GV}_{w,\sigma}^S(\sigma^S) \leq \sum_{I \in \mathcal{I}_2^S} w_I + \text{U}_{w,\sigma}^S + \epsilon$*

*Proof.* This follows from Lemma E.1 and the definitions of  $\epsilon$ -Nash,  $\text{U}_{w,\sigma}^S$ , and  $\text{GV}_{w,\sigma}^S(\sigma^S)$ . □

**Lemma E.3.** *Given an  $\epsilon_0$ -exploitable  $\sigma$  in the original game, if we replace a subgame with a strategy  $\sigma^S$  such that  $\text{BV}_I(\sigma \rightarrow \sigma^S) \leq w_I$  for all  $I \in \mathcal{I}_2^S$ , then the new combined strategy has an exploitability no more than  $\epsilon_0 + \text{EXP}_{w,\sigma}^S$  where*

$$\text{EXP}_{w,\sigma}^S = \sum_{I \in \mathcal{I}_2^S} \max(\text{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \text{BV}_I(\sigma)$$

*Proof.* We only care about the information sets where the opponent's counterfactual value increases, and a worst case upper bound occurs when the opponent best response would reach every such information set with probability 1, and never reach information sets where the value decreased.



Let  $Z[S] \subseteq Z$  be the set of terminal states reachable from some  $h \in S$  and let  $v_2$  be the game value of the full game for player 2. Let  $\sigma_2$  be a best response to  $\sigma$  and let  $\sigma_2^S$  be the part of  $\sigma_2$  that plays in the subtree rooted at  $S$ . Then necessarily  $\sigma_2^S$  achieves counterfactual value  $\text{BV}_I(\sigma)$  at each  $I \in \mathcal{I}_2^S$ .

$$\begin{aligned}
& \max_{\sigma_2^*} (u(\sigma \rightarrow \sigma^S, \sigma_2^*)) \\
&= \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&= \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) - \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) \right. \\
&\quad \left. + \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&\leq \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) - \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) \right] \\
&\quad + \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&\leq \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \sum_{h \in I} \pi_{-2}^{\sigma}(h) \pi_2^{\sigma_2^*}(h) u^{\sigma^S, \sigma_2^*}(h) \right. \\
&\quad \left. - \sum_{I \in \mathcal{I}_2^S} \sum_{h \in I} \pi_{-2}^{\sigma}(h) \pi_2^{\sigma_2^*}(h) u^{\sigma, \sigma_2^S}(h) \right] + \max_{\sigma_2^*} (u(\sigma, \sigma_2^*))
\end{aligned}$$

By perfect recall  $\pi_2(h) = \pi_2(I)$  for each  $h \in I$ :

$$\begin{aligned}
&\leq \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I) \left( \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma^S, \sigma_2^*}(h) - \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma, \sigma_2^S}(h) \right) \right] \\
&\quad + v_2 + \epsilon_O \\
&= \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I) \pi_{-2}^{\sigma}(I) \left( \text{BV}_I(\sigma \rightarrow \sigma^S) - \text{BV}_I(\sigma) \right) \right] + v_2 + \epsilon_O \\
&\leq \left[ \sum_{I \in \mathcal{I}_2^S} \max(\text{BV}_I(\sigma \rightarrow \sigma^S) - \text{BV}_I(\sigma), 0) \right] + v_2 + \epsilon_O \\
&\leq \left[ \sum_{I \in \mathcal{I}_2^S} \max(w_I - \text{BV}_I(\sigma), \text{BV}_I(\sigma) - \text{BV}_I(\sigma)) \right] + v_2 + \epsilon_O \\
&= \left[ \sum_{I \in \mathcal{I}_2^S} \max(\text{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \text{BV}_I(\sigma) \right] + v_2 + \epsilon_O
\end{aligned}$$

□

**Lemma E.4.** *Given an  $\epsilon_O$ -exploitable  $\sigma$  in the original game, if we replace the strategy in a subgame with a strategy  $\sigma^S$  that is  $\epsilon_S$ -exploitable in the gadget game  $G\langle S, \sigma, w \rangle$ , then the new combined strategy has an exploitability no more than  $\epsilon_O + \text{EXP}_{w, \sigma}^S + U_{w, \sigma}^S + \epsilon_S$ .*

*Proof.* We use that  $\max(a, b) = a + b - \min(a, b)$ . From applying Lemma E.3 with  $w_I = \text{BV}_I(\sigma \rightarrow \sigma^S)$  and expanding  $\text{EXP}_{\text{BV}(\sigma \rightarrow \sigma^S), \sigma}^S$  we get exploitability no more than

$\epsilon_O - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma)$  plus

$$\begin{aligned}
& \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \mathbf{BV}_I(\sigma)) \\
& \leq \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \max(w_I, \mathbf{BV}_I(\sigma))) \\
& = \sum_{I \in \mathcal{I}_2^S} \left( \mathbf{BV}_I(\sigma \rightarrow \sigma^S) + \max(w_I, \mathbf{BV}_I(\sigma)) \right. \\
& \quad \left. - \min(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \max(w_I, \mathbf{BV}_I(\sigma))) \right) \\
& \leq \sum_{I \in \mathcal{I}_2^S} \left( \mathbf{BV}_I(\sigma \rightarrow \sigma^S) + \max(w_I, \mathbf{BV}_I(\sigma)) \right. \\
& \quad \left. - \min(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), w_I) \right) \\
& = \sum_{I \in \mathcal{I}_2^S} \left( \max(w_I, \mathbf{BV}_I(\sigma)) + \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - w_I \right) \\
& = \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) + \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - \sum_{I \in \mathcal{I}_2^S} w_I
\end{aligned}$$

From Lemma E.2 we get

$$\leq \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) + \mathbf{U}_{w, \sigma}^S + \epsilon_S$$

Adding  $\epsilon_O - \sum_I \mathbf{BV}_I(\sigma)$  we get the upper bound  $\epsilon_O + \mathbf{EXP}_{w, \sigma}^S + \mathbf{U}_{w, \sigma}^S + \epsilon_S$ .  $\square$

**Lemma E.5.** *Assume we are performing one step of re-solving on subtree  $S$ , with constraint values  $w$  approximating opponent best-response values to the previous strategy  $\sigma$ , with an approximation error bound  $\sum_I |w_I - \mathbf{BV}_I(\sigma)| \leq \epsilon_E$ . Then we have  $\mathbf{EXP}_{w, \sigma}^S + \mathbf{U}_{w, \sigma}^S \leq \epsilon_E$ .*

*Proof.*  $\mathbf{EXP}_{w, \sigma}^S$  measures the amount that the  $w_I$  exceed  $\mathbf{BV}_I(\sigma)$ , while  $\mathbf{U}_{w, \sigma}^S$  bounds the amount that the  $w_I$  underestimate  $\mathbf{BV}_I(\sigma \rightarrow \sigma^S)$  for any  $\sigma^S$ , including the original  $\sigma$ . Thus, together they are bounded by  $|w_I - \mathbf{BV}_I(\sigma)|$ :

$$\begin{aligned}
\mathbf{EXP}_{w, \sigma}^S + \mathbf{U}_{w, \sigma}^S &= \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma) \\
& \quad + \min_{\sigma^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - \sum_{I \in \mathcal{I}_2^S} w_I \\
& \leq \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma) \\
& \quad + \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) - \sum_{I \in \mathcal{I}_2^S} w_I \\
& = \sum_{I \in \mathcal{I}_2^S} [\max(w_I - \mathbf{BV}_I(\sigma), 0) + \max(\mathbf{BV}_I(\sigma) - w_I, 0)] \\
& = \sum_{I \in \mathcal{I}_2^S} |w_I - \mathbf{BV}_I(\sigma)| \leq \epsilon_E
\end{aligned}$$

$\square$

**Lemma E.6.** *Assume we are solving a game  $G$  with  $T$  iterations of CFR-D where for both players  $p$ , subtrees  $S$ , and times  $t$ , we use subtree values  $v_I$  for all information sets  $I$  at the root of  $S$  from some suboptimal black box estimator. If the estimation error is bounded, so that  $\min_{\sigma_s^* \in NE_s} \sum_{I \in \mathcal{I}_2^s} |v^{\sigma_s^*}(I) - v_I| \leq \epsilon_E$ , then the trunk exploitability is bounded by  $k_G/\sqrt{T} + j_G \epsilon_E$  for some game specific constant  $k_G, j_G \geq 1$  which depend on how the game is split into a trunk and subgames.*

*Proof.* This follows from a modified version the proof of Theorem 2 of Burch et al. [Burch et al., 2014], which uses a fixed error  $\epsilon$  and argues by induction on information sets. Instead, we argue by induction on entire public states.

For every public state  $s$ , let  $N_s$  be the number of subgames reachable from  $s$ , including any subgame rooted at  $s$ . Let  $Succ(s)$  be the set of our public states which are reachable from  $s$  without going through another of our public states on the way. Note that if  $s$  is in the trunk, then every  $s' \in Succ(s)$  is in the trunk or is the root of a subgame. Let  $D_{TR}(s)$  be the set of our trunk public states reachable from  $s$ , including  $s$  if  $s$  is in the trunk. We argue that for any public state  $s$  where we act in the trunk or at the root of a subgame

$$\sum_{I \in s} R_{full}^{T,+}(I) \leq \sum_{s' \in D_{TR}(s)} \sum_{I \in s'} R^{T,+}(I) + TN_s \epsilon_E \quad (\text{E.1})$$

First note that if no subgame is reachable from  $s$ , then  $N_s = 0$  and the statement follows from Lemma 7 of [Zinkevich et al., 2007]. For public states from which a subgame is reachable, we argue by induction on  $|D_{TR}(s)|$ .

For the base case, if  $|D_{TR}(s)| = 0$  then  $s$  is the root of a subgame  $S$ , and by assumption there is a Nash Equilibrium subgame strategy  $\sigma_s^*$  that has regret no more than  $\epsilon_E$ . If we implicitly play  $\sigma_s^*$  on each iteration of CFR-D, we thus accrue  $\sum_{I \in s} R_{full}^{T,+}(I) \leq T \epsilon_E$ .

For the inductive hypothesis, we assume that (E.1) holds for all  $s$  such that  $|D_{TR}(s)| < k$ .

Consider a public state  $s$  where  $|D_{TR}(s)| = k$ . By Lemma 5 of [Zinkevich et al., 2007] we have

$$\begin{aligned} \sum_{I \in s} R_{full}^{T,+}(I) &\leq \sum_{I \in s} \left[ R^T(I) + \sum_{I' \in Succ(I)} R_{full}^{T,+}(I') \right] \\ &= \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \sum_{I' \in s'} R_{full}^{T,+}(I') \end{aligned}$$

For each  $s' \in Succ(s)$ ,  $D(s') \subset D(s)$  and  $s \notin D(s')$ , so  $|D(s')| < |D(s)|$  and we can apply the inductive hypothesis to show

$$\begin{aligned} \sum_{I \in s} R_{full}^{T,+}(I) &\leq \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \left[ \sum_{s'' \in D(s')} \sum_{I \in s''} R^{T,+}(I) + TN_{s'} \epsilon_E \right] \\ &\leq \sum_{s' \in D(s)} \sum_{I \in s'} R^{T,+}(I) + T \epsilon_E \sum_{s' \in Succ(s)} N_{s'} \\ &= \sum_{s' \in D(s)} \sum_{I \in s'} R^{T,+}(I) + T \epsilon_E N_s \end{aligned}$$

This completes the inductive argument. By using regret matching in the trunk, we ensure  $R^T(I) \leq \Delta \sqrt{AT}$ , proving the lemma for  $k_G = \Delta |\mathcal{I}_{TR}| \sqrt{A}$  and  $j_G = N_{root}$ .  $\square$

**Lemma E.7.** *Given our strategy  $\sigma$ , if the opponent is acting at the root of a public subtree  $S$  from a set of actions  $A$ , with opponent best-response values  $BV_{I,a}(\sigma)$  after each action  $a \in A$ , then replacing our subtree strategy with any strategy that satisfies the opponent constraints  $w_I = \max_{a \in A} BV_{I,a}(\sigma)$  does not increase our exploitability.*

*Proof.* If the opponent is playing a best response, every counterfactual value  $w_I$  before the action must either satisfy  $w_I = \text{BV}_I(\sigma) = \max_{a \in A} \text{BV}_{I,a}(\sigma)$ , or not reach state  $s$  with private information  $I$ . If we replace our strategy in  $S$  with a strategy  $\sigma'_S$  such that  $\text{BV}_{I,a}(\sigma'_S) \leq \text{BV}_I(\sigma)$  we preserve the property that  $\text{BV}_I(\sigma') = \text{BV}_I(\sigma)$ .  $\square$

**Theorem E.8.** *Assume we have some initial opponent constraint values  $w$  from a solution generated using at least  $T$  iterations of CFR-D, we use at least  $T$  iterations of CFR-D to solve each re-solving game, and we use a subtree value estimator such that  $\min_{\sigma_S^* \in NE_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_S^*}(I) - v_I| \leq \epsilon_E$ , then after  $d$  re-solving steps the exploitability of the resulting strategy is no more than  $(d + 1)k/\sqrt{T} + (2d + 1)j\epsilon_E$  for some constants  $k, j$  specific to both the game and how it is split into subgames.*

*Proof.* Continual re-solving begins by solving from the root of the entire game, which we label as subtree  $S_0$ . We use CFR-D with the value estimator in place of subgame solving in order to generate an initial strategy  $\sigma_0$  for playing in  $S_0$ . By Lemma E.6, the exploitability of  $\sigma_0$  is no more than  $k_0/\sqrt{T} + j_0\epsilon_E$ .

For each step of continual re-solving  $i = 1, \dots, d$ , we are re-solving some subtree  $S_i$ . From the previous step of re-solving, we have approximate opponent best-response counterfactual values  $\widetilde{\text{BV}}_I(\sigma_{i-1})$  for each  $I \in \mathcal{I}_2^{S_{i-1}}$ , which by the estimator bound satisfy  $|\sum_{I \in \mathcal{I}_2^{S_{i-1}}} \text{BV}_I(\sigma_{i-1}) - \widetilde{\text{BV}}_I(\sigma_{i-1})| \leq \epsilon_E$ . Updating these values at each public state between  $S_{i-1}$  and  $S_i$  as described in the main chapter yields approximate values  $\widetilde{\text{BV}}_I(\sigma_{i-1})$  for each  $I \in \mathcal{I}_2^{S_i}$  which by Lemma E.7 can be used as constraints  $w_{I,i}$  in re-solving. Lemma E.5 with these constraints gives us the bound  $\text{EXP}_{w_i, \sigma_{i-1}}^{S_i} + \text{U}_{w_i, \sigma_{i-1}}^{S_i} \leq \epsilon_E$ . Thus by Lemma E.4 and Lemma E.6 we can say that the increase in exploitability from  $\sigma_{i-1}$  to  $\sigma_i$  is no more than  $\epsilon_E + \epsilon_{S_i} \leq \epsilon_E + k_i/\sqrt{T} + j_i\epsilon_E \leq k_i/\sqrt{T} + 2j_i\epsilon_E$ .

Let  $k = \max_i k_i$  and  $j = \max_i j_i$ . Then after  $d$  re-solving steps, the exploitability is bounded by  $(d + 1)k/\sqrt{T} + (2d + 1)j\epsilon_E$ .  $\square$

## E.7 Best-response Values Versus Self-play Values

DeepStack uses self-play values within the continual re-solving computation, rather than the best-response values described in Theorem E.8. Preliminary tests using CFR-D to solve smaller games suggested that strategies generated using self-play values were generally less exploitable and had better one-on-one performance against test agents, compared to strategies generated using best-response values. Figure E.2 shows an example of DeepStack's exploitability in a particular river subgame with different numbers of re-solving iterations. Despite lacking a theoretical justification for its soundness, using self-play values appears to converge to low exploitability strategies just as with using best-response values.

One possible explanation for why self-play values work well with continual re-solving is that at every re-solving step, we give away a little more value to our best-response opponent because we are not solving the subtrees exactly. If we use the self-play values for the opponent, the opponent's strategy is slightly worse than a best response, making the opponent values smaller and counteracting the inflationary effect of an inexact solution. While this optimism could hurt us by setting unachievable goals for the next re-solving step (an increased  $\text{U}_{w, \sigma}^S$  term), in poker-like games we find that the more positive expectation is generally correct (a decreased  $\text{EXP}_{w, \sigma}^S$  term.)

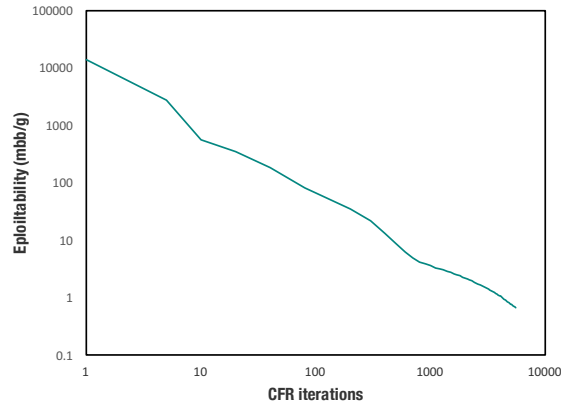


Figure E.2: DeepStack’s exploitability within a particular public state at the start of the river as a function of the number of re-solving iterations.

## E.8 Pseudocode

Complete pseudocode for DeepStack’s depth-limited continual re-resolving algorithm is in Algorithm 1. Conceptually, DeepStack can be decomposed into four functions: RE-SOLVE, VALUES, UPDATESUBTREESTRATEGIES, and RANGEGADGET. The main function is RE-SOLVE, which is called every time DeepStack needs to take an action. It iteratively calls each of the other functions to refine the lookahead tree solution. After  $T$  iterations, an action is sampled from the approximate equilibrium strategy at the root of the subtree to be played. According to this action, DeepStack’s range,  $\vec{r}_1$ , and its opponent’s counterfactual values,  $\vec{v}_2$ , are updated in preparation for its next decision point.

---

**Algorithm 1** Depth-limited continual re-solving

---

**INPUT:** Public state  $S$ , player range  $\mathbf{r}_1$  over our information sets in  $S$ , opponent counterfactual values  $\mathbf{v}_2$  over their information sets in  $S$ , and player information set  $I \in S$

**OUTPUT:** Chosen action  $a$ , and updated representation after the action ( $S(a)$ ,  $\mathbf{r}_1(a)$ ,  $\mathbf{v}_2(a)$ )

```
1: function RE-SOLVE( $S$ ,  $\mathbf{r}_1$ ,  $\mathbf{v}_2$ ,  $I$ )
2:    $\sigma^0 \leftarrow$  arbitrary initial strategy profile
3:    $\mathbf{r}_2^0 \leftarrow$  arbitrary initial opponent range
4:    $R_G^0, R^0 \leftarrow \mathbf{0}$  ▷ Initial regrets for gadget game and subtree
5:   for  $t = 1$  to  $T$  do
6:      $\mathbf{v}_1^t, \mathbf{v}_2^t \leftarrow$  VALUES( $S$ ,  $\sigma^{t-1}$ ,  $\mathbf{r}_1$ ,  $\mathbf{r}_2^{t-1}$ ,  $0$ )
7:      $\sigma^t, R^t \leftarrow$  UPDATESUBTREESTRATEGIES( $S$ ,  $\mathbf{v}_1^t$ ,  $\mathbf{v}_2^t$ ,  $R^{t-1}$ )
8:      $\mathbf{r}_2^t, R_G^t \leftarrow$  RANGEGADGET( $\mathbf{v}_2$ ,  $\mathbf{v}_2^t(S)$ ,  $R_G^{t-1}$ )
9:      $\bar{\sigma}^T \leftarrow \frac{1}{T} \sum_{t=1}^T \sigma^t$  ▷ Average the strategies
10:     $a \sim \bar{\sigma}^T(\cdot|I)$  ▷ Sample an action
11:     $\mathbf{r}_1(a) \leftarrow \langle \mathbf{r}_1, \sigma(a|\cdot) \rangle$  ▷ Update the range based on the chosen action
12:     $\mathbf{r}_1(a) \leftarrow \mathbf{r}_1(a) / \|\mathbf{r}_1(a)\|_1$  ▷ Normalize the range
13:     $\mathbf{v}_2(a) \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{v}_2^t(a)$  ▷ Average of counterfactual values after action  $a$ 
14:    return  $a$ ,  $S(a)$ ,  $\mathbf{r}_1(a)$ ,  $\mathbf{v}_2(a)$ 

15: function VALUES( $S$ ,  $\sigma$ ,  $\mathbf{r}_1$ ,  $\mathbf{r}_2$ ,  $d$ ) ▷ Gives the counterfactual values of the subtree  $S$  under  $\sigma$ ,
    computed with a depth-limited lookahead.
16:   if  $S$  is terminal then
17:      $\mathbf{v}_1(S) \leftarrow U_S \mathbf{r}_2$  ▷ Where  $U_S$  is the matrix of the bilinear utility function at  $S$ ,
18:      $\mathbf{v}_2(S) \leftarrow \mathbf{r}_1^\top U_S$  ▷  $U(S) = \mathbf{r}_1^\top U_S \mathbf{r}_2$ , thus giving vectors of counterfactual values
19:     return  $\mathbf{v}_1(S)$ ,  $\mathbf{v}_2(S)$ 
20:   else if  $d = \text{MAX-DEPTH}$  then
21:     return NEURALNETEVALUATE( $S$ ,  $\mathbf{r}_1$ ,  $\mathbf{r}_2$ )
22:    $\mathbf{v}_1(S)$ ,  $\mathbf{v}_2(S) \leftarrow \mathbf{0}$ 
23:   for action  $a \in S$  do
24:      $\mathbf{r}_{\text{Player}(S)}(a) \leftarrow \langle \mathbf{r}_{\text{Player}(S)}, \sigma(a|\cdot) \rangle$  ▷ Update range of acting player based on strategy
25:      $\mathbf{r}_{\text{Opponent}(S)}(a) \leftarrow \mathbf{r}_{\text{Opponent}(S)}$ 
26:      $\mathbf{v}_1(S(a)), \mathbf{v}_2(S(a)) \leftarrow$  VALUE( $S(a)$ ,  $\sigma$ ,  $\mathbf{r}_1(a)$ ,  $\mathbf{r}_2(a)$ ,  $d + 1$ )
27:      $\mathbf{v}_{\text{Player}(S)}(S) \leftarrow \mathbf{v}_{\text{Player}(S)}(S) + \sigma(a|\cdot) \mathbf{v}_{\text{Player}(S)}(S(a))$  ▷ Weighted average
28:      $\mathbf{v}_{\text{Opponent}(S)}(S) \leftarrow \mathbf{v}_{\text{Player}(S)}(S) + \mathbf{v}_{\text{Opponent}(S)}(S(a))$ 
▷ Unweighted sum, as our strategy is already included in opponent counterfactual values

29:   return  $\mathbf{v}_1$ ,  $\mathbf{v}_2$ 
```

---

---

```

30: function UPDATESUBTREESTRATEGIES( $S, \mathbf{v}_1, \mathbf{v}_2, R^{t-1}$ )
31:   for  $S' \in \{S\} \cup \text{SubtreeDescendants}(S)$  with  $\text{Depth}(S') < \text{MAX-DEPTH}$  do
32:     for action  $a \in S'$  do
33:        $R^t(a|\cdot) \leftarrow R^{t-1}(a|\cdot) + \mathbf{v}_{\text{Player}(S')}(S'(a)) - \mathbf{v}_{\text{Player}(S')}(S')$ 
▷ Update acting player's regrets
34:     for information set  $I \in S'$  do
35:        $\sigma^t(\cdot|I) \leftarrow \frac{R^t(\cdot|I)^+}{\sum_a R^t(a|I)^+}$ 
▷ Update strategy with regret matching
36:   return  $\sigma^t, R^t$ 

37: function RANGEGADGET( $\mathbf{v}_2, \mathbf{v}_2^t, R_G^{t-1}$ )
▷ Let opponent choose to play in the subtree or receive the input value with each hand (see Burch et al. [Burch et al., 2014])
38:    $\sigma_G(\mathbf{F}|\cdot) \leftarrow \frac{R_G^{t-1}(\mathbf{F}|\cdot)^+}{R_G^{t-1}(\mathbf{F}|\cdot)^+ + R_G^{t-1}(\mathbf{T}|\cdot)^+}$ 
▷ F is Follow action, T is Terminate
39:    $\mathbf{r}_2^t \leftarrow \sigma_G(\mathbf{F}|\cdot)$ 
40:    $\mathbf{v}_G^t \leftarrow \sigma_G(\mathbf{F}|\cdot)\mathbf{v}_2^{t-1} + (1 - \sigma_G(\mathbf{F}|\cdot))\mathbf{v}_2$ 
▷ Expected value of gadget strategy
41:    $R_G^t(\mathbf{T}|\cdot) \leftarrow R_G^{t-1}(\mathbf{T}|\cdot) + \mathbf{v}_2 - v_G^{t-1}$ 
▷ Update regrets
42:    $R_G^t(\mathbf{F}|\cdot) \leftarrow R_G^{t-1}(\mathbf{F}|\cdot) + \mathbf{v}_2^t - v_G^t$ 
43:   return  $\mathbf{r}_2^t, R_G^t$ 

```

---

# F. Attachments for Chapter 10

## F.1 An example of Factored-Observation Stochastic Game

Figure F.1 uses the notation introduced in Sec. 10.2 to show an example of simple FOSG game that illustrates how the histories, information states and observations interact. Follow-up Figure F.2 shows a view of a game through public tree perspective.

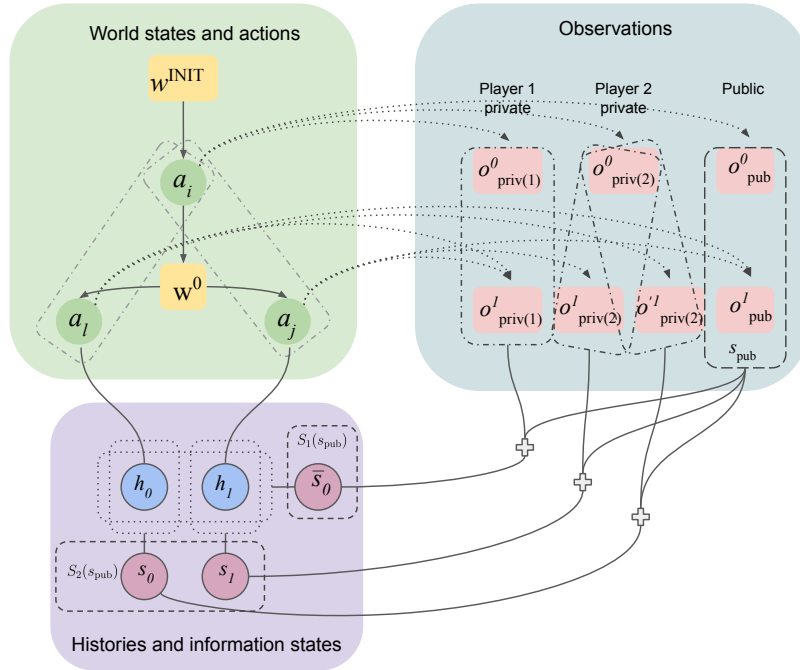


Figure F.1: An example of Factored-Observation Stochastic Game (FOSG). This figure presents the visual view of notation from Sec. 10.2. In this example the game starts in  $w^{init}$  which is the complete state of the environment containing private information for both players. After playing action  $a_i$  the state moves to  $w^0$  where there are two possible actions. Each action emits private and public observations. In this example, actions  $a_j$  and  $a_l$  emit the same private observation  $o_{priv(1)}^l$  for player 1, therefore they cannot distinguish which action happened. On the other hand, player 2 has different observations  $o_{priv(2)}^l$  and  $o'_{priv(2)}^l$  for each of the actions, therefore they have more information about the state of the environment than player 1. The sequence of public observations shared by both players information is denoted as  $s_{pub}$ . Both sequences of actions and factored observations meet in the final 'Histories and information states' view. The two possible action sequences are represented by histories  $h_0$  and  $h_1$ , where  $h_0 = (a_i, a_l)$ ,  $h_1 = (a_i, a_j)$ . Since both actions  $a_l$  and  $a_j$  result in the same observation for player 1, they cannot tell which one of the histories happened and his information state  $\bar{s}_0$  contains them both. This is not the case for player 2, who can separate the histories, and each of his information states  $s_0$  and  $s_1$  contains just one history.



## F.2 Player of Games Algorithm Details

### F.2.1 Re-solving Process

Recall that the re-solving step and the corresponding auxiliary game requires i) the current player’s ranges ii) the opponent’s counterfactual values. This provides succinct and sufficient representation to safely re-solve the subgame rooted in a public state  $s_{pub}$ . DeepStack and Libratus then simply retrieved these invariants from its last search tree (i.e. the search tree from when it acted previously). This was possible because the search tree was fixed to a particular domain (Texas hold’em poker), and thus guaranteed that it included invariants for any possible state the agent could act in next.

As the Player of Games is a general algorithm, it can no longer leverage this special case. The current public state  $s_{pub}$  might not have been included in the last search tree, and the prior computation does not directly provide us with the required invariants for re-solving the subgame rooted in  $s_{pub}$ . POG thus starts its re-solving process in the state closest to the current state that is included in the last search tree:  $s_{pub}^{last}$ . To make sure the search procedure produces policy for the current state, we initialize the search tree to a path from  $s_{pub}^{last}$  to  $s_{current}^{last}$ . See also Figure F.3.

To focus the computation on the states relevant for the current decision, the search tree is being expanded from the current public state  $s_{current}$ . Concretely, recall that the Player of Games algorithm consists of two distinct phases i) the regret update phase and ii) expansion trajectories. The regret update phase is always being run on the full tree (starting in the  $s_{current}^{last}$ ). The expansion simulations then start from  $s_{current}^{pub}$ .

Of course, there are cases when  $s_{current}^{pub} = s_{pub}^{last}$  — that is, the last search tree did include the current state.

### Auxiliary (Gadget) Game for Safe Re-Solving

The construction of the re-solving/auxiliary “gadget” game follows [Burch et al., 2014], where a new state of the opponent is added on top of the subgame. In this state, the opponent can either terminate (T) and receive the constraint values (the re-solving counterfactual

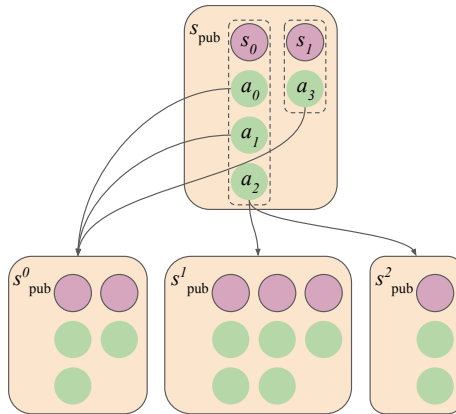


Figure F.2: An example of a public tree. The public tree provides different view of the FOSG. In this example actions  $a_0$  and  $a_1$  emit the same public observation and therefore they lead to the same public tree node  $s_{pub}^0$ . On the other hand, action  $a_2$  can lead to multiple possible states, for instance when a detective in Scotland Yard moves to a location the game can either 1) end because Mr. X was there and he was caught or 2) it continues because he was in a different station.

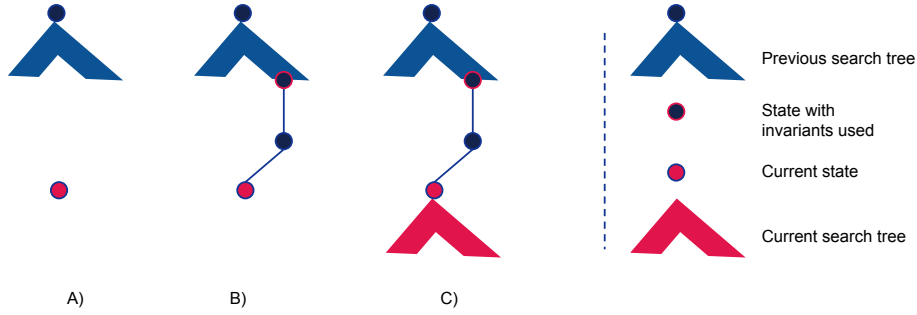


Figure F.3: A) The current state might not be included in the previous search tree. In that case, we lack the corresponding invariants to run the re-solving step. B) The re-solving step is rooted in the state that is closest on the path between the last search state and the current state, while being included in the last search tree. C) The expansion process starts its simulations in the current state, expanding the current search tree only under this node.

values of the opponent), or to follow (F) and play the corresponding subgame. Furthermore, the ranges are used to form the initial distribution of the information states of the player.

Just like DeepStack, the gadget in POG is further modified by mixing in opponent’s ranges as computed from the previous search  $\beta_s$  (when available). This is achieved by modifying the opponent’s initial ranges in the subgame (distribution over their information states after taking the (F) action).  $\beta'_{follow} = \alpha\beta_{follow} + (1 - \alpha)\beta_s$ . Such mixing has been proven to be sound [Moravčík et al., 2017] and empirically improves the performance, and we have chosen  $\alpha = 0.2$ .

## F.2.2 Complexity of the Algorithm

We now analyse the complexity of the algorithm. Due to size of the network, the network call is substantially expensive operation. We thus analyse how many nodes does an algorithm touch, as well as how many times is the network called. Let  $t$  denote the number of iteration, and  $n_t$  the size of the search tree after  $t$  steps (in terms of public states, assuming the number of information states and histories within a public state is bounded by a constant).

**Expansion Phase** The expansion phase traverses a single trajectory, and uses counterfactual values and CFR policy to drive the tree expansion. At the end of the expansion, a new node is added and then evaluated for its value. If a terminal state is hit at the end of the traversal, there is no expansion. The complexity in terms of the number of neural net calls is thus  $\mathcal{O}(t)$ .

Because the simulation touches every node on the sampled trajectory during the traversal, the number of nodes touched is dependant on the structure of the expanded tree. A degenerate case, where the search tree is always a single path then yields  $\mathcal{O}(n_t) = \mathcal{O}(t)$  for each iteration, resulting in  $\mathcal{O}(t^2)$  worst case. Assuming a well-balanced b-ary tree, each iteration touches only  $\mathcal{O}(\log_b(n_t))$ . Note that this phase is analogous to the MCTS variant employed by AlphaZero and matches its complexity.

**Regret Update Phase** The regret update phase traverses the entire search tree during each iteration, and evaluates the leaf nodes using the value function. As all the nodes  $n_t$  are touched during each iteration, the corresponding complexity is  $\mathcal{O}(t^2)$  regardless of the shape of the tree.

While the number of neural net calls is in general also  $\mathcal{O}(t^2)$ , it is possible to optimize the number of network calls down to  $\mathcal{O}(n_t) = \mathcal{O}(t)$  in any perfect information games (e.g. chess and Go). As the CFR reaches the leaf node, it needs to evaluate  $v_\theta(s_{pub}, r_1, r_2)$  where  $r_i$  represents the reach probabilities of individual information states in the public (leaf) state  $s_{pub}$ . In perfect information games, there is a single information state of each player  $|\mathcal{S}_1(s_{pub})| = |\mathcal{S}_2(s_{pub})| = 1$ . Furthermore, by scaling reach probabilities of player  $i$ , the resulting counterfactual values for player  $-i$  are scaled accordingly. Let  $v_1, v_2 = v_\theta(s_{pub}, r_1, r_2)$ ,  $v'_1, v'_2 = v_\theta(s_{pub}, ar_1, br_2)$ , then  $v'_1 = bv_1, v'_2 = av_2$  for any  $a, b \in \mathcal{R}$ . We can leverage this fact for perfect information games and simply scale the value that was returned by the very first neural net call for that public state. This way, there is only a single network call per public state as subsequent evaluations simply scale the previously returned value. Current implementation of POG indeed uses this optimization, substantially speeding up the search process.

Currently, POG uses CFR for the policy improvement and thus needs to traverse the full search tree at each iteration, resulting in  $\mathcal{O}(t^2)$  complexity even for a well balanced tree. There are CFR variants that do not have to traverse the full tree. Namely, MCCFR is a family of sampling based regret minimization methods that only visit and update part of the entire tree and still provide strong convergence guarantees [Lanctot et al., 2009]. Outcome sampling is then a particular variant of MCCFR that samples a single trajectory, and thus its complexity matches the expansion phase. While the sampling introduces variances which can slow down the convergence, it is possible to substantially decrease it using the VR-MCCFR method [Schmid et al., 2019].

### F.2.3 Network Architecture and Optimization

Table F.1 lists neural network architectures and input features used for each game. For chess and Go we use exactly the same architecture and inputs as used by AlphaZero [Silver et al., 2018]. In poker and Scotland Yard we process concatenated belief and public state features by a MLP with ReLU activations [Fukushima, 1980].

The counterfactual value head is optimized by Huber loss [Huber, 1964], while policy for each information state  $i$  is optimized by KL divergence:

$$l(\mathbf{v}, \mathbf{p}, \mathbf{v}_{target}, \mathbf{p}_{target}) = w_v * l_{huber}(\mathbf{v}, \mathbf{v}_{target}) + w_p * \sum_i l_{KL}(\pi^i, \pi_{target}^i)$$

where each head is weighted with the corresponding weight  $w_v$  and  $w_p$ . During training we smoothly decay the learning rate by a factor of  $d$  every  $T_{decay}$  steps. Formally learning rate  $\alpha$  at training step  $t$  is defined as:

$$\alpha_t = \alpha_{init} * d^{t/T_{decay}}$$

When using the policy head’s prediction as prior in PUCT formula the logits are processed with softmax with temperature  $T_{prior}$ . This can decrease weight of the prior in some games and encourage more exploration in the search phase.

### F.2.4 Hyperparameters

Table F.2 lists hyperparameters used for each game, most of these parameters are used in algorithms in Section F.2.5.

Game	Architecture	Belief features	Public state features
Chess	ResNet	Redundant — there is no uncertainty over players state.	One 8x8 plane for each piece type (6) of each player (2) and repetitions planes (2) for last eight moves + scalar planes (7), 119 8x8 planes in total.
Go	ResNet	Redundant	One 19x19 plane for stones of each player (2) for last eight moves plus a single plane encoding player to act, 17 19x19 planes in total.
Poker	MLP 6 x 2048	1326 (possible private card combinations) * 2 (num of players).	N hot encoding of board cards (52) + commitment of each player normalized by his stack (2) + 1 hot encoding of who acts next, including chance player (3).
Scotland Yard	MLP 6 x 512	1 (detectives' position is always certain) + 199 (possible Mr X's position).	1 hot encoding of position of each detective (5*199) + cards of each detective (5*3) + cards of Mr X (5) + who is playing next (6) + was double move just used (1) + how many rounds were played (1).

Table F.1: A neural network architecture and features used for each game.

## F.2.5 Pseudocode

Here we provide pseudocode for the most important parts of the POG algorithm. Algorithm 2 specifies GT-CFR, the core of POG's sound game-theoretic search that scales to large perfect information games introduced in Section 10.3.2. Algorithm 3 presents how GT-CFR is used during selfplay that generates training examples for the neural network, this part was covered in Section 10.3.4. Hyperparameters used in selfplay are specified in Table F.2.

When POG plays against an opponent the search tree is rebuilt also for the opponent's actions (as discussed in Section F.2.1). This way, POG reasons about the opponent's behavior since it directly influences the belief distribution for the current state  $\beta$  where POG is to act.

Note that unlike AlphaZero, POG currently starts its search procedure from scratch. That

Hyperparam	Symbol	Chess	Go	Scot. Yard	HUNL
Batch size		2048	2048	1024	1024
Optimizer		sgd	sgd	sgd	adam
Initial learning rate (LR)	$\alpha_{init}$	0.1	0.02	0.1	0.0001
LR decay steps	$T_{decay}$	40k	200k	2M	2M
LR decay rate	$d$	0.8	0.1	0.5	0.5
Policy head weight	$w_p$	1	1	0.05	0.01
Value head weight	$w_v$	0.25	0.5	1	1
Replay buffer size		50M	50M	1M	1M
Max grad updates per example		1	0.2	5	10
TD(1) target sample probability	$p_{td1}$	0	0.2	0	0
Queries per search	$q_{search}$	1	0	0.3	0.9
Recursive queries per search	$q_{recursive}$	0.2	0	0.1	0.1
Selfplay uniform policy mix	$\epsilon$	0	0	0	0.1
Resign enabled		True	True	False	False
Resign threshold	$resign\_threshold$	-0.9	-0.9	-	-
Min ratio of games without resign	$p_{no\_resign}$	0.2	0.2	-	-
Greedy play after move	$moves_{greedy\_after}$	30	30	never	never
Max moves in one episode	$moves_{max}$	512	722	unlim.	unlim.
Prior softmax temperature	$T_{prior}$	1.5	1.5	1	1

Table F.2: Hyperparameters for each game.

is, the previous computation only provides invariants for the next resolving step. AlphaZero rather warm-starts the MCTS process by initializing values and visit counts from the previous search. For POG, this would also require warm-starting CFR. And while possible [Brown and Sandholm, 2016], there is no warm-starting in the current implementation of POG.

## F.2.6 Implementation

POG is implemented as a distributed system with decoupled actor and trainer jobs. Each actor runs several parallel games and the neural network evaluations are batched for better accelerator utilization. The networks were implemented using TensorFlow [Abadi et al., 2016].

## F.2.7 Poker Betting Abstraction

There are up to 20000 possible actions in no-limit Texas hold’em. To make the problem easier, AI agents are typically allowed to use only a small subset of these [Moravčík et al., 2017, Brown et al., 2018b, 2020, Zarick et al., 2020]. This process of selecting a set of allowed actions for a given poker state is called betting abstraction. Even using betting abstraction the players are able to maintain strong performance in the full game [Moravčík et al., 2017, Brown et al., 2020, Zarick et al., 2020]. Moreover, the local best response

---

**Algorithm 2** Growing Tree CFR. Note that GT-CFR is logging all neural net queries it does since they might be used later in training.

---

**procedure** GT-CFR( $\mathcal{L}^0, \beta, s, c$ )

- ▷  $\mathcal{L}^0$  — a tree including  $\beta$  build as described in Sec. F.2.1.
- ▷  $\beta$  — a public belief state under which the new nodes will be added.
- ▷  $s, c$  — total number of expansion simulations and number of simulations per CFR update.

**for**  $i \in \{0, 1, \dots, \frac{s}{c} - 1\}$  **do**

- CFR( $\mathcal{L}^i, \lceil \frac{1}{c} \rceil$ )      ▷ Store average policy and counterfactual values in the tree.
- $\mathcal{L}^{i+1} \leftarrow \text{GROW}(\mathcal{L}^i)$

▷ Return counterfactual values and average policy from CFR and all NN calls.

**return**  $\mathbf{v}, \mathbf{p}, nn\_queries$

**procedure** GROW( $\mathcal{L}, \beta$ )

**for**  $i \in \{0, 1, \dots, \lceil c \rceil - 1\}$  **do**

- $path \leftarrow \text{SAMPLEPATHDOWNTHETREE}(\mathcal{L}, \beta)$       ▷ The path starts at  $\beta$ .
- ADDTOPKCHILDREN( $\mathcal{L}, path, k$ )      ▷ Choice of  $k$  is discussed in Sec. 10.3.2.
- UPDATEVISITCOUNTSUP( $\mathcal{L}, path$ )

**return**  $\mathcal{L}$

---

evaluation [Lisý and Bowling, 2017b] suggests that there is not an easy exploit for such simplification as long as the agent is able to see full opponent actions [Moravčík et al., 2017].

We use a betting abstraction in the Player of Games to speed up the training and simplify the learning task. Our agent’s action set was limited to just 3 actions: fold (give up), check/call (match the current wager) and bet/raise (add chips to the pot). To improve generalization we used stochastic betting size similarly to ReBeL [Brown et al., 2020]. The single allowed bet/raise size is randomly uniformly selected at the start of each poker hand from the interval  $(0.5, 1.0) * pot\_size$ . This amount is anecdotally similar to one used by human players and had also good performance in our experiments. The same random selection was used in both training and evaluation.

As in [Brown et al., 2020], we have also randomly varied number stack size (number of chips available to the players) at the start of the each round during the training. This number stays fixed during evaluation.

## F.3 Evaluation Details and Additional Experimental Results

### F.3.1 Description of Leduc poker

Leduc is a simplified poker game with two rounds and a 6-card deck in two suits. Each player initially antes a single chip to play and obtains a single private card and there are three actions: fold, call and raise. There is a fixed bet amount of 2 chips in the first round and 4 chips in the second round, and a limit of two raises per round. After the first round, a single public card is revealed. A pair is the best hand, otherwise hands are ordered by their high card (suit is irrelevant). A player’s reward is their gain or loss in chips after the game.

---

**Algorithm 3** Sound Self-play

---

**procedure** SELFPLAY

Get initial history state  $w \leftarrow w^{INIT}$  and corresponding public state  $\beta$

▷ Decide whether the game has to be played till the end.

$do\_not\_resign \leftarrow$  coin flip with probability  $p_{no\_resign}$

**while**  $w$  is not terminal AND played less than  $moves_{max}$  **do**

**if** chance acts in  $w$  **then**

$a \leftarrow$  uniform random action.

**else**

    ▷ POG acts for all non-chance players.

$v_w, \pi_w^{controller} \leftarrow$  POGSELFPLAYCONTROLLER( $w$ )

**if**  $v_w < resign\_threshold$  AND  $not(do\_not\_resign)$  **then**

      ▷ Don't waste compute on already decided game.

**return**

    ▷ Mix controller's policy with uniform prior to encourage exploration.

$\pi_w^{selfplay} \leftarrow (1 - \epsilon) \cdot \pi_w^{controller} + \epsilon \cdot \pi^{uniform}$

**if** moves played  $< moves_{greedy\_after}$  **then**

$a \leftarrow$  sample action from  $\pi_w^{selfplay}$

**else**

$a \leftarrow \arg \max \pi_w^{selfplay}$

$w \leftarrow$  apply action  $a$  on state  $w$

▷ Sampling states with TD(1) targets.

**for** each belief state  $\beta \in$  played trajectory  $tr$  **do**

**if** uniform random sample from unit interval  $< p_{td1}$  **then**

$v \leftarrow$  outcome of  $tr$  assigned to information state visited in  $\beta$

$p \leftarrow$  policy used in  $\beta$

$replay\_buffer.append(\langle \beta, (v, p) \rangle)$

**procedure** POGSELFPLAYCONTROLLER( $w$ )

$\beta \leftarrow$  public state including  $w$

$\mathcal{L} \leftarrow$  the tree including  $\beta$  build as described in Sec. F.2.1.

$v, p \leftarrow$  TRAINING-GT-CFR( $\mathcal{L}$ )

**return**  $v(w), p(w)$

**procedure** TRAINING-GT-CFR( $\mathcal{L}$ )

$v, p, nn\_queries \leftarrow$  GT-CFR( $\mathcal{L}$ )

$queries \leftarrow$  Pick on average  $q_{search}$  neural net queries  $\beta$  from  $nn\_queries$ .

$queries\_to\_solve.extend(queries)$

**return**  $v, p$

**procedure** QUERY SOLVER

**for**  $\beta \leftarrow queries\_to\_solve.pop()$  **do**

$v, p, nn\_queries \leftarrow$  GT-CFR( $\beta$ )

    ▷ Send the example to the trainer.

$replay\_buffer.append(\langle \beta, (v, p) \rangle)$

    ▷ Create recursive queries.

$queries \leftarrow$  Pick on average  $q_{recursive}$  neural net queries  $\beta$  from  $nn\_queries$ .

$queries\_to\_solve.extend(queries)$

---

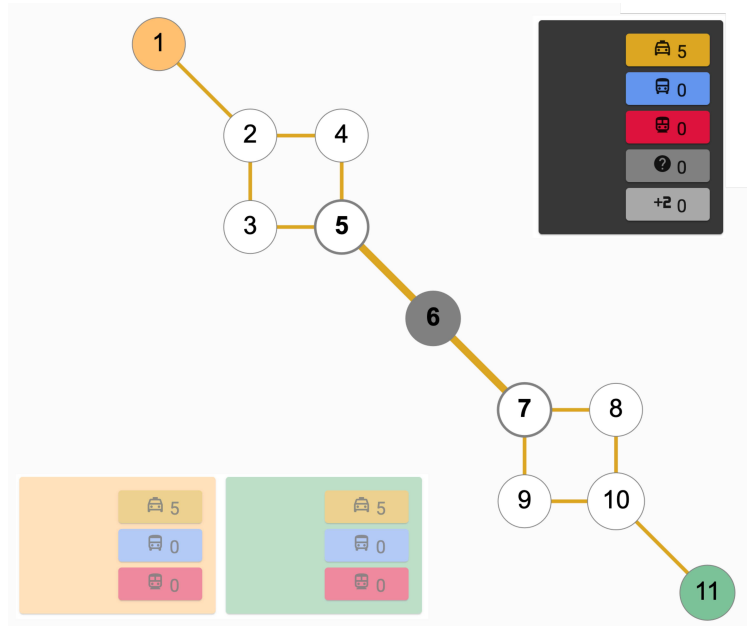


Figure F.4: Initial situation on the glasses map for Scotland Yard. Mr. X starts at station 6 while the two detectives start at stations 1 and 11. All of them have 5 taxi cards (all edges in this map are of the same type) and the game is played for 5 rounds.

### F.3.2 Custom Glasses Map for Scotland Yard

Figure F.4 shows the layout and describes the custom “glasses” map for Scotland Yard.

### F.3.3 Full Results of Go Agent Tournament

Full performance results of the Go tournament are shown in Table F.3.

### F.3.4 Reinforcement Learning and Search in Imperfect Information Games

In this section, we provide some experimental results showing that common RL and widely-used search algorithms can produce highly exploitable strategies, even in small imperfect information games where exploitability is computable exactly. In particular, we show how exploitable Information Set Monte Carlo Tree Search is in Leduc poker, as well as three standard RL algorithms (DQN, A2C and tabular Q-learning) in both Kuhn poker and Leduc poker using OpenSpiel [Lanctot et al., 2019].

#### Information Set Monte Carlo Tree Search

Information Set Monte Carlo Tree Search (IS-MCTS) is a search method that, at the start of each simulation, first samples a world state—consistent with the player’s information state—and uses it for the simulation [Cowling et al., 2012]. Reward and visit count statistics are aggregated over information states so that players base their decisions only on their information states rather than on private information inaccessible to them.

Figure F.4 shows the exploitability of a policy obtained by running separate independent IS-MCTS searches from each information state in the game, over various parameter values. The lowest exploitability of IS-MCTS we found among this sweep was **465 mbb/h**.



Agent	Rel. Elo
AlphaZero(s=16k, t=800k)	+3139
AlphaZero(s=16k, t=400k)	+3021
AlphaZero(s=8k, t=800k)	+2875
AlphaZero(s=8k, t=400k)	+2801
AlphaZero(s=4k, t=800k)	+2643
AlphaZero(s=16k, t=200k)	+2610
AlphaZero(s=4k, t=400k)	+2584
AlphaZero(s=2k, t=800k)	+2451
AlphaZero(s=8k, t=200k)	+2428
AlphaZero(s=2k, t=400k)	+2353
AlphaZero(s=4k, t=200k)	+2234
AlphaZero(s=800, t=800k)	+2099
AlphaZero(s=16k, t=100k)	+2088
AlphaZero(s=2k, t=200k)	+2063
AlphaZero(s=800, t=400k)	+2036
<b>PoG(s=16k, c=10)</b>	<b>+1970</b>
AlphaZero(s=8k, t=100k)	+1940
<b>PoG(s=8k, c=10)</b>	<b>+1902</b>
AlphaZero(s=800, t=200k)	+1812
<b>PoG(s=4k, c=10)</b>	<b>+1796</b>
AlphaZero(s=4k, t=100k)	+1783
<b>PoG(s=2k, c=10)</b>	<b>+1672</b>
AlphaZero(s=2k, t=100k)	+1618
<b>PoG(s=800, c=1)</b>	<b>+1426</b>
AlphaZero(s=800, t=100k)	+1360
Pachi(s=100k)	+869
Pachi(s=10k)	+231
GnuGo(l=10)	+0

Table F.3: Full Go results. Elo of GnuGo with a single thread and 100ms thinking time was set to be 0. AlphaZero(s=16k, t=800k) refers to 16000 search simulations.

### Standard RL algorithms in Imperfect Information Games

As imperfect information games generally need stochastic policies to achieve an optimal strategy, one might wonder how exploitable standard RL algorithms are in this class of games. To test this, we trained three standard RL agents: DQN, policy gradient (A2C) and tabular Q-learning. We used MLP neural networks in DQN and A2C agents. Table F.5 shows the hyper parameters we swept over to train these RL agents.

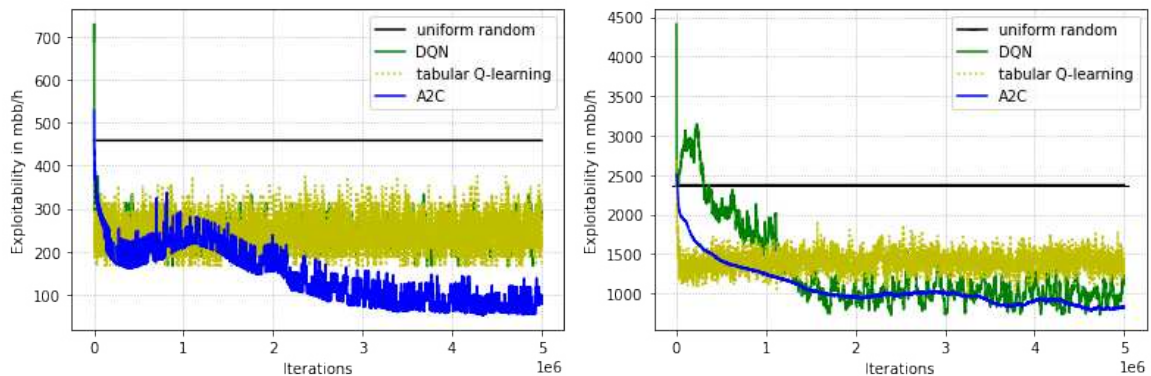
In Kuhn poker, the best performing A2C agent converges to exploitability of 52 mbb/h, and tabular Q-learning and DQN agents converge to around 250 mbb/h. Similarly, in Leduc poker, the best performing A2C agent converges to exploitability of 78 mbb/h, tabular Q-learning and DQN agents converge to about 1300 mbb/h and 900 mbb/h respectively. Figure F.5 shows the exploitability of RL agents in Kuhn poker and Leduc poker.

<b>Num. Sims</b>	<b>UCT const. (<math>C</math>)</b>	<b>Expl. (mvd)</b>	<b>Expl. (mvis)</b>	<b>Expl. (mval)</b>
10	1.0	2168	2449	2173
10	2.0	2058	2408	2341
10	5.0	1902	2615	2517
10	10.0	1738	2555	2360
10	13.0	1799	2517	2598
10	20.0	1821	2830	2349
10	26.0	1888	2861	2669
100	1.0	1489	1509	1333
100	2.0	1404	1587	1395
100	5.0	1239	1145	1094
100	10.0	1213	1195	1245
100	13.0	1218	1292	1227
100	20.0	1350	1456	1342
100	26.0	1448	1747	1568
1000	1.0	1323	1218	1177
1000	2.0	1069	1212	864
1000	5.0	699	778	681
1000	10.0	697	601	632
1000	13.0	741	759	744
1000	20.0	859	962	991
1000	26.0	966	1029	1057
10000	1.0	1348	948	1134
10000	2.0	911	877	763
10000	5.0	516	582	538
10000	10.0	490	485	480
10000	13.0	511	465	470
10000	20.0	572	505	505
10000	26.0	631	575	570

Table F.4: Average exploitability (in mbb/h) over five policy constructions obtained by independent searches of IS-MCTS runs at each information state in Leduc Poker. The parameter  $C$  is the value of the UCT exploration constant. The final policy is obtained either by normalizing the visit counts (mvd), choosing the action with maximum visits (mvis), or choosing the action with the maximal Monte Carlo value estimate (mval).

Parameter	DQN	Tabular Q-Learning	A2C
Learning rate (lr)	1e-1, 1e-2, 1e-3, 1e-4	NA	Actor lr: 1e-3, 1e-4, 1e-5 Critic lr: 1e-2, 1e-3
Decaying exploration rate	1., 0.8, 0.5, 0.2, 0.1	NA	NA
Replay buffer size	100, 1000, 10000, 100000	NA	NA
Hidden layer size	'32', '64', '128', '32, 32', '64, 64'	NA	'32', '64', '128', '32, 32', '64, 64'
Num. of critic updates before every actor update	NA	NA	4, 8, 16
Step size	NA	0.1, 0.2, 0.5, 0.8, 1.0	

Table F.5: Hyper parameters swept over in each RL algorithm.



(a) Exploitability in 2-player Kuhn poker

(b) Exploitability in 2-player Leduc poker

Figure F.5: Comparing performance of DQN, A2C, tabular Q-learning and uniform random policy in (a) Kuhn poker and (b) Leduc poker

## F.4 Scotland Yard example

Figure F.6 describes an example situation from Scotland Yard.

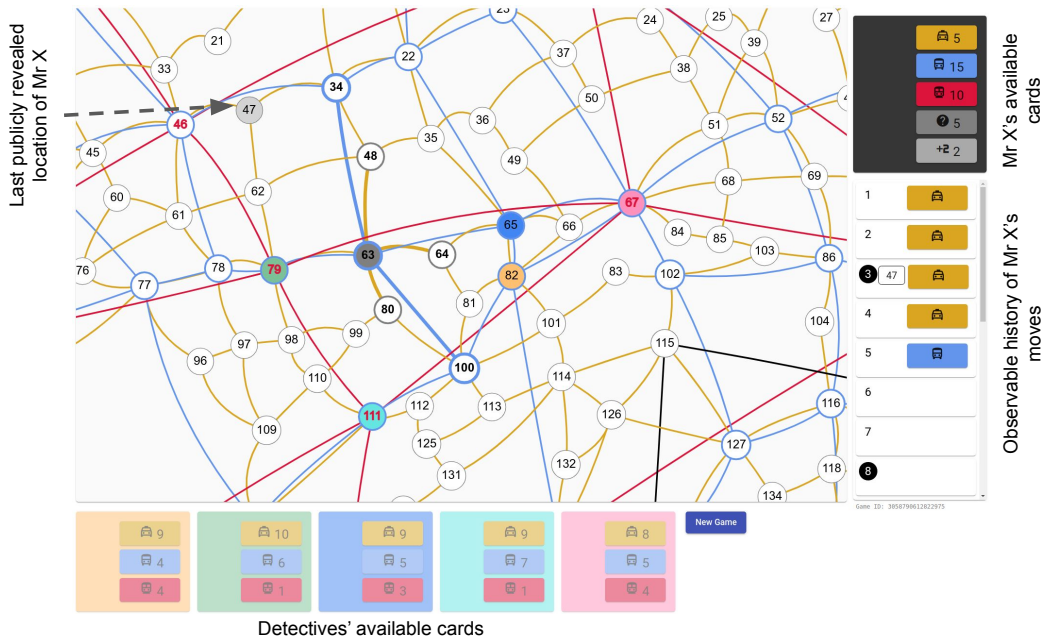


Figure F.6: An example situation in Scotland Yard. The centre shows a game map abstracted as a graph with different types of edges (yellow for taxi, blue for bus, red for subway and black for boat). Both the detectives and Mr. X have corresponding cards that allow them to move edges of the same type. In the current situation, the game is in round 5 and Mr. X is to act. He is in location 63, however the detectives (locations 65, 67, 79, 82 and 111) saw his exact location last time in round 3 when he was in location 47 (he will have to reveal himself again in rounds 8, 13, 18 and the final round 24). Since then the detectives observed just the type of the edge Mr. X moved along (in this case taxi and bus) and they have to "guess" where he is and subsequently where he is going to be in order to catch him. While Mr. X is hidden locations of detectives are public. In single move Mr. X can get to locations 48, 64 or 80 by taxi and to locations 34 or 100 by bus. He can use a card of the corresponding colour (taxi or bus) however that will give unwanted hint to the detectives. He can also use "?" card (he has 5 at the moment) that hides his mode of transport, therefore it increases detectives' uncertainty about his location. Finally, he can use double move card (he has only 2) that would allow him to do two moves instead of one.

## F.5 Proofs of Theorems

There are three substantive differences between the PoG algorithm and DeepStack. First, PoG uses a growing search tree, rather than using a fixed limited-lookahead tree. Second, the PoG search tree may depend on the observed chance events. Finally, PoG uses a continuous self-play training loop operating throughout the entire game, rather than the stratified bottom-up training process used by DeepStack. We address each of these differences below, in turn, after considering how to describe an approximate value function for search in imperfect information games.

### F.5.1 Value Functions for Subgames

Like DeepStack, the PoG algorithm uses a value function, so the quality of its play depends on the quality of the value function. We will describe a value function in terms of its distance to a strategy with low regret. We start with some value and regret definitions that are better suited to subgames.

Consider some strategy profile  $\pi$  which is a tuple containing a strategy for each player, public tree subgame  $S$  rooted at public state  $s_{\text{pub}}$  with player range vectors  $B_i[s_i \in \mathcal{S}_i(s_{\text{pub}})] := P_i(s_i|\pi)$ . First, note that we can re-write counterfactual value  $v$  so that it depends only on  $B$  and  $\pi$  restricted to  $S$ , with no further dependence on  $\pi$ . Let  $s_i$  be a Player  $i$  information state in  $\mathcal{S}_i(s_{\text{pub}})$ , and  $q$  be the opponent of Player  $i$ , then:

$$\begin{aligned} v^{B, \pi^S}(s_i) &:= \sum_{h \in I(s_i)} \sum_{z \sqsupset h} B_q[s_q(h)] P_c(h) P(z|h, \pi^S) u_i(z) \\ &= \sum_{h \in I(s_i)} \sum_{z \sqsupset h} P_{-p}(h|\pi) P(z|h, \pi) u_i(z) = v^\pi(s_i) \end{aligned}$$

We can write a number of quantities in terms of the best-response value at information state  $s_i$ :

$$BV^{B, \pi^S}(s_i) := \max_{\pi_i^*} v^{B, \pi^S \leftarrow \pi_i^*}(s_i)$$

where  $\pi \leftarrow \pi'$  is the strategy profile constructed by replacing action probabilities in  $\pi$  with those in  $\pi'$ . The value function is a substitute for an entire subgame strategy profile, so the regret we are interested in is player  $i$ 's full counterfactual regret [Zinkevich et al., 2007] at  $s_i$ , which considers all possible strategies within subgame  $S$ :

$$R_{s_i}^{\text{full}}(B, \pi^S) := BV^{B, \pi^S}(s_i) - v^{B, \pi^S}(s_i)$$

With these definitions in hand, we can now consider the quality of a value function  $f$  in terms of a regret bound  $\epsilon$  and value error  $\xi$ . Recall that  $f$  maps ranges  $B$  and public state  $s_{\text{pub}}$  to a vector of values  $\tilde{v}(s_i)$  for each player  $i$ .

First, we consider versions of the regret bound and value error which are parameterised by a strategy  $\pi$ . There is some associated bound  $\epsilon(\pi)$  on the sum of regrets across all information states at any subgame, valid for both players.

$$\epsilon(\pi) := \max_B \max_{s_{\text{pub}}} \max_i \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} R_{s_i}^{\text{full}}(B, \pi)$$

There is also some bound  $\xi_f(\pi)$  on the distance between  $f(s_{\text{pub}}, B)$  and the best-response values to  $\pi$ .

$$\xi_f(\pi) := \max_B \max_{s_{\text{pub}}} \max_i \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} |f(s_{\text{pub}}, B)[s_i] - v^{B, \pi}(s_i)|$$

We then say that  $f$  has  $\epsilon, \xi$  quality bounds if there exists some strategy  $\pi$  such that  $\epsilon(\pi) \leq \epsilon$  and  $\xi_f(\pi) \leq \xi$ . As desired, if both  $\epsilon$  and  $\xi$  are low then  $f(s_{\text{pub}}, B)$  is a good approximation of the best-response values to a low-regret strategy, for a subgame rooted at  $s_{\text{pub}}$  with initial beliefs  $B$ .

The DeepStack algorithm [Moravčík et al., 2017] used a similar error metric for value functions, but only considered zero-regret strategies. We introduce a more complicated error measure because the space of values corresponding to low-regret strategies may be much larger than the space of values corresponding to no-regret strategies. For example, consider the public subgame of a matching pennies game after the first player acts with the policy 0.501 heads, 0.499 tails. There are two first-player information states, from playing either heads or tails, with an empty first-player strategy as there are no further first player actions. Let us assume a value function  $f$  is returning the values  $[0 \ 0]$  for these two information states. How good is  $f$ , assuming we restrict our attention to this one subgame?

The unique zero-regret strategy for the second player is to play tails 100% of the time, resulting in first player counterfactual values of -1 for playing heads and 1 for playing tails. The error metric based on zero-regret strategies is therefore measuring  $|f([0.501 \ 0.499]) - [-1 \ 1]|_1$ , so that the DeepStack metric states that  $f$  has an error of 2. However,  $[0 \ 0]$  seems like a very reasonable choice: these are exactly the first player counterfactual values when the second player has a strategy of 0.5 heads, 0.5 tails, which has a regret of only 0.002 in this subgame. Rather than saying  $f$  is a poor quality value function with an error of 2 in a game with utilities in  $[-1, 1]$ , we can now say  $f$  is a great 0.002, 0 value function which exactly describes a low-regret strategy.

The new quality metric also addresses an issue the old DeepStack metric had with discontinuities in the underlying 0-regret value functions. This means that the space of functions with a low DeepStack error metric may not be well suited for learning from data. Continuing with the previous example, if we shifted  $B$  slightly to be 0.499 heads and 0.501 tails for the first player, the unique 0-regret strategy in the subgame flips to playing tails 0% of the time, while the uniform random strategy is still a low-regret strategy for this subgame. In this example, a function can only have a low error with the DeepStack metric if it accurately predicts the values everywhere around the discontinuity at 0.5 heads and 0.5 tails, whereas the new metric can avoid this discontinuity by picking an  $\epsilon > 0$ . More generally, for any constant  $c$ , the objective  $\epsilon + c\xi$  is a continuous function in  $B$ , making it a potentially more attractive learning target than the discontinuous function defined by exact Nash equilibrium values, and which matches a learning procedure based on approximately solving example subgames.

## F.5.2 Growing Trees

One major step in showing soundness of the PoG algorithm is demonstrating that Growing Tree CFR (GT-CFR) can approximately solve games. As a quick recap, GT-CFR is a variant of the CFR algorithm [Zinkevich et al., 2007] that uses limited lookahead and a value function, storing values within a tree that grows over time, in a fashion similar to UCT [Kocsis and Szepesvári, 2006]. We use this new algorithm as a component to solve the problems that the PoG algorithm sets up. At every non-terminal public leaf state  $s_{\text{pub}}$  of the lookahead tree, GT-CFR uses estimated counterfactual values  $\tilde{v}$ , generated from a value function  $f(s_{\text{pub}}, B)$  with player ranges  $B$  induced by Bayes' rule at  $s_{\text{pub}}$  for the current strategy profile  $\pi$ .

Like DeepStack, PoG has two steps which involve solving subgames of the original game. One of the steps is the re-solving step used to play through a game, where we solve a modified subgame based on constraints on opponent values and beliefs about our possible private information, in order to get our policy and new opponent values. The other step is only in the training loop, where we are solving a subgame with fixed beliefs for both players, in order to get values for both players. While the (sub)games for these two cases are slightly different, they are both well-formed games and we can find an approximate Nash equilibrium using GT-CFR.

When running GT-CFR, even though a policy is explicitly defined only at information states in the lookahead tree  $\mathcal{L}$ , at each iteration  $t$  there is implicitly some complete strategy profile  $\pi^t$ . For any information state  $s$  in  $\mathcal{L}$  which is not a leaf,  $\pi^t(s)$  is explicitly defined by the regret-matching policy. For all other  $s$  – either a leaf of  $\mathcal{L}$  or outside of the lookahead tree –  $\pi^t(s)$  is defined by the  $\epsilon$ -regret subgame strategy profile  $\pi^{*,S}$  associated with the value function's  $\epsilon, \xi$  quality bounds. Note that this  $\pi^t$  only exists as a concept which is useful for theoretical analysis: GT-CFR does not have access to the probabilities outside of its lookahead tree, only a noisy estimate of the associated counterfactual values provided by

the value function.

**Lemma F.1.** *Let  $p$  and  $q$  be vectors in  $[0, 1]^n$ , and  $v$  and  $w$  be vectors in  $\mathbb{R}^n$  such that  $v[i] > w[i]$  for all  $i$ . Then  $p \cdot v - q \cdot w \leq \mathbf{1} \cdot (v - w) + p \cdot w - q \cdot w$*

*Proof.*

$$\begin{aligned} p \cdot v - q \cdot w &= p \cdot v - p \cdot w + p \cdot w - q \cdot w \\ &= p \cdot (v - w) + p \cdot w - q \cdot w \\ &\leq \mathbf{1} \cdot (v - w) + p \cdot w - q \cdot w \end{aligned}$$

□

**Lemma F.2.** *Let  $p$  and  $q$  be vectors in  $[0, 1]^n$ , and  $v$  and  $w$  be vectors in  $\mathbb{R}^n$  such that  $\sum_{i=1}^n |v[i] - w[i]| \leq \xi$ . Then  $(p - q) \cdot v \leq \xi + (p - q) \cdot w$ .*

*Proof.*

$$\begin{aligned} (p - q) \cdot v &= (p - q) \cdot (v - w) + (p - q) \cdot w \\ &\leq \sum_{i=1}^n |(p[i] - q[i])(v[i] - w[i])| + (p - q) \cdot w \\ &\leq \sum_{i=1}^n |v[i] - w[i]| + (p - q) \cdot w \\ &\leq \xi + (p - q) \cdot w \end{aligned}$$

□

In GT-CFR, the depth-limited public tree used for search may change at each iteration. Let  $\mathcal{L}^t$  be the public tree at time  $t$ . For any given tree  $\mathcal{L}$ , let  $\mathcal{N}(\mathcal{L})$  be the interior of the tree: all non-leaf, non-terminal public states. The interior of the tree is where regret matching is used to generate a policy, with regrets stored for all information states in interior public states. Let  $\mathcal{F}(\mathcal{L})$  be the frontier of  $\mathcal{L}$ , containing non-terminal leaves, and  $\mathcal{Z}(\mathcal{L})$  be the terminal public states. GT-CFR uses the value function at all public states in the frontier, receiving noisy estimates  $\tilde{v}(s)$  of the true counterfactual values  $v(s)$ . We will distinguish between the true regrets  $R_s^T$  computed from the entire policy, and the regret  $\tilde{R}_s^T$  computed using the estimated values  $\tilde{v}(s)$ . Given a sequence of trees across  $T$  iterations, let  $\mathcal{T}_n(s_{\text{pub}})$  be the set of maximal length intervals  $[a, b] \subseteq [1, T]$  where  $s_{\text{pub}}$  is in  $\mathcal{N}(\mathcal{L}^t)$  for all  $t \in [a, b]$ . Let  $U$  be the maximum difference in counterfactual value between any two strategies, at any information state, and  $A$  be the maximum number of actions at any information state.

**Lemma F.3.** *After running GT-CFR for  $T$  iterations starting at some initial public state  $s_0$ , using a value function with quality  $\epsilon, \xi$ , regret for the strategies satisfies the bound*

$$\begin{aligned} \sum_{s_i \in \mathcal{S}_i(s_0)} R_{s_i}^T &\leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)| (\epsilon + \xi) \\ &\quad + \sum_{s_{\text{pub}} \in \bigcup_{t=1}^T \mathcal{L}^t} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{A} \sum_{[a, b] \in \mathcal{T}_n(s_{\text{pub}})} \sqrt{|[a, b]|} \end{aligned}$$

*Proof.* Starting with the definition of regret, and noting that regrets are independently maximised in a perfect recall game, we can rearrange terms to get

$$\begin{aligned}
\sum_{s_i \in \mathcal{S}_i(s_0)} R_{s_i}^T &= \sum_{s_i \in \mathcal{S}_i(s_0)} \left( \max_{\pi_i^*} \sum_{t=1}^T v^{\pi^t \leftarrow \pi_i^*}(s_i) - \sum_{t=1}^T v^{\pi^t}(s_i) \right) \\
&= \max_{\pi_i^*} \sum_{s_i \in \mathcal{S}_i(s_0)} \left( \sum_{t=1}^T v^{\pi^t \leftarrow \pi_i^*}(s_i) - \sum_{t=1}^T v^{\pi^t}(s_i) \right) \\
&= \max_{\pi_i^*} \sum_{t=1}^T \sum_{s_i \in \mathcal{S}_i(s_0)} \left( v^{\pi^t \leftarrow \pi_i^*}(s_i) - v^{\pi^t}(s_i) \right)
\end{aligned}$$

We can rewrite the counterfactual values of information state  $s_i$  in terms of the counterfactual value of leaves and terminals of the tree.

$$\begin{aligned}
&= \max_{\pi_i^*} \sum_{t=1}^T \left( \sum_{s_{\text{pub}} \in \mathcal{F}(\mathcal{L}^t)} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) v^{\pi^t \leftarrow \pi_i^*}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right) \right. \\
&\quad \left. + \sum_{s_{\text{pub}} \in \mathcal{Z}(\mathcal{L}^t)} \sum_{z \in I(s_{\text{pub}})} \left( P_i(z | \pi^t \leftarrow \pi_i^*) v^{\pi^t}(z) - P_i(z | \pi^t) v^{\pi^t}(z) \right) \right) \quad (\text{F.1})
\end{aligned}$$

Examining part of the first term inside the sum, we can independently maximise the counterfactual values at each information state  $s_i$ . As above, this is equivalent to maximising at public state  $s_{\text{pub}}$ .

$$\begin{aligned}
&\sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) v^{\pi^t \leftarrow \pi_i^*}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right) \\
&\leq \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \max_{\pi_i^{**}} \left( P_i(s_i | \pi_i^*) v^{\pi^t \leftarrow \pi_i^{**}}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right) \\
&= \max_{\pi_i^{**}} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) v^{\pi^t \leftarrow \pi_i^{**}}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right)
\end{aligned}$$

Given that we individually maximised over each minuend, we satisfy the requirements of Lemma F.1. We can then use the value function quality bounds.

$$\begin{aligned}
&\leq \max_{\pi_i^{**}} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( v^{\pi^t \leftarrow \pi_i^{**}}(s_i) - v^{\pi^t}(s_i) \right) \\
&+ \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) v^{\pi^t}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right) \\
&\leq \epsilon + \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) v^{\pi^t}(s_i) - P_i(s_i | \pi^t) v^{\pi^t}(s_i) \right)
\end{aligned}$$

Up to this point, we have used the true counterfactual values for the current strategy profile. At leaves, however, GT-CFR only has access to the value function's noisy estimates of the true values. Applying Lemma F.2, we get

$$\leq \epsilon + \xi + \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) \tilde{v}^{\pi^t}(s_i) - P_i(s_i | \pi^t) \tilde{v}^{\pi^t}(s_i) \right)$$



Placing this back into line F.1 and collecting  $\epsilon$  and  $\xi$  terms, we have

$$\begin{aligned} \sum_{s_i \in \mathcal{S}_i(s_0)} R_{s_i}^T &\leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \max_{\pi_i^*} \sum_{t=1}^T \\ &\quad \left( \sum_{s_{\text{pub}} \in \mathcal{F}(\mathcal{L}^t)} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) \tilde{v}^{\pi^t}(s_i) - P_i(s_i | \pi^t) \tilde{v}^{\pi^t}(s_i) \right) \right. \\ &\quad \left. + \sum_{s_{\text{pub}} \in \mathcal{Z}(\mathcal{L}^t)} \sum_{z \in I(s_{\text{pub}})} \left( P_i(z | \pi^t \leftarrow \pi_i^*) v^{\pi^t}(z) - P_i(z | \pi^t) v^{\pi^t}(z) \right) \right) \end{aligned}$$

We can rearrange the sums to consider the regret contribution for each public state

$$\begin{aligned} &= \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \max_{\pi_i^*} \sum_{s_{\text{pub}} \in \bigcup_{t=1}^T \mathcal{L}^t} \\ &\quad \left( \sum_{t \text{ s.t. } s_{\text{pub}} \in \mathcal{F}(\mathcal{L}^t)} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) \tilde{v}^{\pi^t}(s_i) - P_i(s_i | \pi^t) \tilde{v}^{\pi^t}(s_i) \right) \right. \\ &\quad \left. + \sum_{t \text{ s.t. } s_{\text{pub}} \in \mathcal{Z}(\mathcal{L}^t)} \sum_{z \in I(s_{\text{pub}})} \left( P_i(z | \pi^t \leftarrow \pi_i^*) v^{\pi^t}(z) - P_i(z | \pi^t) v^{\pi^t}(z) \right) \right) \end{aligned}$$

As before we can use Lemma F.1 to separate out regrets at the interior states in  $\mathcal{N} := \mathcal{F}(\mathcal{N}(\bigcup_{t=1}^T \mathcal{L}^t))$ , which always depend only on leaves and terminals. Let  $\mathcal{L}',t$  be  $\mathcal{L}^t$  minus all public states in  $\mathcal{N}$  and any successor states.

$$\begin{aligned} &\leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \mathcal{N}} \sum_{[a,b] \in \mathcal{T}_n(s_{\text{pub}})} \sum_{s_i \in s_{\text{pub}}} \tilde{R}_{s_i}^{a,b} + \max_{\pi_i^*} \sum_{s_{\text{pub}} \in \bigcup_{t=1}^T \mathcal{L}',t} \\ &\quad \left( \sum_{t \text{ s.t. } s_{\text{pub}} \in \mathcal{F}(\mathcal{L}',t)} \sum_{s_i \in \mathcal{S}_i(s_{\text{pub}})} \left( P_i(s_i | \pi_i^*) \tilde{v}^{\pi^t}(s_i) - P_i(s_i | \pi^t) \tilde{v}^{\pi^t}(s_i) \right) \right. \\ &\quad \left. + \sum_{t \text{ s.t. } s_{\text{pub}} \in \mathcal{Z}(\mathcal{L}',t)} \sum_{z \in I(s_{\text{pub}})} \left( P_i(z | \pi^t \leftarrow \pi_i^*) v^{\pi^t}(z) - P_i(z | \pi^t) v^{\pi^t}(z) \right) \right) \end{aligned}$$

Note that the states which were separated out are now effectively terminals in smaller trees. We can repeat this process until regrets for all public states have been separated out.

$$\leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \bigcup_{t=1}^T \mathcal{L}^t} \sum_{[a,b] \in \mathcal{T}_n(s_{\text{pub}})} \sum_{s_i \in s_{\text{pub}}} \tilde{R}_{s_i}^{a,b}$$

Finally, from bounds on regret-matching[Hart and Mas-Colell, 2000],

$$\leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \bigcup_{t=1}^T \mathcal{L}^t} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{A} \sum_{[a,b] \in \mathcal{T}_n(s_{\text{pub}})} \sqrt{|[a,b]|}$$

□

Note that the form of Lemma F.3 implies that regret might not be sub-linear if public states are repeatedly added and removed from the lookahead tree. If we only add states and never remove them, however, we get a standard CFR regret bound plus error terms for the value function.

**Theorem F.4.** *Assume the conditions of Lemma F.3 hold, and public states are never removed from the lookahead tree. Then*

$$R_i^{T,full} \leq \sum_{t=1}^T |\mathcal{F}(\mathcal{L}^t)|(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}^T)} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{AT}$$

*Proof.* This follows from Lemma F.3, noting that the interior of  $\mathcal{L}^t$  monotonically grows over time.  $\square$

### F.5.3 Self-play Values as Re-solving Constraints

By using a value network in solving, we lose the ability to compute our opponent's counterfactual best response values to our average strategy [Šustr et al., 2019]. It is easy to track the opponent's average self-play value across iterations of a CFR variant, but using these values as re-solving constraints does not trivially lead to a bound on exploitability for the re-solved strategy. We show here that average CFR self-play values lead to reasonable, controllable error bounds in the context of continual re-solving. We will use  $(x)^+$  to mean  $\max\{x, 0\}$ . For simplicity, we will also assume that the subgame that is being re-solved is in the GT-CFR lookahead tree for all iterations.

**Theorem F.5.** *Assume we have some average strategy  $\bar{\pi}$  generated by  $T$  iterations of GT-CFR solver using a value function with quality  $\epsilon, \xi$ , with final lookahead tree  $\mathcal{L}^T$  where public states were never removed from the lookahead tree, and a final average regret  $R_i^T$  for the player of interest. Further assume that we have re-solved some public subgame  $S$  rooted public state  $s_{\text{pub}}$ , using the average counterfactual values  $\bar{v}(s_o) := \frac{1}{T} \sum_{t=1}^T v^{\pi^t}(s_o)$  as the opt-out values in the re-solving gadget. Let  $\pi^S$  be the strategy generated from the re-solving game, with some player and opponent average regrets  $\bar{R}_i^S$  and  $\bar{R}_o^S$ , respectively. Then*

$$\begin{aligned} BV_o^{\bar{\pi} \leftarrow \pi^S} - BV_o^{\bar{\pi}} &\leq (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + \bar{R}_i \\ &\quad + 2(\max_t |\mathcal{F}(\mathcal{L}_{s_{\text{pub}}}^t)|)(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}_{s_{\text{pub}}}^T)} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{\frac{A}{T}} \end{aligned}$$

*Proof.* The general outline of the proof has two parts, both asking the question "how much can the opponent best response value increase?" As in Lemma 4 of [Moravčík et al., 2017], we can consider breaking the error in re-solving opt-out values into separate underestimation and overestimation terms. The first part of this proof is a bound that takes into account the re-solving solution quality, and how much the average values underestimate the best response to the average. This underestimation is bounded by the opponent regret at a subgame, which requires the solving algorithm to have low regret everywhere in the game: low regret for the opponent does not directly imply that the opponent has low regret in portions of the game that they do not play. The second part of the proof is placing a bound on the overestimation, using the player's regret rather than the opponent's regret.

We start by noting that from the opponent player  $o$ 's point of view, we can replace an information set  $s_o$  with a terminal that has utility  $BV_o^{\bar{\pi}}(s_o)$ , and the best response utility  $BV_o^{\bar{\pi}, s_o \leftarrow BV_o^{\bar{\pi}}(s_o)}$  in this modified game will be equal to  $BV_o^{\bar{\pi}}$ . We can extend this to the entire subgame  $S$ , replacing each  $s_o$  with a terminal giving the opponent the best response value:  $BV_o^{\bar{\pi}, S \leftarrow BV_o^{\bar{\pi}}(S)} = BV_o^{\bar{\pi}}$ . Using this notation, we can rewrite  $BV_o^{\bar{\pi} \leftarrow \pi^S}$ :

$$\begin{aligned} &BV_o^{\bar{\pi} \leftarrow \pi^S} - BV_o^{\bar{\pi}} \\ &= BV_o^{\bar{\pi}, S \leftarrow BV_o^{\bar{\pi}}(S)} - BV_o^{\bar{\pi}} \end{aligned}$$

Next, note that  $BV^{\pi^S}(s_o)$ , the opponent's counterfactual best response to the re-solved subgame strategy  $\pi^S$  at any  $s_o$  at the root of  $S$ , is no greater than the value of  $\max\{BV^{\pi^S}(s_o), \bar{v}(s_o)\}$ , the value of  $s_o$  within the re-solving game before the gadget where the opponent has decision to opt-out for a fixed value  $\bar{v}(s_o)$ . That is, adding an extra opponent action which terminates the game never decreases the opponent's best response utility. Extending this to the entire subgame  $S$  again, we get

$$\begin{aligned} & \mathbf{BV}_o^{\bar{\pi}, S \leftarrow BV^{\pi^S}(S)} - \mathbf{BV}_o^{\bar{\pi}} \\ & \leq \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\pi^S}(S), \bar{v}\}} - \mathbf{BV}_o^{\bar{\pi}} \end{aligned} \quad (\text{F.2})$$

From Lemma 1 of [Moravčík et al., 2017], the game value of a re-solving game with opt-out values  $\bar{v}(s_o)$  is  $U_{\bar{v}, \bar{\pi}}^S + \sum_{s_o \in \mathcal{S}_o(s_{\text{pub}})} \bar{v}(s_o)$ , for some underestimation error on the opt-out values that is given by

$$U_{\bar{v}, \bar{\pi}}^S := \min_{\pi^* S} \sum_{s_o \in \mathcal{S}_o(s_{\text{pub}})} (BV^{\bar{\pi} \leftarrow \pi^* S}(s_o) - \bar{v}(s_o))^+$$

Given the re-solving regrets, we have  $\mathbf{BV}_o^{\pi^S} \leq (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + U_{\bar{v}, \bar{\pi}}^S + \sum_{s_o \in \mathcal{S}_o(s_{\text{pub}})} \bar{v}(s_o)$ . Because  $\mathbf{BV}_o^{\bar{\pi}, s_o \leftarrow w + \epsilon} \leq \mathbf{BV}_o^{\bar{\pi}, s_o \leftarrow w} + \epsilon$  for  $\epsilon \geq 0$ , we can use this inequality to update Equation F.2. That is, there is some component-wise non-negative vector  $\epsilon$  such that  $BV^{\pi^S}(\tilde{S}) = \bar{v}(\cdot) + \epsilon$  and  $\epsilon \cdot \mathbf{1} \leq (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + U_{\bar{v}, \bar{\pi}}^S$ , so that

$$\begin{aligned} & \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\pi^S}(S), \bar{v}\}} - \mathbf{BV}_o^{\bar{\pi}} \\ & = \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v} + \epsilon} - \mathbf{BV}_o^{\bar{\pi}} \\ & \leq \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v}} + \epsilon \cdot \mathbf{1} - \mathbf{BV}_o^{\bar{\pi}} \\ & \leq \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v}} + (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + U_{\bar{v}, \bar{\pi}}^S - \mathbf{BV}_o^{\bar{\pi}} \end{aligned} \quad (\text{F.3})$$

Looking at  $U_{\bar{v}, \bar{\pi}}^S$ , we note that this minimum is no greater than the case when  $\pi^* = \bar{\pi}$ . The difference  $BV^{\bar{\pi} \leftarrow \pi^* S}(s_o) - \bar{v}(s_o)$  is the average full counterfactual regret  $R_{s_o}$  of strategy  $\bar{\pi}$  at  $s_o$ . Restricting our attention to  $\mathcal{L}_{s_{\text{pub}}}^t$ , the portion of the lookahead tree restricted to  $s_{\text{pub}}$  and its descendants, Theorem F.4 gives us a bound on  $U_{\bar{v}, \bar{\pi}}^S$  and we can update Equation F.3

$$\begin{aligned} & \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v}} + (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + U_{\bar{v}, \bar{\pi}}^S - \mathbf{BV}_o^{\bar{\pi}} \\ & \leq \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v}} - \mathbf{BV}_o^{\bar{\pi}} \end{aligned} \quad (\text{F.4})$$

$$+ (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ \max_t |\mathcal{F}(\mathcal{L}_{s_{\text{pub}}}^t)|(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}_{s_{\text{pub}}}^T)} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{\frac{A}{T}}$$

Looking at just the difference in opponent counterfactual best response values, we can again get an upper bound by giving the opponent the choice at all information sets at the root of subgame  $S$  of playing a best response against the unmodified strategy  $\bar{\pi}$  to get value

$BV^{\bar{\pi}}(S)$ , or opting out to get value  $\bar{v}$ .

$$\begin{aligned}
& \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \bar{v}} - \mathbf{BV}_o^{\bar{\pi}} \\
& \leq \mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \mathbf{BV}_o^{\bar{\pi}} \\
& = (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) - (\mathbf{BV}_o^{\bar{\pi}} - \bar{v}_o) \\
& \leq (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) - (\mathbf{BV}_o^{\pi^*} - \bar{v}_o) \\
& = (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) - (v_o^{\pi^*} - \bar{v}_o) \\
& = (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) + (v_i^{\pi^*} - \bar{v}_i) \\
& \leq (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) + (\mathbf{BV}_i^{\bar{\pi}} - \bar{v}_i) \\
& = (\mathbf{BV}_o^{\bar{\pi}, S \leftarrow \max\{BV^{\bar{\pi}}(S), \bar{v}\}} - \bar{v}_o) + \bar{R}_i
\end{aligned} \tag{F.5}$$

The difference of the first two terms is the regret in the opt-out game described above, where we have lifted each iteration strategy  $\pi^t$  into this game by never selecting the opt-out choice. Consider the immediate counterfactual regret  $\tilde{R}^T(s_o)$  in this situation for any information state  $s_o$  in this augmented game. Writing this in terms of the original immediate counterfactual regret  $R^T(s_o)$  and the opt-out value, we get

$$\begin{aligned}
\tilde{R}^T(s_o) &= \max\{T(\bar{v}[s_o] - \bar{v}[s_o]), R^T(s_o)\} \\
&= (R^T(s_o))^+
\end{aligned}$$

Because the positive immediate regret in the opt-out game is the same as the positive regret in the original game, we can use the Theorem F.4 bound, which is composed from immediate regrets. Putting this together with Equation F.4 and Equation F.5, we get

$$\begin{aligned}
& \mathbf{BV}_o^{\bar{\pi} \leftarrow \pi^S} - \mathbf{BV}_o^{\bar{\pi}} \\
& \leq (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + \bar{R}_i \\
& \quad + 2(\max_t |\mathcal{F}(\mathcal{L}_{s_{\text{pub}}}^t)|)(\epsilon + \xi) + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}_{s_{\text{pub}}}^T)} |\mathcal{S}_i(s_{\text{pub}})| U \sqrt{\frac{A}{T}}
\end{aligned}$$

□

## F.5.4 Continual Re-solving

Continual re-solving puts GT-CFR together with re-solving the previously solved subgame. A bound on final solution quality then follows directly from applications of Theorem F.4 and Theorem F.5.

**Theorem F.6.** *Assume we have played a game using continual re-solving, with one initial solve and  $D$  re-solving steps. Each solving or re-solving step finds an approximate Nash equilibrium through  $T$  iterations of GT-CFR using a value function with quality  $\epsilon, \xi$ , public states are never removed from the lookahead tree, the maximum interior size  $\sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}^T)} |\mathcal{S}_i(s_{\text{pub}})|$  of all lookahead trees is bounded by  $N$ , the sum of frontier sizes across all lookahead trees is bounded by  $F$ , the maximum number of actions at any information sets is  $A$ , and the maximum difference in values between any two strategies is  $U$ . The exploitability of the final strategy is then bounded by  $(5D + 2) \left( F(\epsilon + \xi) + NU \sqrt{\frac{A}{T}} \right)$ .*

*Proof.* The exploitability  $\text{EXP}_0$  of the player's initial strategy from the original solve is bounded by the sum of the regrets for both players. Theorem F.4 provides regret bounds for GT-CFR, so

$$\text{EXP}_0 \leq 2 \left( F(\epsilon + \xi) + NU \sqrt{\frac{A}{T}} \right)$$

Each subsequent re-solve is operating on the strategy of the previous step, using the average values for the opt-out values. That is, the first re-solve will be updating the strategy from the initial solve, the second re-solve will be updating the subgame strategy from the first re-solve, and so on. Theorem F.5 provides a bound on how much the exploitability increases after each re-solving step, with Theorem F.4 providing the necessary regret bounds

$$\begin{aligned} \text{EXP}_d &\leq \text{EXP}_{d-1} + (\bar{R}_o^S)^+ + (\bar{R}_i^S)^+ + \bar{R}_i + 2 \left( F(\epsilon + \xi) + NU \sqrt{\frac{A}{T}} \right) \\ &\leq \text{EXP}_{d-1} + 5 \left( F(\epsilon + \xi) + NU \sqrt{\frac{A}{T}} \right) \end{aligned}$$

Unrolling for  $D$  re-solving steps leads to the final bound. □