Computing Strong Game-Theoretic Strategies and Exploiting Suboptimal Opponents in Large Games

Sam Ganzfried

CMU-CS-15-104

May 2015

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

This research was sponsored by the National Science Foundation under grant numbers IIS-0427858, IIS-0905390, IIS-0964579, CCF-1101668, CCR-0122581, and IIS-1320620, as well as XSEDE computing resources provided by the Pittsburgh Supercomputing Center. I also acknowledge Intel Corporation and IBM for their machine gifts. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government, or any other entity.

Keywords: artificial intelligence, game theory, multiagent systems, game solving, game abstraction, equilibrium computation, qualitative models, opponent modeling.

Abstract

Designing successful agents for multiagent strategic settings is a challenging problem for several reasons. First, many games of interest are far too large to be solved (for a relevant game-theoretic solution concept) by the best current algorithms. For example, no-limit Texas hold 'em has approximately 10^{165} nodes in its game tree, while the best algorithms for computing a Nash equilibrium only scale to games with around 10^{17} states. A second challenge is that it is not even clear that our goal should be computing a Nash equilibrium in the first place. In games with more than two players (or two-player games that are not zero sum (competitive)), playing a Nash equilibrium has no performance guarantee. Furthermore, even in two-player zero-sum games, we can often obtain significantly higher payoffs by learning to exploit mistakes of a suboptimal opponent than by playing a Nash equilibrium.

The leading paradigm for addressing the first challenge is to first approximate the full game with a strategically similar but significantly smaller game, and then to solve this smaller abstract game. All of this computation is done offline in advance, and the strategies are then looked up in a table for actual game play. We have developed new algorithms for improving each step of this paradigm. Specifically, I present new algorithms for performing game abstraction and for computing equilibria in several game classes, as well as new approaches for addressing the problem of interpreting actions for the opponent that have been removed from the abstraction and further post-processing techniques that achieve robustness against limitations of the abstraction and equilibrium-finding phases.

I then describe two new game-solving paradigms: in the first, relevant portions of the game are solved in real time to a better degree of accuracy than the abstract game, which is solved offline according to the leading paradigm, and in the second, qualitative representations of the structure of equilibrium strategies are leveraged to improve the speed of equilibrium finding. The latter paradigm can be utilized to obtain human-understandable knowledge from strategies, which are often represented as massive binary files, thereby enabling improved human decision-making.

In the final portion of the thesis, I address the second challenge by presenting new algorithms for effectively learning to exploit unknown static opponents in large imperfect-information games after only a small number of interactions. Furthermore, I present new algorithms for exploiting weak opponents that are able to guarantee a good worst-case performance even against strong dynamic opponents.

The approaches are domain independent and apply to any games within very broad classes, which include many important real-world situations. While most of the approaches are presented in the context of two-player zero-sum games, they also apply to games with more than two agents, though in some cases this results in a modification of theoretical guarantees. One application domain that was considered was two-player no-limit Texas hold 'em. Several of the approaches were utilized to create an agent that came in first place in the most recent (2014) AAAI Annual Computer Poker Competition, beating each opposing agent with statistical significance.

Acknowledgments

I would like to thank my advisor Tuomas Sandholm for allowing me to pursue an extremely ambitious research agenda; a more conservative one would have been significantly less fulfilling. I am glad that we ended up agreeing on Claudico as the name of our man-machine competition agent as opposed to some of Professor Sandholm's earlier suggestions, which included Carnegietastic, Finntastic, and Sharknegie—I don't think those would have had quite the same ring! I would like to thank Dong Kim, Jason Les, Bjorn Li, and Doug Polk for taking it easy on Claudico so that our loss was slightly below the value needed to be statistically significant at the 95% confidence level!

Claudico may not have been possible without the groundwork that was paved by Andrew Gilpin, my predecessor on the poker project. The "workhorse" file is still called "DoylesGame.java," even though the no-limit competition has not been 500 BBs since 2008! Claudico also certainly benefited from having Noam Brown on the team, and I am confident that I am leaving the code in extremely capable hands at my departure. Rest assured Troels Sørenson your contributions have not been forgotten, and your code has been pivotal in producing some of the finest betting abstractions in the world to this day. I also enjoyed our conversations on various equilibrium refinements, including trembling-hand perfect equilibrium, quasi-perfect equilibrium, and proper equilibrium (both the normal-form and extensive-form variants!) The same goes for Branislav Bosansky, Jiří Čermák, and Nicola Gatti.

I would like to thank the other members of my thesis committee—Avrim Blum, Michael Bowling, Vincent Conitzer, and Geoffrey Gordon, as well as Kevin Leyton-Brown who was on my thesis proposal committee. I appreciate the time, effort, and thoughtfulness you each put into providing valuable feedback. I hope that Geoff was not too devastated to learn that the equilibrium strategies he had computed for one-card poker that had been on his website for many years were actually (weakly) dominated! It was a close call, but Vince wins the award for longest distance traversed to attend—both for the proposal and the defense! I would also like to thank Michael Wellman for insightful discussions—it's too bad that purification has not yet proven to be as helpful for trading agents as it has for poker agents.

I enjoyed working with Ariel Procaccia on a project for optimizing weights for exam questions to minimize the average squared error difference from the homework average, which we used as a proxy for the students' "true ability." It is too bad that much of the improvements of the techniques we considered disappeared when all the scores were normalized by the average, but I am optimistic that there is a bright future ahead for this line of inquiry.

I would like to thank everyone who has been a part of the poker competition and workshop over the years (am I the only person who has attended all four events?). I'd particularly like to thank Michael Johanson, who was extremely helpful in providing details on the information abstraction algorithms used by Alberta's poker agents. Without his numerous detailed emails, it would have taken me eons longer to comprehend the differences between perfect and imperfect-recall abstraction, not to mention distribution-aware abstraction, the merits of earth-mover's distance over L2, and "OCHS." It was a pleasure co-organizing the workshop with Eric Jackson in 2014, and I am very appreciative of Nolan Bard, Neil Burch, Eric Jackson, Jonathan Rubin, and Kevin Waugh for their effort running the computer poker competition, as well as Adrian Koy, Jon Parker, and Tim Reiff for many enjoyable conversations.

I would also like to thank Abe Othman (aka "othasaurus"), Daniel Miller (aka "millertie"), Winslow Strong (aka "WINSLOW STRONG"), and Raja "fitbit" Sambasivan for entertaining all my wacky "startup" ideas. I just know that "SHPRATZ" will go bigtime. I will never forget the dentist of Italian descent who rejoined at just the proper time, shrieking "Oh No!" after ripping backhands that exceeded the baseline by not more than two inches. Thank you HH for informing me that it probably isn't the greatest idea to wear a t-shirt in my main website/profile photo—they don't teach these things in grad school! Andrew "azntracker" Li, now that you are retired, I feel ok telling you that you call wayyy too wide on the bubble, jeez.

Deb Cavlovich and Charlotte Yano have done far more than necessary to ensure the logistical aspects of my time at CMU proceeded impeccably.

And finally, a big thank you to Sally, "Big J," and of course "the cowboy," among other things, for all the couscous.

Contents

I Introduction 1

1 Overview 3

2 Game Theory Background 5
- 2.1 Extensive-form games . . . 7
- 2.2 Repeated games . . . 9
- 2.3 Other game representations . . . 9

3 Poker 11

4 Leading Game-Solving Paradigm 13

II New Approaches for Game Solving within the Leading Paradigm 17

5 Potential-Aware Imperfect-Recall Abstraction with Earth Mover's Distance in Imperfect-Information Games 19
- 5.1 Potential-aware abstraction . . . 21
  - 5.1.1 Domain-independent example . . . 22
  - 5.1.2 Poker example . . . 23
- 5.2 Algorithm for potential-aware imperfect-recall abstraction, with EMD . . . 25
- 5.3 Experiments . . . 28
  - 5.3.1 Head-to-head performance vs. best prior abstraction algorithm . . . 29
  - 5.3.2 Evaluating the approximation of potential-aware EMD . . . 29
- 5.4 Summary and extensions . . . 30

6 Hierarchical Abstraction Algorithm that Enables Massive Distributed Equilibrium Computation 33
- 6.1 Abstraction algorithm . . . 34
  - 6.1.1 Main abstraction algorithm . . . 35
  - 6.1.2 Algorithm for computing abstraction of public information . . . 36
  - 6.1.3 Public abstraction clustering algorithm . . . 38
- 6.2 Equilibrium-finding algorithm . . . 39
- 6.3 Experiments . . . 40
- 6.4 Summary and extensions . . . 41

7 Action Translation **43**
7.1 Problem formulation . . . 44
7.2 No-Limit poker . . . 44
7.2.1 Clairvoyance game . . . 44
7.2.2 Kuhn poker . . . 44
7.2.3 Leduc hold 'em . . . 45
7.2.4 Texas hold 'em . . . 45
7.3 Action translation desiderata . . . 45
7.4 Prior mappings . . . 46
7.4.1 Deterministic arithmetic . . . 46
7.4.2 Randomized arithmetic . . . 46
7.4.3 Deterministic geometric . . . 47
7.4.4 Randomized geometric 1 . . . 47
7.4.5 Randomized geometric 2 . . . 47
7.5 Our new mapping . . . 47
7.6 Graphical examples . . . 49
7.7 Theoretical analysis . . . 49
7.8 Comparing exploitability . . . 52
7.8.1 Clairvoyance game . . . 52
7.8.2 Kuhn poker . . . 53
7.8.3 Leduc hold 'em . . . 55
7.9 Experiments in Texas hold 'em . . . 56
7.10 Summary and extensions . . . 57

8 Post-Processing **59**
8.1 Purification and thresholding . . . 60
8.1.1 Purification . . . 61
8.1.2 Thresholding . . . 61
8.2 Evaluation metrics . . . 61
8.2.1 Empirical performance . . . 62
8.2.2 Worst-case exploitability . . . 62
8.2.3 Performance against full equilibrium . . . 62
8.3 Theory: selective superiority . . . 63
8.4 Random matrix games . . . 64
8.4.1 Evaluation methodology . . . 64
8.4.2 Experimental results and theory . . . 64
8.5 Leduc hold 'em . . . 66
8.5.1 Experimental evaluation and setup . . . 66
8.5.2 Purification vs. no purification . . . 67
8.5.3 Purification vs. thresholding . . . 67
8.6 Texas hold 'em . . . 67
8.6.1 A champion no-limit Texas hold 'em program . . . 67
8.6.2 Assessing worst-case exploitability in limit Texas hold 'em . . . 70
8.7 New generalized family of post-processing techniques . . . 71

8.8 Summary and extensions 72

9 Computing Equilibria in Multiplayer Stochastic Imperfect-Information Games 75

9.1 Stochastic games 76
9.2 Poker tournaments and jam/fold strategies 77
9.3 *VI-FP* algorithm 78
9.3.1 Inner loop 79
9.3.2 Outer loop 79
9.4 Our *ex post* check 81
9.4.1 Policy iteration 82
9.4.2 *Ex post* check algorithm 82
9.5 New algorithms for solving stochastic games 84
9.5.1 *PI-FP*: Policy Iteration as outer loop; Fictitious Play as inner loop 84
9.5.2 *FP-MDP*: switching the roles of Fictitious Play and MDP solving 85
9.5.3 *FTPL-MDP*: a polynomial time algorithm for regret minimization 86
9.6 Experiments 86
9.7 Implementation discussion 87
9.7.1 11-card rollout 87
9.7.2 Indexing within the equilibrium finding 89
9.8 Poker strategy observations 90
9.8.1 Prior results for a two-player poker tournament 90
9.8.2 Independent Chip Model (ICM) 91
9.8.3 Assessing the Independent Chip Model 91
9.8.4 Comparing tournament strategies and single-hand strategies 92
9.8.5 Nonexistence of a fixed ranking of hands 97
9.8.6 Rarity of randomization 98
9.9 Summary and extensions 98

III New Game-Solving Paradigms 101

10 Endgame Solving in Large Imperfect-Information Games 103

10.1 Endgame solving 105
10.2 Benefits of endgame solving 108
10.2.1 Exact computation of Nash equilibrium in abstracted endgames 108
10.2.2 Ability to compute certain equilibrium refinements 108
10.2.3 Finer-grained, history-aware, and strategy-biased abstraction 109
10.2.4 A solution to the off-tree problem 109
10.3 Endgame solving algorithm 110
10.4 Example 114
10.5 Experiments on no-limit Texas hold ’em 115
10.6 Variance-reduction technique 116
10.7 Equilibrium refinements that can be integrated with endgame solving 118
10.7.1 Undominated Nash equilibrium 118

10.7.2 ϵ-Quasi-perfect equilibrium 121
10.7.3 One-card poker 121
10.7.4 Prior experiments against 2012 poker competition agents 123
10.8 Summary and extensions 123

11 Computing Equilibria by Incorporating Qualitative Models 125
11.1 Continuous Games 126
11.2 Measure theory background 127
11.3 Motivating example 128
11.4 Our setting 133
11.5 Parametric models 134
11.6 Our main algorithm 135
11.6.1 Constructing a MILFP 135
11.6.2 Obtaining mixed strategies from the MILFP solution 136
11.6.3 Algorithm correctness 136
11.7 Extensions to more general settings 137
11.7.1 Continuous private signal distributions 137
11.7.2 Dependent private signals 138
11.7.3 Many players 138
11.7.4 Multiple parametric models 141
11.8 Experiments 143
11.8.1 Approximating large finite games by continuous games 143
11.8.2 Two-player limit Texas hold ’em 145
11.8.3 Multiplayer experiments 149
11.9 Summary and extensions 154

IV Exploiting Suboptimal Opponents 157

12 DBBR: A Scalable, Domain-Independent Opponent Exploitation Algorithm 159
12.1 Introduction 160
12.2 DBBR: an efficient real-time opponent modeling algorithm 162
12.2.1 Overview of the algorithm 162
12.2.2 Computing posterior action probabilities 162
12.2.3 Computing posterior bucket probabilities 163
12.2.4 Computing the opponent model 164
12.2.5 Full algorithm 167
12.3 Experiments and discussion 167
12.3.1 Experimental results 168
12.3.2 Comparing the opponent modeling algorithms 169
12.3.3 Win rates over time 170
12.4 Summary and extensions 172

13 Safe Opponent Exploitation 173

13.1 Uses, applicability, and generality of the approach 175
13.1.1 Two alternative uses of the methodology 175
13.1.2 Bounds suffice for using the methodology 176
13.1.3 The methodology also applies to infinitely repeated, general-sum, and multiplayer games 176
13.1.4 Safe exploitation can be viewed as selection among equilibria of the repeated game 177
13.2 Safety 177
13.2.1 A game in which safe exploitation is not possible 177
13.2.2 A game in which safe exploitation is possible 178
13.3 Characterizing gifts 178
13.4 Safety analysis of some natural exploitation algorithms 181
13.4.1 Risk What You've Won (RWYW) 181
13.4.2 Risk What You've Won in Expectation (RWYWE) 182
13.4.3 Best equilibrium strategy 184
13.4.4 Regret minimization between an equilibrium and an opponent exploitation algorithm 184
13.4.5 Regret minimization in the space of equilibria 185
13.4.6 Best equilibrium followed by full exploitation (BEFFE) 185
13.4.7 Best equilibrium and full exploitation when possible (BEFEWP) 186
13.5 A full characterization of safe strategies in strategic-form games 186
13.6 Safe exploitation in extensive-form games 188
13.6.1 Extensive-form games of perfect information 188
13.6.2 Extensive-form games of imperfect information 191
13.6.3 Gift detection and exploitation within a game iteration 192
13.7 Experiments 193
13.7.1 Kuhn poker 193
13.7.2 Experimental setup 195
13.7.3 Algorithm for computing safe responses in extensive-form games 196
13.7.4 Experimental results 197
13.8 Summary and extensions 197

V Conclusion **201**

14 Reiteration of Game-Solving Challenges **203**

15 Contributions **205**

16 Future Research **209**

Bibliography **221**

List of Figures

4.1 Leading paradigm for solving large games. 14

5.1 Equity distribution for 6c6d. 20
5.2 Equity distribution for KcQc. 20
5.3 Information equity tree for private signal x_1. 22
5.4 Information equity tree for private signal x_2. 22
5.5 Histogram of equity for both private information x_1 and x_2 at round 1, assuming the game reaches the end of the final round. 23
5.6 Histogram for private signal x_2 at round 1 over non-ordinal information states at round 2. 23
5.7 Equity distribution for TcQd-7h9hQh on the river (final betting round). 24
5.8 Equity distribution for 5c9d-3d5d7d on the river (final betting round). 24
5.9 Equity distribution for TcQd-7h9hQh on the turn (next betting round). Each point is the expected hand strength for a given turn card assuming a uniform random distribution for the river card and the opponent's cards. 25
5.10 Equity distribution for 5c9d-3d5d7d on the turn (next betting round). Each point is the expected hand strength for a given turn card assuming a uniform random distribution for the river card and the opponent's cards. 25

6.1 Win rates over time against the two strongest agents from the 2013 poker competition. 41

7.1 Randomized mappings with $A = 0.01$, $B = 1$. 49
7.2 Randomized mappings with $A = 1$, $B = 4$. 50

8.1 Abstraction equilibrium strategies can be overfit to the abstraction. 60
8.2 Game where purification leads to arbitrarily large improvement. 63
8.3 Effects of thresholding on performance of abstract strategies against an equilibrium opponent in mb/h. 69

9.1 Epsilon as a function of running time. 87

10.1 Endgame solving (re-)solves the relevant endgame that we have actually reached in real time to a greater degree of accuracy than in the offline computation. 103
10.2 Player 1 selects his action a_i, then the players play imperfect-information game G_i.107

11.1 Worst-case payoff of playing the projection of the equilibrium of G_∞ ($\pi(\sigma_n)$) versus the value of G_n ($v(G_n)$). 144
11.2 Experiments for G_{250}. 144
11.3 First parametric model. 146
11.4 Second parametric model. 147
11.5 Third parametric model. 148
11.6 Parametric model for the three-player poker game. 150
11.7 Experimental values of, and the theoretical bound on, solution quality as function of number of breakpoints. 151
11.8 Running time (in seconds) as a function of the number of breakpoints. 152
11.9 Conditional parametric model representation of a qualitative model for limit Texas hold 'em strategies. 153

12.1 Profits and win rates over time of DBBR-EM against AlwaysFold. 171
12.2 Profits and win rates over time of DBBR-EM against AlwaysRaise. 171
12.3 Profits and win rates over time of DBBR-EM against GUS2. 171

13.1 Our safe exploitation methodology used as a meta-algorithm which makes any opponent exploitation architecture safe. An opponent exploitation architecture consists of two components: an opponent modeling algorithm and a strategy selection algorithm. 175
13.2 Our safe exploitation methodology used to replace the strategy selection component while retaining the opponent modeling component of any opponent exploitation architecture. 176
13.3 Payoff matrix of Rock-Paper-Scissors. 178
13.4 Payoff matrix of RPST. 178
13.5 A game with a gift strategy that is not weakly iteratively dominated. 179
13.6 Strategy R is not in the support of an equilibrium for player 2, but is also not a gift. 179
13.7 Payoff matrix for first stage game of extensive-form game where we can detect and exploit a gift within a game iteration. 193
13.8 Payoff matrix for second stage game of extensive-form game where we can detect and exploit a gift within a game iteration. 193
13.9 Extensive-form representation of a game where we can detect and exploit a gift within a game iteration. *X*:*Y* denotes the *Y*th information set of Player *X*. Dotted lines tie together nodes within an information set. At leaves of the game tree, the payoff of Player 1 is listed first, followed by the payoff of Player 2. 194
13.10 Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, both k^t and profits steadily increase. 198
13.11 Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, k^t stays relatively close to 0 throughout the match, while profit steadily increases. 199

List of Tables

6.1 Win rate of our two-player no-limit Texas hold ’em agent in the 2014 Computer Poker Competition against opposing agents. 41

7.1 Effect of increasing A while holding $B = 1$ and $x = 0.25$ fixed. 52
7.2 Exploitability of mappings for the clairvoyance game, using betting abstraction $\{$fold, check, pot, all-in$\}$. 53
7.3 Exploitability of mappings for no-limit Kuhn poker, using betting abstraction $\{$fold, check, pot, all-in$\}$. 54
7.4 Exploitability of mappings for no-limit Kuhn poker, using betting abstraction $\{$fold, check, 0.4 pot, all-in$\}$. 54
7.5 Optimal bet sizes for player 2 of action translation mappings for no-limit Kuhn poker, using betting abstraction with fold, check, all-in, and one additional bet size. 55
7.6 Exploitability of mappings for each player in no-limit Leduc hold ’em using the fcpa betting abstraction. 55
7.7 No-limit Texas hold ’em results in milli big blinds per hand. The entry is the profit of our agent Tartanian5 using the mapping given in the column against the opponent listed in the row. 56

8.1 Results for experiments on 1.5 million random 4×4 matrix games using random 3×3 abstractions. The $\pm$ given is the 95% confidence interval. 64
8.2 Effects of purification on performance of abstract strategies against an equilibrium opponent in mb/h. 68
8.3 Results from the 2010 Annual Computer Poker Competition for two-player no limit Texas hold ’em. Values are in milli big blinds per hand (from the row player’s perspective) with 95% confidence intervals shown. IRO and TBR both use the same abstraction and equilibrium-finding algorithms. The only difference is that IRO uses thresholding with a threshold of 0.15 while TBR uses purification. 70
8.4 Exploitabilities of limit Texas hold ’em agents for several post-processing approaches. 70
8.5 Win rate (in mbb/h) of several post-processing techniques against the strongest two-player no-limit Texas hold ’em agents from the 2013 Computer Poker Competition. 72

9.1 Deviations between ICM and our payoffs 92
9.2 Button 93
9.3 Small blind after button jams 93

9.4 Small blind after button folds 93
9.5 Big blind after jam/fold 93
9.6 Big blind after fold/jam 94
9.7 Big blind after jam/jam 94
9.8 Button 95
9.9 Small blind after button jams 95
9.10 Small blind after button folds 95
9.11 Big blind after jam/fold 95
9.12 Big blind after fold/jam 96
9.13 Big blind after jam/jam 96
9.14 Differences between tournament and single hand strategy with equal stacks. . . . 96
9.15 Small blind strategy after button folds with stacks (3300, 1500, 8700). 97
9.16 Karlson-Sklansky hand rankings 98

10.1 Improvement from using endgame solving against strongest two-player no-limit Texas hold 'em competition agents. 116
10.2 Equilibrium strategy for player 1 in one-card poker, computed using the LP formulation. The first row is the probability of betting with each hand in the initial betting round, and the second row is the probability of calling after player 1 checks initially and player 2 bets. 122
10.3 Equilibrium strategy for player 2 in one-card poker. The first row is the probability of betting with each hand after player 1 checks, and the second row is the probability of calling after player 1 bets. 122
10.4 No-limit Texas hold 'em results against the strongest competitors from the 2012 Annual Computer Poker Competition. The payoffs are given in milli big blinds per hand for the agent listed in the column, and the 95% confidence intervals are reported. 123

12.1 Win rate in small bets/hand of the bot listed in the row. The ± given is the standard error (standard deviation divided by the square root of the number of hands). 169

13.1 Win rates of safe exploitation algorithms in Kuhn poker 197

Part I

Introduction

Chapter 1

Overview

Important problems in nearly all disciplines and on nearly all application domains involve multiple agents behaving strategically; for example, deploying officers to protect ports, determining optimal thresholds to protect against phishing attacks, and finding robust policies for diabetes management. Such problems are modeled under the framework of game theory. In many important games there is information that is private to only some agents and not available to other agents—for instance, in auctions each bidder may know his own valuation and only know the distribution from which other agents' valuations are drawn.

This thesis presents new approaches for strategic agents acting in large imperfect-information games. It includes novel algorithms, theoretical analysis, and detailed discussion from large-scale implementation of the approaches.

There are several major challenges that must be confronted when designing successful agents for large multiagent strategic environments. First, standard solution concepts such as Nash equilibrium lack theoretical justification in certain classes (e.g., games with more than two players). Second, computing these concepts is difficult in certain classes from a complexity-theoretic perspective. Third, computing these concepts is difficult in practice for many important games even for cases when they are well-motivated and polynomial-time algorithms exist (e.g., two player zero-sum (competitive) games), due to enormous state spaces. And fourth, for all game classes, it is not clear if the goal should even be to compute a Nash equilibrium; one could achieve significantly higher payoff by learning to exploit opponents' mistakes. However, such exploitation must be done in a way that does not open oneself up to being exploited in turn by strong deceptive opponents.

While the approaches are domain independent, most of them have been motivated by and applied to the domain of poker. Poker has emerged as a major AI challenge problem. Poker is not simply a toy game; it is tremendously popular for humans, and online poker is a multibillion dollar industry. For the past ten years, there has been a competition between the strongest computer poker agents held annually at the top AI conference. The version of two-player no-limit Texas hold 'em played in the competition has approximately 10^{165} states in its game tree. Several of the techniques presented in this thesis were utilized to create an agent for two-player no-limit Texas hold 'em that is currently the strongest agent in the world: it beat each opposing agent with statistical significance in the most recent (2014) AAAI Annual Computer Poker Competition.

Chapter 2

Game Theory Background

A *game* is an abstract model of strategic interaction between multiple agents, or players. Formally, a *strategic-form game* G consists of a finite set of *players* $N = \{1, \ldots, n\}$, a finite set of *pure strategies* S_i for each player, and a *utility function* $u_i : \times S_i \rightarrow \mathbb{R}$ for each player. Here $\times S_i$ denotes the space of *pure strategy profiles*—vectors of pure strategies, one per player. To play a game, each agent i simultaneously selects a pure strategy $s_i \in S_i$, and then receives a payoff of $u_i(s_1, \ldots, s_n)$. In general, players are allowed to randomize over their pure strategies, and need not play deterministically. Let Σ_i denote the space of probability distributions over S_i, which we call the *mixed strategy space* of player i. When each agent i plays $\sigma_i \in \Sigma_i$, the expected payoff to player i is

$$u_i(\sigma_1, \ldots, \sigma_n) = \sum_{s_1 \in S_1} \cdots \sum_{s_n \in S_n} \left[u_i(s_1, \ldots, s_n) \prod_{j=1}^{n} \sigma_j(s_j) \right].$$

Note that we have overloaded the utility operator to be defined over $\Sigma = \times \Sigma_i$, the space of *mixed strategy profiles*. If the players are following the mixed strategy profile $\sigma \in \Sigma$, let σ_{-i} denote the vector of strategies taken by all players other than i, and let Σ_{-i} denote the space of mixed strategies for these players. The *support* of a mixed strategy σ_i is the set of pure strategies for player i played with nonzero probability under σ_i. Mixed strategy σ_i *weakly dominates* σ_i' if $u_i(\sigma_i, \sigma_{-i}^*) \geq u_i(\sigma_i', \sigma_{-i}^*)$ for all $\sigma_{-i}^* \in \Sigma_{-i}$, where the inequality is strict for at least one σ_{-i}^*.

If the other agents are playing strategy profile σ_{-i}, then a *best response* (aka *nemesis*) for player i is any strategy in $\arg\max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i})$. A *Nash equilibrium* is a strategy profile σ such that σ_i is a best response to σ_{-i} for all i. Thus, in a Nash equilibrium, all players are simultaneously playing a best response to the strategy profile of the other agents, and no agent has an incentive to deviate to a different strategy given that the other agents follow the prescribed profile.

John Nash first introduced the Nash equilibrium in 1951, and in that paper he proved that a Nash equilibrium exists in every strategic-form game [86]. Subsequently, the Nash equilibrium has emerged as the central solution concept in the field of game theory. If all agents were perfectly rational, then we would intuitively expect them to follow a Nash equilibrium; if they instead followed a non-equilibrium strategy profile, then at least one agent could improve his performance by playing a different strategy, in which case it would not be rational for him to follow the prescribed strategy profile.

The Nash equilibrium solution concept is particularly compelling in a class of games known as two-player zero-sum games (aka *matrix games*). A two-player game is *zero sum* if $u_1(s) + u_2(s) = 0$ for all $s \in \times_i S_i$. These are fully non-cooperative, competitive games where one player's loss is exactly equal to the other player's gain. In this class of games, we have the following result, which is called the *minimax theorem*:

$$v^* = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} u_1(\sigma_1, \sigma_2) = \min_{\sigma_2 \in \Sigma_2} \max_{\sigma_1 \in \Sigma_1} u_1(\sigma_1, \sigma_2).$$

The minimax theorem was first published by John von Neumann [113] in 1928, several decades before Nash's existence theorem. This theorem states that there exists a unique value v^* such that player 1 can guarantee himself an expected payoff of at least v^* regardless of the strategy chosen by player 2, and similarly that player 2 can guarantee himself an expected payoff of at least $-v^*$ regardless of the strategy chosen by player 1. We refer to v^* as the *value* of the game. Sometimes we will write $v_1 = v^*$ as the value of the game to player 1, and $v_2 = -v^*$ as the value of the game to player 2. *Any* equilibrium strategy for a player will guarantee an expected payoff of at least the value of the game to that player.

Define the *exploitability* of σ_i to be the difference between the value of the game and the performance of σ_i against its nemesis, formally:

$$\text{expl}(\sigma_i) = v_i - \min_{\sigma'_{-i}} u_i(\sigma_i, \sigma'_{-i}).$$

Since there always exists a nemesis that is a pure strategy, this expression is equal to $v_i - \min_{s_{-i} \in S_{-i}} u_i(\sigma_i, s_{-i})$. For any $\epsilon \geq 0$, define SAFE$(\epsilon) \subseteq \Sigma_i$ to be the set of strategies with exploitability at most ϵ. The set SAFE(ϵ) is defined by linear constraints: $\sigma_i \in$ SAFE(ϵ) if and only if $u_i(\sigma_i, s_{-i}) \geq v_i - \epsilon$ for all $s_{-i} \in S_{-i}$. Define an *ϵ-safe best response* of player i to σ_{-i} to be any strategy in

$$\text{argmax}_{\sigma_i \in \text{SAFE}(\epsilon)} u_i(\sigma_i, \sigma_{-i}).$$

In two-player zero-sum games, the Nash equilibrium strategies for player 1 are precisely those strategies that guarantee a worst-case expected payoff of at v^* (and similarly, the Nash equilibrium strategies for player 2 are precisely those strategies that guarantee a worst-case expected payoff of at least $-v^*$). For any non-equilibrium strategy for player 1, there exists some strategy for player 2 such that player 1's expected payoff is strictly less than v^*. Thus, Nash equilibrium strategies have a strictly better worst-case guarantee than all other strategies. If we assume players took turns having the role of player 1 and player 2, then a Nash equilibrium strategy would guarantee at least breaking even against any opponent, while for every non-equilibrium strategy, there exists some counter strategy against which it would lose.

An additional property of Nash equilibria in two-player zero-sum games is that they are *exchangeable*: if (σ_1, σ_2) and (σ'_1, σ'_2) are Nash equilibria, then (σ_1, σ'_2) and (σ'_1, σ_2) are also Nash equilibria. Thus, if player 1 follows his portion of the strategy profile from one Nash equilibrium, and player 2 follows his portion of the strategy profile from a different Nash equilibrium, the overall strategy profile played still constitutes a Nash equilibrium.

One final property of Nash equilibria in two-player zero-sum games is that they can be computed in polynomial time using a linear programming (LP) formulation [22]. This means that, at least in theory, an efficient procedure exists for computing a Nash equilibrium that will scale to large games. As we will see, this does not necessarily mean that we can compute a Nash equilibrium in a satisfactory amount of time for specific games we are interested in, which may be extremely large. Best responses can be computed much more efficiently than Nash equilibria. Computing a best response involves a single matrix-vector multiplication followed by a traversal up the game tree, both of which take linear time in the size of the game tree.

Unfortunately, none of these properties that make Nash equilibrium compelling in two-player zero-sum games hold in more general classes of games. In two-player general-sum games and games with more than two players, there can exist multiple equilibria, each yielding different payoffs to the players. If one player follows one equilibrium while other players follow a different equilibrium, the overall strategy profile is not guaranteed to be an equilibrium. And furthermore, if one player plays an equilibrium strategy, he could do arbitrarily poorly if the opponents do not follow their components of that same equilibrium. In addition, the problem of computing a Nash equilibrium in these game classes has recently been shown to be PPAD-complete, and it is widely conjectured that no efficient algorithms exist [21, 23]. So even if we wanted to play a Nash equilibrium in these games, we may not be able to compute one, even in relatively small games. In two-player general-sum and multiplayer games, the Nash equilibrium is a much less satisfactory solution concept than in two-player zero-sum games.

Even in two-player zero-sum games, the Nash equilibrium is not the end of the story. For one, algorithms may not scale to specific games we are interested in. Furthermore, we can often obtain a significantly higher payoff than the value of the game against suboptimal opponents who are not playing an equilibrium strategy. Against such opponents, it may be desirable to try to learn and exploit their mistakes rather than to simply follow a static equilibrium strategy. Of course, such *opponent exploitation* would be similarly beneficial in general-sum and multiplayer games as well.

Despite the theoretical limitations described above, we will follow traditional terminology and refer to the problem of computing an (approximate) Nash equilibrium of a game as *solving* the game. The first portion of the thesis will focus on new approaches for solving games, while the latter portion will address the problem of developing game-playing agents that potentially deviate from a Nash equilibrium strategy in order to exploit opponents' mistakes.

2.1 Extensive-form games

While the strategic form can be used to model simultaneous actions, another representation, called the *extensive form*, is generally preferred when modeling settings that have sequential moves. The extensive form can also model simultaneous actions, as well as chance events and imperfect information (i.e., situations where some information is available to only some of the agents and not to others). Extensive-form games consist primarily of a game tree; each non-terminal node has an associated player (possibly *chance*) that makes the decision at that node, and each terminal node has associated utilities for the players. Additionally, game states are partitioned into *information sets*, where the player whose turn it is to move cannot distinguish among the states in the same information set. Therefore, in any given information set, a player must choose actions with the same distribution at each state contained in the information set. If no player forgets information that he previously knew, we say that the game has *perfect recall*.

A (behavioral) *strategy* for player i, $\sigma_i \in \Sigma_i$, is a function that assigns a probability distribution over all actions at each information set belonging to i.

In theory, every extensive-form game can be converted to an equivalent strategic-form game; however, there is an exponential blowup in the size of the game representation, and therefore such a conversion is undesirable. Instead, new algorithms have been developed that operate on the extensive form representation directly. It turns out that the complexity of computing equilibria in extensive-form games is similar to that of strategic-form games; a Nash equilibrium can be computed in polynomial time in two-player zero-sum games (with perfect recall) [72], while the problem is hard for two-player general-sum and multiplayer games.

For many years, the standard algorithm for computing an equilibrium in two-player zero-sum extensive-form games with perfect recall was a linear programming formulation [72]. This formulation works by modeling each sequence of actions for each player as a variable, and is often called the *sequence form LP* algorithm. The most scalable current general-purpose linear programming technique (CPLEX's barrier method) scales to games with around 10^8 nodes in their game tree, and runs into memory limitations for larger games. By contrast, full best responses can be computed in time linear in the size of the game tree, while the best known techniques for computing ϵ-safe best responses have running times roughly similar to an equilibrium computation [65].

Unfortunately, many interesting games have far more than 10^8 states in their game tree. To solve such games, newer algorithms have been developed that are able to scale to games with approximately 10^{17} states in their game tree. These algorithms are iterative and converge to a Nash equilibrium in the limit. While the LP algorithm is able to compute an exact equilibrium, in practice these iterative algorithms can only compute an approximate, or ϵ-, equilibrium. An *ϵ-equilibrium* is a strategy profile in which each player achieves a payoff of within ϵ of his best response.

Two main iterative algorithms have been used for solving these larger games. The first, called EGT, is based on a generalization of Nesterov's excessive gap technique [55]. Recently, a more scalable version has been developed that converges to an ϵ-equilibrium in $O(\ln(\frac{1}{\epsilon}))$ iterations [49]. The other algorithm, called *counterfactual regret minimization* (CFR), stores the cumulative regret of each action at each information set, contingent on the information set being reached [125], as well as the average action probability vector at each information set. At each iteration, each action is selected in proportion to its counterfactual regret. This algorithm is run against itself in self play, and the average strategy for each player is proven to converge to an equilibrium. CFR guarantees convergence to ϵ-equilibrium in $O(\frac{1}{\epsilon^2})$ iterations, though each individual iteration is much faster than an iteration of EGT. Several sampling schemes have been used that significantly improve the performance of CFR in practice in various classes of games [40, 41, 67, 77].

Both algorithms parallelize well, and have been shown to scale effectively in practice to very large games, such as Texas hold 'em. While EGT has a better asymptotic performance guarantee in terms of the number of iterations needed for convergence, each iteration of EGT takes much longer than each iteration of CFR. Overall, these algorithms have selective superiority, and it is not clear which will perform best on a given game. Unlike EGT, CFR can still be run on games that have imperfect recall, as well as two-player general-sum and multiplayer games [1, 39, 78, 120], though there are no significant general theoretical guarantees in such settings.

2.2 Repeated games

In *repeated games*, the *stage game* is repeated for a finite number T of iterations. At each iteration, players can condition their strategies on everything that has been observed so far. In extensive-form games, generally only the actions of the opponent along the path of play are observed; in games with imperfect information, the opponent's private information may also be observed in some situations.

2.3 Other game representations

While the majority of this thesis will deal with strategic-form and extensive-form games (both one shot and repeated), some parts will deal with other game representations, in particular *stochastic games* and *continuous games*. A stochastic game is a collection of games (often these are strategic-form games, though we will consider the case when they are extensive-form imperfect-information games); the agents repeatedly play a game from this collection, and then transition probabilistically to a new game depending on the previous game played and the actions taken by all agents in that game. Continuous games generalize finite strategic-form games to the case of (uncountably) infinite strategy spaces. Many natural games have an uncountable number of actions; for example, games in which strategies correspond to an amount of time, money, or space. While Nash equilibria have been proven to exist in some classes of games, simple examples have also been constructed that do not contain an equilibrium. Algorithms have been developed for computing equilibria in certain subclasses; however, there are natural game classes for which neither the algorithms nor the existence results apply.

Chapter 3

Poker

While all of the new algorithms and techniques we present in this thesis are domain-independent and apply to broad classes of games, we will primarily be evaluating them in the domain of Texas hold 'em poker. Poker has received significant academic interest since the founding of the field of game theory [86, 114]. This interest has been heightened in recent years due to the emergence of poker as a central AI challenge problem and the development of the Annual Computer Poker Competition (ACPC). Two-player poker is a two-player zero-sum extensive-form game with perfect recall; therefore, the algorithms described in Section 2.1 will apply. We will be considering several variants of poker including no-limit Texas hold 'em, the most popular variant of poker among humans. Two-player no-limit Texas hold 'em is played competitively by humans, and it is the game of most active research in the computer poker community currently. For further information about AI research in poker, we refer the reader to recent survey articles [96, 99].

Two-player no-limit Texas hold 'em (NLHE) works as follows. Initially two players each have a *stack* of chips (worth \$20,000 in the computer poker competition). One player, called the *small blind*, initially puts \$50 worth of chips in the middle, while the other player, called the *big blind*, puts \$100 worth of chips in the middle. The chips in the middle are known as the *pot*, and will go to the winner of the hand.

Next, there is an initial round of betting. The player whose turn it is to act can choose from three available options:

- *Fold:* Give up on the hand, surrendering the pot to the opponent.
- *Call:* Put in the minimum number of chips needed to match the number of chips put into the pot by the opponent. For example, if the opponent has put in \$1000 and we have put in \$400, a call would require putting in \$600 more. A call of zero chips is also known as a *check*.
- *Bet:* Put in additional chips beyond what is needed to call. A bet can be of any size from 1 chip up to the number of chips a player has left in his stack, provided it exceeds some minimum value[1] and is a multiple of the smallest chip denomination (by contrast, in the *limit* variant, all bets must of a fixed size, which equals the big blind for the first two rounds and twice the big blind for the final two rounds). A bet of all of one's remaining chips is

[1]The minimum allowable bet size is the big blind for the first bet of a round and the size of the previous bet in the current round for subsequent bets.

called an *all-in* bet. If the opponent has just bet, then our additional bet is also called a *raise*. In some variants, the number of raises in a given round is limited (for limit it is limited to three and for no-limit it is unlimited), and players are forced to either fold or call at that point.

The initial round of betting ends if a player has folded, if there has been a bet and a call, or if both players have checked. If the round ends without a player folding, then three public cards are revealed face-up on the table (called the *flop*) and a second round of betting takes place. Then one more public card is dealt (called the *turn*) and a third round of betting, followed by a fifth public card (called the *river*) and a final round of betting. If a player ever folds, the other player wins all the chips in the pot. If the final betting round is completed without a player folding, then both players reveal their private cards, and the player with the best five-card hand (out of his two private cards and the five public cards) wins the pot (it is divided equally if there is a tie).

In the AAAI computer poker competitions, each *match* consists of 3000 *duplicate hands*: 3000 hands are played normally, then the players switch positions and play the same 3000 hands (with no memory of the previous hands). This is a well-known technique for reducing the variance so that fewer hands are needed to obtain statistical significance.

Some techniques presented in this thesis will be analyzed on simplified poker variants. The rules of these additional variants will be described in the chapter where they are first studied.

Chapter 4

Leading Game-Solving Paradigm

Two-player no-limit Texas hold ’em has about 10^{165} states in its game tree, while the limit variant has about has about 10^{17} game states [63]; so neither of these can be solved directly using EGT or CFR, which only scale to games with up to 10^{17} states. The traditional approach for solving games of this magnitude is depicted in Figure 4.1. First, the original game is approximated by a smaller *abstract game* that hopefully retains much of the strategic structure of the initial game. Abstraction in games is quite different than in single-agent settings. For example, it is not monotonic: if we refine an abstraction, we may get strategies that have higher exploitability in the original game [119]. The first abstractions for two-player Texas hold ’em were manually generated [9, 105], while current abstractions are computed automatically [42, 43, 48, 68, 120]. For smaller games, such as Rhode Island hold ’em, abstraction can be performed losslessly, and the abstract game is actually isomorphic to the full game [44]. However, for larger games, such as Texas hold ’em, we must be willing to incur some loss in the quality of the modeling approximation due to abstraction.

In general, extensive-form games can have enormous strategy spaces for two primary reasons: the game tree has many information sets, or players have many actions available at each information set (e.g., when actions correspond to real numbers from some large set). There are two kinds of abstraction to deal with these two sources of complexity: *information abstraction* and *action abstraction*.[1] In information abstraction, one groups information sets of a player together in order to reduce the total number of information sets, coarsening the moves of chance. (Essentially this forces the player to play the game the same way in two different states of knowledge.) In action abstraction, one reduces the size of the action space. The typical approach for performing action abstraction is to discretize an action space into a smaller number of allowable actions; for example, instead of allowing agents to bid any integral amount between \$1 and \$1000, perhaps we limit the actions to only multiples of \$10 or \$100. This approach applies to almost any game where action sizing is an issue, such as bet sizing in poker, bid sizing in auctions, offer sizing in negotiations, allocating different quantities of attack resources or defense resources in security games, and so on. While there has been some recent work on algorithmic

[1]There has also been some recent work on *player abstraction* that constructs abstractions that have fewer agents than the initial game [121, 123]. In this thesis I focus on studying games with a small number of agents; however, I expect player abstraction to play a very important role as game-theoretic algorithms continue to scale and games with more agents are studied more seriously.

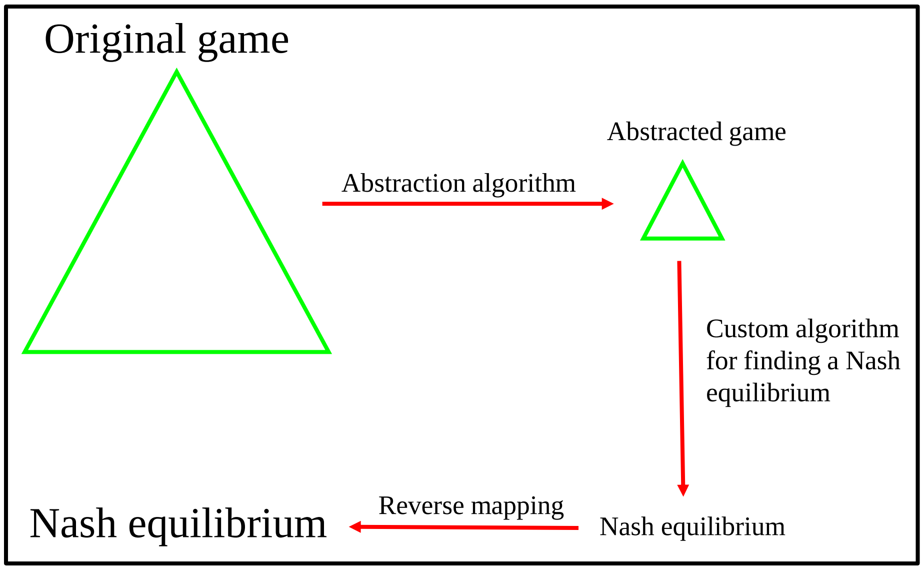


Figure 4.1: Leading paradigm for solving large games.

approaches to action abstraction [17, 53, 54] the leading approach has been to generate action abstractions manually using knowledge from domain experts. By contrast, the leading approaches for information abstraction have all been algorithmic, and I will present describe several of them in Part II.

The second step in the leading game-solving paradigm is to compute an ϵ-equilibrium in the smaller abstracted game, using a custom equilibrium-finding algorithm such as CFR or EGT.

The final step is to construct a strategy profile in the original game from the approximate equilibrium of the abstracted game by means of a *reverse mapping* procedure. When the action spaces of the original and abstracted games are identical, the final step is often straightforward, since the equilibrium of the abstracted game can be played directly in the full game. However, I will show that, even in this simplified setting, often significant performance improvements can be obtained by applying a nontrivial reverse mapping. I will introduce several procedures that modify the action probabilities of the abstract equilibrium strategies by placing more weight on certain actions [38]. These *post-processing* procedures are able to achieve robustness against limitations of the abstraction and equilibrium-finding phases of the paradigm.

When the action spaces of the original and abstracted games differ, an additional procedure is needed to interpret actions taken by the opponent that are not allowed in the abstract game model. Such a procedure is called an *action translation mapping*. The typical approach for performing action translation is to map the opponent's action to a nearby action that is in the abstraction (perhaps probabilistically), and then respond as if the opponent had taken this action. I will present a new approach with theoretical advantages over the best prior approaches that also outperforms them empirically.

While the first two steps of the paradigm have received significant attention over the last several years, the final step has received considerably less attention and is often overlooked. I will show that there are significant benefits to rigorous, theoretically-principled study of reverse mapping. Even with great abstraction and equilibrium-finding algorithms, the performance improvement by using more sophisticated reverse mapping techniques can be enormous.

Part II

New Approaches for Game Solving within the Leading Paradigm

Chapter 5

Potential-Aware Imperfect-Recall Abstraction with Earth Mover's Distance in Imperfect-Information Games

As described in Chapter 4, significant amounts of abstraction, and particularly information abstraction, are needed to apply algorithms for approximating equilibrium strategies in large imperfect-information games. One approach for determining which information sets should be grouped together is to come up with a measure of 'strength' for each state, then run a clustering algorithm, such as k-means, using the difference in 'strength' as the distance metric. For example, in poker, a natural measure of a hand's strength is the *equity* (probability of winning plus one-half the probability of tying) against a uniform random draw of private cards for the opponent, assuming a uniform random rollout of the remaining public (i.e., shared) cards. This is also known as the *expected hand strength (EHS)* metric. For example, if a player is dealt two aces as his two private cards in Texas hold 'em, he will win 84.93% of the time and will tie 0.55% of the time against a random hand assuming a random rollout of the public cards, giving his hand an equity of 0.852. Similarly, the equity of two kings is 0.824. Since these two values are similar, it is likely that they would be grouped together by a clustering algorithm. Early approaches for abstraction in poker used EHS (or EHS exponentiated to some power) to cluster hands [9, 42, 120, 125].

While EHS is a reasonable first-order-approximation for the strength of a hand, it fails to account for the entire probability distribution of hand strength. For example, the hands KcQc (king and queen of clubs) and 6c6d (six of clubs and six of diamonds) have expected hand strengths of 0.634 and 0.633 respectively, which suggests that they have very similar strength. However, looking at the full distributions of hand strength, as opposed to just its expectation, paints a very different picture, as previous work has shown [47, 68]. Figures 5.1 and 5.2 (which are similar to part of Figure 2 from Johanson et al.'s work [68]) show the full histograms of expected hand strength for the two hands, where each bar corresponds to the probability mass of the given level of hand strength. For example, if the public board cards are 7dQh4h2s3c, then KcQc has an equity of 0.856 against a uniform random opponent hand; so the histogram entry corresponding to the column for an equity of 0.84–0.86 is incremented by one for this hand (prior work has assumed that the equities are divided into 50 regions of size 0.02, as do these figures).

As the figures indicate, despite the fact that these hands have similar expected hand strengths, their full distributions are very different. For example, 6c6d frequently has an equity between 0.5 and 0.7 and rarely has an equity between 0.7 and 0.9, while the reverse is true for KcQc.

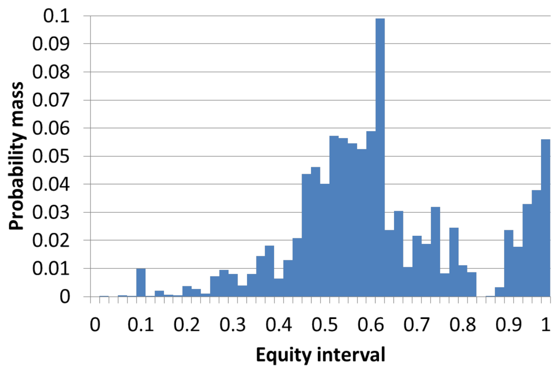


Figure 5.1: Equity distribution for 6c6d.

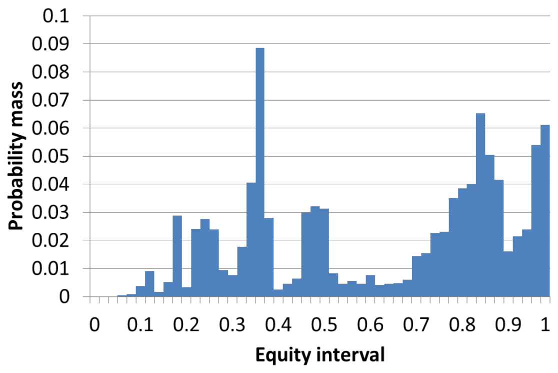


Figure 5.2: Equity distribution for KcQc.

An abstraction algorithm that considers the full distributions of hand strength, as opposed to just the expectation, is called *distribution aware*. The leading abstraction algorithm for imperfect-information games generates abstractions that are distribution aware [68], and it has been shown empirically that the distribution-aware approach significantly outperforms EHS-based approaches [45, 68]. The natural metric for computing distances between histograms of hand-strength distributions is the *earth mover's distance (EMD)*. Informally, EMD is the "minimum cost of turning one pile into the other, where the cost is assumed to be amount of dirt moved times the distance by which it is moved." Earlier work on distribution-aware abstraction used the L_2 distance metric instead [47], which has been shown to be significantly less effective because it does not properly account for how far the "dirt" needs to be moved (only how much needs to be moved). Using one-dimensional histograms as done above, EMD can be computed by a straightforward linear time procedure that scans the histogram and keeps track of how much dirt needs to be transported between consecutive bins. However, computing EMD is much more challenging as the dimensionality of the data increases, and as we will present later, multi-dimensional EMD computation will be needed in more sophisticated abstraction algorithms.

In the domain of Texas hold 'em poker, the leading abstraction algorithm works as follows [68]. In the first round, there is no card abstraction, and each hand is in its own bucket. In the second and third rounds, abstractions are computed as follows. First, an equity histogram is constructed for each hand, similarly to those in Figures 5.1 and 5.2. For example, for the flop, we will create a histogram for the hand where the private cards are Kc3h and the public cards are KsTd8h. Then k-means is used to compute an abstraction with a desired number of clusters, using the EMD between each pair of histograms as the distance metric. One important feature of these abstractions is that they have *imperfect recall*: a player can be made to forget information that he knew earlier in the hand. For example, the hands Kc3h-KsTd8h and Kc4h-KsTd8h will likely be grouped together on the flop, even though the player could distinguish between Kc3h and Kc4h in the preflop round. Imperfect recall abstractions have been demonstrated to lead to significantly stronger performance than perfect recall for an abstraction of a given size, because they allow the player to have a more refined view of the present since he is allowed to forget details about the past [120].[1] That algorithm computes abstractions for the flop and turn rounds independently using this approach. It computes the abstraction for the final round using a different approach (k-means with L_2 over vectors of EHS against first-round clusters of the opponent).

As described above, the equity histograms for the flop (and turn) rounds consider distributions over future strength at the final round of the game (i.e., after all the public cards are dealt). However, it is possible that two flop hands have very similar distributions over strength after the river is dealt, but they realize the equity at very different rates throughout the hand. Section 5.1 provides example situations of how this can arise, both in poker and in a general domain-independent game. Thus, a natural direction to explore is whether one might benefit by considering the distribution over strength in all future rounds, not just the final round. An abstraction algorithm that takes all future rounds into account is called *potential aware*. Prior work on potential-aware abstraction [47] applied only to perfect-recall abstraction and used the L_2 distance metric, both of which have significant shortcomings, as described above.

I will present the first algorithm for computing potential-aware imperfect-recall abstractions, using EMD as the distance metric [35]. We design a new abstraction algorithm that combines these three threads, each of which has been shown helpful separately in the past. Computing imperfect-recall abstractions is significantly more challenging than in the perfect-recall case, since the set of hands that must be clustered at each step is much larger. Additionally, computing EMD in this setting is significantly more challenging than in the one-dimensional distribution-aware setting, and is also much more challenging than computing L_2 distance in the potential-aware setting. The best commercially-available algorithm for computing (multi-dimensional) EMD [87, 88] is far too slow to compute abstractions in poker, and we develop a fast custom heuristic for approximating EMD in our setting. Experiments on no-limit Texas hold 'em show that our algorithm leads to a statistically significant improvement in performance over the previously best abstraction algorithm.

5.1 Potential-aware abstraction

In this section, we present examples that demonstrate the difference between potential-aware abstraction and the leading distribution-aware approach, which considers distributions over future strength at the *final* round of the game. The examples show that it is possible for two different states of private information to have very similar (even identical) histograms over equity at the end of the game, but to realize this equity in very different ways throughout the play of the game. We first present a domain-independent example in Section 5.1.1, followed by an example of a poker situation demonstrating this phenomenon in Section 5.1.2.

[1]A downside of using imperfect-recall abstractions is that they typically cause equilibrium-finding algorithms to lose their convergence guarantees.

5.1.1 Domain-independent example

We consider the following game. A player is initially given private signal x_i, and then chance makes up to two moves before the game ends. The information equity trees for x_1 and x_2 are given in Figures 5.3 and 5.4. The opponent is also given a private signal from some distribution, and the equities of the initial player against the opponent's distribution are given in the leaves of the trees. If the player has x_1, then chance selects the right branch with probability 1 in the first round, then selects each branch with probability $\frac{1}{2}$ in the second round. If chance selects the right branch in the second round, then the player has an equity of 1; otherwise, he has equity 0. If the player has x_2, then chance selects each branch with probability $\frac{1}{2}$ in the first round, and selects the right branch with probability 1 in the second round (for each choice of actions in the first round). If chance selected the left branch in the first round, then the player's equity is 0; if chance selected the right branch in the first round, then his equity is 1.

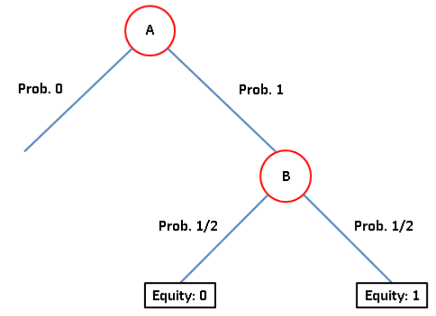


Figure 5.3: Information equity tree for private signal x_1.

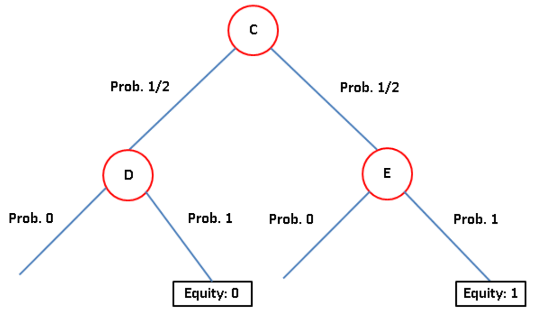


Figure 5.4: Information equity tree for private signal x_2.

If we use the traditional distribution-aware approach of considering equity assuming the end of the game is reached, then equity histograms for both x_1 and x_2 are the same and given in Figure 5.5: with probability $\frac{1}{2}$, the player will have equity 0, and with probability $\frac{1}{2}$, he will have equity 1. For this example, we assume that the equities are broken into five equally-sized intervals. (Several of the strongest poker agents use fifty intervals each of width 0.02.) Since these histograms are identical, the EMD between the two states corresponding to the two private signals respectively would be zero, so they would be treated as being identical.

However, these two states are actually quite different if we consider how the equity changes between the first and second round, as opposed to just jumping to the end of the game. With x_2, the player will know for sure whether he has an equity of 0 or 1 after chance's first move, while with x_1 he will not. To perform potential-aware abstraction, the first step is to compute the histograms for both players at the possible states in the second round. The histogram for x_1 after chance selects the right branch (i.e., at B) is also the histogram given in Figure 5.5; the histogram for x_2 after chance selects the left branch (i.e., at D) has unit mass in the left-most column (equity of 0–0.2); and the histogram for x_2 after chance selects the right branch (i.e., at E) has unit mass in the right-most column (equity of 0.8–1).

Next, we compute the histograms for x_1 and x_2 at the first round, with respect to the possible states that could be reached at the second round. The histogram for x_2 is given in Figure 5.6.

The possible states B, D, and E correspond to the second round states in the information equity trees. We omit additional states that could have originated from x_3, x_4, etc. (they will all have probability 0). As the figure shows, with x_2 the player will be in states D and E with probability $\frac{1}{2}$. The histogram for x_1 will have unit mass in the column for state B. Unlike the histograms above, whose x-axis is cardinal (i.e., equity), the x-axis of these histograms is not even ordinal (the next-round states can be listed in any arbitrary order).

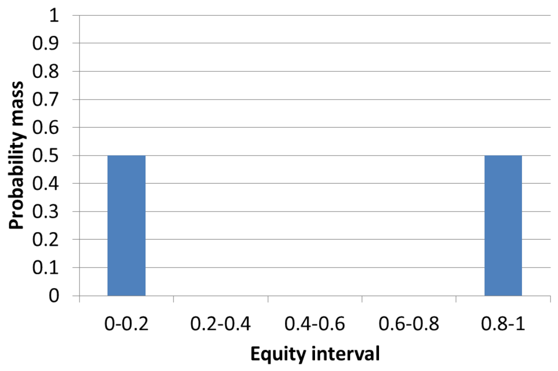


Figure 5.5: Histogram of equity for both private information x_1 and x_2 at round 1, assuming the game reaches the end of the final round.

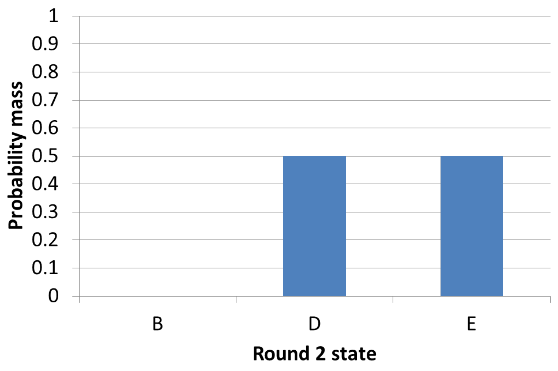


Figure 5.6: Histogram for private signal x_2 at round 1 over non-ordinal information states at round 2.

To transform this new histogram for x_1 into the histogram for x_2, we must move a mass of $\frac{1}{2}$ from the B column to both the D and E columns. Thus, the EMD is $\frac{1}{2}d(B, D) + \frac{1}{2}d(B, E)$, where the *ground distances* $d(B, D)$ and $d(B, E)$ are computed using the second-round histograms described above. To transform the histogram at B into the histogram at D, we must move a mass of $\frac{1}{2}$ from the rightmost column to the leftmost column; so $d(B, D) = \frac{1}{2} \cdot 4 = 2$. Similarly, $d(B, E)$ also equals 2. So the EMD between the two, non-ordinal first-round histograms is $\frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 2 = 2$. Thus, potential-aware abstraction will treat x_1 and x_2 differently, and potentially group them into different clusters (while the distribution-aware approach will treat them as identical, as shown above).

5.1.2 Poker example

Earlier in this chapter (Figures 5.1 and 5.2) we provided a canonical example of two Texas hold 'em hands with similar EHS, but very different histograms over equity at the end of the hand (KcQc vs. 6c6d). We now present an example of two hands that have similar histograms over equity at the final round (and thus also similar EHS), but realize their equity in very different ways throughout the hand.

Consider the two flop hands TcQd-7h9hQh and 5c9d-3d5d7d (the first two cards are the private cards, and the next three are the public flop cards). These are both relatively strong hands. The first hand has top pair (a pair of queens), and the second hand has second pair (a pair of fives) plus a flush draw (another diamond on the board would complete a flush). The hands both have very similar EHS, assuming both the turn and river are dealt; TcQd-7h9hQh has EHS 0.683 and 5c9d-3d5d7d has EHS 0.679. These two hands also have very similar full distributions over equity at the final round, as can be seen in Figures 5.7 and 5.8. The EMD between these two distributions is 0.559, where the unit is equity intervals (assuming fifty intervals of width 0.02, and a total unit mass for both distributions). This value is sufficiently low that these two hands are grouped into the same bucket by the leading distribution-aware abstraction algorithm (and therefore, they would be treated as being identical by equilibrium-finding algorithms).

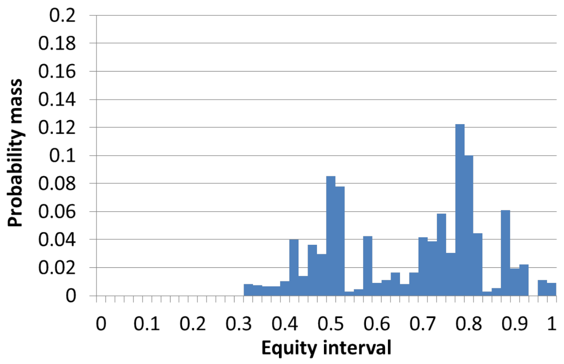


Figure 5.7: Equity distribution for TcQd-7h9hQh on the river (final betting round).

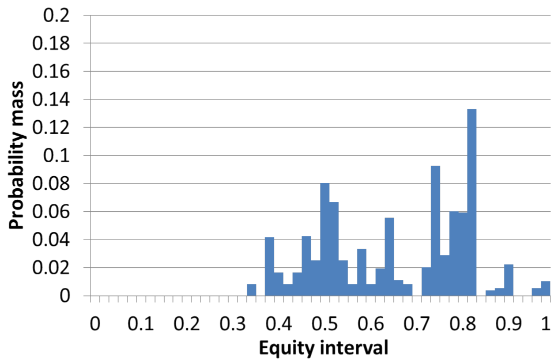


Figure 5.8: Equity distribution for 5c9d-3d5d7d on the river (final betting round).

However, despite similarities between the equity distributions after the river is dealt, these two hands realize their equity very differently throughout the hand. Figures 5.9 and 5.10 show their expected hand strength distributions on the turn (the next public card), assuming a uniform random distribution for the river card and the opponent's cards. These two distributions are very different; for example, a large portion of the time TcQd-7h9hQh will have a turn equity between 0.68 and 0.78, while 5c9d-3d5d7d has a large portion of its equity in the 0.56–0.66 range.

We note that Figures 5.9 and 5.10 depict the distributions of expected turn hand strength, as opposed to full distributions over distributions of river hand strength (since each turn card will lead to a distribution over strength in the next round, not a single value). For example, for the hand TcQd-7h9hQh, if the turn card is Ad, the hand's expected equity, assuming a uniform random river card and uniform random hand for the opponent, is 0.695 (though it will vary for individual river cards); so the interval for 0.68–0.7 in the histogram would be incremented by one for that turn card. This is significantly more simplistic than our potential-aware approach, which takes into account the full distribution of turn 'buckets', which are themselves distributions over equity intervals after the river. However, comparing these distributions is still useful for several reasons. First, if the full distributions over turn hand strength (which are, themselves, distributions over river hand strength) were similar, then the distributions over the expectation of turn hand strength distributions would necessarily be similar as well; thus, the fact that the expectation distributions differ significantly indicates that the full distributions also differ significantly. And second, it is not feasible to compactly represent the full distributions visually, while the distributions of expected hand strength can be represented easily as two-dimensional histograms.

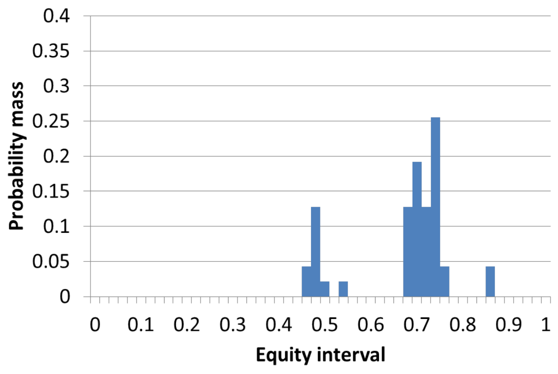


Figure 5.9: Equity distribution for TcQd-7h9hQh on the turn (next betting round). Each point is the expected hand strength for a given turn card assuming a uniform random distribution for the river card and the opponent's cards.

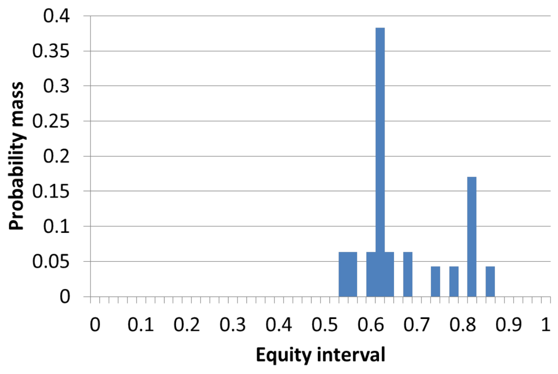


Figure 5.10: Equity distribution for 5c9d-3d5d7d on the turn (next betting round). Each point is the expected hand strength for a given turn card assuming a uniform random distribution for the river card and the opponent's cards.

While the EMD between the river equity distributions depicted in Figures 5.7 and 5.8 is 0.559, the EMD between full turn distributions using the potential-aware approach is 4.519 (using comparable units). Potential-aware abstraction is able to correctly identify that these hands are quite different, and places them into different buckets due to their large EMD (while the prior abstraction algorithm places them in the same bucket).

5.2 Algorithm for potential-aware imperfect-recall abstraction, with EMD

In this section, we present our new algorithm for computing potential-aware imperfect-recall abstractions using EMD. We first present our main algorithm, followed by a heuristic for quickly approximating the EMD in our setting that we use to make the algorithm practical for large games such as Texas hold 'em.

Our abstraction algorithm, depicted in Algorithm 1, works as follows. Assume the information tree has $r+1$ levels (0–r), and for each level n, a number of clusters C^n is specified as input. For the final rounds $n = \hat{r}, \ldots, r$, an arbitrary abstraction algorithm S^n is used, with distance function d^n, to produce C^n clusters; let A^n denote the resulting abstraction, and let m_i^n denote the mean of the i'th cluster in A^n. Next, we compute the abstraction at round $\hat{r}-1$ as follows. First, we compute the distance $d_{i,j}^n$ between each pair of round-(n+1) means m_i^{n+1} and m_j^{n+1}, using the distance metric d^{n+1} (for the application to poker, the means are multidimensional vectors (histograms) and d^{n+1} is the earth mover's distance). Next, we compute histograms $H^n(x^n)$, where the i-th element of $H^n(x^n)$ is the fraction of the time that chance's next move will send x^n into cluster i in A^{n+1}. Finally, we compute the abstraction A^n at round n, by clustering the histograms H^n into C^n clusters using clustering algorithm L^n (prior work in poker uses k-means).

The distance metric used, denoted d^n, is the EMD between the histograms, using $d^n_{i,j}$ as the ground distance between components i and j of a histogram. We then compute the new cluster means m^n_i, and continue in the same fashion for $n = \hat{r} - 2, \ldots, 0$. The resulting abstraction, A^n, has imperfect recall since we cluster all of the histograms $H^n(x^n)$ without any regard for the information known at the corresponding states at previous stages of the game, and potentially we cluster two states together that contain information that we could distinguish between at earlier rounds.

Algorithm 1 Main algorithm for computing potential-aware imperfect-recall abstractions

```
Inputs: $\{C^n\} : n = 0, \ldots, r$; $\{S^n\}, \{d^n\} : n = \hat{r}, \ldots, r$; $\{L^n\} : n = 0, \ldots, \hat{r} - 1$
  for $n = r$ to $\hat{r}$ do
      Compute abstraction $A^n$ with $C^n$ clusters using abstraction algorithm $S^n$ with distance function $d^n$
  end for
  for $n = \hat{r} - 1$ to 0 do
      for $i = 1$ to $C^n$ do
          $m^{n+1}_i \leftarrow$ mean of cluster $i$ in $A^{n+1}$
      end for
      for $i = 1$ to $C^n - 1$ do
          for $j = i + 1$ to $C^n$ do
              $d^n_{i,j} \leftarrow$ distance between $m^{n+1}_i$ and $m^{n+1}_j$ using distance function $d^{n+1}$
          end for
      end for
      for each point $x^n$ at round $n$ do
          $H^n(x^n) \leftarrow$ histogram for $x^n$ over clusters from $A^{n+1}$
      end for
      Compute abstraction $A^n$ over histograms $H^n$ using clustering algorithm $L^n$ and distance function $d^n$ (i.e., EMD with $d^n_{i,j}$ as the ground distance) to produce $C^n$ clusters
  end for
```

To compute the distances in the main loop of Algorithm 1 we implemented the fastest commercially-available multi-dimensional EMD algorithm [87, 88]; however, it was far too slow for the domain of Texas hold 'em. So we were forced to develop a faster heuristic for approximating EMD in this setting. Our heuristic is given in Algorithm 2. The context is that we are running k-means to compute an abstraction at level n of the tree, for which we must compute the distance between each 'point' and each mean. The 'points' correspond to the histograms $H^n(x^n)$ over clusters in A^{n+1}, and were computed in the previous step of Algorithm 1. The naïve way of representing them would be as vectors of dimension C^{n+1}. However, this vector may be very sparse. For example, if $C^{n+1} = 5000$ (as in our experiments), but the current point can only transition into 50 next-round clusters with positive probability, we would like to take advantage of a sparser representation rather than represent it as a vector of size 5000. Instead, we represent the point as a vector of length 50 (in this example), where each index corresponds to the index of the next-round cluster we transition to. For example, if a point can transition to clusters 3, 5, or 10, for different chance moves, then we represent the point as the vector $(3, 5, 10)$, where each of

Algorithm 2 Algorithm for efficiently approximating EMD in our setting

```
Inputs: Point $x^n$ with $N$ elements; mean $m$ with $Q$ elements; sortedDistances[i][j], orderedClusters[i][j], for $1 \leq i \leq C^{n+1}$, $1 \leq j \leq Q$
  targets[] $\leftarrow$ array of size $N$ with all elements equal to $\frac{1}{N}$
  meanRemaining[] $\leftarrow$ copy of $m$
  done[] $\leftarrow$ array of size $N$ with all elements set to false
  totCost $\leftarrow 0$
  for $i = 1$ to $Q$ do
      for $j = 1$ to $N$ do
          if done[j] == true then
              continue
          end if
          pointCluster $\leftarrow x^n[j]$
          meanCluster $\leftarrow$ orderedClusters[pointCluster][i]
          amtRemaining $\leftarrow$ meanRemaining[meanCluster]
          if amtRemaining == 0 then
              continue
          end if
          $d \leftarrow$ sortedDistances[pointCluster][i]
          if amtRemaining $<$ targets[j] then
              totCost += amtRemaining * d
              targets[j] -= amtRemaining
              meanRemaining[meanCluster] $\leftarrow 0$
          else
              totCost += targets[j] * d
              targets[j] $\leftarrow 0$
              meanRemaining[meanCluster] -= targets[j]
              done[j] $\leftarrow$ true
          end if
      end for
  end for
  return totCost
```

these will have probability $\frac{1}{3}$. Each point x^n in Algorithm 2 corresponds to such a vector, where N denotes the length. For simplicity we assume that all of the elements are distinct, though repeated elements can be dealt with straightforwardly.

We similarly take advantage of sparsity in representing the means. While each mean could potentially have C^{n+1} entries, many of these entries may be zero. Instead, we simply represent the mean m as the vector of the nonzero entries, of which we assume there are Q. In order to identify which clusters the entries correspond to, and to make our overall implementation more efficient, we utilize several data structures. First, we precompute an array called sortedDistances, where sortedDistances[i][j] is the distance between next-round cluster i and the j-th closest cluster to i for which the current mean has non-zero probability, where distances have already been computed using d^{n+1}. We also use a structure orderedClusters, where orderedClusters[i][j] is the index of the cluster that the mean assigns non-zero probability to that is j-th closest to cluster i. These arrays are precomputed in advance of the EMD computation; while they require some time to compute, the EMD computations are by far the bottleneck of the algorithm. This additional computation helps us overall since it significantly speeds up the EMD computations.

Given these data structures as input, we now approximate EMD as follows. First, we start with the first entry of x^n; we call the round-(n+1) cluster to which this belongs 'pointCluster.' We then find the closest cluster to pointCluster that corresponds to a nonzero element in the mean; this will be the element orderedClusters[pointCluster][1], which we call 'meanCluster.' We shift as much mass as possible between from this mean element to the corresponding point element. The cost is increased by the amount of mass we shift multiplied by the distance. We then update the remaining point mass and mean mass at the elements we have considered, and continue for the remaining point indices $j = 2, \dots, N$. Next, we set $i = 2$, and repeat the same procedure, now shifting as much mass as is available from the second closest nonzero mean cluster to each cluster of the point. We repeat this for $i = 3, \dots, Q$, until the mean vector has been fully transformed into the point vector. We then output the resulting total cost.

As mentioned above, the fastest commercially-available algorithm for computing EMD [87, 88] is far too slow to be effective in Texas hold 'em. (This was despite the fact that we integrated the data structures described above with this algorithm to exploit sparsity, as well as applied several other enhancements to improve performance of k-means, such as a pruning technique that exploits the triangle inequality [26] and parallelizing each step using 64 cores.) For the point-mean distances in the first round of k-means, the exact EMD algorithm averaged 11.4 ms per computation, while our heuristic averaged only 0.008 ms per computation. Furthermore, the exact algorithm scales extremely poorly as the dimensionality of the inputs increases. Since the initial means are themselves data points, their dimensionality is small; however, for future rounds of k-means, the means are weighted averages over all points in their cluster, and have higher dimensionality. Our new algorithm performs well even in the future rounds of k-means as this dimensionality increases, while the exact algorithm scales very poorly.

There are many potential further improvements to approximating EMD and doing clustering in this context. However, even with the techniques we already developed and tested, the approach outperforms the previously best abstraction algorithm, as the experiments in the next section will show.

5.3 Experiments

We evaluated our abstraction algorithm in a very large sequential imperfect-information game, two-player no-limit Texas hold 'em. While our abstraction algorithm works for any number of agents and does not assume a zero-sum game, we focused on this two-player zero-sum game in the experiments—as is most common in this field—so that we can compute a near equilibrium to a large abstraction, and thereby evaluate the results.

5.3.1 Head-to-head performance vs. best prior abstraction algorithm

We ran two different sets of experiments corresponding to two different manually-generated betting abstractions. In both experiments, we compared performance to the previously best abstraction algorithm [68]. In each experiment, we used the same betting abstraction for us and for the benchmark. In the first experiment, we used the betting abstraction that was used by the agent that finished in 2nd place in the 2012 Annual Computer Poker Competition. We chose to test on this relatively small betting abstraction so that we could obtain a good level of convergence to equilibrium in the abstract game. In the second experiment, we used a very large betting abstraction; it is about 8 times larger than the version used by our 2013 competition agent, and currently beats the winner from the 2013 competition.

In both experiments, we used 169, 5000, 5000, and 5000 card buckets respectively in the four betting rounds for both the new algorithm and the prior algorithm. This card abstraction is used by our strongest agent that now beats the winner from the 2013 competition, and was also used by the 2013 competition agent that finished in 3rd. Also, as is typical nowadays among all the top teams, the first round has 169 buckets corresponding to no abstraction at all.

In each of the two experiments, we created an agent that was identical to the corresponding opponent, except that it used our new algorithm to compute the abstraction for the flop round (i.e., second betting round). For both flop abstraction algorithms, we conducted 25 restarts using the k-means++ initialization procedure [6], and selected the run that produced the lowest within-cluster sum of squares. We chose to focus on the flop round for the following reasons. First, the strongest agents do not use any abstraction preflop (i.e., on the first betting round), and there is no potential on the river (i.e., last betting round) since no further cards will be dealt; so potential-aware abstraction would not help on those rounds. The approach is potentially useful on the turn (i.e., third betting round) as well, but it appears to be computationally intractable, even using our fast heuristic for EMD (there are around 1.3 million hands to be clustered on the flop, and 55 million on the turn). For each of the generated abstractions (two new and two benchmarks), we computed an approximate equilibrium for the abstraction using a sampled version of counterfactual regret minimization [77].

In each of the two experiments, we ran 20,000 duplicate matches between our new agent and the respective benchmark agent. In both experiments, our new approach led to a statistically significant improvement over the old approach. In the first experiment, the new agent beat its benchmark by 2.58 milli big blinds per hand (mbb/h) with a 95% confidence interval of ± 1.56 mbb/h. In the second experiment, the new agent beat its benchmark by 2.22 mbb/h (± 1.28 mbb/h).

5.3.2 Evaluating the approximation of potential-aware EMD

To evaluate how closely the distance computed by Algorithm 2 approximates the true potential-aware EMD, we repeatedly generated the histograms (over turn buckets) for two random flop hands, and computed the exact EMD between the histograms. If this distance was less than some threshold, then we also ran Algorithm 2 to approximate the EMD. (We chose a threshold of 3000^2 since it appears that the vast majority of the closest point-mean distances were in the 0–2500 range, and we are mostly interested in how well our heuristic does at approximating EMD for the point-mean pairs with low distance, since those represent the cluster to which a point might actually be assigned. We are not as concerned about how far off our heuristic is for distances that are extremely large, as they will likely be pruned and have no chance of being assigned as the closest mean.) We computed the relative error between the EMD computed by our heuristic and the true EMD, and averaged it over many samples. Our algorithm had average relative error of 0.1496 (with 95% confidence interval ± 0.0014).

For comparison, we also computed the EMD between the same flop hands using the previously best distribution-aware approach, where the histograms consider equity assuming the end of the game is reached. That approach produced an average relative error of 0.2084 (± 0.0014) compared to the potential-aware EMD over the same sample. Thus, our potential-aware, but heuristically-computed EMD, approximates the true potential-aware EMD 28.2% better than the prior approach for computing EMD, which used exact calculation but was not potential aware.

Though we already demonstrated the superiority of our abstraction algorithm over the prior algorithm in the experiments described in Section 5.3.1, these results provide a further sanity check that Algorithm 2 does in fact lead to a better degree of approximation of potential-aware EMD than the prior method. The results also indicate that there is still room for significant improvement toward more accurate potential-aware EMD computation.

5.4 Summary and extensions

I presented the first algorithm for computing potential-aware imperfect-recall abstractions using earth mover's distance as a distance metric. This is the first algorithm that combines potential-aware abstraction with imperfect recall. It is also the first time earth mover's distance has been used in potential-aware abstraction. Both of these are conceptually clear improvements, and experiments showed in the large that the new algorithm outperforms the best prior abstraction algorithm with statistical significance.

A future direction would be to develop more accurate and/or faster heuristics for approximating EMD in our setting, or faster algorithms for computing exact EMD. One technique for achieving the latter would be to use a clustering algorithm that selects cluster centers that have lower dimensionality than the centers selected by k-means, since the best commercially-available algorithm for computing EMD is slow for points with high dimensionality. A promising approach is the k-medoids algorithm, which only uses data points as the cluster centers. However, k-medoids is significantly slower than k-means and it requires additional memory, so it is unclear whether it will be feasible to apply to poker or other large games. We would also like to extend our approach to the turn round as well, though this appears to be infeasible, even using our fast heuristic, due to the large number of turn hands that need to be clustered. One technique that may help is to sample a subset of the turn hands and perform clustering only over that subset.

[2]These values must be divided by 1081 for the total histogram mass to be normalized to one, since for each flop hand there are $\frac{47 \cdot 46}{2} = 1081$ combinations of turn and river cards, and these combinations constitute the elements in the histograms.

We expect our algorithm to be applicable to games beyond no-limit Texas hold ’em (NLHE). We would expect it to lead to an even more significant performance improvement over the prior approach in certain other popular variants of poker, such as pot-limit Omaha hold ’em (aka PLO), since the strength of hands changes much more significantly between rounds than in Texas hold ’em, so taking the full trajectory of strength into account would be more important in that game. PLO also has a significantly larger state space than Texas hold ’em (in all rounds), and therefore we would expect abstraction to play a larger role than in NLHE. In particular, there are many more hands for the preflop and flop rounds in PLO than in NLHE, and we expect our approach to help the most when performing abstraction in the earlier rounds, since that is where there is the biggest difference between the distribution of hand strength assuming the final round is reached and the full trajectory of hand strength distributions in all future rounds.

We would also like to apply our algorithm to games outside of poker. It would be applicable to any large sequential game of imperfect information where information abstraction is necessary, and it would be especially useful for games where the strength of private information can change significantly between rounds, since the algorithm accounts for the full trajectory of strength over all future rounds.

Chapter 6

Hierarchical Abstraction Algorithm that Enables Massive Distributed Equilibrium Computation

The equilibrium-finding algorithm used by today's strongest Texas hold 'em agents is a Monte Carlo version of the counterfactual regret minimization algorithm (MCCFR) [77]. That algorithm involves repeatedly sampling chance outcomes and actions down the tree, and updating regret and average strategy values that are stored at each information set.

On a shared-memory architecture, MCCFR can be parallelized straightforwardly; however, true shared-memory architectures typically come with relatively little memory and relatively few cores, and it would be desirable for scalability to be able to run on architectures that have more memory (in order to be able to run on larger, more detailed abstractions) and more cores (for speed). However, on distributed architectures and supercomputers with high inter-blade[1] memory access latency, straightforward MCCFR parallelization approaches lead to impractically slow runtimes because when a core does an update at an information set it needs to read and write memory with high latency. Our approach solves this problem.[2]

To obtain these benefits, our algorithm creates an information abstraction that allows us to assign different components of the game tree to different blades so the trajectory of each sample only accesses information sets located on the same blade [18]. At a high level, the first stage of our hierarchical approach is to cluster public information at some early point in the game (public flop cards in the case of Texas hold 'em poker—see Chapter 3 for the rules), giving a global basis for distributing the rest of the game into non-overlapping pieces; then our algorithm conducts clustering of private information. A key contribution is the specific way to cluster the public information. As we will detail in Section 6.1, two prior abstraction algorithms motivated by similar considerations have been developed for poker by others [60, 120], but ours differs in

[1]Such supercomputers consists of *blades*, which are themselves computers that are plugged into racks. A core can access memory on its blade faster than memory on other blades—seven times faster on the computer we used. On regular distributed systems, the difference between local and remote memory access is even greater.

[2]Note that a recent parallel implementation of MCCFR that uses a new version of CFR called CFR+ and involves breaking the game into public subgames also successfully keeps the memory local [15]. Vanilla CFR+ is now the fastest solving variant.

that it does not use hand-crafted poker features, is applicable to the large, and does not have the conceptual weaknesses from which they suffer.

We developed an equilibrium-finding algorithm that can be applied to this abstraction. It is a modified version of external-sampling MCCFR [77]. Applied to TH, it samples one pair of preflop (i.e., first betting round) hands per iteration. For the later betting rounds, each blade samples public cards from its public cluster and performs MCCFR within each cluster. Our algorithm weighs the samples to remove bias. Ours is similar to the algorithm of Jackson [60]. However, we implement MCCFR instead of chance-sampled CFR, and split only based on public information (chance actions) rather than players' actions. Another related prior approach used vanilla CFR (which converges significantly slower in practice) and split based only on players' actions (which does support nearly as much parallelization) [62].

The new abstraction and equilibrium-finding algorithms enabled an equilibrium computation of unprecedented size on a supercomputer with high inter-blade memory access latency. Experiments also show that this run outperforms the strongest prior approach executed on a large shared-memory server with low memory latency but fewer cores. An agent for two-player no-limit Texas hold 'em that was generated using these techniques won the 2014 Annual Computer Poker Competition (ACPC), beating each opponent with statistical significance.

6.1 Abstraction algorithm

Our new hierarchical abstraction algorithm is domain independent, although in many places of the description we present it in the context of poker for concreteness. In order to enable distributed equilibrium finding, it creates an information abstraction that assigns disjoint components of the game tree to different blades so that sampling in each blade will only access information sets that are located on that blade.

At a high level, the first stage of our hierarchical abstraction algorithm is to cluster public information at some early point in the game (public flop boards, i.e., combinations of public flop cards, in the case of TH), giving a global basis for distributing the rest of the game into non-overlapping pieces. Then, as a second stage our algorithm conducts clustering of information states (that can include both public and private information) in a way that honors the partition generated in the first stage.

As an example, suppose that in the first stage we cluster public flop boards into 60 buckets. Suppose bucket 4 contains only the boards AsKhQd and AsKhJd. Then we cluster all private hands for each betting round, starting with the flop, i.e., the second round (we assume the abstraction for the preflop round has already been computed—the strongest agents, including ours, use no abstraction preflop). We perform abstraction over full (five-card) flop hands separately for each of the 60 blades. For blade 4, only the hands for which the public board cards are AsKhQd or AsKhJd are considered (for example, 5s4s-AsKhQd and QcJc-AsKhJd). There are 2,352 such hands. If we allowed an abstraction at the current round with 50 private buckets per blade, we would then group these 2,352 hands into 50 buckets (using some abstraction algorithm; we discuss ours in detail later). We then perform a similar procedure for the third (aka turn) and fourth (aka river) rounds, ensuring that the hands for each blade are limited only to the hands that contain a public flop board that was assigned to that blade in the first stage of the algorithm.

A game has *perfect recall* if, informally, no player ever forgets information that he knew at an earlier point in the game. This is a useful concept for several reasons. First, certain equilibrium-finding algorithms can only be applied to games with perfect recall [55, 72]. Second, other equilibrium-finding algorithms, such as CFR [125] and its sampling variants, have no theoretical guarantees in games that have imperfect recall, though they can still be applied. (One notable exception is recent work giving a theoretical guarantee of the performance of CFR in one class of imperfect-recall games called well-formed games [78].) And third, Nash equilibria are not even guaranteed to exist in general in behavioral strategies in games with imperfect recall.

Despite these limitations, poker agents using abstractions with imperfect recall have consistently been shown to outperform agents that use perfect recall abstractions [120]. Intuitively, perfect-recall abstractions force agents to distinguish all information at a later round in the tree that they were able to distinguish at an earlier round, even if such a distinction is not very significant at the later round. For example, if an agent can distinguish between Kh3c and Kh4c in the preflop round (as is the case in the abstractions of the best agents), then a perfect-recall abstraction would force them to be able to distinguish between Kh3c on a KsJd9h flop, and Kh4c on the same flop, despite the fact that the 3c vs. 4c distinction is extremely unlikely to play a strategic role in the hand. On the other hand, with imperfect recall, agents are not forced to remember all of these distinctions simply because they knew them at a previous round, and are free to group any hands together in a given round without regard to what information was known about them in prior rounds of the abstraction. The most successful prior abstraction algorithms use imperfect recall [35, 68].

Unfortunately, running CFR on imperfect-recall abstractions on a machine with high inter-blade memory access latency can be problematic, since regrets and strategy values at different buckets along a sample may be located on different blades. We now describe in detail our new approach that enables us to produce strong abstractions for this setting. Our approach requires players to remember certain information throughout the hand (public flop bucket), but does not force players to distinguish between other pieces of information that they may have been able to distinguish between previously (if such distinctions are no longer relevant). Thus, our approach achieves the benefits of imperfect recall to a large extent (though not the flexibility of full imperfect recall) while achieving partitioning of the game into disjoint pieces for different blades to work on independently.

6.1.1 Main abstraction algorithm

Our main abstraction algorithm, Algorithm 3, which is domain independent, works as follows. Let $\hat{r}$ be the special round of the game where we perform the public clustering. For the initial $\hat{r}-1$ rounds, we compute a (potentially imperfect-recall) abstraction using an arbitrary algorithm A_r for round r. For example, in poker the strongest agents use no abstraction in the preflop round (and even if they did use abstraction for it, it would not require public clustering and could be performed separately). Next, the public states at round $\hat{r}$ are clustered into C buckets. The algorithm for this public clustering is described in Section 6.1.2. Once this public abstraction has been computed, we compute abstractions for each round from $\hat{r}$ to R over all states of private information separately for each of the public buckets that have been previously computed. These abstractions can be computed using any arbitrary approach, A_r. For our poker agent, we used an abstraction algorithm that had previously been demonstrated to perform well as the A_r's [68].

Algorithm 3 Main abstraction algorithm

```
Inputs: number of rounds $R$; round where public information abstraction is desired $\hat{r}$; number of public buckets $C$; number of desired private buckets per public bucket at round $r$, $B_r$; abstraction algorithm used for round $r$, $A_r$
  for $r = 1$ to $\hat{r} - 1$ do
    cluster information states at round $r$ using $A_r$
  end for
  cluster public information states at round $\hat{r}$ into $C$ buckets (e.g., using Algorithm 4)
  for $r = \hat{r}$ to $R$ do
    for $c = 1$ to $C$ do
      cluster private information states at round $r$ that
      have public information in public bucket $c$ into $B_r$
      buckets using abstraction algorithm $A_r$
    end for
  end for
```

6.1.2 Algorithm for computing abstraction of public information

The algorithm used to compute the abstraction of public information at round $\hat{r}$ is shown as Algorithm 4. For TH, this corresponds to computing a bucketing of the public flop boards. To do this, we need a distance function $d_{i,j}$ between pairs of public states (or, equivalently, a similarity function $s_{i,j}$ that can be transformed into a distance function). We use this distance function to compute the public abstraction using the clustering algorithm described in Section 6.1.3.

Two prior approaches have been applied to abstract public flop boards. One uses poker-specific features that have been constructed manually [60]. The second, due to Waugh et al., uses k-means clustering with L_2 distance over transition tables that were constructed from a small perfect-recall *base abstraction* with 10 preflop buckets and 100 flop buckets [120]. The entry $T[f][i][j]$ in the table gives the probability of transitioning from preflop bucket i to flop bucket j in the abstraction when the public flop board is f. In addition to potentially prohibitive computational challenges of scaling that approach to large base abstractions (such as the one we will use, which has 169 preflop and 5,000 flop buckets), there are also conceptual issues, as the following example illustrates. Consider the similar public flop boards AhKs3d and AhKs2d. Suppose the base abstraction does not perform abstraction preflop and places 4c3s-AhKs3d and 4c2s-AhKs2d into the same flop bucket, (which we would expect, as they are very similar—both have bottom pair with a 4 "kicker"), say bucket 12, while it places 4c3s-AhKs2d and 4c2s-AhKs3d into bucket 13 (these hands are also very similar—the worst possible non-pair hand with a "gutshot" straight draw). Suppose 4c3s is in bucket 7 preflop and 4c2s is in bucket 8. Then the transition table for AhKs2d would have value 0 for the probability of transitioning from preflop bucket 7 into flop bucket 12, while it would have value 1 for transitioning from preflop bucket 8 into flop bucket 12 (and the reverse for AhKs3d). So the L_2 distance metric would maximally penalize the boards for this component, despite the fact that they should actually be considered very similar based on this component, since they map hands that are extremely similar to the same bucket. Our new approach accounts for this problem by building a distance function based on how often public boards result in a given flop bucket in the base abstraction for *any* private cards (not necessarily the same private cards, as the prior approach has done).

We have developed an efficient approach that was able to use the strong 169-5,000-5,000-5,000 imperfect-recall abstraction as its base. We refer to this abstraction as A. The algorithm is game independent, and pseudocode (that is not specific to poker) is presented in Algorithm 4. As in Waugh's approach described above, we first compute a transition table T that will be utilized later in the algorithm, though our table will contain different information than theirs. For concreteness, and to demonstrate the implementation used by our agent so that it can be replicated, we will describe how the table is constructed in the context of TH poker.

We first construct a helper table called PublicFlopHands. The entry PublicFlopHands[i][j] for $1 \leq i \leq 1,755$, $1 \leq j \leq 3$ gives the j'th public flop card corresponding to index i, using a recently developed indexing algorithm that accounts for all suit isomorphisms [118] (there are $\frac{52 \cdot 51 \cdot 50}{6} = 22,100$ total public flop hands, but only 1,755 after accounting for all isomorphisms). We specify one such canonical hand for each index. Next, using this table, we create the transition table T, where the entry $T[i][j]$ for $1 \leq i \leq 1,755$, $1 \leq j \leq 5,000$ gives the number of private card combinations for which a hand with public flop i transitions into bucket j of the abstraction A, which has $B = 5,000$ buckets. This is computed by iterating over all public flop indices, then looking up the canonical hand in PublicFlopHands, and iterating over the $\frac{49 \cdot 48}{2} = 1,176$ possible private card combinations given that public flop hand. We then construct the 5-card flop hand by combining the two private cards with the given public flop hand, look up the index of this hand (again using Waugh's indexing algorithm), and then look up what bucket A places that flop hand index into. Thus, the creation of the transition table involves iterating over $1,755 \cdot 1,176 = 2,063,880$ combinations, which can be done quickly.

In poker-independent terms, $T[i][j]$ stores how often public state i will lead to bucket j of the base abstraction, aggregated over all possible states of private information. In contrast, Waugh's table stores separate transition probabilities for each state of private information.

We would like our distance function to assign a small value between public states that are frequently grouped into the same bucket by A, since we already know A to be a very strong abstraction. We compute distances by iterating over the B (private) buckets in round $\hat{r}$ of A. We initialize a variable $s_{i,j}$ which corresponds to the similarity between i and j to be zero. For each bucket b, let c_i denote the number of private states with public state i that are mapped to b under A (and similarly for c_j). For example, suppose i corresponds to the public flop board of AsQd6h and $b = 7$. Then c_i would denote the number of private preflop card combinations (x,y), such that the flop hand xy-AsQd6h is placed in bucket 7 under A. We then increment $s_{i,j}$ by the minimum of c_i and c_j. For example, if $c_i = 4$ and $c_j = 12$, this would mean that i and j are *both* placed into the current bucket b four times. Then the distance $d_{i,j}$ is defined as $\frac{V - s_{i,j}}{V}$, which corresponds to the fraction of private states that are not mapped to the same bucket of A when paired with public information i and j.[3]

[3]Note that d is not a distance metric. It is possible to have $d_{i,j} = 0$ for boards that are different, if the boards send the same number of preflop hands into each flop bucket in A. Thus, we view d as an arbitrary matrix of distances rather than viewing the space as a metric space. This will affect selection of the clustering algorithm, described in Section 6.1.3.

Algorithm 4 Algorithm for computing abstraction of public information

```
Inputs: number of public buckets $C$; number of public states $M$; number of private information sets per public state $V$; prior abstraction $A$ with $B$ buckets; transition table $T$ for public states into buckets of $A$; clustering algorithm $L$
  for $i = 1$ to $M - 1$ do
      for $j = i + 1$ to $M$ do
          $s_{i,j} \leftarrow 0$
          for $b = 1$ to $B$ do
              $c_i \leftarrow T[i][b]$, $c_j \leftarrow T[j][b]$, $s_{i,j}$ += min($c_i$, $c_j$)
          end for
          $d_{i,j} \leftarrow \frac{V - s_{i,j}}{V}$
      end for
  end for
  Cluster the $M$ public states into $C$ clusters using $L$ with distance function $d$
```

For our application of Algorithm 4 to poker, the number of public buckets we used is $C = 60$, the total number of private states for each public state is $V = 1,176$, and $B = 5,000$ as described above. The full number of public flop boards after accounting for all suit isomorphisms is $M = 1,755$. Thus, to compute all of the distances we must iterate over $\frac{BN(N-1)}{2}$ = 7.7 billion triples. This can be performed quickly in practice, since for each item we only need to perform lookups in the precomputed transition table.

6.1.3 Public abstraction clustering algorithm

Given the distance function we have computed, we next perform the clustering of the public states into C public clusters, using the procedure shown in Algorithm 5. The initial clusters c^0 are computed by applying k-means++ [6], using the pairwise point distance function $d_{i,j}$, which is taken as an input. The k-means++ initialization procedure only requires knowing distances between data points, not distances from a point to a non-data-point. Next, for each iteration t, we iterate over all points i. We initialize clusterDistances to be an array of size K of all zeroes, which will denote the distance between point i and each of the current clusters. We then iterate over all other points $j \neq i$, and increment clusterDistances[$c^{t-1}[j]$] by $d_{i,j}$. Once we have iterated over all values of j, we let $c^t[i]$ denote the cluster with smallest distance from i. If no clusters changed from the clustering at the previous iteration, we are done. Otherwise, we continue this procedure until T iterations have been performed, at which point we output $c^T[i]$ as the final abstraction.

This algorithm only takes into account distances between pairs of data points, and not distances between points in the space that are not data points (such as means). Clustering algorithms that are designed for metric spaces, such as k-means, are not applicable to this setting.[4]

[4]We could have used the k-medoid algorithm (though it has a significant computational overhead over our approach, both in terms of running time and memory), or used the objective of minimizing the average distance of each point from the points in a cluster (rather than the sum). It would be interesting to explore the effect of using different choices for the clustering objective on abstraction quality. We chose the sum objective because it is computationally

Algorithm 5 Clustering algorithm for public abstraction

Inputs: Number of public states to cluster M; desired number of clusters K; distances $d_{i,j}$ between each pair of points; number of iterations to run T

```
Compute initial clusters $c^0$ (e.g., using k-means++)
for $t$ = 1 to $T$ do
    for $i = 1$ to $M$ do
        clusterDistances ← array of size $K$ of zeroes
        for $j = 1$ to $M$, $j \neq i$ do
            clusterDistances[$c^{t-1}[j]$] += d$_{i,j}$
        end for
        $c^t[i]$ ← cluster with smallest distance
    end for
    if no clusters were changed from previous iteration then break
    end if
end for
```

6.2 Equilibrium-finding algorithm

To solve the abstract game, one needs an algorithm that converges to a Nash equilibrium. The most commonly used equilibrium-finding algorithm for large imperfect-information extensive-form games is counterfactual regret minimization (CFR) and its extensions.

There is a large benefit to not needing to sample all actions at every iteration of CFR, and the variants that selectively sample more promising actions more often are Monte Carlo CFR (MCCFR) and Pure CFR. The external sampling variant of MCCFR converges faster than Pure CFR in practice but requires twice as much memory [39]. We build our equilibrium-finding algorithm starting from MCCFR because it converges faster and we are no longer memory constrained since we can run on distributed architectures.

External-Sampling MCCFR (ES-MCCFR) does a separate iteration for each player. On a player's iteration, ES-MCCFR samples opponent action and chance nodes down the tree (while exploring all of the player's actions). Actions are selected according to regret minimization. Regret is updated in the player's information sets, while average strategy is updated for encountered opponent information sets. This is problematic on a machine with high inter-blade memory access latency because the information sets traversed on a single iteration can be located on different blades. On the supercomputer we used, for example, accessing memory on the same blade takes 130 nanoseconds, while accessing memory on different blades takes about one microsecond.

As discussed in the previous section, our new abstraction addresses this issue by ensuring that after a certain point (for the flop through river rounds in the case of TH) all remaining information sets encountered in the current MCCFR iteration are stored on the same blade (i.e., the blade that the public flop was assigned to in the first stage of the abstraction algorithm.)

My collaborator Noam Brown has developed a modification of MCCFR specifically for ar-

feasible and gives a clustering with clusters of more balanced sizes than the average objective.

chitectures with high inter-blade memory access latency, which is able to capitalize on the abstraction that has been computed using the new hierarchical abstraction algorithm described in Section 6.1. The algorithm designates one blade as the "head" blade, which stores the regrets and average strategies for the top part of the game tree (preflop round in TH). The algorithm samples private information and conducts MCCFR on the head blade. When an action sequence is reached that transitions outside the top of the game tree (to the flop in TH), the algorithm sends the current state to each of the child blades. Each child blade then samples public information from its public bucket and continues the iteration of MCCFR. Once all the child blades complete their part of the iteration, their values are returned to the head blade. The head blade calculates a weighted average of these values, weighing them by the number of choices of public information (possible flops in TH) that they sampled from. The head node then continues its iteration of MCCFR, repeating the process whenever the sample exits the top part (a flop sequence is encountered) until the iteration is complete.

6.3 Experiments

We experimented on the version of two-player no-limit Texas hold 'em (nlhe) used in the ACPC, which has 10^{165} nodes [63] in its game tree. Our agent used the new hierarchical abstraction and distributed equilibrium-finding algorithms described in this chapter, as well as the new action translation mapping which will be described in Chapter 7 and the new post-processing technique which will be described in Section 8.7.

We used our new abstraction algorithm to create an information abstraction with 169 preflop buckets, 60 public flop buckets, and 500 private buckets for the flop, turn, and river for each of the public flop buckets, that is, 30,000 total private buckets for each of the three postflop rounds. Our action abstraction had 6,104,546 nodes (including leaves). In total, our abstract game then had $5.49 \cdot 10^{15}$ nodes (including leaves), $6.6 \cdot 10^{10}$ information sets (not including leaves), and $1.8 \cdot 10^{11}$ *infoset actions* (a new measure of game size that is directly proportional to the amount of memory that CFR uses [63]). This is six times larger than the largest abstractions used by prior nlhe agents—and, to our knowledge, the largest imperfect-information game ever tackled by an equilibrium-finding algorithm. This scale was enabled by our new, distributed approach.

We ran our equilibrium-finding algorithm for 1,200 hours on a supercomputer (Blacklight) with a high inter-blade memory access latency using 961 cores (60 blades of 16 cores each, plus one core for the head blade), for a total of 1,153,200 core hours. Each blade had 128 GB RAM.

The results from the 2014 ACPC against all opponents are shown in Table 6.1. The units are milli big blinds per hand (mbb/h), and the $\pm$ indicates 95% confidence intervals. Our agent beat each opponent with statistical significance, with an average win rate of 479 mbb/h. It won both of the scoring categories: the total bankroll (which ranks agents by the total amount won against all opposing agents) and the bankroll instant run-off (which iteratively removes the agent with lowest total bankroll—this scoring rule favors agents that are close to equilibrium).

We also compared our algorithm's performance to using the prior best approach on a low-latency shared-memory server with 64 cores and 512 GB RAM. This is at the upper end of shared-memory hardware commonly available today. The algorithm run on the server used external sampling MCCFR on an imperfect-recall card abstraction with 169, 5,000, 5,000, and

O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13
261 ± 47	121 ± 38	21 ± 16	33 ± 16	20 ± 16	125 ± 44	499 ± 68	141 ± 45	214 ± 57	516 ± 61	980 ± 34	1474 ± 180	1819 ± 111

Table 6.1: Win rate (in mbb/h) of our agent in the 2014 Computer Poker Competition against opposing agents.

5,000 bucket in the four respective betting rounds (this size was selected because it is slightly under the capacity of 512 GB RAM). We computed that information abstraction using the state-of-the-art non-distributed abstraction algorithm [35]. We used the same action abstraction as for the distributed case. The abstract game then had $1.5 \cdot 10^{14}$ nodes (including leaves), $1.1 \cdot 10^{10}$ information sets (not including leaves), and $3.1 \cdot 10^{10}$ infoset actions.

We benchmarked both against the two strongest agents from the 2013 competition, Figure 6.1.[5] The new approach outperformed the old against both agents for all timestamps tested. So, it is able to effectively take advantage of the additional distributed cores and RAM.

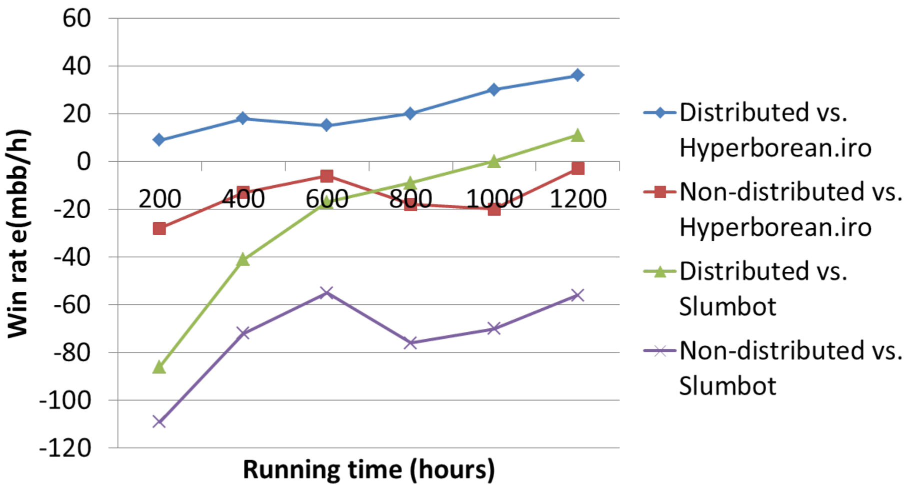


Figure 6.1: Win rates over time against the two strongest agents from the 2013 poker competition.

6.4 Summary and extensions

We introduced a distributed version of the most commonly used algorithm for large-scale equilibrium computation, counterfactual regret minimization (CFR), which enables CFR to scale to dramatically larger abstractions and numbers of cores. Specifically, we based our algorithm on external-sampling Monte Carlo CFR. The new algorithm begets constraints on the abstraction so as to make the pieces running on different computers disjoint. We introduced an algorithm for generating such abstractions while capitalizing on state-of-the-art abstraction ideas such as imperfect recall and the earth-mover's-distance similarity metric. Our techniques enabled an equi-

[5]Both our distributed and parallel algorithms were evaluated in play with purification (except no post-processing of the first action), which had been shown to perform best among prior techniques. This is also one of the benchmarks we evaluate in the experiments presented in Table 8.5.

librium computation of unprecedented size on a supercomputer with a high inter-blade memory latency. Prior approaches run slowly on this architecture. Our approach also leads to a significant improvement over using the prior best approach on a large shared-memory server with low memory latency. We applied these techniques to generate an agent for two-player no-limit Texas hold 'em. It won the 2014 Annual Computer Poker Competition, beating each opponent with statistical significance.

The techniques are game independent. While we presented them for a setting that does not require abstraction before the public information arrives, and there is only one round of public information, they can be extended to settings with any sequence of interleaved public and private information delivery—while keeping the information sets on different blades disjoint. Also, while we presented techniques for two levels in the distribution tree (one blade to handle the top part and the rest split disjointly among the other blades), it is easy to see how the same idea can be directly extended to trees with more than two levels of blades.

Chapter 7

Action Translation

Thus far, I have described approaches for the first two steps of the leading paradigm from Figure 4.1. In many games, once the abstract equilibrium has been computed, it can be implemented directly in the real game, and the "reverse mapping" step is trivial. However, as I will show, there can be significant benefits to performing more sophisticated reverse-mapping algorithms. First, if we performed action abstraction when constructing the abstraction, we need a technique for determining how to respond when the opponent takes an action that has been removed from the model [34]. For example, we may have limited bids to multiples of \$100, but the opponent makes a bid of \$215. We need an intelligent way of interpreting and responding to such actions which are not in our abstraction. The standard approach for doing this is to apply an *action translation mapping* [48], [103]), which maps the observed action a of the opponent to an action a' in the abstraction; then we simply respond as if the opponent had played a' instead of a. A natural action translation mapping would be to map the observed action to the closest action in our abstraction (according to a natural distance metric); in the example just described, this mapping would map the bid of \$215 to \$200. However, this is just one possible mapping, and significantly more sophisticated ones are possible.

This chapter will present a new approach for the action translation problem, which an essential step if action translation has been performed. Next, in Chapter 8, I will describe further post-processing approaches can be applied in addition to action translation, and also in games where no action abstraction was performed. Those approaches work by modifying the action probabilities of the abstraction equilibrium in desirable ways. Action translation and these further post-processing approaches together comprise the final "reverse mapping" step of the leading paradigm.

Several prior action translation mappings have been proposed for the domain of no-limit Texas hold 'em [3, 48, 97, 103]. However, these have all been based on heuristics and lack any theoretical justification. We show that most of the prior approaches violate certain natural desiderata and that all of them are highly exploitable in simplified games. (Exploitability in such simplified games is a standard evaluation technique since it cannot be computed in the large.) We present a new mapping, called the *pseudo-harmonic mapping*, that satisfies these desiderata and has significantly lower exploitability than the prior mappings. Thus, we expect our mapping to perform much better than the prior ones against sophisticated adaptive opponents who are specifically trying to exploit our mapping. (For one, any strong human poker player would try this against a computer program.) Furthermore, we observe that the cost of this worst-case performance benefit (low exploitability) is not high in practice; our mapping performs competitively with the prior mappings against no-limit Texas hold 'em agents submitted to the 2012 Annual Computer Poker Competition.

7.1 Problem formulation

Suppose the set of allowable actions at a given information set is some subset of the real interval $S = [\underline{T}, \overline{T}]$. (In no-limit poker, $\underline{T}$ will be zero and $\overline{T}$ will be the stack size of the player to act.) An action abstraction at this information set will correspond to a finite increasing sequence $(A_0, \ldots, A_k)$ with $\underline{T} \leq A_0$ and $A_k \leq \overline{T}$. (In our experiments we will set $A_0 = \underline{T}$ and $A_k = \overline{T}$; that is, the interval boundaries will be in our abstraction. In abstractions where that is not the case, actions that fall outside of $[A_0, A_k]$ can simply be mapped to A_0 or A_k.)

Now suppose the opponent takes some action $x \in S$. Let $A = \max\{A_i : A_i \leq x\}$, and let $B = \min\{A_i : A_i \geq x\}$. Then $x \in [A, B]$, where $\underline{T} \leq A \leq B \leq \overline{T}$. The action translation problem is to determine whether we should map x to A or to B (perhaps probabilistically). Thus, our goal is to construct a function $f_{A,B}(x)$, which denotes the probability that we map x to A ($1 - f_{A,B}(x)$ denotes the probability that we map x to B). This is our *action translation mapping*. Ideally we would like to find the mapping that produces the lowest exploitability when paired with a given action abstraction and equilibrium-finding algorithm. We call the value x^* for which $f_{A,B}(x^*) = \frac{1}{2}$ the *median* of f (if it exists).

7.2 No-Limit poker

We will evaluate different action translation mappings empirically in several variants of two-player no-limit poker. The rules of no-limit Texas hold 'em are described in Chapter 3, and the rules of several new variants considered are described here.

7.2.1 Clairvoyance game

In the *clairvoyance game* [4], player P2 is given no private cards, and P1 is given a single card drawn from a distribution that is half winning hands and half losing hands. Both players have stacks of size n, and they both ante \$0.50 (so the initial size of the pot is \$1). P1 is allowed to bet any amount $x \in [0, n]$. Then P2 is allowed to call or fold (but not raise).

7.2.2 Kuhn poker

No-limit Kuhn poker is similar to the clairvoyance game, except that both players are dealt a single private card from a three-card deck containing a King, Queen, and a Jack [4, 76].[1] For Kuhn poker and the clairvoyance game, we restrict all bets to be multiples of \$0.10.

[1]In limit Kuhn poker, player 2 is allowed to bet following a check of player 1; this is not allowed in no-limit Kuhn poker.

7.2.3 Leduc hold ’em

In Leduc hold ’em, both players are dealt a single card from a 6-card deck with two Kings, two Queens, and two Jacks. Both players start with \$12 in their stack, and ante \$1 [103, 119]. There is initially a round of betting, then one community card is dealt and there is a second round of betting. Any number of bets and raises is allowed (up to the number of chips remaining in one’s stack).

7.2.4 Texas hold ’em

For comparison to the new variants, I briefly summarize Texas hold ’em again here. In Texas hold ’em, both players are dealt two private cards from a 52-card deck. Using the parameters of the Annual Computer Poker Competition, both players have initial stacks of size 20,000, with a small blind of 50 and big blind of 100. The game has four betting rounds. The first round takes place before any public information has been revealed. Then three public cards are dealt, and there is a second betting round. One more public card is then dealt before each of the two remaining betting rounds.

7.3 Action translation desiderata

Before presenting an analysis of action translation mappings for the domain of poker, we first introduce a set of natural domain-independent properties that any reasonable action translation mapping should satisfy.

1. **Boundary Constraints.** If the opponent takes an action that is actually in our abstraction, then it is natural to map his action to the corresponding action with probability 1. Hence we require that $f(A) = 1$ and $f(B) = 0$.
2. **Monotonicity.** As the opponent’s action moves away from A towards B, it is natural to require that the probability of his action being mapped to A does not increase. Thus we require that f be non-increasing.
3. **Scale Invariance.** This condition requires that scaling A, B, and x by some multiplicative factor $k > 0$ does not affect the mapping. In poker for example, it is common to scale all bet sizes by the size of the big blind or the size of the pot. Formally, we require

$$\forall k > 0, x \in [A, B], f_{kA,kB}(kx) = f_{A,B}(x).$$

4. **Action Robustness.** We want f to be robust to small changes in x. If f changes abruptly at some x^*, then the opponent could potentially significantly exploit us by betting slightly above or below x^*. Thus, we require that $f_{A,B}$ is continuous in x, and preferably Lipschitz continuous as well.[2]

[2]A function $f : X \to Y$ is *Lipschitz continuous* if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in X$, $d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$.

5. **Boundary Robustness.** We also want f to be robust to small changes in A and B. If a tiny change in A (say from A_1 to A_2) caused $f_{A,B}(x)$ to change dramatically, then it would mean that f was incorrectly interpreting a bet of size x for either $A = A_1$ or $A = A_2$, and could be exploited if the boundary happened to be chosen poorly. Thus, we require that f be continuous and ideally Lipschitz continuous in A and B.

7.4 Prior mappings

Several action translation mappings have been proposed for no-limit Texas hold 'em [3, 48, 97, 103]. In this section we describe them briefly. In later sections, we will analyze the mappings in more detail, both empirically and theoretically. For all the mappings, we assume that the pot initially has size 1 and that all values have been scaled accordingly.

7.4.1 Deterministic arithmetic

The deterministic arithmetic mapping is the simple mapping described in the introduction. If $x < \frac{A+B}{2}$, then x is mapped to A; otherwise x is mapped to B. In poker, this mapping can be highly exploitable. For example, suppose A is a pot-sized bet (e.g., of 1) and B is an all-in (e.g., of 100). Then the opponent could significantly exploit us by betting slightly less than $\frac{A+B}{2}$ with his strong hands. Since we will map his bet to A, we will end up calling much more often than we should with weaker hands. For example, suppose our strategy calls a pot-sized bet of 1 with probability $\frac{1}{2}$ with a medium-strength hand. If the opponent bets 1 with a very strong hand, his expected payoff will be $1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} = 1.5$. However, if instead he bets 50, then his expected payoff will be $1 \cdot \frac{1}{2} + 51 \cdot \frac{1}{2} = 26$. In fact, this phenomenon was observed in the 2007 Annual Poker Competition when the agent Tartanian1 used this mapping [48].

7.4.2 Randomized arithmetic

This mapping improves upon the deterministic mapping by incorporating randomness [3, 48]:

$$f_{A,B}(x) = \frac{B - x}{B - A}$$

Now a bet at $x^* = \frac{A+B}{2}$ is mapped to both A and B with probability $\frac{1}{2}$. While certainly an improvement, it turns out that this mapping is still highly exploitable for similar reasons. For example, suppose the opponent bets 50.5 in the situation described above, and suppose that we will call an all-in bet with probability $\frac{1}{101}$. Then his expected payoff will be

$$\frac{1}{2}(1 \cdot \frac{1}{2} + 51.5 \cdot \frac{1}{2}) + \frac{1}{2}(1 \cdot \frac{100}{101} + 51.5 \cdot \frac{1}{101}) = 13.875.$$

This mapping was used by the agent AggroBot [3].

7.4.3 Deterministic geometric

In contrast to the arithmetic approaches, which consider differences from the endpoints, the deterministic geometric mapping uses a threshold x^* at the point where the ratios of x^* to A and B to x^* are the same [48]. In particular, if $\frac{A}{x} > \frac{x}{B}$ then x is mapped to A; otherwise x is mapped to B. Thus, the threshold will be $x^* = \sqrt{AB}$ rather than $\frac{A+B}{2}$. This will diminish the effectiveness of the exploitation described above; namely to make a large value bet just below the threshold. This mapping was used by the agent Tartanian2 in the 2008 Annual Computer Poker Competition [48].

7.4.4 Randomized geometric 1

Two different randomized geometric approaches have also been used by strong poker agents. Both behave similarly and satisfy $f_{A,B}(\sqrt{AB}) = \frac{1}{2}$. The first has been used by at least two strong agents in the competition, Sartre and Hyperborean [97, 103]:

$$g_{A,B}(x) = \frac{\frac{A}{x} - \frac{A}{B}}{1 - \frac{A}{B}} \qquad h_{A,B}(x) = \frac{\frac{x}{B} - \frac{A}{B}}{1 - \frac{A}{B}}$$

$$f_{A,B}(x) = \frac{g_{A,B}(x)}{g_{A,B}(x) + h_{A,B}(x)} = \frac{A(B-x)}{A(B-x) + x(x-A)}$$

7.4.5 Randomized geometric 2

The second one was used by another strong agent, Tartanian4, in the 2010 competition:

$$f_{A,B}(x) = \frac{A(B+x)(B-x)}{(B-A)(x^2+AB)}$$

7.5 Our new mapping

The prior mappings have all been based on heuristics without theoretical justification. We propose a new mapping that is game-theoretically motivated as the generalization of the solution to a simplified game—specifically, the clairvoyance game described in Section 7.2.1. The clairvoyance game is small enough that its solution can be computed analytically [4]:

- P1 bets n with probability 1 with a winning hand.
- P1 bets n with probability $\frac{n}{1+n}$ with a losing hand (and checks otherwise).
- For all $x \in [0, n]$, P2 calls a bet of size x with probability $\frac{1}{1+x}$.

It was shown by Ankenman and Chen [4] that this strategy profile constitutes a Nash equilibrium.[3] Here is a sketch of that argument.

[3]In fact, these betting and calling frequencies have been shown to be optimal in many other poker variants as well.

Proposition 1. *The strategy profile presented in Section 7.5 is a Nash equilibrium of the clairvoyance game.*

Proof. First, it is shown that player 2 must call a bet of size x with probability $\frac{1}{1+x}$ in order to make player 1 indifferent between betting x and checking with a losing hand. For a given x, player 1 must bluff $\frac{x}{1+x}$ as often as he value bets for player 2 to be indifferent between calling and folding. Given these quantities, the expected payoff to player 1 of betting size x will be $v(x) = \frac{x}{2(1+x)}$. This function is monotonically increasing, and therefore player 1 will maximize his payoff by setting $x = n$ and going all-in. $\square$

It turns out that player 2 does not need to call a bet of size $x \neq n$ with exact probability $\frac{1}{1+x}$: he need only not call with such an extreme probability that player 1 has an incentive to change his bet size from n to x (with either a winning or losing hand). In particular, it can be shown that player 2 need only call a bet of size x with any probability (which can be different for different values of x) in the interval $\left[\frac{1}{1+x}, \min\left\{\frac{n}{x(1+n)}, 1\right\}\right]$ in order to remain in equilibrium. Only the initial equilibrium is reasonable, however, in the sense that we would expect a rational player 2 to maintain the calling frequency $\frac{1}{1+x}$ for all x so that he continues to play a properly-balanced strategy in case player 1 happens to bet x.

Using this as motivation, our new action translation mapping will be the solution to

$$f_{A,B}(x) \cdot \frac{1}{1+A} + (1 - f_{A,B}(x)) \cdot \frac{1}{1+B} = \frac{1}{1+x}.$$

Specifically, our mapping is

$$f_{A,B}(x) = \frac{(B-x)(1+A)}{(B-A)(1+x)}.$$

This is the only mapping consistent with player 2 calling a bet of size x with probability $\frac{1}{1+x}$ for all $x \in [A, B]$.

This mapping is not as susceptible to the exploitations previously described. The median of f is

$$x^* = \frac{A + B + 2AB}{A + B + 2}.$$

As for the arithmetic and geometric mappings, we define both deterministic and randomized versions of our new mapping. The randomized mapping plays according to f as described above, while the deterministic mapping plays deterministically using the threshold x^*.

If we assumed that a player would call a bet of size x with probability $\frac{1}{x}$ instead of $\frac{1}{1+x}$, then the median would be the *harmonic mean* of the boundaries A and B: $\frac{2AB}{A+B}$. Because of this resemblance,[4] we will call our new mapping the *pseudo-harmonic mapping*. We will abbreviate the deterministic and randomized versions of the mapping as Det-psHar and Rand-psHar.

[4]We call our mapping *pseudo*-harmonic because it is actually quite different from the one based on the harmonic series. For example, for $A = 0$ and $B = 1$ the median of the new mapping is $\frac{1}{3}$, while the harmonic mean is 0.

7.6 Graphical examples

In Figure 7.1 we plot all four randomized mappings using $A = 0.01$ and $B = 1$. As the figure shows, both of the randomized geometric mappings have a median of 0.1 pot, while the median of the arithmetic mapping is around 0.5 pot and the median of the pseudo-harmonic mapping is around 0.34 pot. In this case, the mappings differ significantly.

In Figure 7.2, we plot the mappings using $A = 1$ and $B = 4$. In this case the pseudo-harmonic mapping is relatively similar to the geometric mappings, while the arithmetic mapping differs significantly from the others.

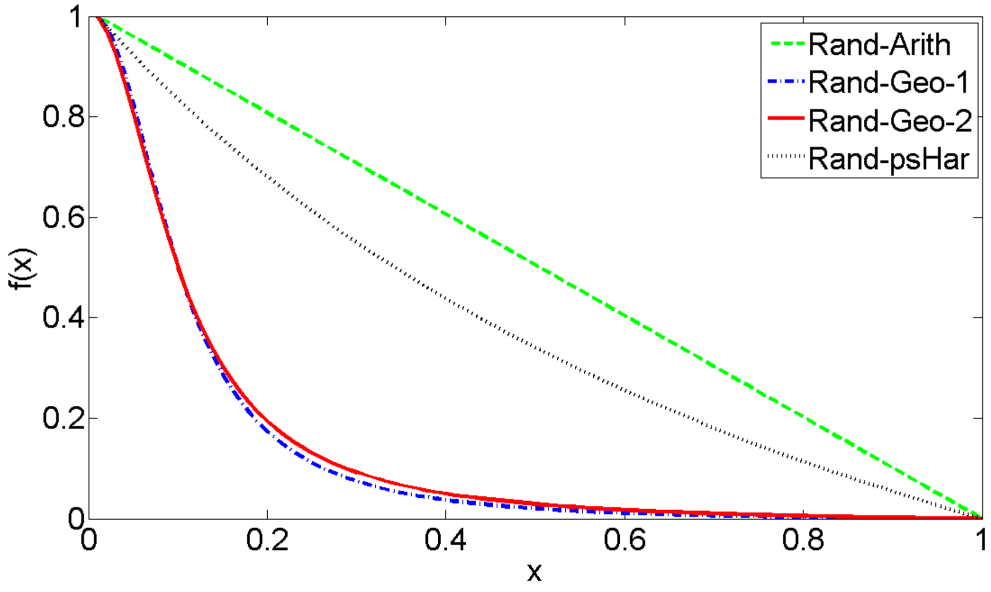


Figure 7.1: Randomized mappings with $A = 0.01$, $B = 1$.

7.7 Theoretical analysis

Before we present an axiomatic analysis of the mappings, we first note that $A = 0$ is somewhat of a degenerate special case. In particular, the geometric mappings are the constant function $f = 0$ for $A = 0$, and they behave much differently than they do for $A > 0$ (even for A arbitrarily small). So we will analyze these mappings separately for the $A = 0$ and $A > 0$ cases. In many applications it is natural to have $A = 0$; for example, for the interval between a check and a pot-sized bet in poker, we will have $A = 0$ and $B = 1$. So the degenerate behavior of the geometric mappings for $A = 0$ can actually be a significant problem in practice.[5]

[5]Some poker agents never map a bet to 0, and map small bets to the smallest positive betting size in the abstraction (e.g., $\frac{1}{2}$ pot). This approach could be significantly exploited by an opponent who makes extremely small bets as bluffs, and is not desirable.

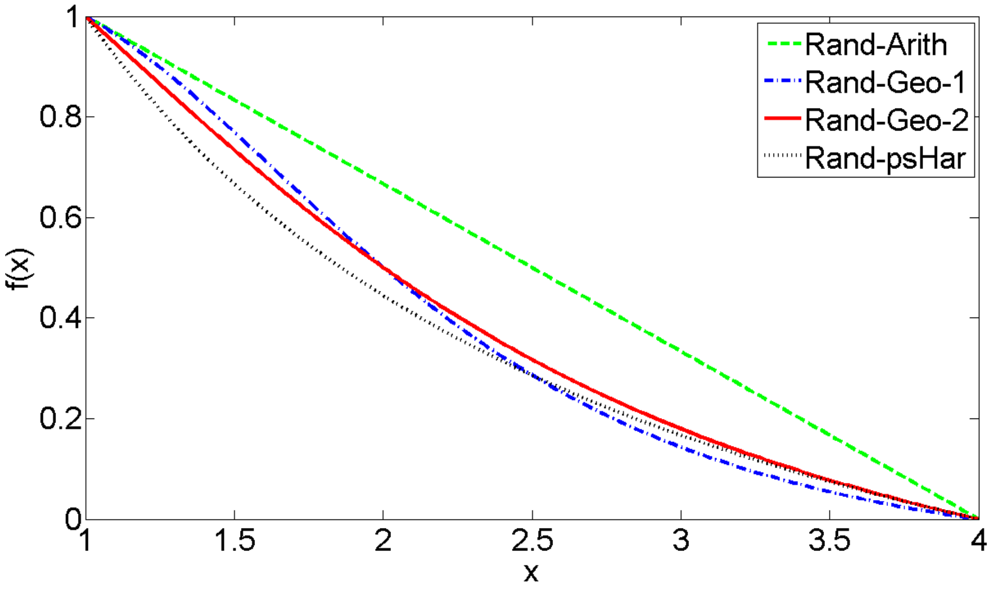


Figure 7.2: Randomized mappings with $A = 1$, $B = 4$.

All of the mappings satisfy the Boundary Conditions for $A > 0$, while the geometric mappings violate them for $A = 0$, since they map A to 0 instead of 1. All of the mappings satisfy (weak) Monotonicity (though the deterministic ones violate strict Monotonicity, as do the geometric ones for $A = 0$). All mappings satisfy Scale Invariance.

It is easy to see that the deterministic mappings violate Action Robustness, as they are clearly discontinuous at the threshold (this is true for any deterministic mapping). The randomized mappings satisfy Action Robustness, as their derivatives are bounded. The deterministic mappings all violate Boundary Robustness as well, since increasing A from A_1 to A_2 will cause $f(x)$ to change abruptly from 0 to 1 for some values of x near the threshold. It is natural to use the L^∞ norm to define distances between mappings, since a mapping could be exploited if it behaves poorly on just a single action. Formally,

$$d(f_{A_1,B_1}, f_{A_2,B_2}) = \max_{x \in S} |f_{A_1,B_1}(x) - f_{A_2,B_2}(x)|,$$

where $S = [A_1, B_1] \cap [A_2, B_2]$ is nonempty. Using this definition, Rand-Arith and Rand-psHar are Lipschitz continuous in both A and B (even for $A = 0$), while Rand-Geo-1 and Rand-Geo-2 are discontinuous in A for $A = 0$, and Lipschitz discontinuous in A for $A > 0$. We present proofs for Rand-psHar and Rand-Geo-2 (the proofs of the results for Rand-Geo-1 are analogous to the proofs for Rand-Geo-2).

Proposition 2. *Rand-psHar is Lipschitz continuous in A.*

Proof. Let $A_1, A_2 \in (0, B]$, $A_1 \neq A_2$ be arbitrary, and without loss of generality assume $A_1 < A_2$. Let

$$K = \frac{1 + B}{(B - A_1)(1 + A_2)}.$$

Then

$$\max_{x\in[A_2,B]}\left|\frac{(B-x)(1+A_1)}{(B-A_1)(1+x)}-\frac{(B-x)(1+A_2)}{(B-A_2)(1+x)}\right|$$
$$=\frac{(A_2-A_1)(1+B)}{(B-A_1)(B-A_2)}\max_{a\in[A_2,B]}\left|\frac{B-x}{1+x}\right|$$
$$=\frac{(A_2-A_1)(1+B)}{(B-A_1)(B-A_2)}\cdot\frac{B-A_2}{1+A_2}=K|A_2-A_1|$$

□

Proposition 3. *For any $B>0$, Rand-Geo-1 and Rand-Geo-2 are not continuous in A, where A has domain $[0,B)$.*

Proof. We present the proof for Rand-Geo-2. It is similar for Rand-Geo-1. Let $B>0$ be arbitrary, let $\epsilon=0.5$, and let $\delta>0$ be arbitrary. Let $A_1=0$ and $A_2=\frac{\delta}{2}$. Then $f_{A_1,B}(A_2)=0$ and $f_{A_2,B}(A_2)=1$. So we have

$$\max_{x\in[A_2,B]}|f_{A_2,B}(x)-f_{A_1,B}(x)|$$
$$\geq\ |f_{A_2,B}(A_2)-f_{A_1,B}(A_2)|=1>\epsilon.$$

But $|A_2-A_1|=\frac{\delta}{2}<\delta$. So Rand-Geo-2 is not continuous in A at $A=0$. □

Proposition 4. *For any $B>0$, Rand-Geo-1 and Rand-Geo-2 are not Lipschitz continuous in A, where A has domain $(0,B)$.*

Proof. We present the proof for Rand-Geo-2. It is similar for Rand-Geo-1. Let $B>0, K>0$ be arbitrary. For now, assume that $0<A<A'<B$. Then

$$\frac{\max_{x\in[A',B]}|f_{A,B}(x)-f_{A',B}(x)|}{|A'-A|}$$
$$\geq\frac{|f_{A,B}(A')-f_{A',B}(A')|}{|A'-A|}=\frac{1-f_{A,B}(A')}{|A'-A|}$$
$$=\frac{B(A'+A)}{(B-A)(A'^2+AB)}$$

This quantity is greater than K if and only if

$$A^2(BK)+A(B+A'^2K-KB^2)+(A'B-A'^2BK)>0.$$

Let $\mu(A)$ denote the LHS of the final inequality. Note that $\mu(A)\to(A'B-A'^2BK)$ as $A\to 0$. Since $\mu(A)$ is continuous, there exists some interval $I=(\underline{A},\overline{A})$ with $0<\underline{A}<\overline{A}<\min\{\frac{1}{4K},\frac{B}{2}\}$ such that $\mu(A)>0$ for all $A\in I$. Let A be any value in I, and let $A'=2A$. Then we have found A, A' satisfying $0<A<A'<B$ such that

$$\frac{\max_{x\in[A',B]}|f_{A,B}(x)-f_{A',B}(x)|}{|A'-A|}>K.$$

So Rand-Geo-2 is not Lipschitz continuous in A. □

To give some intuition for why Boundary Robustness is important, we examine the effect of increasing A gradually from 0 to 0.1, while holding $B = 1$ and $x = 0.25$ fixed. Table 7.1 shows the value of $f_{A,B}(x)$ for several values of A, for each of the randomized mappings. For the two mappings that satisfy Boundary Robustness—Rand-Arith and Rand-psHar—the values increase gradually as A is increased: Rand-Arith increases from 0.75 at $A = 0$ to 0.833 at $A = 0.1$, while Rand-psHar increases from 0.6 to 0.733. The two geometric mappings increase much more sharply, from 0 to 0.667 and 0.641 respectively. In practice, we may not know the optimal values to use in our abstraction ex ante, and may end up selecting them somewhat arbitrarily. If we end up making a choice that is not quite optimal (for example, 0.01 instead of 0.05), we would like it to not have too much of an effect. For non-robust mappings, the effect of making poor decisions in these situations could be much more severe than desired.

	A				
	0	0.001	0.01	0.05	0.1
Rand-Arith	0.75	0.751	0.758	0.789	0.833
Rand-Geo-1	0	0.012	0.111	0.429	0.667
Rand-Geo-2	0	0.015	0.131	0.439	0.641
Rand-psHar	0.6	0.601	0.612	0.663	0.733

Table 7.1: Effect of increasing A while holding $B = 1$ and $x = 0.25$ fixed.

7.8 Comparing exploitability

The *exploitability* of a strategy is the difference between the value of the game and worst-case performance against a nemesis. In particular, Nash equilibrium strategies are precisely those that have zero exploitability. Since our main goal is to approximate equilibrium strategies, minimizing exploitability is a natural metric for evaluation. The clairvoyance game, Kuhn poker, and Leduc hold ’em are small enough that exploitability can be computed exactly.

7.8.1 Clairvoyance game

In Table 7.2, we present the exploitability of the mappings described in Section 7.4 in the clairvoyance game. We varied the starting stack from $n = 1$ up to $n = 100$, experimenting on 7 games in total. (A wide variety of stack sizes relative to the blinds are encountered in poker in practice, so it is important to make sure a mapping performs well for many stack sizes.) For these experiments, we used the betting abstraction $\{$fold, check, pot, all-in$\}$ (fcpa). This abstraction is a common benchmark in no-limit poker [48, 53, 54, 103]: “previous expert knowledge [has] dictated that if only a single bet size [in addition to all-in] is used everywhere, it should be pot sized” [54].

For the abstract equilibrium, we used the equilibrium strategy described in Section 7.5.[6] The

[6]We also experimented using the Nash equilibrium at the other extreme (see Section 7.5), and the relative performances of the mappings were very similar. This indicates that our results are robust to the abstract equilibrium strategies selected by the solver.

entries in Table 7.2 give player 2's exploitability for each mapping. The results show that the exploitability of Rand-psHar stays constant at zero, while the exploitability of the other mappings steadily increases as the stack size increases. As we have predicted, the arithmetic mappings are more exploitable than the geometric ones, and the deterministic mappings are more exploitable than the corresponding randomized ones.

	Stack Size (n)						
	1	3	5	10	20	50	100
Det-Arith	0.01	0.24	0.49	1.12	2.38	6.12	12.37
Rand-Arith	0.00	0.02	0.09	0.36	0.96	2.82	5.94
Det-Geo	0.23	0.28	0.36	0.63	0.99	1.68	2.43
Rand-Geo-1	0.23	0.23	0.23	0.24	0.36	0.66	1.01
Rand-Geo-2	0.23	0.23	0.23	0.25	0.36	0.65	1.00
Det-psHar	0.15	0.19	0.33	0.47	0.59	0.67	0.71
Rand-psHar	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 7.2: Exploitability of mappings for the clairvoyance game, using betting abstraction {fold, check, pot, all-in}.

7.8.2 Kuhn poker

We conducted similar experiments on the more complex game of Kuhn poker; the results are given in Table 7.3. As in the clairvoyance game, Rand-psHar significantly outperformed the other mappings, with an exploitability near zero for all stack sizes. Interestingly, the relative performances of the other mappings differ significantly from the results in the clairvoyance game. Rand-Arith performed second-best while Det-psHar performed the worst.[7]

It turns out that for each stack size, player 1 has a unique equilibrium strategy that uses a bet size of 0.4 times the pot (recall that we only allow bets that are a multiple of 0.1 pot). So we thought it would be interesting to see how the results would change if we used the bet size of 0.4 pot in our abstraction instead of pot. Results for these experiments are given in Table 7.4. Surprisingly, all of the mappings became more exploitable (for larger stack sizes) when we used the "optimal" bet size, sometimes significantly so (for $n = 100$ Det-Arith had exploitability 0.301 using the first abstraction and 3.714 using the second abstraction)! This is very counterintuitive, as we would expect performance to improve as we include "better" actions in our abstraction. It also casts doubt on the typical approach for selecting an action abstraction for poker-playing programs; namely, emulating the bet sizes that human professional poker players use.

[7]We do not allow player 2 to fold when player 1 checks for these experiments, since he performs at least as well by checking. The results are even more favorable for Rand-psHar if we remove this restriction because player 2 is indifferent between checking and folding with a Jack, and the abstract equilibrium strategy our solver output happened to select the fold action. The geometric mappings are unaffected by this because they never map a bet down to a check, but the other mappings sometimes do and will correctly fold a Jack more often to a small bet. In particular, Rand-psHar obtained exploitability 0 for all stack sizes using fcpa.

We decided to investigate this paradox further, and computed the bet size that minimized exploitability for each of the mappings. The results are given in Table 7.5.[8] Interestingly, the unique full equilibrium bet size of 0.4 was very rarely the optimal bet size to use. The optimal bet size varied dramatically as different stack sizes and mappings were used. In some cases it was quite large; for example, for $n = 100$ it was 71.5 for Det-psHar and 29.8 for Det-Geo. The results indicate that the optimal action abstraction to use may vary considerably based on the action translation mapping used, and can include surprising actions while excluding actions that are played in the full equilibrium (even when these are the only actions played in any full equilibrium). This suggests that when using multiple actions in an abstraction, a mix of both "optimal" offensive actions (which are actually taken by the agent) and defensive actions (which are not taken themselves, but reduce exploitability due to an imperfect abstraction) may be more successful than focusing exclusively on the offensive ones. This is consistent with the approach that some teams in the competition have been using where they insert large defensive actions into the abstraction on the opponent's side.

	Stack Size (n)						
	1	3	5	10	20	50	100
Det-Arith	0.205	0.205	0.244	0.271	0.287	0.298	0.301
Rand-Arith	0.055	0.055	0.055	0.055	0.055	0.055	0.055
Det-Geo	0.121	0.121	0.121	0.217	0.297	0.366	0.399
Rand-Geo-1	0.121	0.121	0.121	0.121	0.121	0.121	0.121
Rand-Geo-2	0.121	0.121	0.121	0.121	0.121	0.121	0.121
Det-psHar	0.171	0.171	0.233	0.365	0.454	0.520	0.545
Rand-psHar	0.029	0.029	0.029	0.029	0.029	0.029	0.029

Table 7.3: Exploitability of mappings for no-limit Kuhn poker, using betting abstraction {fold, check, pot, all-in}.

	Stack Size (n)						
	1	3	5	10	20	50	100
Det-Arith	0.088	0.110	0.285	0.485	0.848	1.926	3.714
Rand-Arith	0.012	0.033	0.068	0.157	0.336	0.871	1.764
Det-Geo	0.086	0.114	0.294	0.425	0.548	0.714	0.873
Rand-Geo-1	0.071	0.085	0.095	0.116	0.145	0.203	0.269
Rand-Geo-2	0.071	0.083	0.094	0.114	0.144	0.203	0.269
Det-psHar	0.064	0.090	0.302	0.420	0.500	0.556	0.574
Rand-psHar	0.008	0.010	0.017	0.027	0.037	0.047	0.054

Table 7.4: Exploitability of mappings for no-limit Kuhn poker, using betting abstraction {fold, check, 0.4 pot, all-in}.

[8]For ties, we reported the smallest size.

	Stack Size (n)						
	1	3	5	10	20	50	100
Det-Arith	0.3	0.4	0.7	0.9	0.9	1.0	1.0
Rand-Arith	0.3	0.5	0.6	0.8	0.9	1.0	1.0
Det-Geo	0.3	0.2	1.0	2.6	2.5	14.6	29.8
Rand-Geo-1	0.2	0.1	0.3	0.4	1.0	1.0	1.0
Rand-Geo-2	0.2	0.1	0.3	0.3	1.0	1.0	1.0
Det-psHar	0.4	0.3	2.3	7.9	4.5	49.9	71.5
Rand-psHar	0.1	0.4	0.5	0.6	0.6	0.7	0.7

Table 7.5: Optimal bet sizes for player 2 of action translation mappings for no-limit Kuhn poker, using betting abstraction with fold, check, all-in, and one additional bet size.

7.8.3 Leduc hold ’em

We also compared exploitability on Leduc hold ’em—a much larger poker variant than the clairvoyance game and Kuhn poker. Unlike these smaller variants, Leduc hold ’em allows for multiple bets and raises, multiple rounds of betting, and shared community cards. Thus, it contains many of the same complexities as the variants of poker commonly played by humans—most notably Texas hold ’em—while remaining small enough that exploitability computations are tractable.

	P1 exploitability	P2 exploitability	Avg. exploitability
Det-Arith	0.427	0.904	0.666
Rand-Arith	0.431	0.853	0.642
Det-Geo	0.341	0.922	0.632
Rand-Geo-1	0.295	0.853	0.574
Rand-Geo-2	0.296	0.853	0.575
Det-psHar	0.359	0.826	0.593
Rand-psHar	0.323	0.603	0.463

Table 7.6: Exploitability of mappings for each player in no-limit Leduc hold ’em using the fcpa betting abstraction.

Exploitabilities for both players using the fcpa abstraction are given in Table 7.6. The results indicate that Rand-psHar produces the lowest average exploitability by a significant margin, while Det-Arith produces the highest exploitability. Interestingly, Rand-psHar did not produce the lowest exploitability for player 1; however, its exploitability was by far the smallest for player 2, making its average the lowest. Player 2’s exploitability was higher than player 1’s in general because player 1 acts first in both rounds, causing player 2 to perform more action translation to interpret bet sizes.

7.9 Experiments in Texas hold 'em

We next tested the mappings against the agents submitted to the no-limit Texas hold 'em division of the 2012 Annual Computer Poker Competition. We started with our submitted agent, Tartanian5 [33], and varied the action translation mapping while keeping everything else about it unchanged.[9] Then we had it play against each of the other entries.

	Action Translation Mapping						
	Det-Arith	Rand-Arith	Det-Geo	Rand-Geo-1	Rand-Geo-2	Det-psHar	Rand-psHar
azure.sky	3135 ± 106	3457 ± 90	2051 ± 96	2082 ± 97	2057 ± 97	2954 ± 96	3041 ± 109
dcubot	880 ± 52	752 ± 51	169 ± 47	156 ± 47	141 ± 46	754 ± 36	622 ± 47
hugh	137 ± 84	122 ± 86	-103 ± 50	-98 ± 52	-117 ± 52	-102 ± 30	42 ± 72
hyperborean	-189 ± 79	-272 ± 77	-216 ± 78	-203 ± 77	-161 ± 75	-161 ± 36	-276 ± 77
little.rock	-115 ± 100	-107 ± 95	-48 ± 92	-22 ± 91	-85 ± 89	165 ± 63	93 ± 87
lucky7.12	772 ± 104	510 ± 105	465 ± 82	471 ± 78	462 ± 78	536 ± 94	565 ± 74
neo.poker.lab	6 ± 97	-37 ± 106	11 ± 101	24 ± 98	31 ± 100	8 ± 31	-9 ± 103
sartre	94 ± 65	-3 ± 65	51 ± 64	86 ± 64	26 ± 64	56 ± 38	50 ± 65
spewy.louie	457 ± 118	421 ± 116	572 ± 102	530 ± 106	475 ± 103	614 ± 60	484 ± 109
uni.mb.poker	856 ± 84	900 ± 87	1588 ± 102	1567 ± 101	1657 ± 104	1148 ± 61	1103 ± 90
Avg.	609	571	454	459	449	597	568

Table 7.7: No-limit Texas hold 'em results in milli big blinds per hand. The entry is the profit of our agent Tartanian5 using the mapping given in the column against the opponent listed in the row.

The results are in Table 7.7 (with 95% confidence intervals included). Surprisingly, Det-Arith performed best using the metric of average overall performance, despite the fact that it was by far the most exploitable in simplified games. Det-psHar, Rand-Arith, and Rand-psHar followed closely behind. The three geometric mappings performed significantly worse than the other four mappings, (and similarly to each other).

One interesting observation is that the performance rankings of the mappings differed significantly from their exploitability rankings in simplified domains (with Det-Arith being the most extreme example). The results can be partially explained by the fact that none of the programs in the competition were attempting any exploitation of bet sizes or of action translation mappings (according to publicly-available descriptions of the agents available on the competition website). Against such unexploitative opponents, the benefits of a defensive, randomized strategy are much less important.[10] As agents become stronger in the future, we would expect action exploitation to become a much more important factor in competition performance, and the mappings with high exploitability would likely perform significantly worse. In fact, in the 2009 competition, an entrant in the bankroll category (Hyperborean) used a simple strategy (that did not even look at its own cards) to exploit opponents' betting boundaries and came in first place [103].

[9]Tartanian5 used Det-psHar in the actual competition.

[10]This is similar to a phenomenon previously observed in the poker competition, where an agent that played a fully deterministic strategy outperformed a version of the same agent that used randomization [38].

7.10 Summary and extensions

We have formally defined the action translation problem and analyzed all the prior action translation mappings which have been proposed for the domain of no-limit poker. We have developed a new mapping which achieves significantly lower exploitability than any of the prior approaches in the clairvoyance game, Kuhn poker, and Leduc hold 'em for a wide variety of stack sizes. In no-limit Texas hold 'em, our mapping significantly outperformed the mappings used by the strongest agents submitted to the most recent Annual Computer Poker Competitions (Det-Geo, Rand-Geo-1, and Rand-Geo-2). It did not outperform the two less sophisticated (and highly exploitable) mappings Det-Arith and Rand-Arith because the opponents were not exploitative (though the performance differences were small). We also introduced a set of natural domain-independent desiderata and showed that only our new randomized mapping (and Rand-Arith, which we showed to be highly exploitable) satisfy all of them.

In the course of this work, we observed several paradoxical and surprising results. In Kuhn poker, all of the action translation mappings had lower exploitability for large stack sizes when using an abstraction with a suboptimal action (a pot-sized bet) than when using an abstraction that contained the optimal action (a 0.4 times pot bet), even when all equilibrium strategies use the latter bet size. When we computed what the optimal action abstractions would have been for each mapping, they often included actions that differed significantly from the unique equilibrium actions. In addition, we observed that the naïve deterministic arithmetic mapping actually outperformed all the other mappings against agents submitted to the 2012 Annual Computer Poker Competition despite the fact that it had by far the highest exploitability in simplified domains (and violated many of the desiderata).

This work suggests many avenues for future research. One idea would be to consider more complex action translation mappings in addition to the ones proposed in this paper. For example, one could consider mappings that take into account game-specific information (as opposed to the game-independent ones considered here which only take as input the action size x and the adjacent abstract actions A and B). It also might make sense to use different mappings at different information sets (or even between different actions at the same information set). For example, we may want to use one mapping to interpret smaller bets (e.g., between 0 and a pot-sized bet), but a different one to interpret larger bets. In addition, our paradoxical results in Kuhn poker suggest that, when using multiple actions in an abstraction, a mix of both "optimal" offensive actions and defensive actions may be more successful than focusing exclusively on the offensive ones. Finally, we would like to use our framework (and the new mapping) in other domains. Action abstraction, and therefore also translation, are necessary for solving almost any game where action sizing is an issue, such as bid sizing in auctions, offer sizing in negotiations, and allocating different quantities of attack resources or defense resources in security games.

Chapter 8

Post-Processing

In Chapter 7, I described a new approach for interpreting actions of the opponent that have been removed from the abstraction by mapping them probabilistically to actions in the abstraction. In this chapter, I will describe further post-processing techniques that can be applied in addition to action translation, and also in games where no action abstraction was performed. These approaches, combined with the action translation approach presented in the previous chapter, comprise the final "reverse mapping" step of the leading paradigm depicted in Figure 4.1.

One of the key ideas that motivate our approaches is that the exact action probabilities of a mixed strategy equilibrium in an abstraction can exemplify overfitting to the particular abstraction used. (Our results confirm this.) Ideally, we would like to extrapolate general principles from the strategy rather than just use values that were finely tuned for a specific abstraction. This is akin to the classic example from machine learning, where we would prefer a degree-one polynomial that fits the training data quite well to a degree-hundred polynomial that may fit it slightly better.[1] (Subsequent to the appearance of the earlier versions of our paper, others have also shown that overfitting strategies to a particular abstraction is a very significant and real problem in large imperfect-information games [66].) An example depicting this overfitting phenomenon, which was presented in that prior work [66], appears in Figure 8.1. As the figure shows, exploitability within the abstraction decreases monotonically as the equilibrium-finding algorithm is run longer (as we would expect), while exploitability in the full game starts increasing at an intermediate point in time to do the strategies being overfit to the abstraction.

We present a family of modifications to the standard approach that work by constructing non-equilibrium strategies in the abstract game, which are then played in the full game. Our new procedures, called *purification* and *thresholding*, modify the action probabilities of an abstract equilibrium by placing a preference on the higher-probability actions [38]. The main intuition behind our algorithms is that we should ignore actions that are played with small probability in the abstract equilibrium, as they are likely due to abstraction coarseness, overfitting, or failure of the equilibrium-finding algorithm to fully converge.

Using a variety of experimental domains, we show that our new approach leads to significantly stronger play than the standard abstraction/equilibrium approach. For example, our pro-

[1]Note that an important difference between our setting and the classic machine learning setting is that we are applying our techniques after the equilibrium computation has been performed, while regularization in machine learning is done during the learning phase.

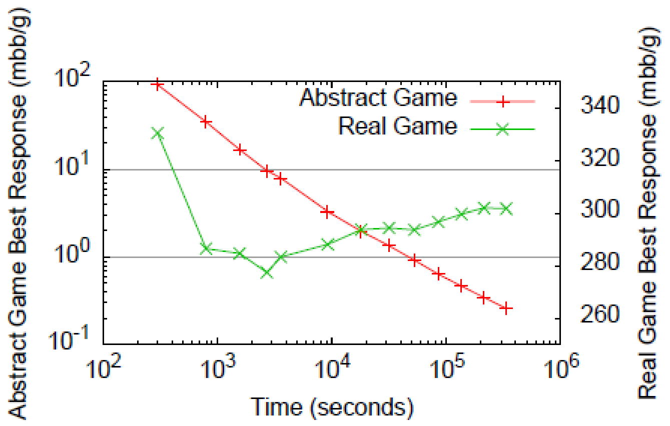


Figure 8.1: Abstraction equilibrium strategies can be overfit to the abstraction.

gram that uses purification won the two-player no-limit Texas hold 'em total bankroll division of the 2010 Annual Computer Poker Competition (ACPC), held at AAAI. Surprisingly, we also show that purification significantly improves performance (against the full equilibrium strategy) in random 4×4 matrix games using random 3×3 abstractions. We present additional results (both theoretical and empirical), including: worst-case theoretical results, empirical and theoretical results on specific support properties for which purification helps in matrix games, and experimental results in well-studied large imperfect-information games (Leduc hold 'em and Texas hold 'em).

Finally, we introduce a family of post-processing techniques that generalize purification and thresholding, leading to improved performance in two-player no-limit Texas hold 'em against the strongest prior agents. Our techniques combine 1) the observation that rounding action probabilities mitigates the above-mentioned issues, 2) the new observation that similar abstract actions should be bucketed before such rounding so that fine-grained action discretization (aka action abstraction) does not disadvantage those actions, and 3) the new observation that biasing toward actions that reduce variance is helpful in a strong agent and our experiments show that this increases expected value as well.

8.1 Purification and thresholding

Suppose we are playing a game Λ that is too large to solve directly. As described in Chapter 4, the standard approach would be to construct an abstract game Λ', compute an equilibrium σ' of Λ', then play the strategy profile σ induced by σ' in the full game Λ.

One possible problem with this approach is that the specific strategy profile σ' might be very finely tuned for the abstract game Λ', and it could perform arbitrarily poorly in the full game (see the results in Section 8.3). Ideally we would like to extrapolate the important features from σ' that will generalize to the full game and avoid playing a strategy that is overfit to the particular abstraction. This is one of the key motivations for our new approaches, *purification* and *thresholding*.

8.1.1 Purification

Let σ_i be a mixed strategy for player i in a strategic-form game, and let $S_i = \arg\max_j \sigma_i(j)$, where j ranges over all of player i's pure strategies. Then we define the *purification* pur(σ_i) of σ_i as follows:

$$\text{pur}(\sigma_i)(j) = \left\{ \begin{array}{ccc} 0 & : & j \notin S_i \\ \frac{1}{|S_i|} & : & j \in S_i \end{array} \right.$$

If σ_i plays a single pure strategy with highest probability, then the purification will play that strategy with probability 1. If there is a tie between several pure strategies of the maximum probability played under σ_i, then the purification will randomize equally between all maximal such strategies. Thus the purification will usually be a pure strategy, and will only be a mixed strategy in degenerate special cases when several pure strategies are played with identical probabilities.

If σ_i is a behavioral strategy in an extensive-form game, we define the purification similarly; at each information set I, pur(σ_i) will play the purification of σ_i at I.

8.1.2 Thresholding

The effects of purification can be quite extreme in some situations. For example, if σ_i plays action a with probability 0.51 and action b with probability 0.49, then b will never be played after purification. We also consider a more relaxed approach, called *thresholding*, that only eliminates actions below a prescribed ϵ to help alleviate this concern.

Thresholding works by setting all actions that have weight below ϵ to 0, then renormalizing the action probabilities. Pseudocode is given below in Algorithm 6. One intuitive interpretation of thresholding is that actions with probability below ϵ may just have been given positive probability due to noise from the abstraction (or because an equilibrium-finding algorithm had not yet taken those probabilities all the way to zero), and really should not be played in the full game. Additionally, low probability actions are often played primarily to protect a player from being exploited, and this may be an overstated concern against realistic opponents (as discussed further in Section 8.2.2).

Algorithm 6 Threshold(σ_i, ϵ)

```
for $j = 1$ to $|S_i|$ do
    if $\sigma_i(j) < \epsilon$ then
        $\sigma_i(j) \leftarrow 0$
    end if
end for
normalize($\sigma_i$)
return $\sigma_i$
```

8.2 Evaluation metrics

In recent years, several different metrics have been used to evaluate strategies in large games.

8.2.1 Empirical performance

The first metric, which is perhaps the most meaningful, is *empirical performance* against other realistic strategies. For example, in the ACPC, programs submitted from researchers and hobbyists from all over the world compete against one another. Empirical performance is the metric we will be using in Section 8.6 when we assess our performance in Texas hold 'em.

8.2.2 Worst-case exploitability

The worst-case *exploitability* of player i's strategy σ_i is the difference between the value of the game to player i and the payoff when the opponent plays his best response to σ_i (aka his *nemesis strategy*). Formally it is defined as follows:

$$\text{expl}(\sigma_i) = v_i - \min_{\sigma_{-i} \in \Sigma_{-i}} u_i(\sigma_i, \sigma_{-i}).$$

Worst-case exploitability is a common metric for assessing the quality of strategies in games [45, 66, 119].

Any equilibrium has zero exploitability, since it receives payoff v_i against its nemesis. So if our goal were to approximate an equilibrium of the full game, worst-case exploitability would be a good metric to use, since it approaches zero as the strategy approaches equilibrium.

Unfortunately, the worst-case exploitability metric has several drawbacks. First, it cannot be computed in very large games (though very recent advancements have made it possible to compute full best responses offline in two-player limit Texas hold 'em, which has about 10^{18} game states [66], and we will be leveraging that algorithm in our experiments).

Second, exploitability is a worst-case metric that assumes the opponent is able to *optimally* exploit us in the full game (i.e., he knows our full strategy and is able to efficiently compute a full best response in real time). In fact, it is quite common in very large games for agents to simply play static, fixed strategies the entire time, since the number of interactions is generally tiny compared to the size of the game, and it is usually quite difficult to learn to effectively exploit opponents online. For example, in recent computer poker competitions, almost all submitted programs simply play a fixed strategy. In the 2010 ACPC, many of the entrants attached summaries describing their algorithm. Of the 17 bots for which summaries were included, 15 played fixed strategies, while only 2 included some element of attempted exploitation. If the opponents are just playing a fixed strategy and not trying to exploit us, then worst-case exploitability is too pessimistic of an evaluation metric. Furthermore, if the opponents have computational limitations and use abstractions, then they will not be able to fully exploit us in the full game.

8.2.3 Performance against full equilibrium

In this paper, we will also evaluate strategies based on performance against equilibrium in the full game. The intuition behind this metric is that in many large two-player zero-sum games, the opponents are simply playing fixed strategies that attempt to approximate an equilibrium of the full game (using some abstraction). For example, most entrants in the ACPC do this. Against such static opponents, worst-case exploitability is not very significant, as the agents are not generally adapting to exploit us.

This metric, like worst-case exploitability, is not feasible to apply on very large games. However, we can still apply it to smaller games as a means of comparing different solution techniques. In particular, we will use this metric in Sections 8.4 and 8.5 when presenting our experimental results on random matrix games and Leduc hold 'em. This metric has similarly been used on solvable problem sizes in the past to compare abstraction algorithms [45].

8.3 Theory: selective superiority

So which approach is best: purification, thresholding, or the standard abstraction/equilibrium approach? It turns out that using the performance against full equilibrium metric, there exist games for which each technique can outperform each other. Thus, from a worst-case perspective, not much can be said in terms of comparing the approaches.

Proposition 5 shows that, for *any* equilibrium-finding algorithm, there exists a game and an abstraction such that purification does arbitrarily better than the standard approach.

Proposition 5. *For any equilibrium-finding algorithms A and A', and for any $k > 0$, there exists a game Λ and an abstraction Λ' of Λ, such that*

$$u_1(pur(\sigma'_1), \sigma_2) \geq u_1(\sigma'_1, \sigma_2) + k,$$

where σ' is the equilibrium of Λ' computed by algorithm A', and σ is the equilibrium of Λ computed by A.

Proof. Consider the game in Figure 8.2. Let Λ denote the full game, and let Λ' denote the

	L	M	R
U	2	0	$-3k-1$
D	0	1	-1

Figure 8.2: Two-player zero-sum game used in the proof of Proposition 5.

abstraction in which player 2 (the column player) is restricted to only playing L or M, but the row player's strategy space remains the same. Then Λ' has a unique equilibrium in which player 1 plays U with probability $\frac{1}{3}$, and player 2 plays L with probability $\frac{1}{3}$. Since this is the unique equilibrium, it must be the one output by algorithm A'. Note that player 1's purification pur(σ'_1) of σ' is the pure strategy D.

Note that in the full game Λ, the unique equilibrium is (D,R), which we denote by σ. As before, since this equilibrium is unique it must be the one output by algorithm A. Then we have

$$\begin{aligned} u_1(\sigma'_1, \sigma_2) = \frac{1}{3}(-3k-1) + \frac{2}{3}(-1) &= -k-1 \\ u_1(\text{pur}(\sigma'_1), \sigma_2) &= -1. \end{aligned}$$

So $u_1(\sigma'_1, \sigma_2) + k = -1$, and therefore

$$u_1(\text{pur}(\sigma'_1), \sigma_2) = u_1(\sigma'_1, \sigma_2) + k.$$

□

We can similarly show that purification can also do arbitrarily worse against the full equilibrium than standard unpurified abstraction, and that both procedures can do arbitrarily better or worse than thresholding (using any threshold cutoff). We can also show similar results using an arbitrary multiplicative (rather than additive) constant k.

8.4 Random matrix games

The first set of experiments we conduct to demonstrate the power of purification is on random matrix games. This is perhaps the most fundamental and easy to analyze class of games, and is a natural starting point when analyzing new algorithms.

8.4.1 Evaluation methodology

We study random 4×4 two-player zero-sum matrix games with payoffs drawn uniformly at random from [-1,1]. We repeatedly generated random games and analyzed them using the following procedure. First, we computed an equilibrium of the full 4×4 game Λ; denote this strategy profile by σ^F. Next, we constructed an abstraction Λ' of Λ by ignoring the final row and column of Λ. In essence, Λ' is a naïve, random abstraction of Λ, since there is nothing special about the final row or column. As in Λ, we computed an equilibrium σ^A of Λ'. We then compared $u_1(\sigma_1^A, \sigma_2^F)$ to $u_1(\text{pur}(\sigma_1^A), \sigma_2^F)$ to determine the effect of purification on performance of the abstract equilibrium strategy for player 1 against the full equilibrium strategy of player 2.

To solve the full and abstract games, we used two different procedures. For our first set of experiments comparing the overall performance of purified vs. unpurified abstract equilibrium strategies, we used a standard algorithm involving solving a single linear program [72]. For our results on supports, we used a custom support enumeration algorithm (similar to the approach of Porter et al. [93]). We note that it is possible that the specific algorithm used may have a significant effect on the results (i.e., certain algorithms may be more likely to select equilibria with specific properties when several equilibria exist).

$u_1(\text{pur}(\sigma_1^A), \sigma_2^F)$ (purified average payoff)	-0.050987 ± 0.00042
$u_1(\sigma_1^A, \sigma_2^F)$ (unpurified average payoff)	-0.054905 ± 0.00044
Number of games where purification led to improved performance	261569 (17.44%)
Number of games where purification led to worse performance	172164 (11.48%)
Number of games where purification led to no change in performance	1066267 (71.08%)

Table 8.1: Results for experiments on 1.5 million random 4×4 matrix games using random 3×3 abstractions. The $\pm$ given is the 95% confidence interval.

8.4.2 Experimental results and theory

In our experiments on 4×4 random games, we performed 1.5 million trials; the results are given in Table 8.1. The first row gives the average value of $u_1(\text{pur}(\sigma_1^A), \sigma_2^F)$ over all trials, while the second row gives the average value of $u_1(\sigma_1^A, \sigma_2^F)$. We conclude that purified abstraction outperforms the standard unpurified abstraction approach using 95% confidence intervals.

The next three rows of Table 8.1 report the number of trials for which purification led to an increased, decreased, or unchanged payoff of the abstract equilibrium strategy of player 1 against the full equilibrium strategy of player 2. While purification clearly improved performance more often than it hurt performance (17.44% vs. 11.48%), for the overwhelming majority of cases it led to no change in performance (71.08%). In particular, Proposition 6 gives two general sets of conditions under which purification leads to no change in performance.

Proposition 6. *Let Λ be a two-player zero-sum game, and let Λ' be an abstraction of Λ. Let σ^F and σ^A be equilibria of Λ and Λ' respectively. Then*

$$u_1(\sigma_1^A, \sigma_2^F) = u_1(\mathit{pur}(\sigma_1^A), \sigma_2^F)$$

if either of the following conditions is met:

1. *σ^A is a pure strategy profile*
2. *$support(\sigma_1^A) \subseteq support(\sigma_1^F)$*

Proof. If the first condition is met, then pur$(\sigma_1^A) = \sigma_1^A$ and we are done. Now suppose the second condition is true and let $s, t \in support(\sigma_1^A)$ be arbitrary. This implies that $s, t \in support(\sigma_1^F)$ as well, which means that $u_1(s, \sigma_2^F) = u_1(t, \sigma_2^F)$, since a player is indifferent between all pure strategies in his support at an equilibrium. Since s and t were arbitrary, player 1 is also indifferent between all strategies in $support(\sigma_1^A)$ when player 2 plays σ_2^F. Since purification will just select one strategy in $support(\sigma_1^A)$, we are done. $\square$

To understand our results further, we investigated whether they would vary for different supports of σ^F. In particular, we kept separate tallies of the performance of pur(σ_1^A) and σ_1^A for each support of σ^F. We observed that pur(σ_1^A) outperformed σ_1^A on many of the supports, while they performed equally on some (and σ_1^A did not outperform pur(σ_1^A) on any). These results are all statistically significant using 95% confidence intervals. A summary of the results from these experiments is given in Observation 1.

Observation 1. *In random 4×4 matrix games using 3×3 abstractions, pur(σ_1^A) performs better than σ_1^A using a 95% confidence interval for each support of σ^F except for supports satisfying one of the following conditions, in which case neither pur(σ_1^A) nor σ_1^A performs significantly better:*

1. *σ^F is the pure strategy profile in which each player plays his fourth pure strategy*
2. *σ^F is a mixed strategy profile in which player 1's support contains his fourth pure strategy, and player 2's support does not contain his fourth pure strategy.*

To interpret Observation 1, consider the following example. Suppose the support for player 1 includes his first three pure strategies, while the support for player 2 includes his final three pure strategies; denote this support profile by S^*. Now consider the set U of all games for which our equilibrium-finding algorithm outputs an equilibrium profile σ^F with support profile S^*. Since S^* does not satisfy either condition of Observation 1, this means that, in expectation over the set of all games in U,

$$u_1(\text{pur}(\sigma_1^A), \sigma_2^F) > u_1(\sigma_1^A, \sigma_2^F)$$

(i.e., purification improves the performance of the abstracted equilibrium strategy of player 1 against the full equilibrium strategy of player 2).

We find it interesting that there is such a clear pattern in the support structures for which pur(σ_1^A) outperforms σ_1^A. We obtained identical results using 3×3 games with 2×2 abstractions. We did not experiment on games larger than 4×4. While we presented experimental results that are statistically significant at the 95% confidence interval, rigorously proving that the results of Observation 1 hold even on 4×4 games with 3×3 abstractions remains a challenging open problem. Resolving this problem would shed some light on the underlying reasons behind the observed performance improvements of purification in random matrix games, which are quite surprising and unintuitive. In addition, we conjecture that a more general theoretical result will hold for general matrix games with any size, using any size random abstractions.

8.5 Leduc hold 'em

Leduc hold 'em is a simplified poker variant that has been used in previous work to evaluate imperfect-information game-playing techniques (e.g., [119]). Leduc hold 'em is large enough that abstraction has a non-trivial impact, but unlike larger games of interest (e.g., Texas hold 'em) it is small enough that equilibrium solutions in the full game can be quickly computed. That is, Leduc hold 'em allows for rapid and thorough evaluation of game-playing techniques against a variety of opponents, including an equilibrium opponent or a best responder.

Prior to play, a deck of six cards containing two Jacks, two Queens, and two Kings is shuffled and each player is dealt a single private card. After a round of betting, a public card is dealt face up for both players to see. If either player pairs this card, he wins at showdown; otherwise the player with the higher ranked card wins. For a complete description of the betting, we refer the reader to Waugh et al. [119].

8.5.1 Experimental evaluation and setup

To evaluate the effects of purification and thresholding in Leduc hold 'em, we compared the performance of a number of abstract equilibrium strategies altered to varying degrees by thresholding against a single equilibrium opponent averaged over both positions.[2] The performance of a strategy (denoted EV for expected value) was measured in millibets per hand (mb/h), where one thousand millibets is a small bet. As the equilibrium opponent is optimal, the best obtainable performance is 0 mb/h. Note that the expected value computations in this section are exact.

We used card abstractions mimicking those produced by state-of-the-art abstraction techniques to create our abstract equilibrium strategies. Specifically, we used the five Leduc hold 'em card abstractions from prior work [119], denoted *JQK*, *JQ.K*, *J.QK*, *J.Q.K* and *full*. The abstraction *full* denotes the null abstraction (i.e., the full unabstracted game). The names of the remaining abstractions consist of groups of cards separated by periods. All cards within a group are indistinguishable to the player prior to the flop. For example, when a player using the *JQ.K* abstraction is dealt a card, he will know only if that card is a king, or if it is not a king. These abstractions can only distinguish pairs on the flop. By pairing these five card abstractions,

[2]The experiments in this section were conducted by Kevin Waugh.

one abstraction per player, we learned twenty four abstract equilibrium strategies using linear programming techniques. For example, the strategy *J.Q.K-JQ.K* denotes the strategy where our player of interest uses the *J.Q.K* abstraction and he assumes his opponent uses the *JQ.K* abstraction.

8.5.2 Purification vs. no purification

In Table 8.2 we present the performance of the regular and purified abstract equilibrium strategies against the equilibrium opponent. We notice that purification improves the performance in all but 5 cases. In many cases this improvement is quite substantial. In the cases where it does not help, we notice that at least one of the players is using the *JQK* card abstraction, the worst abstraction in our selection. Prior to purification, the best abstract equilibrium strategy loses at 43.8 mb/h to the equilibrium opponent. After purification, 14 of the 24 strategies perform better than the best unpurified strategy, the best of which loses at only 1.86 mb/h. That is, only five of the strategies that were improved by purification failed to surpass the best unpurified strategy.

8.5.3 Purification vs. thresholding

In Figure 8.3 we present the results of three abstract equilibrium strategies thresholded to varying degrees against the equilibrium opponent. We notice that, the higher the threshold used the better the performance tends to be. Though this trend is not monotonic, all the strategies that were improved by purification obtained their maximum performance when completely purified. Most strategies tended to improve gradually as the threshold was increased, but this was not the case for all strategies. As seen in the figure, the *JQ.K-JQ.K* strategy spikes in performance between the thresholds of 0.1 and 0.15.

From these experiments, we conclude that purification tends to improve the performance of an abstract equilibrium strategy against an unadaptive equilibrium opponent in Leduc hold ’em. Though thresholding is itself helpful, it appears that the improvement generally increases with the threshold whenever thresholding improves a strategy, with the biggest improvement achieved using full purification.

8.6 Texas hold ’em

In the 2010 Annual Computer Poker Competition, the CMU team (Ganzfried, Gilpin, and Sandholm) submitted bots that used both purification and thresholding to the two-player no-limit Texas hold ’em division. We present the results in Section 8.6.1. Next, in Section 8.6.2, we observe how varying the amount of thresholding used effects the exploitabilities of two bots submitted to the two-player limit Texas hold ’em division.

8.6.1 A champion no-limit Texas hold ’em program

The two-player no-limit competition consists of two sub-competitions with different scoring rules. In the *instant-runoff* scoring rule, each pair of entrants plays against each other, and the

Strategy	**Base EV**	**Purified EV**	**Improvement**
JQ.K-J.QK	-119.46	-37.75	81.71
J.QK-full	-115.63	-41.83	73.80
J.QK-J.Q.K	-96.66	-27.35	69.31
JQ.K-J.Q.K	-96.48	-28.76	67.71
JQ.K-full	-99.30	-39.13	60.17
JQ.K-JQK	-80.14	-24.50	55.65
JQ.K-JQ.K	-59.97	-8.31	51.66
J.Q.K-J.QK	-60.28	-13.97	46.31
J.Q.K-J.Q.K	-46.23	-1.86	44.37
J.Q.K-JQ.K	-44.61	-3.85	40.76
full-JQK	-43.80	-10.95	32.85
J.QK-J.QK	-96.60	-67.42	29.18
J.QK-JQK	-95.69	-67.14	28.55
full-J.QK	-52.94	-24.55	28.39
J.QK-JQ.K	-77.86	-52.62	25.23
J.Q.K-full	-68.10	-46.43	21.66
full-JQ.K	-55.52	-36.38	19.14
full-J.Q.K	-51.14	-40.32	10.82
JQK-J.QK	-282.94	-279.44	3.50
JQK-full	-273.87	-279.99	-6.12
JQK-J.Q.K	-258.29	-279.99	-21.70
J.Q.K-JQK	-156.35	-188.00	-31.65
JQK-JQK	-386.89	-433.64	-46.75
JQK-JQ.K	-274.69	-322.41	-47.72

Table 8.2: Effects of purification on performance of abstract strategies against an equilibrium opponent in mb/h.

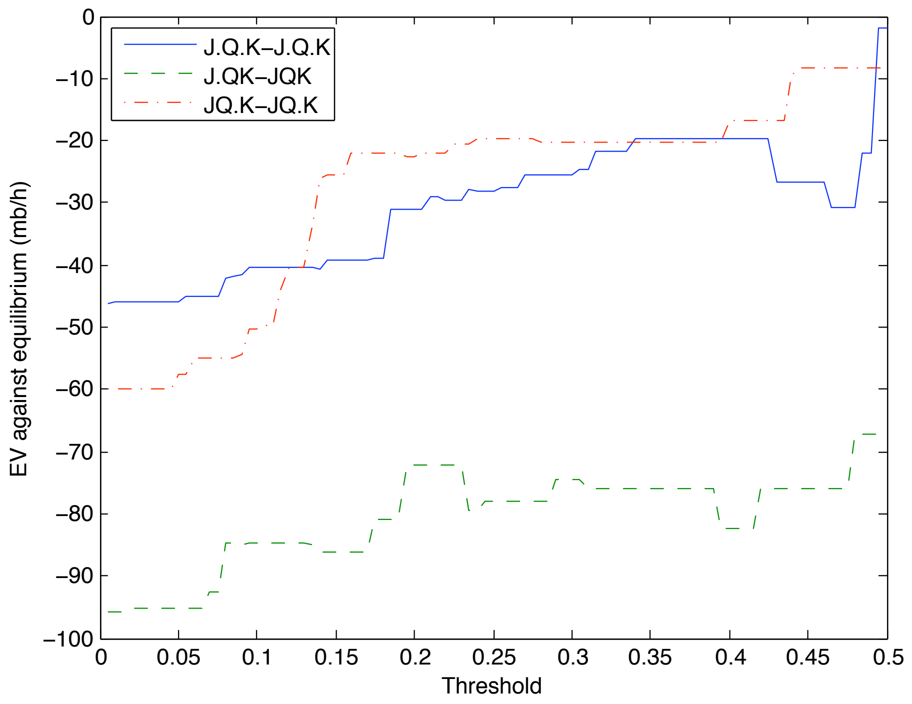


Figure 8.3: Effects of thresholding on performance of abstract strategies against an equilibrium opponent in mb/h.

bot with the worst head-to-head record is eliminated. This procedure is continued until only a single bot remains. The other scoring rule is known as *total bankroll*. In this competition, all entrants play against each other and are ranked in order of their total profits. While both scoring metrics serve important purposes, the total bankroll competition is considered by many to be more realistic, as in many real-world multiagent settings the goal of agents is to maximize total payoffs against a variety of opponents.

We submitted bots to both competitions: Tartanian4-IRO (IRO) to the instant-runoff competition and Tartanian4-TBR (TBR) to the total bankroll competition. Both bots use the same abstraction and equilibrium-finding algorithms. They differ only in their reverse-mapping algorithms: IRO uses thresholding with a threshold of 0.15 while TBR uses purification. IRO finished third in the instant-runoff competition, while TBR finished first in the total bankroll competition.

Although the bots were scored only with respect to the specific scoring rule and bots submitted to that scoring rule, all bots were actually played against each other, enabling us to compare the performances of TBR and IRO. Table 8.3 shows the performances of TBR and IRO against all of the bots submitted to either metric in the 2010 two-player no-limit Texas hold 'em competition.

One obvious observation is that TBR actually beat IRO when they played head-to-head (at a rate of 80 milli big blinds per hand). Furthermore, TBR performed better than IRO against every single opponent except for one (c4tw.iro). Even in the few matches that the bots lost, TBR lost at a lower rate than IRO. Thus, even though TBR uses less randomization and is perhaps more exploitable in the full game, the opponents submitted to the competition were either not trying

	c4tw.iro	c4tw.tbr	Hyperborean.iro	Hyperborean.tbr	PokerBotSLO	SartreNL	IRO	TBR
IRO	5334 $\pm$ 109	8431 $\pm$ 156	-248 $\pm$ 49	-364 $\pm$ 42	108 $\pm$ 46	-42 $\pm$ 38		-80 $\pm$ 23
TBR	4754 $\pm$ 107	8669 $\pm$ 168	-122 $\pm$ 38	-220 $\pm$ 39	159 $\pm$ 40	13 $\pm$ 33	80 $\pm$ 23	

Table 8.3: Results from the 2010 Annual Computer Poker Competition for two-player no limit Texas hold 'em. Values are in milli big blinds per hand (from the row player's perspective) with 95% confidence intervals shown. IRO and TBR both use the same abstraction and equilibrium-finding algorithms. The only difference is that IRO uses thresholding with a threshold of 0.15 while TBR uses purification.

or not able to find successful exploitations. Additionally, TBR would have still won the total bankroll competition even if IRO were also submitted.

These results show that purification can in fact yield a big gain over thresholding (with a lower threshold) even against a wide variety of realistic opponents in very large games.

8.6.2 Assessing worst-case exploitability in limit Texas hold 'em

Despite the performance gains we have seen from purification and thresholding, it is possible that these gains come at the expense of worst-case exploitability (see Section 8.2.2). Exploitabilities for several variants of a bot we submitted to the two-player limit division of the 2010 ACPC (GS6.iro) are given in Table 8.4; the exploitabilities were computed in the full unabstracted game using a recently developed approach [66].

Interestingly, using no rounding at all produced the most exploitable bot, while the least exploitable bot used an *intermediate* threshold of 0.15. There is a natural explanation for this seemingly surprising phenomenon. If there is too much thresholding, the resulting strategy does not have enough randomization, so it signals too much to the opponent about the agent's private information. On the other hand, if there is too little thresholding, the strategy is overfit to the particular abstraction.

Hyperborean.iro was submitted by the University of Alberta to the same competition; exploitabilities of its variants are shown as well. Hyperborean's exploitabilities increased monotonically with the threshold, with no rounding producing the least exploitable bot.

Threshold	**Exploitability of GS6**	**Exploitability of Hyperborean**
None	463.591	**235.209**
0.05	326.119	243.705
0.15	**318.465**	258.53
0.25	335.048	277.841
Purified	349.873	437.242

Table 8.4: Worst-case exploitabilities of several strategies in two-player limit Texas hold 'em. Results are in milli big blinds per hand. Bolded values indicate the lowest exploitability achieved for each strategy.

These results show that it can be hard to predict the relationship between the amount of rounding and the worst-case exploitability, and that it may depend heavily on the abstraction and/or equilibrium-finding algorithm used. While exploitabilities for Hyperborean are more in line with what one might intuitively expect, results from GS6 show that the minimum exploitability can actually be produced by an intermediate threshold value. One possible explanation of this difference is that thresholding and purification help more when coarser abstractions (i.e., smaller abstract games relative to the full game) are used, while in finer-grained abstractions, they may not help as much, and may even hurt performance.[3] The fact that the exploitability of Hyperborean is smaller than that of GS6 suggests that it was computed using a finer-grained abstraction.[4]

8.7 New generalized family of post-processing techniques

We observe that combining reverse mapping and thresholding leads to the issue that discretizing actions finely in some area of the action space disfavors those actions because the probability mass from the equilibrium finding gets diluted among them. To mitigate this problem, we propose to *bucket* abstract actions into similarity classes for the purposes of thresholding (but not after thresholding). For example, in no-limit poker any bet size is allowed up to the number of chips a player has left. In a given situation our betting abstraction may allow the agent to fold, call, bet 0.5 pot, 0.75 pot, pot, 1.5 pot, 2 pot, 5 pot, and all-in. If the action probabilities are (0.1, 0.25, 0.15, 0.15, 0.2, 0.15,0, 0, 0), then purification would select the call action, while the vast majority of the mass (0.65) is on betting actions. In this example, our approach—detailed below—would make a pot-sized bet (the highest-probability bet action).

Finally, we observe that biasing toward conservative actions that reduce variance (e.g., the fold action in poker) is helpful in a strong agent (variance increases the probability that the weaker opponent will win).[5] Our experiments show that preferring the conservative "fold" action in TH increases expected value as well. One reason may be that if an agent is uncertain about what should be done in a given situation (the equilibrium action probabilities are mixed), the agent will likely be uncertain also later down that path and it may be better to end the game here instead of continuing to play into a part of the game where the agent is weak.

Our new post-processing technique combines all the ideas listed above [18].[6] It first separates the available actions into three categories: fold, call, and bet. If the probability of folding exceeds a threshold parameter, we fold with probability 1. Otherwise, we follow purification between the three options of fold, call, and the "meta-action" of bet. If bet is selected, then we follow purification within the specific bet actions. (Purification is a special case of our family where the threshold parameter is 1 and there is no "bundling" of similar actions.)

Clearly, there are many variations of this technique—so it begets a family—depending on what threshold for definitely using the conservative action (fold) is used, how the actions are

[3]It is worth noting that purification and thresholding cannot help us against an equilibrium strategy if the abstraction is lossless; but even if it is lossless the algorithms may still help against actual (non-equilibrium) opponents.

[4]While such a monotonicity of exploitability vs. abstraction size is not guaranteed [119], empirically it has been demonstrated that in practice [68] exploitability typically decreases with increasing abstraction size.

[5]The idea folding with probability 1 in information sets where the approximate equilibrium gives a reasonably high probability for folding came from discussions with Eric Jackson.

[6]Noam Brown helped contribute to the development of this technique.

bucketed for thresholding, what thresholding value is used among the buckets, and what thresholding value is used within (each of possibly multiple) meta-actions.

We studied the effect of using our new post-processing techniques on the final strategies computed by our distributed equilibrium computation that was described in Chapter 6. We compared using no threshold, purification, a threshold of 0.15,[7] and using the new technique with a threshold of 0.2.[8] We tested against the two strongest agents from the 2013 competition (these were the same opponents that we evaluated our approaches against in Chapter 6). Results are shown in Table 8.5. The new post-processor outperformed the prior ones both on average performance and on worst observed performance.

	Hyperborean	Slumbot	Avg	Min
No Threshold	+30 $\pm$ 32	+10 $\pm$ 27	+20	+10
Purification	+55 $\pm$ 27	+19 $\pm$ 22	+37	+19
Thresholding-0.15	+35 $\pm$ 30	+19 $\pm$ 25	+27	+19
New-0.2	+39 $\pm$ 26	+103 $\pm$ 21	+71	+39

Table 8.5: Win rate (in mbb/h) of several post-processing techniques against the strongest two-player no-limit Texas hold 'em agents from the 2013 Computer Poker Competition.

8.8 Summary and extensions

We presented two new reverse-mapping algorithms for large games: purification and thresholding. One can view these approaches as ways of achieving robustness against one's own lossy abstraction. From a theoretical perspective, we proved that it is possible for each of these algorithms to help (or hurt) arbitrarily over the standard approach, and that each can perform arbitrarily better than the other. However, in practice both purification and thresholding seem to consistently help over a wide variety of domains.

Our experiments on random matrix games show that, perhaps surprisingly, purification helps even when random abstractions are used. Our experiments on Leduc hold 'em show that purification leads to improvements on most abstractions, especially as the abstractions become more sophisticated. Additionally, we saw that thresholding generally helps as well, and its performance improves overall as the threshold cutoff increases, with optimal performance usually achieved at full purification. We also saw that purification outperformed thresholding with a lower threshold cutoff in the Annual Computer Poker Competition against a wide variety of realistic opponents. In particular, our bot that won the 2010 two-player no-limit Texas hold 'em bankroll competition used purification. Finally, we saw that these performance gains do not necessarily come at the expense of worst-case exploitability, and that intermediate threshold values can actually produce the lowest exploitability. There is a natural explanation for this seemingly surprising phenomenon. If there is too much thresholding, the resulting strategy does not have enough ran-

[7]This value was a prior benchmark [38]. Our exploratory data analysis concurred that it is a good choice.

[8]This was a good choice based on exploratory analysis, and it performed clearly better than 0.1 against both opponents.

domization, so it signals too much to the opponent about the agent's private information. On the other hand, if there is too little thresholding, the strategy is overfit to the particular abstraction.

Our results open up many interesting avenues for future work. In Section 8.4, we presented several concrete theoretical open problems related to understanding the performance of purification in random matrix games. In particular, larger games (with different degrees of abstraction) should be studied, and perhaps general theorems can be proven to augment our (statistically significant) empirical findings.

Future work should also investigate possible deeper connections between purification, abstraction, and overfitting from a learning-theoretic perspective. Is there a formal sense in which purification and thresholding help diminish the effects of overfitting strategies to a particular abstraction? Is such overfitting more prone to occur with coarser abstractions, or with some abstraction algorithms more than others? Perhaps the results also depend crucially on the equilibrium-finding algorithm used (especially for games with many equilibria). A better understanding of these phenomena could have significant practical and theoretical implications.

In addition, we have just considered some of the families of modifications to the leading abstraction/equilibrium paradigm. Many other approaches are possible; for example, rounding probabilities to intermediate values (rather than to 0), or randomizing equally between the k highest-probability actions. We would like to experiment further with the different approaches within the family we have developed, as well as study new families of approaches, in order to understand better when certain approaches would be preferred over others.

Chapter 9

Computing Equilibria in Multiplayer Stochastic Imperfect-Information Games

Thus far, I have described approaches which were applied to solve two-player zero-sum games. In this chapter, I describe algorithms that we have applied to a three-player game. As discussed in Chapter 2, games with more than two agents are significantly more challenging to reason about and compute strong strategies for than two-player zero-sum games: computing a Nash equilibrium is PPAD-hard, and there is little theoretical justification for why Nash equilibrium strategies would perform well against unknown opponents (or even against "rational" opponents, for many possible definitions of rationality). That said, the study of algorithms for approximating Nash equilibria in these games is still worthwhile for several reasons. First, it could be possible to develop algorithms that work well in practice despite a lack of theoretical performance or runtime guarantees. Second, it could be possible to develop algorithms with theoretical performance and/or runtime guarantees in certain interesting game classes, despite a lack of a general worst-case guarantee. And third, it is my personal belief that Nash equilibrium strategies would still perform extremely well in many interesting games with more than two players, despite the lack of a general worst-case guarantee. For instance, many of the strongest high-stakes human players for 6-player variants of poker try to approximate Nash equilibrium strategies for many situations. Part of my longer-term research vision beyond this thesis is to develop a better understanding for why Nash equilibrium is a compelling solution concept even in these game classes.

In this chapter, I will present new algorithms for approximating equilibrium strategies in a class of multiplayer games—undiscounted multiplayer stochastic games, where the stage games are imperfect-information games. This game class includes a format of multiplayer poker tournaments that are played competitively for high stakes on several online poker sites. While the algorithms are not guaranteed to converge to equilibrium, we have observed empirically that they converge to ϵ-equilibrium strategies for extremely small ϵ in a realistic three-player poker tournament endgame. I also present proofs that several of the algorithms cannot converge to anything but a Nash equilibrium (i.e., if they converge, then the resulting strategies are guaranteed to be an equilibrium). I present detailed analysis of the computed approximate equilibrium strategies and draw several conclusions about poker strategy, some of which challenge popular heuristics from the poker community.

The algorithms presented are the first algorithms for provably computing an ϵ-equilibrium of a large stochastic game for small ϵ. An efficient algorithm that minimizes external regret in both the perfect and imperfect information case is also described.

A *stochastic game* is a collection of games (often these are strategic-form games); the agents repeatedly play a game from this collection, and then transition probabilistically to a new game depending on the previous game played and the actions taken by all agents in that game. We will consider stochastic games where the individual *stage games* are themselves extensive-form imperfect-information games. Unlike extensive-form games, stochastic games have a potentially infinite duration. In *discounted stochastic games*, the expected payoff of a player is the weighted sum of payoffs from each iteration, where weights are multiplied by some constant factor λ at each time step; in *undiscounted stochastic games*, the expected payoff is the limit of the average iteration payoff. Prior algorithms are guaranteed to converge to an equilibrium in certain classes of stochastic games [16, 58, 70, 79, 116]. However, no known algorithms are guaranteed to converge to an equilibrium in three-player stochastic games (even in the zero-sum case). In fact, it is unknown whether a Nash equilibrium is even guaranteed to exist in undiscounted stochastic games with more than two players.

9.1 Stochastic games

A *stochastic game* is a collection of games (often these are strategic-form games); the agents repeatedly play a game from this collection, and then transition probabilistically to a new game depending on the previous game played and the actions taken by all agents in that game. (Stochastic games generalize Markov decision processes (MDPs) to the multiagent setting.) We will consider stochastic games where the individual *stage games* are themselves extensive-form imperfect-information games. Unlike extensive-form games, stochastic games have a potentially infinite duration. In *discounted stochastic games*, the expected payoff of a player is the weighted sum of payoffs from each iteration, where weights are multiplied by some constant factor λ at each time step; in *undiscounted stochastic games*, the expected payoff is the limit of the average iteration payoff.

As in other classes of games, the standard solution concept in stochastic games is the Nash equilibrium. Prior algorithms are guaranteed to converge to an equilibrium in certain classes of games; however, no known algorithms are guaranteed to converge to an equilibrium in three-player stochastic games (even in the zero-sum case). In fact, it is unknown whether a Nash equilibrium is even guaranteed to exist in undiscounted stochastic games with more than two players.

Friend-or-Foe Q-learning [79] and *Nash-Q* [58] is proven to converge if the overall stochastic game has a global optimum (a strategy profile that maximizes the payoff for each agent) or a saddle point (Nash equilibrium such that if any agent deviates, all others become better off)—and furthermore, the algorithm needs to know which of these two settings it is operating in. Other algorithms are guaranteed to converge to an optimal Nash equilibrium in team stochastic games—games in which all agents receive the same payoff [16, 116]. Interesting algorithmic results have also been proven for planning in general-sum stochastic games [70]. That paper does not specify a Nash selection function; we provide such a function that is effective in practice. Their algorithm *Finite-VI* resembles our algorithm *VI-FP* from [29], which we will review, but they do not have algorithms resembling those that we will introduce in this paper. Furthermore, their algorithms find equilibria only in finite-horizon games; ours also do in infinite games. In summary, the settings of prior work are quite restrictive, and all of those algorithms have only been tested on tiny problems, if at all.

In contrast, I will present algorithms for computing approximate jam/fold equilibrium strategies in a three-player no-limit Texas hold 'em tournament [29, 30]. The tournament does not fall into the classes of games for which prior algorithms are guaranteed to converge, and furthermore the game is significantly larger than the games previously solved. Additionally, the stage games have imperfect information: players are dealt private cards at each game state and must choose their action without knowing the cards dealt to the other players. The tournament uses the actual parameters of tournaments played on the most popular poker site, *Pokerstars*. It can be formulated as a stochastic game with 946 states, each with $2^{6\times 169}$ possible actions.

Formally, a stochastic game G is a tuple (N, S, A, p, r) where $N = \{1, \ldots, n\}$ denotes a finite set of players, and S denotes a finite set of states. A is a tuple $(A_1, \ldots, A_n)$ where for each $i \in N$, $s \in S$, $A_i(s)$ is a finite set corresponding to player i's available actions at state s. For $s \in S$ let $A(s)$ denote the vector $(A_1(s), \ldots, A_n(s))$. For each $s, t \in S$ and $a \in A(s)$, $p_{s,t}(a)$ denotes the probability that we transition from state s to state t when all players play their component of the action vector a. Finally, $r : S \to \mathbb{R}^n$ denotes the payoff function, where for each $s \in S$, $r(s)$ is a vector whose i'th component is the payoff to player i when state s is reached (in some formulations the reward function has domain $S \times A$).

At each state s, player i can choose any probability distribution σ over actions in $A_i(s)$. Let $\sigma(a_i)$ denote the probability he selects action a_i. Let $a = (a_1, \ldots, a_n)$, and define $\sigma(a) = \prod_i \sigma(a_i)$. Then we extend our definition of the transition function so that

$$p_{s,t}(\sigma) = \sum_{a \in A(s)} \left[\sigma(a) p_{s,t}(a)\right].$$

A *policy* π for player i is a choice of probability distributions over actions in $A_i(s)$ at each state s.

If we let s_t denote the current state at time t then the goal of agent i is to maximize

$$\sum_{t=0}^{\infty} \gamma^t r_i(s_t),$$

where $0 < \gamma \leq 1$ is the discount rate. If $\gamma = 1$ then we say that the game is *undiscounted*, and otherwise we say that it is *discounted*.

We say that a state s is *terminal* if $p_{s,t}(a) = 0$ for all $t \neq s, a \in A(s)$ and $p_{s,s}(a) = 1$ for all $a \in A(s)$. If we ever arrive at a terminal state s, then we will remain at s for the remainder of the game. A state is *nonterminal* if it is not a terminal state.

A stochastic game with one agent is called a *Markov decision process (MDP)*.

9.2 Poker tournaments and jam/fold strategies

Tournaments are an extremely popular form of poker; they work as follows. Some number of players (say 10) all pay an entry fee (say \$10) and are given a number of chips (say 1500), which have no monetary value *per se* except that a player is eliminated from the tournament when he runs out of chips. The first six players eliminated from the tournament lose their entry fee, while the three top finishers receive 50%, 30%, and 20% of the sum of the entry fees ($100) respectively. Usually the blinds increase every five or ten minutes in online tournaments, starting at SB = 10, BB = 20 and approximately doubling at every level. Since chips have no explicit monetary value, tournaments are actually stochastic games, where each *state* corresponds to a vector of stack sizes.

When blinds become sufficiently large relative to stack sizes, rationality dictates more aggressive play. For example, suppose a player has 1000 chips at the 100/200 level (SB = 100, BB = 200) and is considering making a preflop raise smaller than going all-in. If he makes a minimum raise to 400 and someone re-raises for all his chips, then the original raiser will only need to put in 600 more chips into a pot of 1700. If all chips had the same value, then he would only need to expect to win with probability $\frac{600}{1700+600} = 0.261$ to make calling correct. Since even the worst starting hand has close to this probability of beating the best starting hand, it will almost always be correct to call. If he had gone all-in preflop instead of raising only to 400, the result would probably have been the same in this case; furthermore, he would have been more likely to win the blinds for free, since opponents are more likely to call a raise of 200 than a raise of 800.

In line with this example, common strategy when blinds become sufficiently high relative to stack sizes is to either go all-in or fold preflop (this is known as a *jam/fold strategy*). Following this reasoning, Miltersen and Sørensen [82] compute jam/fold strategies for tournaments with two players and the fixed parameters SB = 300, BB = 600, and 8000 total chips (at the time, these defined the normal parameters for the two-player endgame in tournaments on PartyPoker.com). In fact, if we let G_{i,s_i} denote the restriction of the original tournament in which player i starts with s_i chips and is limited to playing a jam/fold strategy (while the other player is not), they show that for any value of s_1, $V_1(G_{1,s_1}) + V_2(G_{2,8000-s_1}) \geq 0.986$ (where the winner gets payoff 1, the loser gets payoff 0, and V_i denotes the value of the game to player i). Thus, neither player can guarantee more than 0.014 by deviating to a non-jam/fold strategy; this provides a justification for restricting attention to jam/fold strategies. Additionally, they note that the probability each player wins the tournament is very close to the proportion of chips he has (i.e., player i will win with probability very close to $\frac{s_i}{s_1+s_2}$).

9.3 *VI-FP* algorithm

When blinds become sufficiently large relative to stack sizes in tournaments, rationality dictates more aggressive play. In particular, prior work suggests that it is near optimal for players to either go all-in or fold preflop [29, 82] (such strategies are called *jam/fold strategies*). In this section I present a new algorithm for computing an approximate jam/fold equilibrium in a three-player tournament [29]. The algorithm consists of iteration at two nested levels. The "inner loop" uses an extension of smoothed fictitious play to determine ϵ-equilibrium strategies for playing a hand at a given state (stack vector) given the values of possible future states. This is done for all states. The iteration of the "outer loop" adjusts the values of the different states in light of the new payoffs obtained from the inner loop. Then the inner loop is executed again until convergence, then the outer loop, and so on. The algorithm terminates when no state's payoff changes by more than δ between outer loop iterations. As the outer loop resembles Value Iteration and the inner loop uses smoothed Fictitious Play, we refer to this algorithm as *VI-FP*.

9.3.1 Inner loop

In standard fictitious play, each player plays a best response to the average strategies of his opponents thus far. Similarly, in smoothed fictitious play each player i applies the following update rule at each time step t to obtain his current strategy s_i^t:

$$s_i^t = \left(1 - \frac{1}{t}\right) s_i^{t-1} + \frac{1}{t} s_i'^t, \tag{9.1}$$

where $s_i'^t$ is a best response of player i to the profile s_{-i}^{t-1} of the other players played at time $t-1$ (strategies can be initialized arbitrarily at $t = 0$). This algorithm was originally developed as a simple learning model for repeated games, and was proven to converge to a Nash equilibrium in two-player zero-sum games [27]. However, it is not guaranteed to converge in two-player general-sum games or games with more than two players.

In poker tournaments the strategy spaces are exponential in the number of possible private signals. In particular, in Texas hold 'em there are 169 strategically distinct preflop hands (see, e.g., [42]) and two jam/fold moves each player can make at each information set. Therefore, the strategy spaces (for any given stack vector) are of size 2^{169}, $2^{2\times169}$, and $2^{3\times169}$ for the button, small blind, and big blind, respectively (the big blind does not need to act when the other two players fold). To deal efficiently with this exponential blowup, the algorithm works with the extensive form of the game instead of enumerating the strategies. In the extensive form, each player's best response can be computed by traversing his information sets one at a time. The button has 169 information sets, the small blind has 2×169, and the big blind has 3×169. Therefore, the best response (for each player in turn) can be computed efficiently.

While fictitious play is not guaranteed to converge, we fortunately have the following result:
Theorem 1. *(Fudenberg and Levine [27]) Under fictitious play, if the empirical distributions over each player's choices converge, the strategy profile corresponding to the product of these distributions is a Nash equilibrium.*

9.3.2 Outer loop

As stated earlier, poker tournaments are stochastic games in which each state corresponds to a vector of stack sizes. In particular, we analyze the tournament in which there are 13500 total chips, and blinds are SB = 300, BB = 600. These correspond to common endgame parameters of sit-and-go tournaments at *Pokerstars*, the most popular online poker site. Each state in our game G is defined by a triple (x_1, x_2, x_3), where x_1 denotes the stack of the button, x_2 denotes the stack of the small blind, x_3 denotes the stack of the big blind, $\sum_i x_i = 13500$, and all x_i are multiples of 300. When one of the x_i becomes zero, that player is eliminated, and we can already solve the remaining game with the prior techniques because there are at most two players left. So we can ignore states in which stacks are zero from our model and just substitute the corresponding payoffs to players when we arrive in such states. G has 946 non-terminal states.

The algorithm for solving the game works as follows. First we initialize the assignment V^0 of payoffs to each player at each game state using a heuristic from the poker community known as the Independent Chip Model (ICM), which asserts that a player's probability of winning is equal to his fraction of all the chips; his probability of coming in second is asserted to be the fraction of remaining chips conditional on each other player coming in first, etc. [29]. ICM has been shown to be quite accurate when two players remain in the tournament [82]; when three players remain it has been shown to be relatively accurate overall, although in some states it is significantly inaccurate [29].

Next, suppose we start at some game state x^0. If we assume that the payoffs of V^0 at all states are the actual values of those game states to the players, then the whole stochastic game just becomes a standard game in which every transition from x^0 leads to a terminal payoff. So we can run the fictitious play algorithm described in the previous section at state x^0 until it (hopefully) converges. Supposing it does converge to an approximate equilibrium, each player will now have some new expected payoff at state x^0 if that equilibrium is played. (These payoffs might differ from V^0.) If we do this for all 946 game states, we come up with a new assignment V^1 of payoffs to each player at each game state. Now suppose we repeat this process and that hypothetically the payoffs remain the same between iterations k and $k+1$ (all the payoffs in V^k and V^{k+1} are equal). This would suggest that the strategies computed by fictitious play at iteration k (assuming they converge) are close to an equilibrium of the full game G.

Now, stated explicitly with tolerances, the overall algorithm works as follows. First, fix $\gamma, \delta > 0$. Initialize the V^0 to ICM, and run fictitious play using V^0 as payoffs until an γ-equilibrium is reached (a strategy profile in which no player can gain more than γ in expectation by deviating). Then use the payoffs V^1 which result from the computed strategy profile and repeat. The algorithm halts when it reaches an iteration k such that each payoff of V^k differs from the corresponding payoff of V^{k-1} by less than δ.

Algorithm 7 *VI-FP*(γ, δ)

```
$V^0$ = ICM
diff = $\infty$
$i = 0$
while diff $> \delta$ do
    $i = i + 1$
    regret $\leftarrow \infty$
    $S \leftarrow$ initializeStrategies()
    while regret $> \gamma$ do
        $S$ = fictPlay()
        regret $\leftarrow$ maxRegret($S$)
    end while
    $V^i \leftarrow$ getNewValues($V^{i-1}$,$S$)
    diff $\leftarrow$ maxDev($V^i, V^{i-1}$)
end while
return $S$
```

9.4 Our *ex post* check

In the experimental results with *VI-FP*, both the inner and outer loops converged. However, we observe that this does not guarantee that the final strategies constitute an approximate equilibrium. For example, suppose we initialized all the payoffs at nonterminal states to be \$100 (higher than the first-place payoff) and initialized payoffs at the terminal states using ICM. Then the optimal strategy for each player would be to fold every hand (except for the shortest stack if he is all-in), and the algorithm would converge to this profile (in a single outer-loop iteration). However, this profile is clearly not an equilibrium of the game, as the players will not receive any payoff (while they would receive at least \$20 if the game ended after some finite time). Thus there exist initializations for which *VI-FP* converges to a non-equilibrium profile. I suspect also that VI-FP can fail to converge even if some nonterminal states are initialized pessimistically and/or there is positive probability that the game will end (unlike in the example above where the optimal strategy is for every player to fold every hand, under which the game will not end); this merits further study.

Despite this fact, we suspected that the computed strategies formed an approximate equilibrium (as the strategies seemed very reasonable and we used an intelligent initialization). To verify this, we developed an *ex post* checking procedure to make sure. It computes the best response of each player to the computed strategy profile and determines the improvement in payoff (while the overall algorithm is somewhat obvious, it is difficult in our setting for reasons described below). If the improvement is less than ϵ for each player, then the original strategies indeed constitute an ϵ-equilibrium.

Stepping back to the big picture, *VI-FP* is an incomplete algorithm; but if it converges, our *ex post* check can be used to check the quality of the solution. Furthermore, the check can be applied at every iteration of the outer loop, even if *VI-FP* does not converge. As we will see, we have developed several new algorithms for solving stochastic games that cannot converge to a non-equilibrium strategy profile (as *VI-FP* can). Thus, if the new algorithms converge we will not need to waste extra resources performing the *ex post* check after every iteration, as we are guaranteed to be at an equilibrium.

We will now describe the *ex post* check in detail. The first step is to construct the MDP induced by the strategy profile s^* (computed by *VI-FP*). Denote this MDP by M. Each nonterminal state s_i of M is determined by a nonterminal state v_i of the stochastic game G (stack vector) and a position α_i (button, small blind, or big blind). So, M contains $946 \times 3 = 2838$ nonterminal states. In addition, M contains a terminal state for each terminal of G corresponding to each stack vector with at most two nonzero stacks that is reachable from a nonterminal state (say that a state s_2 is *reachable* from state s_1 if there exists an $a \in A(s_1)$ such that $p_{s_1,s_2}(a) > 0$).

The set of actions available at each state is essentially the set of jam/fold strategies available at that stack vector in G. If α_i is the button then there are 2^{169} possible actions at state s_i; if α_i is the small blind there are $2^{2\times169}$ actions; and if α_i is the big blind there are $2^{3\times169}$ actions. The transitions of M are determined by the probability distribution over states when player α_i chooses action a_i and the other players follow the strategy profile s^*_{-i}. It is important to note that in a transition to a nonterminal state the position changes; that is, if we are at state s_i with position α_i equal to the button and transition to state s_j, the new position α_j is the big blind (because blinds move clockwise). Finally, the rewards at each nonterminal state are 0, and at each terminal state they are the ICM payoff of the corresponding stack vector (since previous two-player results showed that ICM payoffs very closely approximate equilibrium payoffs in this case).

An optimal policy for M corresponds to a best response of each player in G to the profile s^*. However, we cannot apply the standard value or policy iteration algorithms for solving the MDP because we are in the undiscounted setting (since the players only care about their final payoffs). Instead we turn to variants of these algorithms that work when the objective is expected total reward and the following conditions hold: 1) for all states s and policies π, the value of state s under π is finite and 2) for each s there exists at least one available action a that gives non-negative reward. In the results below, we refer to these conditions as "our setting."

For value iteration, the change from the classic discounted setting is the initialization:

Theorem 2. *(Puterman [95]) In our setting, if v^0 is initialized pessimistically (i.e., $\forall s, v^0(s) \leq v^*(s)$), value iteration converges (pointwise and monotonically) to v^*.*

9.4.1 Policy iteration

We refer the reader to [95] for a thorough description of the standard policy iteration algorithm, and present the modified algorithm needed to obtain convergence in our setting here (while noting deviations from the classic setting). Policy iteration converges to an optimal policy in our setting if 1) our initial policy obtains non-negative expected total reward (which is obviously the case here), 2) we choose the minimal non-negative solution of the system of equations in the evaluation equation in the evaluation step (if there is a choice), and 3) if the action chosen for some state in the previous iteration is still among the optimal actions for that state, then select it again.

Since step 2 is critical for our new algorithms for solving multi-player stochastic games in the next section, we will explain it in more detail here. For each state i in the MDP M, $v(i)$ is a variable corresponding to its value. Thus, the equation in step 2 is linear in the unknowns $v(i)$, $i = 1, \ldots, |S|$, where $|S|$ is the number of states in M. Thus we have a linear system of $|S|$ equations with $|S|$ unknowns. If the equations are linearly independent, then there will be a unique solution which can be computed by matrix inversion. If there are no solutions, then we are in a degenerate case and the MDP has no optimal policy (it can be proven that this can never be the case in our setting). If there are multiple solutions, then we select the minimal non-negative solution (see [95] for a further explanation). In all of our experiments the coefficient matrix had full rank and hence the system had a unique solution.

In the rest of this paper we will refer to step 2 of Algorithm 8 as EvaluatePolicy().

Theorem 3. *(Puterman [95]) Let S be the set of states in M. Suppose S and $A(s)$ are finite. Then, for some finite N, $v^N = v^*$ and $\pi^N = \pi^*$.*

9.4.2 *Ex post* check algorithm

Since we already have a policy s^* that we expect to be near-optimal (and the corresponding values from the final iteration of the outer loop) from *VI-FP*, we would like to use these values to obtain a warm start for v^0 in the *ex post* check. However, this would not work with the value

Algorithm 8 Policy iteration for our setting

```
1. Set $n = 0$ and initialize the policy $\pi^0$ so it has non-negative expected reward.
2. Let $v^n$ be the solution to the system of equations
       $$v(i) = r(i) + \sum_j p_{ij}^{\pi^n} v(j)$$
   where $p_{ij}^{\pi^n}$ is the probability of moving from state $i$ to state $j$ under policy $\pi^n$. If there are
   multiple solutions, let $v^n$ be the minimal non-negative solution.
3. For each state $s$ with action space $A(s)$, set
       $$\pi^{n+1}(s) \in \operatorname*{argmax}_{a \in A(s)} \sum_j p_{ij}^a v^n(j),$$
   breaking ties so that $\pi^{n+1}(s) = \pi^n(s)$ whenever possible.
4. If $\pi^{n+1}(s) = \pi^n(s)$ for all $s$, stop and set $\pi^* = \pi^n$. Otherwise increment $n$ by 1 and return
   to Step 2.
```

iteration algorithm because v^0 would not be simultaneously pessimistic for each player (at any state) because we are in a zero-sum game (unless the values from *VI-FP* were precisely correct). On the other hand, we can use s^* as the initial policy in policy iteration because it clearly obtains non-negative expected total reward. Therefore we opt for using the warm start and choose policy iteration in the *ex post* check.

Algorithm 9 *Ex post* check

```
Create MDP $M$ from the strategy profile $s^*$.
Run Algorithm 8 on $M$ (using initial policy $\pi^0 = s^*$) to get $\pi^*$.
**return** $\max_{i \in N, s \in S} u_i(\pi_i^*(s), s_{-i}^*(s)) - u_i(s_i^*(s), s_{-i}^*(s))$
```

Proposition 7. *Algorithm 9 correctly computes the largest amount any agent can improve its expected utility by deviating from s^*.*

Proof. By Theorem 3, the policy iteration step of the *ex post* check converges in finite time to an optimal policy of the MDP M determined by s^*. Since the agent assumes that all other players are following s^* at every state of M, the final policy π^* is a best response to s^*. Thus the algorithm returns the largest amount any agent can gain by deviating from s^*. □

As in *VI-FP*, for efficiency in Step 3 of the policy iteration (Algorithm 8) we do not compute best responses in the exponential strategy space directly, but instead we consider the extensive form and compute best responses by traversing each agent's information sets in turn. Again, this avoids the exponential blowup.

We executed the *ex post* check on the strategies output by *VI-FP*. It converged in just two iterations. The conclusion from that experiment is that for any starting state of the tournament, no player can gain more than \$0.049 by deviating from s^*. This represents less than 0.5% of the tournament entry fee. In other words, while *VI-FP* does not guarantee that convergence is to an equilibrium, it happened to indeed converge to a strategy profile that is an ϵ-equilibrium for very small ϵ.

9.5 New algorithms for solving stochastic games

The approaches used in the *ex post* check motivate new algorithms for equilibrium finding as well. We now present those new algorithms. They apply to multi-player stochastic games of either perfect or imperfect information in which best responses can be computed efficiently. Unlike *VI-FP*, each of the new algorithms has the following guarantee: if the algorithm converges, the final strategy profile is an equilibrium.

9.5.1 *PI-FP*: Policy Iteration as outer loop; Fictitious Play as inner loop

Our first new algorithm is similar to *VI-FP* except now the value updates take the form of Step 2 of Algorithm 8. After the inner loop of fictitious play converges at each stack vector to an ϵ-equilibrium s' (assuming the currently outer-loop values), we evaluate the expected payoffs at all stack vectors under the profile s' by constructing the induced MDP and finding the minimal solution of the system of equations determined by Step 2 of Algorithm 8. (We initialize the payoffs using ICM.) Thus, *PI-FP* differs from *VI-FP* in how the values are updated in the outer loop, while the inner loop remains the same.

Algorithm 10 *PI-FP*(γ, δ)

```
$V^0 \leftarrow$ ICM
diff $\leftarrow \infty$
$i \leftarrow 0$
**while** diff $> \delta$ **do**
    $i \leftarrow i + 1$
    regret $\leftarrow \infty$
    $S^0 \leftarrow$ initializeStrategies()
    **while** regret $> \gamma$ **do**
        $S^i$ = fictPlay()
        regret $\leftarrow$ maxRegret($S^i$)
    **end while**
    $M^i \leftarrow$ createTransitionMatrix($S^i$)
    $V^i \leftarrow$ evaluatePolicy($M^i$)
    diff $\leftarrow$ maxDev($V^i, V^{i-1}$)
**end while**
**return** $S^i$
```

Proposition 8. *If the sequence of strategies $\{s^n\}$ determined by iterations of the outer loop of this algorithm converges, then the final strategy profile s^* is an equilibrium.*

Proof. Let M denote the induced MDP determined by the profile s^* and define π^0 to be the policy such that $\pi^0(s) = s_i^*(s)$ for all states s in M, where i is the player corresponding to state s. Now suppose we run Algorithm 8 on M using π^0 as our initial policy. By the convergence of *PI-FP*, we must have $\pi^1(s) = \pi^0(s)$ for all s. Thus Algorithm 8 converges in one iteration on M, and we have $\pi^* = \pi^0 = s_i^*$. So, all players are playing a best response to each other under profile s^*, i.e., s^* is an equilibrium. □

While *VI-FP* might converge to a non-equilibrium given a poor initialization (e.g., if all values are initialized to be above the first-place payoff), *PI-FP* can still recover from a poor initialization since it uses the values resulting from evaluating the policy, not the values from the initialization.

9.5.2 *FP-MDP*: switching the roles of Fictitious Play and MDP solving

Our next new algorithm involves reversing the roles of the inner and outer loops from the previous two algorithms: now fictitious play serves as the outer loop and MDP solving as the inner loop. As argued previously, we prefer policy iteration to value iteration for this application because it allows us to get a good warm start more easily (although value iteration may be preferable in some applications). (We initialize values using ICM and generate our initial strategy profile s^0 as before.) As in the *ex post* check, we construct the MDP induced from the strategy profile s^0 and compute the best response for each player using policy iteration. We combine the computed best responses with s^0 using the fictitious play updating rule to obtain s^1, and repeat until the sequence $\{s^n\}$ (hopefully) converges. We can use several different termination criteria, such as specifying a number of outer loop iterations or specifying a minimum deviation for the values or strategies between two outer loop iterations.

Algorithm 11 *FP-MDP*

```
$S^0$ = initializeStrategies()
$i = 0$
**while** termination criterion not met **do**
    $M^i$ = constructMDP($S^i$)
    $S'$ = solveMDP($M^i$)
    $S^{i+1} = \frac{i}{i+1}S^i + \frac{1}{i+1}S'$
    $i = i + 1$
**end while**
**return** $S^i$
```

Proposition 9. *If the sequence of strategies $\{s^n\}$ determined by iterations of the outer loop of this algorithm converges, then the final strategy profile s^* is an equilibrium.*

Proof. By Theorem 3, policy iteration will converge to an optimal (deterministic) policy at each step t of the inner loop, which corresponds to a best response to the strategy profile s_{-i}^{t-1}. Thus by Theorem 1, convergence of the outer loop implies we are at an equilibrium. □

Again, unlike *VI-FP*, *FP-MDP* can potentially recover from poor initializations because it only uses the values resulting from evaluating the policy.

9.5.3 *FTPL-MDP*: a polynomial time algorithm for regret minimization

Note that we could use any suitable MDP-solving algorithm in the inner loop of *FP-MDP*. In particular there exists a linear programming (LP) formulation for our setting [95]. We can treat this as a polynomial-time best-response oracle. If we replace fictitious play in the outer loop of *FP-MDP* with the follow-the-perturbed-leader (FTPL) algorithm [69], we obtain an efficient procedure for minimizing external regret in large multi-player stochastic games. FTPL is similar to fictitious play (Eq. 9.1) except that instead of playing the best response to the profile s_{-i}^{t-1} at time t, player i picks the strategy j that maximizes $u_i(j, s_{-i}^{t-1}) + Z_{j,t}$, where $Z_{j,t}$ corresponds to random noise. The FTPL algorithm minimizes external regret; that is, if the game is repeated n times and a player follows the algorithm, then the per-period difference between the expected payoff of his best fixed strategy in hindsight and his expected payoff by following the FTPL algorithm approaches 0 (as $\frac{1}{\sqrt{n}}$).

Algorithm 12 *FTPL-MDP*

```
$S^0 \leftarrow$ initializeStrategies()
$i \leftarrow 0$
while termination criterion not met do
    $\hat{S}^i \leftarrow$ randomPerturbation($S^i$)
    $M^i \leftarrow$ constructMDP($\hat{S}^i$)
    $S' \leftarrow$ solveMDP-LP($M^i$)
    $S^{i+1} \leftarrow \frac{i}{i+1} S^i + \frac{1}{i+1} S'$
    $i \leftarrow i + 1$
end while
return $S^i$
```

9.6 Experiments

We conducted experiments comparing the performance of *VI-FP* with *PI-FP* and *FP-MDP* on the tournament using the payouts \$50, \$30, and \$20. The results are shown in Figure 9.1. The x-axis is the running time (in minutes, using 16 3GHz processors), and the y-axis is the maximal amount (in dollars) a player could improve by deviating from the current outer loop strategy profile of the algorithm (i.e., the ϵ of ϵ-equilibrium). The y-axis value is the maximum ϵ over all players and over all possible starting stack vectors of the tournament. The solid line denotes *VI-FP*, the dashed line denotes *PI-FP*, and the dashed/dotted line denotes *FP-MDP*. The graph was generated by running the *ex post* check algorithm over the strategies computed at the end of each outer-loop iteration of the algorithms.

Since the outer loops for the algorithms take different amounts of time, the graph contains different numbers of data points for the different algorithms (the outer loop for *VI-FP* and *PI-FP* takes about 2 hours while it takes about 40 minutes for *FP-MDP*). We halted the first two algorithms when ϵ fell below 0.05, which represents 0.1% of the first place payoff; we halted the second new algorithm after 20 hours, as it failed to achieve $\epsilon = 0.05$.

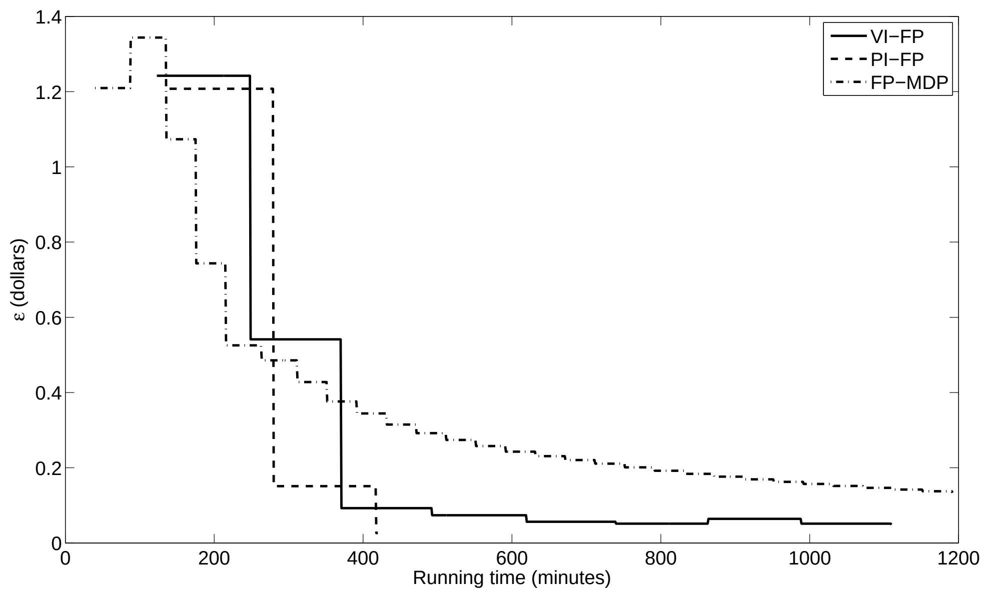


Figure 9.1: Epsilon as a function of running time.

PI-FP was the fastest to reach the target accuracy. *VI-FP* also reached it, but took almost three times as long. *FP-MDP* did not reach that accuracy in 20 hours.

9.7 Implementation discussion

9.7.1 11-card rollout

The main computational challenge of this project was precomputing the probabilities of all of the events that occur when all three players are all-in preflop (e.g., player 1 wins, players 2 and three tie for second): there are 13 such probabilities for each set of hole cards. In order to do this, we had to perform an 11-card rollout—essentially iterating over all possible hole cards for the players (6 total) and all possible sets of community cards (5 total). The straightforward approach of iterating over all

$$\binom{52}{2}\binom{50}{2}\binom{48}{2}\binom{46}{5} = 2.51 \times 10^{15}$$

possibilities would take too long, and we were forced to find ways of exploiting suit symmetries to hopefully reduce the running time.

Exploiting suit symmetries

Gilpin, Sandholm and Sørensen [47] faced similar issues when they performed a nine-card rollout for two players. Unfortunately their techniques do not extend directly to three players, and we were forced to come up with a new method of exploiting symmetries which works as follows.

First, fix an order of the players a, b, c and number the cards from 0 to 51 (0 is 2♣, 1 is 2♢, etc.). Also order the suits from 0 to 3: (♣, ♢, ♡, ♠). For each player, fix an ordering of his hole cards: $a_1 < a_2$, $b_1 < b_2$, and $c_1 < c_2$. We restrict a_1 to be of suit 0, then proceed as follows in the order a_2, b_1, b_2, c_1, c_2. Each of these cards can either be a suit of one of its predecessors, or can be 1 more than the maximum suit of one of its predecessors. For example, a_2 can either have suit 0 or 1, since its only predecessor a_1 had suit 0. Then every set of hole cards for the three players is equivalent (up to a permutation of players and suits) to a set of cards meeting the above requirements; thus we only need to iterate over this smaller set of hole cards in the rollout.

We will now give an example that demonstrates how to transform a particular hand to the canonical form discussed in the preceding paragraph. Suppose the three sets of hole cards are as follows: $(7\spadesuit, K\heartsuit), (A\diamondsuit, A\clubsuit), (6\diamondsuit, J\spadesuit)$. Since the $6\diamondsuit$ is the lowest of the six cards, we fix the last player to be player 1. Similarly we fix the first player to be player 2 since $7\spadesuit$ is the lowest card of the remaining players, and the middle player becomes player 3. Since player 1's lowest card must be a club, we must apply a permutation mapping $\diamondsuit \to \clubsuit$. Since $6\diamondsuit$ and $J\spadesuit$ are of different suits, we must map $\spadesuit \to \diamondsuit$. So far we have $a_1 = 6\clubsuit$, $a_2 = J\diamondsuit$. Since $\spadesuit$ has already been mapped to a suit, we set $b_1 = 7\diamondsuit$. Now $K\heartsuit$ does not match any of the suits seen so far: so we must set it to the lowest ranked unseen suit, which is $\heartsuit$. So our permutation maps $\heartsuit \to \heartsuit$ and therefore $\clubsuit \to \spadesuit$. Thus $b_2 = K\heartsuit$, $c_1 = A\clubsuit$, $c_2 = A\spadesuit$.

Indexing in the rollout

The previous section discussed how we were able to reduce the number of possible hole card combinations we needed to iterate over. Once we fixed a set of six hole cards for the players, we now had to iterate over all possible sets of five community cards. Let us denote the hole cards as in the previous section and call the community cards $t_1, \ldots, t_5$. Without loss of generality, we can assume that $t_1 < t_2 < \ldots < t_5$: this reduces the number of community card possibilities we need to iterate over by a factor of $5! = 120$. However, while the authors of [47] were able to come up with some clever ways of further reducing the number of community card possibilities with two players, this proved to be more difficult with three players and we were not able to do so. So for each set of hole cards, we were forced to iterate over $\binom{46}{5} = 1370754$ combinations of community cards.

Given a fixed set of hole cards and community cards, our strategy for computing the desired probabilities was the following. First, we construct the seven-card hand for the first player $(a_1, a_2, t_1, t_2, t_3, t_4, t_5)$, and do the same for the other two players. We next compute the ranking of this hand, which denotes the value of the best possible five-card hand our of these seven cards. Since performing all of these calculations at run time would make the rollout take too long, we precomputed all of these rankings of seven card hands and output them to a file, which we read into an array before we started the seven card rollout. The straightforward method of doing this would require an array with $52^7 = 10^{12}$ elements. Such a large array would slow down the rollout too much, so we were forced to find a way to make it smaller. Fortunately, we were able to apply the same indexing technique used by Gilpin, Sandholm and Sørensen [13, 47]. The *colexicographical index* of a set of integers $x = \{x_1 \ldots, x_k\} \subset \{0, \ldots, n-1\}$ with $x_i < x_j$

whenever $i < j$ is

$$colex(x) = \sum_{i=1}^{k} \binom{x_i}{i}.$$

This index has the important property that for a given n, each of the $\binom{n}{k}$ sets of size k has a distinct colexicographical index. Furthermore, these indices are compactly encoded as the integers from 0 to $\binom{n}{k} - 1$. Since the ranking of a hand is not affected by the order of the seven cards, we can assume without loss of generality that the cards are given in increasing order. So if we index each seven-card hand by the index obtained after permuting the cards so they are in increasing order, we only require an array of size $\binom{52}{7} = 133784650$. This results in a reduction of memory by a factor of 7685, which proved to be crucial given the number of iterations we had to perform.

With these techniques, the computation of determining the 11-card rollout took a month using 16 processors. The output includes for each combination of three pairs of hole cards the probability distribution over rollout outcomes (who wins, who comes second, and who comes third, treating outcomes with ties counting as separate outcomes). We then stored that database, which is used heavily in our equilibrium-finding algorithm discussed above.

9.7.2 Indexing within the equilibrium finding

Another implementation issue of note arose when we needed to access the rollout probabilities within our equilibrium computation. First, notice that each player is playing a strategy that only depends on his hand ranking and not the specific cards per se. That is, each player treats $Q\heartsuit 8\spadesuit$ and $Q\diamondsuit 8\clubsuit$ the same strategically, even though the actual cards are different (there are 169 strategically distinct hands). The straightforward method of reading in the output of the 11-card rollout would involve using an array with $52^6 \times 13 = 2.57 \times 10^{11}$ entries (13 outcomes for each set of 6 hole cards). However, for the purposes of our algorithm, we can combine all of the results in which a player has two different but strategically equivalent hands together into a single entry of the array. Thus we can collapse this array into one of size $169^3 \times 13$ which has about 63 million entries. However, even after this reduction the time needed to access elements of this array made the running time too long (since the accesses were nested inside several loops in each iteration of the inner loop).

We observed that we could obtain a further reduction in memory by fixing a permutation of the hand rankings. That is, suppose the three hand rankings are a, b, and c. The array described in the previous paragraph would have up to six different entries corresponding to different permutations of these rankings. However, suppose we fix an ordering of the hand rankings and suppose $a \leq b \leq c$ using this ordering (note that there can be equalities: for example, two players can be dealt pocket aces). Then our new array would just have an element corresponding to the triple (a, b, c) and not all of the other permutations. However, we cannot use the indexing scheme of the previous section because of the possibility of equalities between hand rankings. Fortunately, Troels Sørenson suggested to us the following indexing scheme, which is able to deal with multisets of this form. Now we define the *multiset-colexicographical index* of a set of integers $x = \{x_1 \ldots, x_k\} \subset \{0, \ldots, n-1\}$ with $x_i \leq x_j$ whenever $i < j$ to be

$$multi - colex(x) = \sum_{i=1}^{k} \binom{x_i + i - 1}{i}.$$

Using this indexing scheme, we only require an array of size 881805×13, which has about 11 million entries. This is about a factor of 5.5 further reduction and proved to be enough for our algorithm to run sufficiently fast.

We also found it useful to map each stack vector $s = (s_1, s_2, s_3)$ to a unique index using the following formula, where we assume $0 < s_i \leq n$ and s_i is an integer for each i (we divided the stacks by 300 before applying this):

$$\begin{aligned} stack - colex(s) &= (s_2 - 1) + \sum_{i=1}^{s_1 - 1} \sum_{j=1}^{n-i-1} 1 \\ &= -0.5s_1^2 + s_1(n - 0.5) + s_2 - n. \end{aligned}$$

9.8 Poker strategy observations

9.8.1 Prior results for a two-player poker tournament

A recent paper [82] computes near-optimal jam/fold strategies for tournaments with two players and the fixed parameters SB = 300, BB = 600, and 8000 total chips (at the time, these defined the normal parameters for the heads-up endgame in tournaments on PartyPoker.com). In fact, if we let G_{i,s_i} denote the restriction of the original tournament in which player i starts with s_i chips and is limited to playing a jam/fold strategy (while the other player is not), they show that for any value of s_1, $V_1(G_{1,s_1}) + V_2(G_{2,8000-s_1}) \geq 0.986$ (where the winner gets payoff 1, the loser gets payoff 0, and V_i denotes the value of the game to player i). Thus, neither player can guarantee more than 0.014 by deviating to a non-jam/fold strategy; this provides a justification for restricting attention to jam/fold strategies. They compute the optimal jam/fold strategies for the full stochastic games G_{1,s_1} and G_{2,s_2}, where they consider all states in which stack sizes are a multiple of 50. One important conclusion they draw is that for any starting stack sizes s_1, s_2, the probability that player i will win is very close to

$$\frac{s_i}{s_1 + s_2}. \tag{9.2}$$

In addition to being simple, this formula is nice for another reason. Consider the following game: two players start with stacks s_1 and s_2, and each round they flip a coin and the loser pays the winner some fixed number ϵ chips until one player has no more chips. This is just the gambler's ruin problem, with the same solution: player i wins with probability $\frac{s_i}{s_1+s_2}$.

An additional noteworthy conclusion of [82] is that the optimal strategy involves very little randomization: at each state each player only needs to randomize (play an action with probability not equal to 0 or 1) with at most one hand. They also note that there is no general strength-ranking of hands: there exist hands A and B and stack vectors v and w such that A should be jammed and B should be folded at v, but B should be jammed and A should be folded at w. Finally, they prove that the optimal strategy for a single hand of a cash game is almost identical to that of a tournament.

9.8.2 Independent Chip Model (ICM)

It is not clear which of the conclusions that hold for two players extend to more players; however, one thing that is clear is that chips and money do not necessarily coincide with more than two players (while they do with two players). For example, suppose the chip stacks are 5000, 4900, and 100 with blinds at 50/100. If the player with 100 folds and the player with 5000 jams, should the player A with 4900 be indifferent between taking a 50-50 for all his chips (in this hypothetical example we assume he is sure it will be a 50-50 if he calls)? In a single hand of a cash game he should be indifferent, because the expected payoff of calling and folding are both 0 (ignoring the blinds which are negligible compared to the relevant stack sizes). However this is not the case in a tournament with 50/30/20 payoffs. If he folds, then the short stack will be eliminated very soon and A will expect to come in first and second with probability $\frac{1}{2}$: so his expected payoff is \$40. If he calls, then he will basically guarantee winning the tournament if he wins, but will come in third if he loses. So his expected payoff is \$35. So he will gain \$5 on average by folding.

Unfortunately, there is no obvious way of generalizing equation (9.2) to come up with a heuristic for performing expected value calculations in tournaments with more than two players; however, the following formula has been proposed and has received widespread acceptance in the poker community for the last several years [110]:

$$\begin{aligned} U_i &= P_1 * \frac{s_i}{S} + P_2 \times \sum_{j \neq i} \left[\frac{s_j}{S} \times \frac{s_i}{S - s_j} \right] \\ &+ P_3 \times \sum_{j \neq i} \sum_{k \neq i, k \neq j} \left[\frac{s_j}{S} \times \frac{s_k}{S - s_j} \times \frac{s_i}{S - s_j - s_k} \right], \end{aligned}$$

where U_i denotes the expected payoff (ignoring the entry fee) of player i, s_i denotes his stack, $S = \sum_j s_j$, and P_1, P_2, P_3 are the payoffs to the top three finishers. This formula is referred to as the *Independent Chip Model* (ICM), and all of the popular tournament software tools (such as [107]) use ICM to determine expected payoffs.

9.8.3 Assessing the Independent Chip Model

After we computed the approximate equilibrium strategies in the tournament, we compared the payoffs for each player at each stack vector (when our new equilibrium strategy is played) to ICM predictions. Our findings are listed in Table 9.1. The left column denotes the absolute value of the difference between the ICM prediction and our final result in dollars (where the total prize pool is \$100). Since there are 946 possible stack scenarios and three players, there are 2838 total deviations. The average deviation was \$0.3703, and the range was from $\$4.42 \times 10^{-5}$ to \$2.99.

The largest deviation of \$2.99 occurs at the following stacks: $s_1 = 300, s_2 = 12300, s_3 = 900$ (for button, small blind, big blind). ICM predicts that the prize winnings of the big blind are \$28.848, while we obtain \$25.856. ICM also undervalues the button's prize equity by \$2.49

Table 9.1: Deviations between ICM and our payoffs

Range	Frequency
0-0.1	510
0.1-0.2	460
0.2-0.3	368
0.3-0.4	363
0.4-0.5	403
0.5-1	626
1-3	108

($22.96 vs. $25.45), and undervalues the small blind's equity by $0.50 ($48.19 vs. $48.69) at these stacks. The reason for this large deviation is that ICM thinks that the big blind is three times as likely to win the tournament as the button (because his stack is three times as large); however, this does not take into account the fact that the big blind must post $\frac{2}{3}$ of his stack as blinds the next hand. If the button folds on the next hand and the small blind jams, the big blind will be getting 6–1 odds to call. If he calls and loses then he will finish in 3rd, while if he wins then he will almost ensure finishing in second. On the other hand, if he folds then both he and the button will be all-in on the next hand, and it will be a coin-flip as to who will finish in second and third. Thus, the prize equities of the button and big blind should actually be very close to each other (as our results confirm).

9.8.4 Comparing tournament strategies and single-hand strategies

In this section we compare the strategies we computed for tournaments with the strategies we computed for a single hand. Interestingly, the conclusions are very different from those resulting from the same comparison in the two-player game [82]. In the multiplayer setting, there is a big difference between the tournament case and the single-hand case, while in the two-player setting there was not.

Tables 9.2– 9.7 give our computed approximate equilibrium strategies for all players in a tournament with even chip stacks $s_1 = s_2 = s_3 = 4500$ and blinds at $SB = 300$, $BB = 600$, while tables 9.8–!9.13 give the strategies for a single hand of a cash game with the same parameters. Each square in the tables represents one of the 169 distinct starting hands, with suited hands being in the upper right and unsuited hands in the lower left. The numbers in each square denote the probability the player should jam with that hand, rounded to the nearest 1%. A 'P' (for 'push') means that the probability of jamming is 100%, and 'F' (for 'fold') means the probability of jamming is 0%. So for example, in Table 9.4 the small blind should jam 93 suited with probability 0.23, and should always fold 93 unsuited.

Tournament strategy with equal stacks

(suited hands in upper right, unsuited in lower left)

Table 9.2: Button

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	99	99	99	99	99	97	25
Q	P	P	P	P	99	99	99	96	95	31	8	1	1
J	P	P	99	P	99	99	99	94	F	F	F	F	F
T	P	P	99	99	P	99	99	94	F	F	F	F	F
9	P	99	95	93	94	P	99	95	6	F	F	F	F
8	P	13	1	F	F	F	P	96	17	F	F	F	F
7	P	5	F	F	F	F	F	P	82	F	F	F	F
6	99	2	F	F	F	F	F	F	P	F	F	F	F
5	99	1	F	F	F	F	F	F	F	P	F	F	F
4	99	1	F	F	F	F	F	F	F	F	P	F	F
3	99	F	F	F	F	F	F	F	F	F	F	99	F
2	99	F	F	F	F	F	F	F	F	F	F	F	99

Table 9.3: Small blind after button jams

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	99	98	F	F	F	F	F	F	F
K	P	P	F	F	F	F	F	F	F	F	F	F	F
Q	P	F	P	F	F	F	F	F	F	F	F	F	F
J	P	F	F	P	F	F	F	F	F	F	F	F	F
T	99	F	F	F	P	F	F	F	F	F	F	F	F
9	F	F	F	F	F	P	F	F	F	F	F	F	F
8	F	F	F	F	F	F	99	F	F	F	F	F	F
7	F	F	F	F	F	F	F	98	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F	F	F	F	F

Table 9.4: Small blind after button folds

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	P	P	P	P
Q	P	P	P	P	P	P	P	P	99	99	99	99	99
J	P	P	P	P	P	P	P	99	99	99	99	99	94
T	P	P	P	P	P	P	P	99	99	99	99	93	88
9	P	P	P	99	99	P	P	99	99	99	90	23	2
8	P	P	99	99	99	99	P	99	99	99	99	2	2
7	P	P	91	91	99	99	99	P	99	99	99	3	2
6	P	P	90	3	3	87	94	99	P	99	99	92	2
5	P	99	3	2	1	2	2	3	91	P	99	99	85
4	P	99	3	2	1	1	1	1	2	3	P	93	2
3	P	99	2	1	F	F	F	F	F	1	1	P	2
2	P	99	2	1	F	F	F	F	F	F	F	F	P

Table 9.5: Big blind after jam/fold

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	99	99	89	87	F	F	F
K	P	P	99	99	89	F	F	F	F	F	F	F	F
Q	P	99	P	F	F	F	F	F	F	F	F	F	F
J	P	F	F	P	F	F	F	F	F	F	F	F	F
T	P	F	F	F	P	F	F	F	F	F	F	F	F
9	99	F	F	F	F	P	F	F	F	F	F	F	F
8	99	F	F	F	F	F	P	F	F	F	F	F	F
7	F	F	F	F	F	F	F	P	F	F	F	F	F
6	F	F	F	F	F	F	F	F	P	F	F	F	F
5	F	F	F	F	F	F	F	F	F	99	F	F	F
4	F	F	F	F	F	F	F	F	F	F	15	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F	F	F	F	F

Table 9.6: Big blind after fold/jam

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	99
K	P	P	P	P	P	P	99	99	87	4	F	F	F
Q	P	P	P	99	99	98	3	F	F	F	F	F	F
J	P	P	99	P	98	1	F	F	F	F	F	F	F
T	P	P	91	F	P	F	F	F	F	F	F	F	F
9	P	99	F	F	F	P	F	F	F	F	F	F	F
8	P	6	F	F	F	F	P	F	F	F	F	F	F
7	P	2	F	F	F	F	F	P	F	F	F	F	F
6	99	F	F	F	F	F	F	F	P	F	F	F	F
5	99	F	F	F	F	F	F	F	F	P	F	F	F
4	99	F	F	F	F	F	F	F	F	F	P	F	F
3	99	F	F	F	F	F	F	F	F	F	F	3	F
2	5	F	F	F	F	F	F	F	F	F	F	F	F

Table 9.7: Big blind after jam/jam

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	96	1	F	F	F	F	F	F	F	F	F	F
K	2	P	F	F	F	F	F	F	F	F	F	F	F
Q	F	F	P	F	F	F	F	F	F	F	F	F	F
J	F	F	F	P	F	F	F	F	F	F	F	F	F
T	F	F	F	F	1	F	F	F	F	F	F	F	F
9	F	F	F	F	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F	F	F	F	F

Single hand strategy with equal stacks

(suited hands in upper right, unsuited in lower left)

Table 9.8: Button

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	P	P	98	F
Q	P	P	P	P	P	P	P	96	94	F	F	F	F
J	P	P	P	P	P	P	98	95	F	F	F	F	F
T	P	P	P	P	P	P	98	96	F	F	F	F	F
9	P	P	F	F	20	P	99	97	F	F	F	F	F
8	P	2	F	F	F	F	P	98	95	F	F	F	F
7	P	F	F	F	F	F	F	P	97	F	F	F	F
6	P	F	F	F	F	F	F	F	P	94	F	F	F
5	P	F	F	F	F	F	F	F	F	P	F	F	F
4	P	F	F	F	F	F	F	F	F	F	P	F	F
3	P	F	F	F	F	F	F	F	F	F	F	P	F
2	P	F	F	F	F	F	F	F	F	F	F	F	P

Table 9.9: Small blind after button jams

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	1	1	1	F	F	F	F
Q	P	P	P	P	P	1	F	F	F	F	F	F	F
J	P	P	91	P	P	1	F	F	F	F	F	F	F
T	P	P	1	F	P	1	F	F	F	F	F	F	F
9	P	1	F	F	F	P	F	F	F	F	F	F	F
8	P	F	F	F	F	F	P	F	F	F	F	F	F
7	P	F	F	F	F	F	F	P	F	F	F	F	F
6	P	F	F	F	F	F	F	F	P	F	F	F	F
5	P	F	F	F	F	F	F	F	F	P	F	F	F
4	1	F	F	F	F	F	F	F	F	F	P	F	F
3	1	F	F	F	F	F	F	F	F	F	F	P	F
2	1	F	F	F	F	F	F	F	F	F	F	F	P

Table 9.10: Small blind after button folds

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	P	P	P	P
Q	P	P	P	P	P	P	P	P	P	P	P	P	P
J	P	P	P	P	P	P	P	P	P	P	99	99	99
T	P	P	P	P	P	P	P	P	99	99	99	17	F
9	P	P	P	P	P	P	P	P	99	99	F	F	F
8	P	P	P	P	P	99	P	P	99	99	98	F	F
7	P	P	P	P	99	99	99	P	99	99	98	F	F
6	P	P	P	F	F	F	8	98	P	99	99	11	F
5	P	P	P	F	F	F	F	F	F	P	99	98	F
4	P	P	P	F	F	F	F	F	F	F	P	98	F
3	P	P	4	F	F	F	F	F	F	F	F	P	F
2	P	P	F	F	F	F	F	F	F	F	F	F	P

Table 9.11: Big blind after jam/fold

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	92	1	1	1
Q	P	P	P	P	P	P	89	1	1	P	F	F	F
J	P	P	P	P	P	95	1	F	F	F	F	F	F
T	P	P	P	48	P	95	1	F	F	F	F	F	F
9	P	P	1	F	F	P	F	F	F	F	F	F	F
8	P	1	F	F	F	F	P	F	F	F	F	F	F
7	P	1	F	F	F	F	F	P	F	F	F	F	F
6	P	1	F	F	F	F	F	F	P	F	F	F	F
5	P	1	F	F	F	F	F	F	F	P	F	F	F
4	P	F	F	F	F	F	F	F	F	F	P	F	F
3	P	F	F	F	F	F	F	F	F	F	F	P	F
2	P	F	F	F	F	F	F	F	F	F	F	F	P

Table 9.12: Big blind after fold/jam

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	P	P	P	P
Q	P	P	P	P	P	P	P	P	P	P	99	96	1
J	P	P	P	P	P	P	P	98	1	F	F	F	F
T	P	P	P	P	P	P	P	5	F	F	F	F	F
9	P	P	P	P	97	P	99	1	F	F	F	F	F
8	P	P	P	93	F	F	P	F	F	F	F	F	F
7	P	P	97	F	F	F	F	P	F	F	F	F	F
6	P	P	1	F	F	F	F	F	P	F	F	F	F
5	P	P	F	F	F	F	F	F	F	P	F	F	F
4	P	P	F	F	F	F	F	F	F	F	P	F	F
3	P	P	F	F	F	F	F	F	F	F	F	P	F
2	P	99	F	F	F	F	F	F	F	F	F	F	P

Table 9.13: Big blind after jam/jam

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	5	3	2
K	P	P	P	P	P	P	2	1	1	1	F	F	F
Q	P	P	P	P	P	P	2	F	F	F	F	F	F
J	P	P	P	P	P	P	3	F	F	F	F	F	F
T	P	8	2	2	P	P	P	F	F	F	F	F	F
9	P	1	F	F	F	P	P	2	F	F	F	F	F
8	4	F	F	F	F	F	P	P	F	F	F	F	F
7	2	F	F	F	F	F	F	P	7	F	F	F	F
6	1	F	F	F	F	F	F	F	P	F	F	F	F
5	1	F	F	F	F	F	F	F	F	P	F	F	F
4	1	F	F	F	F	F	F	F	F	F	P	F	F
3	1	F	F	F	F	F	F	F	F	F	F	P	F
2	F	F	F	F	F	F	F	F	F	F	F	F	2

Table 9.14 gives the total probabilities of the different outcomes when both players use their computed strategies in a tournament and a single hand. The chart suggests that the equilibrium strategies are fairly similar in a tournament and single hand for the first player to enter the pot, but that players should be much more aggressive in a single hand when someone else has already jammed. The difference is most significant in the final situation: big blind jams with probability 0.197 in a single hand after both of the other players jam, but only probability 0.021 in a tournament (a factor of 9.4 difference). In a tournament, the big blind only calls with the four highest pocket pairs and AKs ('s' denotes 'suited,' 'o' denotes 'offsuit'). In a single hand, his range includes hands like 33, A9o, K9s, QJo, T8s, and 87s.

Table 9.14: Differences between tournament and single hand strategy with equal stacks.

Situation	Tournament prob.	Single hand prob.
Button jam	0.376	0.355
SB jam after jam	0.088	0.234
SB jam after fold	0.652	0.633
BB jam after jam/fold	0.145	0.308
BB jam after fold/jam	0.273	0.466
BB jam after jam/jam	0.021	0.197

Interestingly, despite the fact that the small blind jams slightly more often following a fold in the tournament than single hand, the big blind jams following fold/jam almost twice as often in a single hand as in a tournament. Hands that the small blind jams following a fold in a tournament but not a single hand include T3s, 94s, 52s, 96o, 65o. The small blind also jams some hands in a single hand that he folds in a tournament such as Q5o and Q4o. Despite the fact that the small blind is jamming less often following a fold in the single hand, the big blind still calls with dozens of marginal hands that he would fold in a tournament, including 33, 22, A2o, K5s–K2s, K8o–K2o, Q8s–Q3s,Q9o–Q7o, J9s–J7s, JTo–J8o, T9s–T8s, T9o, and 98s.

9.8.5 Nonexistence of a fixed ranking of hands

Two player results [82] show that there is no single ranking of the hands (i.e., there exist two hands h and h' such that at one stack vector, h should be jammed and not h', but at another stack vector, h' should be jammed and not h). In particular, when SB = 1800, BB = 6200 the small blind should fold 43s and jam J2o, but when SB = 3600 and BB = 4400 the small blind should jam 43s and fold J2o. We obtain similar results with three players. For example, we observe such a phenomenon in the small blind's strategy given a fold between the cases of equal stacks (Table 9.4) and the stacks: BUT = 3300, SB = 1500, BB = 8700 (Table 9.15). In the latter, the small blind jams with Q2o–Q5o, J5o–J6o but folds T2s–T4s, 94s–95s, 96o–97o, 84s–85s, 86o–87o, 74s–75s, 76o, 63s–65s, 65o,52s–54s, 43s (all of which mark deviations from the first case).[1]

Table 9.15: Small blind strategy after button folds with stacks (3300, 1500, 8700).

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	P	P	P	P	P	P	P	P	P	P	P	P	P
K	P	P	P	P	P	P	P	P	P	P	P	P	P
Q	P	P	P	P	P	P	P	P	P	P	P	P	P
J	P	P	P	P	P	P	P	P	P	P	99	97	91
T	P	P	P	P	P	P	P	P	99	90	F	F	F
9	P	P	P	P	P	P	P	P	95	F	F	F	F
8	P	P	P	P	P	97	P	97	90	F	F	F	F
7	P	P	P	P	95	21	F	P	85	F	F	F	F
6	P	P	P	96	F	F	F	F	P	F	F	F	F
5	P	P	P	91	F	F	F	F	F	P	F	F	F
4	P	P	P	F	F	F	F	F	F	F	P	F	F
3	P	P	97	F	F	F	F	F	F	F	F	P	F
2	P	P	93	F	F	F	F	F	F	F	F	F	P

It is also interesting to compare our results to the Karlson-Sklansky (K-S) hand ranking system; these rankings are given in Table 9.16, where 0 denotes the best hand and 168 denotes the worst hand. This system has often been proposed as a good heuristic for evaluating hand strength, and many software tools use hand ranges based on K-S rankings. However, our results for the even stack case show that equilibrium strategies might drastically contradict K-S rankings. For example, in the even stack case after the button has folded (Table 9.4) the small blind should fold Q5o (K-S ranking 84), but jam with 52s (K-S ranking 155). Interestingly, it seems

[1]Note that we are just computing one equilibrium, and it is still possible that other equilibria do exist that are consistent with a fixed ranking of hands. So we cannot not conclude with certainty that there is no fixed ranking of hands consistent with an equilibrium. This is a question I would like to study further in the future.

that equilibrium strategy actually agrees pretty closely with K-S rankings for players acting after another player has already jammed. For example, strategy for the big blind contradicts K-S rankings only on a few hands.

Table 9.16: Karlson-Sklansky hand rankings

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0	2	6	10	12	17	19	23	27	25	28	32	34
K	4	1	20	24	30	42	46	47	50	53	55	58	59
Q	8	33	3	39	45	52	60	67	69	72	75	77	81
J	13	38	51	5	48	62	71	79	87	89	91	96	99
T	16	44	56	65	7	68	78	88	97	106	107	112	116
9	22	49	64	76	86	9	83	94	104	114	123	127	132
8	26	54	74	85	95	102	11	100	108	118	128	138	141
7	31	57	80	92	103	111	117	14	113	122	133	142	151
6	36	61	82	101	110	119	126	131	15	125	137	146	155
5	35	63	84	105	120	129	136	140	144	18	134	145	154
4	37	66	90	109	124	139	147	150	153	152	21	149	157
3	40	70	93	115	130	143	156	159	161	160	163	29	162
2	43	73	98	121	135	148	158	164	166	165	167	168	41

9.8.6 Rarity of randomization

As in the two-player case [82], we observe that approximate equilibrium strategy for three players involves little randomization (many of the probabilities are actually very close — but not equal to — 0% or 100% because we halted fictitious play as soon as the ϵ-approximation guarantee was met which did not allow it to fully converge). For example, the tables show that in the case of even stacks, our equilibrium involves randomization on only a few hands. In fact, the small blind's strategy after button jams, and the big blind's strategy following two jams, involve no randomization. In all other stack sizes we make the similar observation that each player only needs to randomize on at most a small number of hands.

9.9 Summary and extensions

Computing a Nash equilibrium in multi-player stochastic games is a notoriously difficult problem. We pointed out that the best prior algorithm could converge to a non-equilibrium strategy profile; however, we developed an *ex post* check procedure that confirmed that the actual strategy profile computed for a large three-player poker tournament actually constitutes an ϵ-equilibrium for very small ϵ. We also presented several new algorithms for solving multi-player stochastic games of imperfect information and proved that they can never converge to a non-equilibrium. Experiments on the poker tournament showed that one of the new algorithms (*PI-FP*) outperforms the prior algorithm *VI-FP*; thus the equilibrium guarantee comes at no cost in run time.

Finally, we presented a polynomial-time algorithm for minimizing external regret in the same class of games.

We find it interesting that the algorithms converged to an equilibrium consistently and quickly despite the fact that they are not guaranteed to do so. None of the algorithms are guaranteed to converge at all, and the original algorithm was not even guaranteed to be at an equilibrium even if it did converge. Hopefully the results of this paper could lead to the investigation of more general settings under which these convergence properties can be proven. While fictitious play is not guaranteed to converge in three-player games, it converged in every iteration of all of our algorithms; perhaps there is a more general class of multi-player games (that includes poker tournaments) for which fictitious play is guaranteed to converge. Furthermore, the value iteration of the outer loop of the original algorithm converged despite the fact that the initializations were clearly not pessimistic for every player; perhaps the assumptions of Theorem 2 can be weakened to allow limited optimistic initializations in some settings.

Part III

New Game-Solving Paradigms

Chapter 10

Endgame Solving in Large Imperfect-Information Games

So far we have presented new approaches for game solving that fit within the leading paradigm. In the leading paradigm, strategies are computed offline in advance, and the strategies are then looked up in a table for actual game play. I now present a new paradigm, in which we retain the abstract equilibrium strategies for the initial portion of the game tree (called the *trunk*), and discard the strategies for the final portion (called the *endgames*). Then, in real time, we solve the relevant endgame that we have reached to a greater degree of accuracy than the initial abstract strategy, where we use Bayes' rule to compute the distribution of players' private information leading into the endgames from the precomputed trunk strategies. This approach, which we call *endgame solving*, is depicted in Figure 10.1 [36].

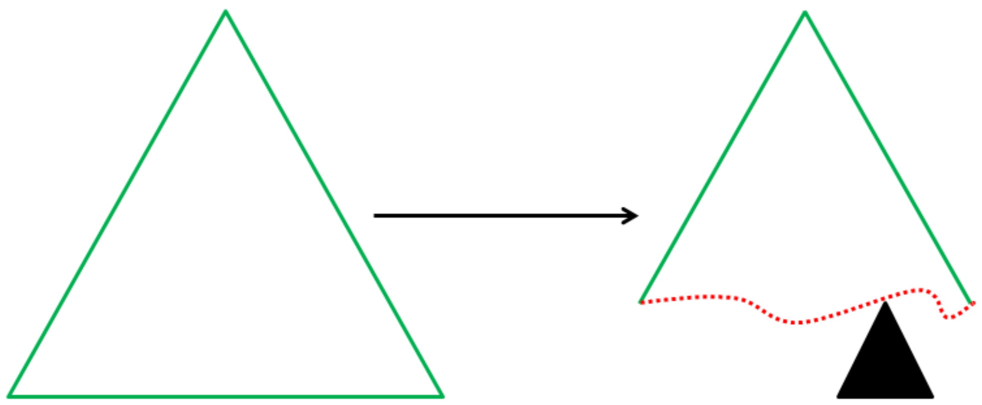


Figure 10.1: Endgame solving (re-)solves the relevant endgame that we have actually reached in real time to a greater degree of accuracy than in the offline computation.

We present the first theoretical analysis of endgame solving in imperfect-information games, and show that it can actually produce highly exploitable strategies in some games. In fact, we show that it can fail even in a simple game with a unique equilibrium and a single endgame, even if our base strategy were an exact equilibrium (of the full game) and we were able to compute an exact equilibrium in the endgame. However, we show that endgame solving can guarantee a low *exploitability* (difference between game value and payoff against a nemesis) in some games when the opponent is given sufficient exploitative power within the endgame.

Endgame solving has been used by several prior agents for the limit variation of TH (where bets must be of a single fixed size). The agent GS1 precomputed strategies only for the first two rounds, using rough approximations for the payoffs at the leaves of that trunk based on the (unrealistic) assumption that there was no betting in future rounds [42]. Then in real time, the relevant endgame consisting of the final two rounds was solved using the LP algorithm. GS2 precomputed strategies for the first three rounds, using simulations to estimate the payoffs at the leaves; it then solved the endgames for the final two rounds in real time [43].

However endgame solving has not been implemented by any competitive agents for the significantly larger and more challenging domain of no-limit Texas hold 'em (NLTH) prior to our work. We present a new algorithm that is capable of scaling to extremely large games such as no-limit Texas hold 'em, and incorporates several algorithmic improvements over the prior approaches (the benefits described in this paragraph would be improvements over the prior approaches even for the limit variant). First, the prior approaches assume that the private hand distributions leading into the endgame are independent, while they are actually dependent and the full joint distribution should be computed. The naïve way of accomplishing this would require $O(n^2)$ strategy table lookups, where n is the number of private hands (1081 for the final round of poker), and computing these distributions would become the bottleneck of the algorithm and make the real-time computation intractable; however, we developed a technique for computing the joint distributions that requires just $O(n)$ strategy table lookups. Second, the prior approaches use a single perfect-recall card abstraction that has been precomputed offline (which assumes a uniform random distribution for the opponent's hand distributions). In contrast, we use an imperfect-recall card abstraction[1] that is computed in real time in a finer granularity than the initial offline abstraction and that is tailored specifically to the relevant distribution of the opponent's hands at the given hand history. Furthermore, the prior approaches did not compare performance between endgame solving and not using it (since the base strategies were not computed for the endgames), while we provide such a comparison.

Very recent work, which appeared subsequently to the first version of our present work, has presented an approach for decomposing imperfect-information games into smaller games that can be solved independently offline that provides theoretical guarantees on full-game exploitability. This approach, called CFR-D, has been applied to the relatively small domain of limit Leduc hold 'em, which has 936 information sets in its game tree [19]. It is possible that this approach will soon be practical for larger games such as NLTH, though no successful implementation on this domain has been reported. Challenges for extending this approach to larger domains include the computation of counterfactual values for each state of private information at the start of the endgame, and needing to construct and solve larger endgames that result from adding in an additional action for each state of private information for each agent that obtains this counterfactual value for avoiding the endgame. I have also learned from personal communication with the authors that CFR-D has performed worse empirically in head-to-head performance than the procedure described in this work, despite the additional theoretical guarantee.

[1]Imperfect-recall abstractions allow for greater flexibility in which hands can be grouped together, and significantly improve performance over perfect-recall abstractions [68, 120].

A second related (offline) approach includes counterfactual values for game states that could have been reached off the path to the endgames [61]. This approach has been demonstrated to be effective in limit Leduc hold 'em, and has also been implemented in NLTH, though no experimental results are given for that domain. For NLTH, it is implemented by first solving the game in a coarse abstraction, then fixing the strategies for the preflop (first) round, and re-solving for certain endgames starting at the flop (second round) after common preflop betting sequences have been played. All of this computation is done offline. In contrast, our approach enables us to solve endgames at the river (final round) in real time. It is infeasible to solve the river endgames using the prior approach for several reasons. First, there are far too many of them to be solved individually in advance (there is a different one for each sequence of public cards and betting actions). Second, by the time play gets down to the river, there are many possible alternative actions that a player could have taken to avoid reaching the given endgame, and counterfactual values for each of these would need to be computed and then included in the solution to the endgame solver; this would likely be infeasible to do in real time. Solving the river endgames, as opposed to the flop endgames which the prior approach does, is very important because CFR only occasionally samples from a specific river endgame during the course of the initial equilibrium computation, while it very frequently samples from the flop endgames that follow common preflop betting sequences. So, our approach is addressing a more pressing limitation.

Our approach has significant benefits over the standard approach for solving large imperfect-information games, including computation of exact (rather than approximate) equilibrium strategies (within a given abstraction), the ability to compute certain equilibrium refinements that cannot be computed in the full offline computation (such as undominated and ϵ-quasi-perfect equilibrium), finer-grained abstraction in the endgames, abstraction that takes into account realistic distributions of players' private information entering the endgame (as opposed to the typical assumption of uniform random distributions), and a solution to the "off-tree" problem that arises when the opponent has taken actions that are not allowed in the abstraction. We present an efficient algorithm for performing endgame solving in large imperfect-information games, and present a novel variance-reduction technique for evaluating the performance of an agent that uses endgame solving. Experiments on no-limit Texas hold 'em show that using our algorithm leads to a significantly stronger performance against the strongest 2013 poker competition agents. I demonstrate that certain equilibrium refinements are feasible to compute in large imperfect-information games for the first time—particularly undominated Nash equilibrium and ϵ-quasi-perfect equilibrium. I show that undominated Nash equilibrium is a useful concept in realistic large imperfect-information games by showing that the equilibrium computed in a well-studied poker variant by the standard approach is dominated. I show that these concepts can be integrated with endgame solving and present results indicating that they can lead to stronger performance.

10.1 Endgame solving

Definition 1. (E, t_i, t_{-i}) *(or* E *for brevity) is an* endgame *of game* G *if the following properties hold:*

1. *The set of E's nodes is a subset of the set of G's nodes.*
2. *If s' is a child of s in G and s is a node in E, then s' is also a node in E.*
3. *If s is in the same information set as s' in G and s is a node in E, then s' is also a node in E.*
4. *The initial probabilities of private information for the agents entering E are defined by the Bayesian conditional probabilities assuming the agents have followed strategies t_i, t_{-i} in the portion of the game preceding E.*

For example, we can consider endgames in poker where several rounds of betting have taken place and several public cards have already been dealt. In these endgames, we can assume players have a joint distribution of private information from nodes prior to the endgame that are induced from the precomputed base approximate-equilibrium strategy using Bayes' rule. Given this distribution as input, we can then solve individual endgames in real time using more accurate abstractions. We can view the endgames as new games where nature makes an initial move by assigning private information according to the joint distribution induced by the trunk strategies.

Unfortunately, this approach has some fundamental theoretical shortcomings. It turns out that even if we computed an exact equilibrium in the trunk (which is an unrealistically optimistic assumption in large games) and in the endgame, the combined strategies for the trunk and endgame may fail to be an equilibrium in the full game. One obvious reason for this is that the game may contain many equilibria, and we might choose one for the trunk that does not match up correctly with the one for the endgame; or we may compute different equilibria in different endgames that do not balance appropriately. However, Proposition 10 shows that it is possible for this procedure to output a non-equilibrium strategy profile in the full game even if the full game has a unique equilibrium and a single endgame.

Proposition 10. *There exist games—even with a unique equilibrium and a single endgame—for which endgame solving can produce a non-equilibrium strategy profile.*

Proof. Consider a sequential version of Rock-Paper-Scissors where player 1 acts, then player 2 acts without observing player 1's action. This game has a single endgame—when it is player 2's turn to act—and a unique equilibrium—where each player plays each action with probability $\frac{1}{3}$. Now suppose we restrict player 1 to follow the equilibrium in the initial portion of the game. Any strategy for player 2 is an equilibrium in the endgame, because each one yields her expected payoff 0. In particular, suppose our equilibrium solver outputs the pure strategy Rock for her. This is clearly not an equilibrium of the full game. $\square$

Rock-Paper-Scissors (RPS) is somewhat of an extreme example though, because player 1 does not actually make any moves in the endgame. At the other extreme, if the endgame were the entire game, then endgame solving would produce an exact equilibrium. As a slightly less extreme example, consider the game in Figure 10.2, where P1 selects an action a_i, and then a sequential imperfect-information game G_i is played. Suppose we are solving endgames after P1's initial action. Then we will solve the endgame G_i and produce strategies with zero exploitability in the full game. Endgame solving could be very useful in this game for several reasons. First, if the number of initial actions n for P1 were extremely large, it may be infeasible to solve and/or store solutions to all of the endgames in advance of game play. Endgame solving would only require solving the endgames that are actually reached during game play, and would be feasible even if n is extremely large as long as the number of game repetitions were relatively small. And second, the typical approach would actually not even involve solving each of the G_i separately in advance; it would be to solve the full game, which includes each of the G_i as well as P1's initial actions. It is very possible that equilibrium-finding algorithms would not scale to the full game and/or it would not fit in memory, while equilibria could be computed quickly and fit into memory for the individual endgames G_i.

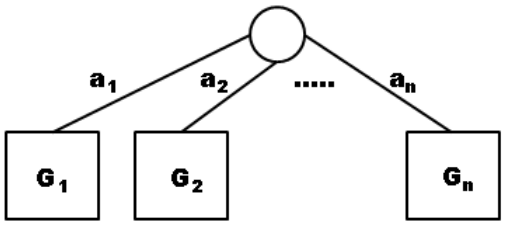


Figure 10.2: Player 1 selects his action a_i, then the players play imperfect-information game G_i.

One could imagine much more complex trunk games than the above example with imperfect information and multiple actions for both players where it is difficult to know for sure how "important" the trunk strategies are for the endgames. In such games, it may be possible for endgame solving to still guarantee a reasonably low exploitability in the full game. As Proposition 11 shows, in general, the more exploitative power the opponent has within the endgame, the lower the full-game exploitability of the strategies produced by (approximate) endgame solving are.

Proposition 11. *If every strategy that has exploitability strictly more than ϵ in the full game has exploitability of strictly more than δ within the endgame, then the strategy output by a solver that computes a δ-equilibrium in the endgame induced by a trunk strategy t would constitute an ϵ-equilibrium of the full game when paired with t.*

Proof. Suppose a strategy is a δ-equilibrium in the endgame induced by t, but not an ϵ-equilibrium in the full game when paired with t. Then by assumption, it has exploitability of strictly more than δ within the endgame, which leads to a contradiction. □

Intuitively, Proposition 2 says that endgame solving produces strategies with low exploitability in games where the endgame is a significant strategic portion of the full game, that is, in games where any endgame strategy with high full-game exploitability can be exploited by the opponent by modifying his strategy just within the endgame.

One could classify different games according to how they fall regarding the premise of Proposition 11, given a subdivision of the game into a trunk and endgames, and given fixed strategies for the trunk. If the premise is satisfied, then we can say that the game/subdivision satisfies the (ϵ, δ)-*endgame property*. An interesting property would be the smallest value $\epsilon^*(\delta)$ such that the game satisfies the (ϵ, δ)-endgame property for a given δ. For instance, the game in Figure 10.2 would have $\epsilon^*(\delta) = \delta$ for all $\delta \geq 0$, while RPS would only have $\epsilon^*(\delta) = 1$ for each $\delta \geq 0$. While Proposition 11 is admittedly somewhat trivial, such a classification could be useful in developing a better understanding of when endgame solving would be helpful in general.

10.2 Benefits of endgame solving

Even though we showed in the previous section that endgame solving may lead to highly exploitable strategies in some games, it has many clear benefits in large imperfect-information games, which we now describe. These benefits and techniques are *enabled by* using endgame solving (rather than being techniques that help alongside endgame solving).

10.2.1 Exact computation of Nash equilibrium in abstracted endgames

The best algorithms for computing approximate equilibria in large games of imperfect information scale to about 10^{17} nodes. However, they are iterative and guarantee convergence only in the limit; in practice they only produce approximations of equilibrium strategies (within a given abstraction). Sometimes the approximation error is quite large. For example, a recent nlhe agent reported having an exploitability of 800 milli big blinds per hand (mbb/h) even within the abstract game [33] (an agent that folds every hand would only have an exploitability of 750 mbb/h). The best general-purpose LP algorithms find an exact equilibrium, though they only scale to games with around 10^8 nodes [42]. While the LP algorithms do not scale to reasonable abstractions of full TH, we can (and do) use them to exactly solve abstracted endgames that have up to around 10^8 nodes.[2]

10.2.2 Ability to compute certain equilibrium refinements

The Nash equilibrium (NE) solution concept has some theoretical limitations, and several equilibrium refinements have been proposed which rule out NEs that are not rational in various senses. In general, these solution concepts guarantee that we behave sensibly against an opponent who does not follow his prescribed equilibrium strategy (i.e., he takes actions that should be taken with probability zero in equilibrium). Specialized algorithms have been developed for computing many of these concepts [81, 83, 84]. However, those algorithms do not scale to large games. In TH, computing a reasonable approximation of a single Nash equilibrium already takes months (using the leading algorithms, CFR or EGT), and no algorithms have been published in the literature that compute any of the common refinements that scale to games of that size (though based on personal communication with Michael Bowling, there may be modifications of CFR that compute approximations of some of these). However, when solving endgames that are significantly smaller than the full game, it can be possible to compute certain refinements. An undominated Nash equilibrium (UNE) can be computed by solving two LPs instead of one and an ϵ-quasi-perfect-equilibrium by solving a single LP (though the second one is not technically

[2]I note that the distinction between "exact" and "approximate" equilibrium computation is not quite as clearcut as this section has suggests. Barrier methods for a linear program have error, and historically their use in games has not included an examination of the extent of this error. On the other hand, the excessive gap technique (EGT) has such a fast convergence rate that after a polynomial number of iterations it is guaranteed to reach error below the rational number precision needed for the solution, thus giving an exact solution. Based on personal communication with several people I suspect that the LP outperforms the other approaches on sufficiently small games, but this has not been thoroughly investigated and hopefully future work will do so. I thank Michael Bowling for pointing out this clarification.

a refinement and has documented numerical stability issues). We have implemented algorithms for computing both of these on large no-limit Texas hold 'em endgames, which demonstrates for the first time that they are feasible to compute in imperfect-information games of this magnitude. Preliminary experiments indicate that in UNE is useful is no-limit Texas hold 'em, though those results were not statistically significant; these are described in Section 10.7.

10.2.3 Finer-grained, history-aware, and strategy-biased abstraction

Another important benefit of endgame solving in large games is that we can compute better abstractions in the endgame that is actually played than if we are forced to abstract the entire game at once in advance. In addition to allowing us to compute finer-grained abstractions, endgame solving enables us to compute an abstraction specifically for the situation at hand. In other words, we can condition the abstraction on the path of play so far (both the players' actions and nature's actions). For example, in poker, we can condition the abstraction on the betting history (which offline game-solving approaches do not do) and on the board cards (which offline game-solving approaches cannot afford to do at an equally fine granularity).

The standard approach for performing information abstraction is to bucket information sets together for hands that perform similarly against a uniform distribution of the opponent's private information [42, 68].[3] However, the assumption that the opponent has a hand uniformly at random is extremely unrealistic in many situations; for example, if the opponent has called large bets throughout the hand, he is unlikely to hold a very weak hand. Ideally, a successful information abstraction algorithm would group hands together that perform similarly against the relevant distribution of hands the opponent actually has—not a naïve uniform random distribution. Fortunately, we can accomplish such *strategy-biased information abstraction* in endgames. Our algorithm is detailed in Section 10.3.

10.2.4 A solution to the off-tree problem

When we perform action abstraction, the opponent may take an action that falls outside of our action model for him. When this happens, an *action translation mapping* (aka *reverse mapping*) is necessary to interpret his action by mapping it to an action in our model [34, 103]. However, this mapping may ignore relevant game state information. In poker, action translation works by mapping a bet of the opponent to a 'nearby' bet size in our abstraction; however, it does not account for the size of the pot or remaining stacks. For example, suppose remaining stacks are 17,500, the pot is 5,000, and our abstraction allows for bets of size 5,000 and 17,500. Suppose the opponent bets 10,000, which we map to 5,000 (if we use a randomized mapping, we will do this with some probability). So we map his action to 5,000, and simply play as if he had bet 5,000. If we call his bet, we will think the pot has 15,000 and stacks are 12,500. However, in reality the pot has 25,000 and stacks are 7,500. These two situations are completely different and should be played very differently (for example, we should be more reluctant to bluff in the latter case because the opponent will be getting much better odds to call). This is known as the *off-tree*

[3]Recent work has also considered an approach where the opponent's preflop hands are first grouped into several buckets, then hands for the later rounds are grouped together if they perform similarly against each of the preflop buckets [68].

problem. Even if one is using a very sophisticated translation algorithm, one will run into the off-tree problem.[4]

When performing endgame solving in real time, we can solve the off-tree problem completely. Regardless of the action translation used to interpret the opponent's actions prior to the endgame, we can take the stack and pot sizes (or any other relevant game state information) as inputs to the endgame solver. Our endgame solver in poker takes the current pot size, stack sizes, and prior distributions of the cards of both players as inputs. Therefore, even if we mapped the opponent's action to 5,000 in the above example, we correct the pot size to 25,000 (and the stack sizes accordingly) before solving the endgame.

10.3 Endgame solving algorithm

In this section we present our algorithm for endgame solving in imperfect-information games with very large state and action spaces. Pseudocode is given in Algorithm 13. The core algorithm is domain independent, although we present the signals as card-playing hands for concreteness. An example poker hand illustrating each step of the algorithm is given in Section 10.4.

Algorithm 13 Algorithm for endgame solving

Inputs: number of information buckets per agent k_i; abstraction parameter T; action abstractions B_i with b_i action sequences; clustering algorithms C_i; equilibrium-finding algorithm Q; number of private hands H; hand rankings $R[]$

```
Compute joint hand-strength distribution $D[i][j]$
$E_1, E_2 \leftarrow$ array of dimension $H$ of zeroes
for $h_1 = 1$ to $H$ do
    $r_1 \leftarrow R[h_1]$, $s_1, s_2 \leftarrow 0$
    for $h_2 = 1$ to $H$ do
        $r_2 \leftarrow R[h_2]$, $s_1$ += $D[h_1][h_2]$, $s_2$ += $D[h_2][h_1]$
        if $r_2 < r_1$ then
            $E_1[h_1]$ += $D[h_1][h_2]$, $E_2[h_1]$ += $D[h_2][h_1]$
        else if $r_1 == r_2$ then
            $E_1[h_1]$ += $\frac{D[h_1][h_2]}{2}$, $E_2[h_1]$ += $\frac{D[h_2][h_1]}{2}$
        end if
    end for
    $E_1[h_1] = \frac{E_1[h_1]}{s_1}$, $E_2[h_1] = \frac{E_2[h_1]}{s_2}$
end for
$k_i \leftarrow \lfloor \frac{T}{b_i} \rfloor$ for $i = 1, 2$
$A_i \leftarrow$ information abstraction created by clustering elements of $E_i$ into $k_i$ buckets using $C_i$ for $i = 1, 2$
Solve game with information abstractions $A_i$ and action abstractions $B_i$ using $Q$
```

[4]Some agents try to solve this problem by taking an action that is designed specifically to get us back "on-path"; however, this is not always desirable or even possible.

The first step is to compute the joint input distribution of private information using Bayes' rule. The naïve approach for doing this would require iterating over all possible private hand combinations h_1, h_2 for the players, and for each pair looking up the probability that the base agent would have taken the given action sequence. This requires $O(n^2)$ lookups to the strategy table, where n is the number of possible hands ($n = 1081$ for the final round in poker). It turns out that this computation would become the bottleneck of the entire endgame solving algorithm and would make real-time endgame solving computationally infeasible. For this reason, prior approaches for endgame solving have made the (significantly) simplifying assumption that the distributions are independent [42, 43]. However, we developed an algorithm that does this with just $O(n)$ table lookups. Pseudocode for our algorithm is given in Algorithm 14.

In short, the algorithm first computes the distributions separately for each player (as done by the independent approach), then multiplies the probabilities together for hands that do not share a common card (and setting the joint probability to zero otherwise). In order to make sure hands are indexed properly in the array, we must make use of two helper indexing functions, Algorithms 15 and 16. The former gives an algorithm for indexing the two-card private hands, and the latter gives an algorithm for indexing the 7-card river hand consisting of the two private cards and five public cards. Then, in Algorithm 14, we iterate over all sets of private hands (p_1, p_2), and create an array called IndexMap that maps the 7-card hand index to the corresponding 2-card hand index. In the course of this loop, we also look up the probability that each player would play according to the observed betting history in the precomputed trunk strategies, which we then normalize in accordance with Bayes' rule.

In advance of applying Algorithm 14, we compute a table of the conflicts between each pair of private-hand indices, where we set IC$[i][j]$ to 1 if hand with indices i and j share a card in common, and 0 otherwise. Then, we set the joint probability $D[i][j]$ to equal the product of the two independent probabilities $D_1[i], D_2[j]$ if there is no constraint between the indices, and we set it to zero otherwise. Note that this algorithm actually runs in $O(n^2)$, where n is the number of private hands. However, the n^2 loop only involves the simple step of looking up an element in the IC array, which is performed extremely quickly. The time-consuming part of the computation is looking up the strategy probabilities P_1, P_2, which involves accessing several elements in the massive binary strategy file. Our algorithm performs this task only $O(n)$ times, while the naïve approach would do this $O(n^2)$ time, and make real-time endgame solving intractable. (Note that each private hand consists of the two cards p_1, p_2, so while the main loop in Algorithm 14 iterates over both p_1 and p_2, it is only iterating once over the H private hands and is $O(n)$).[5]

Next we compute arrays E_1, E_2 that contain the *equities* for each state of private information against the opponent's distribution. For player 1, we do this by adding $D[h_1][h_2]$ to $E_1[h_1]$ for each hand h_2 such that the rank of it on the given board is lower than that of h_1, and adding $\frac{D[h_1][h_2]}{2}$ for each hand with equal rank.[6] We then normalize the entries of $E_1[h_1]$, and compute E_2 analogously. $E_1[h_1]$ is now the probability that player 2 has a hand worse than h_1, given the prior distribution D and the current history of betting and public cards.

[5]Note that a similar "inclusion-exclusion" trick was also helpful in reducing the running time of a crucial subroutine of an algorithm for best-response calculation from $O(n^2)$ to $O(n)$ to enable best-response computation in limit Texas hold 'em [66].

[6]The rank of a hand given a set of public cards is an integral-valued mapping such that stronger hands have a higher value; for example, a royal flush has the highest rank.

Algorithm 14 Algorithm for computing hand distributions

Inputs: Public board B; number of possible private hands H; betting history of current hand h; array of index conflicts IC[][]; base strategy s^*

```
$D_1, D_2 \leftarrow$ array of dimension $H$ of zeroes
for $p_1 = 0$ to $50$, $p_1$ not already on $B$ do
    for $p_2 = p_1 + 1$ to $51$, $p_2$ not already on $B$ do
        $I \leftarrow$ IndexFull($B, p_1, p_2$)
        IndexMap[$I$] $\leftarrow$ IndexHoles($p_1, p_2$)
        $P_1 \leftarrow$ probability P1 would play according to $h$
        with $p_1, p_2$ in $s^*$
        $P_2 \leftarrow$ probability P2 would play according to $h$
        with $p_1, p_2$ in $s^*$
        $D_1[I]$ += $P_1$, $D_2[I]$ += $P_2$
    end for
end for
Normalize $D_1$ and $D_2$ so all entries sum to 1
for $i = 0$ to $H$ do
    for $j = 0$ to $H$ do
        if !IC[IndexMap[i]][IndexMap[j]] then
            $D[i][j] \leftarrow D_1[i] \cdot D_2[j]$
        else
            $D[i][j] \leftarrow 0$
        end if
    end for
end for
Normalize $D$ so all entries sum to 1
return $D$
```

Algorithm 15 Algorithm for computing private hand index

Inputs: Private hole cards h_1, h_2

```
if $h_2 < h_1$ then $t \leftarrow h_1$, $h_1 \leftarrow h_2$, $h_2 \leftarrow t$
end if
return $\binom{h_2}{2} + \binom{h_1}{1}$
```

In advance of gameplay, we have computed separate action abstractions for the endgame solver to use for each pot/stack size that could be encountered. This allows us to solve the "off-tree problem," since we are taking into account the actual pot size even the opponent took an action outside the action abstraction earlier in the hand. We have constructed these abstractions so that the larger pot sizes (which have shallower stacks) have more bet sizes available for each history, for several reasons; the first is that the tree is smaller in these situations due to the shallower stack sizes (once players are "all-in," no additional bets are allowed), and the second is that hands with larger pot sizes are more important, since more money is won and lost on them,

Algorithm 16 Algorithm for computing index of 7-card hands on a given board

```
Inputs: Private hole cards $h_1, h_2$, board $B$ consisting of five public cards
  if $h_2 < h_1$ then, $t \leftarrow h_1, h_1 \leftarrow h_2, h_2 \leftarrow t$
  end if
  $n_1 \leftarrow 0, n_2 \leftarrow 0$
  for $i = 1$ to 5 do
      for $j = 1$ to 2 do
          if $B[i] < h_j$ then ++$n_j$
          end if
      end for
  end for
  return $\binom{h_2-n_2}{2} + \binom{h_1-n_1}{1}$
```

and we would like to ensure that more bet sizes are accounted for on these hands. B_i denotes the action abstraction to use for the given pot size at hand, and b_i denotes the number of betting sequences of B_i, for $i = 1, 2$.

Next, we compute a card abstraction A_i by grouping E_i into k_i buckets, using some clustering algorithm C_i, for $i = 1, 2$. Here $k_i = \frac{T}{b_i}$, where T is a parameter of the algorithm (for our agent we used $T = 7500$). While much prior work on poker has used k-means as the standard clustering algorithm, the following example demonstrates why this would be problematic. Suppose there are many hands with an equity of 0.775, and also many hands with an equity of 0.772. Then k-means would likely create separate clusters for these two equity values, and possibly group hands with very different equities (e.g., 0.2 and 0.3) together if few hands have those equities. To address this concern we used percentile hand strength, which also happens to be easier to compute. To do this, we break up the interval [0,1] into k_i regions of equal length (each of size $\frac{1}{k_i}$). We then group hand h_i into bucket $\lfloor \frac{E_i[h_i]}{k_i} \rfloor$. (For our poker agent we actually use a slight modification of this approach where we create a special bucket just for the hands with $E_i[h_i] \geq \alpha$, to ensure that the strongest hands are grouped separately (we used $\alpha = 0.99$ for our agent). Then the remaining α mass is divided according to the previously described procedure.) Sometimes this algorithm results in significantly fewer than k_i buckets, since there may be zero hands with E_i within certain intervals. We take this into account, and reduce the number of buckets in the card abstraction accordingly before solving the endgame. Note that the card abstractions A_i may be very different for the two players (and have different numbers of buckets); this differs from all the standard approaches, which use the same abstraction for all players.

Finally, we compute an (exact) equilibrium in the abstracted endgame by applying an equilibrium-finding algorithm Q to the game with card abstractions A_i and betting abstractions B_i. While the card abstractions were computed independently (based on equities derived from the joint distribution), we use the joint distribution for determining the probabilities that players are dealt hands from their respective buckets when constructing the endgame. For our agent, we used Gurobi's parallel LP solver [59] as Q. As discussed in Section 10.2.2, one could instead use an algorithm that is guaranteed to compute a certain equilibrium refinement.

Algorithm 17 Algorithm for computing endgame information abstractions

Inputs: Equity arrays E_i; desired number of buckets per agent k_i; parameter for top bucket α; total number of possible private hands H

```
$J \leftarrow \frac{\alpha}{k_1 - 1}$
$A_1 \leftarrow$ array of zeroes of size $H$
$U_1 \leftarrow$ array of booleans initialized to false of size $H$
for $h = 1$ to $H$ do
    if $E_1[h] \geq \alpha$ then $b \leftarrow k_1 - 1$
    else
        $b \leftarrow \lfloor \frac{E_1[h]}{J} \rfloor$
    end if
    if $U_1[b] ==$ FALSE then $U_1[h] \leftarrow$ TRUE
    end if
end for
$M_1 \leftarrow$ array of zeroes of size $k_1$, $g \leftarrow 0$
for $i = 0$ to $k_i$ do
    $M_1[i] \leftarrow g$
    if $U_1[i] ==$ TRUE then $g = g + 1$
    end if
end for
for $h = 1$ to $H$ do
    if $E_1[h] \geq \alpha$ then $A_1[h] \leftarrow M_1[k_1 - 1]$
    else
        $A_1[h] \leftarrow M_1 \left[ \lfloor \frac{E_1[h]}{J} \rfloor \right]$
    end if
end for
Compute $A_2$ analogously
```

10.4 Example

In this section we demonstrate the operation of our algorithm on an example hand of no-limit Texas hold 'em. Recall that blinds are \$50 and \$100 and that both players start with \$20,000. In the example hand, we are in the small blind with 8dTh. We raise to \$250, the opponent re-raises to \$750, and we call (there is now \$1500 in the pot). The flop is Jc6s2c. The opponent checks and we check. The turn is Kd. The opponent checks, we bet \$375, and he calls (there is now \$2250 in the pot). The river is Qc. Up until this point we have just played according to the precomputed base strategy; the endgame solving algorithm begins now.

According to the pseudocode for Algorithm 13, the first step is to compute the joint prior hand distribution D from the base strategies, using Algorithm 14. This took 0.433 seconds. We then compute the equities E_i for each player, using Algorithm 13. This took 0.015 seconds.

The next step is to look at the betting abstraction that has been precomputed for this specific pot/stack size (pot size of \$2250 and stack sizes of \$18875). Note that for this particular hand all of the opponent's actions before the river fell inside of our betting abstraction; however, if they had not, and we were forced to use an action translation mapping to map his action to an action in our betting abstraction, we would be able to correct our misperception of the pot size at this point, by selecting the precomputed betting abstraction for the actual pot/stack size (as opposed to the size that assumed he played an action in our betting abstraction). This solves the "off-tree" problem, discussed in the paper.

The betting abstraction for a pot size of \$2250 has 196 betting sequences for each player. For this hand we used a betting abstraction parameter of $T = 10000$ (while for the experiments described in the paper, we used $T = 7500$). Therefore, we will use $k_i = \lfloor \frac{10000}{196} \rfloor = 51$ card buckets for each player for this hand.

Next, we compute card abstractions for both players, using Algorithm 17. We used used a top bucket parameter of $\alpha = 0.995$ (while for the experiments described in the paper, we used $\alpha = 0.99$). After applying our card abstraction algorithm for both players, the resulting abstractions had 38 and 35 buckets respectively for the two players (since not all of the 51 hand equity intervals contained hands). Computing these took 0.008 seconds.

Our actual hand (8dTh) had rank 296 (out of 1081) and actually had an equity of 0 vs. the opponent's hand distribution (we thought the opponent would never play the hand the way he did so far with a worse hand than 8dTh). This places us in bucket 0 (the worst bucket, out of 35). By contrast, if the opponent had our hand, he would have an equity of 0.336 against our hand distribution, and would be in bucket 8 (where his buckets range from 0–37).

We then construct the LP matrices for the resulting abstracted endgame, which took 0.15 seconds, and then compute an exact equilibrium by solving the LP using Gurobi's parallel LP solver (it took 1.051 seconds to construct the LP instance and 5.328 seconds to solve it). If we were computing an undominated equilibrium, we would need to solve a second LP, using the procedure described in Section 10.7.1. Overall, the endgame solving algorithm took 6.985 seconds for this hand.

The opponent checked for his initial action on the river. The betting abstraction for this hand had nine available options for the first action for each player: check, 0.1 pot, $\frac{1}{3}$ pot, $\frac{2}{3}$ pot, pot, 1.5 pot, 2 pot, 3 pot, all-in. The strategy from our endgame solver said for us to check with probability 0.742, bet $\frac{2}{3}$ pot with probability 0.140, bet pot with probability 0.103, and bet 2 pot with probability 0.014. [7] We ended up betting $\frac{2}{3}$ pot and the opponent folded.

10.5 Experiments on no-limit Texas hold 'em

We tested our algorithm against the two strongest agents from the 2013 poker competition. The base agent was a version of the agent we submitted to the 2014 AAAI computer poker compe-

[7]We find it very interesting that the exact equilibrium strategies (for the given abstraction) involve so much randomization between different bet sizes. Even strong human players generally do not randomize between different bet sizes at a single information set (though they often bet different amounts at different information sets). It is very difficult to draw such conclusions about properties of exact equilibrium strategies from iterative algorithms for approximating equilibria, such as counterfactual regret minimization, since we are not sure whether the algorithm has fully converged at a given information set. Computing exact equilibria in endgames can give us more of a lens into properties of exact equilibrium strategies that could perhaps lead to improved game-solving algorithms, and furthermore are interesting on their own.

tition (that came in first place) from shortly before the competition. Ordinarily it would be very time consuming to differentiate the performance of the base strategies from the endgame solver with statistical significance, since the endgame solver plays relatively slowly (it averaged around 8 seconds per hand, which still kept us well within the competition time limit of 7 seconds per hand on average, since only around 25% of hands make it to the final betting round). A useful variance-reduction technique is to only consider hands where both agents make it to an endgame. In Section 10.6 we prove that this technique is unbiased. The results using this evaluation metric are given in Table 10.1, where the $\pm$ indicates 95% confidence intervals.

Hyperborean.iro	Slumbot
+87 $\pm$ 50	+29 $\pm$ 25

Table 10.1: Improvement by using endgame solving against the strongest agents from the 2013 poker competition over all hands where both agents made it to some endgame (i.e., to the river betting round). Units are milli big blinds per hand.

The base agent used a procedure called purification on all rounds (except for the first preflop action); this procedure selects the maximal probability action at each information set with probability 1 instead of randomizing according to the abstract equilibrium strategy (ties are broken uniformly at random) [38]. This parameter setting was shown to be the best in our thorough experiments in prior years, and we had used this as the standard setting when evaluating our base agent. The main motivation for purification is that it compensates for the failure of iterative equilibrium-finding algorithms to fully converge to equilibrium in the abstract game (a phenomenon that has been documented by prior agents, e.g., [33]). The endgame solving agent did not use any rounding for the river, as the endgame equilibria are exact (within the chosen abstraction), and the problem of the equilibrium-finding algorithm failing to converge is not present. Both agents used the pseudo-harmonic action translation mapping [34] for all rounds to interpret actions taken by the opponent that fall outside of the action abstraction.

The results are from 100 duplicate matches against Hyperborean and 155 duplicate matches against Slumbot. Since each match is 3,000 hands, this means we played 600,000 and 930,000 hands; out of these hands, both versions of our agent made it to the river round on 173,568 and 318,700 hands against the respective opponents. If we had used the standard duplicate approach for evaluating performance, we would not have been able to statistically differentiate the base agent from the endgame solver over this sample. However, we were able to obtain statistically significant results using our new evaluation approach. Additional results using a prior version of our algorithm against the strongest 2012 agents, which also include experiments for undominated equilibrium, are in Section 10.7.4.

10.6 Variance-reduction technique

When comparing the performance of one version of an agent A_1 to another version that is identical except that it plays differently on endgames A_2, one would like to take advantage of the fact that the agents play identically up until the endgames in order to evaluate the performance difference more efficiently. Ideally, we could play A_1 against a given opponent, and when the endgame is reached, evaluate how both A_1 and A_2 would do on that same endgame given the trunk history. However, such a technique is not possible on the poker competition test server. All that is allowed is to play A_1 and A_2 against an opponent for a full set of matches. The agents may reach endgames on different hands, or may reach different endgames even on the same hands (since both our agent and the opponent may be playing randomized strategies before the endgames).

One possible approach for reducing variance would be to only consider hands where both A_1 and A_2 arrive at the same endgame (the same betting history was played). It turns out that this approach is actually biased, so it cannot be applied to accurately measure performance. A second approach, that it turns out is unbiased, would be to only consider the hands where both agents arrive at *some* endgame (though not necessarily the same one). If we only consider these hands, then the difference in performance between the two agents is an unbiased estimator of their true performance difference. This would allow us to achieve statistical significance using a smaller sample of hands.

Proposition 12. *Let A_1 and A_2 be two algorithms that differ in play only for endgames. Then the difference in performance looking at only the hands where both make it to the same endgame is not an unbiased estimator of the overall performance difference.*

Proof. Suppose there were only two betting sequences and both make it to the river, where the first one (A) happens 99% of the time and the second one (B) happens 1% of the time. Then the probability that both hands hit the river with B on any particular hand is 0.01%, and the probability that both hands hit the river with A with any particular hand is 98.01%. So if you look at all hands where both hit the river with the same sequence, there would be only 1 (B) for every 9802 (A) sequences. □

Proposition 13. *Let A_1 and A_2 be two algorithms that differ in play only for endgames. Then the difference in performance looking at only the hands where both make it to some (but not necessarily the same) endgame is an unbiased estimator of the overall performance difference.*

Proof. For each history that leads into an endgame h_i, let p_i be the probability that h_i is played when we use the base strategy against the opponent O. Let $U(A_1, O, h_i)$ denote the expected payoff when following algorithm A_1 against an opponent who plays strategy O following history h_i (define $U(A_2, O, h_i)$ analogously). Then the expectation of the difference in payoff between playing A_1 (base strategy) and A_2 (endgame solver) against O is

$$\begin{aligned} & \sum_i \left[p_i \left(U(A_1, O, h_i) - U(A_2, O, h_i)\right)\right] \\ = \; & \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_i \left[p_i U(A_2, O, h_i)\right] \end{aligned}$$

Suppose that we look at performance over all hands where both algorithms make it to some endgame. The probability that A_1 makes it to the endgame with history h_i and A_2 makes it to the endgame with history h_j is $p_i p_j$. Thus, the expectation of the payoff difference is

$$\begin{aligned}
& \sum_i \sum_j \left[p_i p_j \left(U(A_1, O, h_i) - U(A_2, O, h_j)\right)\right] \\
= \;& \sum_i \sum_j \left[p_i p_j U(A_1, O, h_i)\right] - \sum_i \sum_j \left[p_i p_j U(A_2, O, h_j)\right] \\
= \;& \sum_i \left[p_i U(A_1, O, h_i) \sum_j p_j\right] - \sum_j \left[p_j U(A_2, O, h_j) \sum_i p_i\right] \\
= \;& \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_j \left[p_j U(A_2, O, h_j)\right] \\
= \;& \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_i \left[p_i U(A_2, O, h_i)\right]
\end{aligned}$$

□

10.7 Equilibrium refinements that can be integrated with endgame solving

As discussed in Section 10.2.2, the Nash equilibrium (NE) solution concept has some theoretical limitations, and several equilibrium refinement concepts have been proposed which rule out NE strategy profiles that are not rational in various senses. Common equilibrium refinements for extensive-form games of imperfect information include *undominated Nash equilibrium*, *perfect Bayesian equilibrium*, *sequential equilibrium*, *normal-form trembling hand perfect equilibrium*, *extensive-form trembling hand perfect equilibrium*, *normal-form proper equilibrium*, and *extensive-form proper equilibrium*. In general, these solution concepts guarantee that we behave sensibly against an opponent who does not follow his prescribed equilibrium strategy (i.e., he takes actions that should be taken with probability zero in equilibrium).

Specialized algorithms have been developed for computing many of these concepts [81, 83, 84]. However, those algorithms do not scale to large games. In Texas hold 'em, computing a reasonable approximation of a single Nash equilibrium already takes months (using the leading algorithms, CFR or EGT), and there are no known algorithms for computing any of the refinements listed above that scale to games of that size. However, when solving endgames that are significantly smaller than the full game, we will show that it can be possible to compute certain refinements. Specifically, we will consider undominated Nash equilibrium and ϵ-quasi-perfect equilibrium. (Note that while a quasi-perfect equilibrium is guaranteed to be a Nash equilibrium (in fact, it is a refinement of undominated Nash equilibrium), an ϵ-QPE is not, and technically it is not an equilibrium refinement.)

10.7.1 Undominated Nash equilibrium

An *undominated Nash equilibrium* (UNE) is a Nash equilibrium where no player's strategy is weakly dominated by another strategy.

Definition 2. *A strategy s^* for player i* weakly dominates *strategy s' if for all pure strategies s_{-i} for the opponent, $u_i(s^*, s_{-i}) \geq u_i(s', s_{-i})$, where the inequality is strict for at least one s_{-i}.*

In two-player games, a strategy is a UNE if and only if it is a trembling-hand perfect equilibrium (a popular equilibrium refinement).

It seems intuitive that we would never want to play a dominated strategy s', since we could guarantee at least as good a payoff if we played some other strategy s^* instead. However, standard equilibrium-finding algorithms may find an equilibrium that is dominated. One example of this phenomenon is the equilibrium strategy profile computed by Gordon for one-card poker, which used the LP formulation to find an equilibrium. This strategy profile, as well as the rules of one-card poker, are presented in Section 10.7.3.

In these equilibrium strategies, player 1 always checks initially with a 5–8 and bets with positive probability with a 2–4 and 9–A. In response to a bet by player 1, player 2 always folds with a 2–4, always calls with a 9–A, and calls with probability strictly between 0 and 1 with 5–8. In particular, player 2 calls with a 5 with probability 0.251, with a 6 with probability 0.408, with a 7 with probability 0.583, and with an 8 with probability 0.759. Denote this strategy by s'. Now suppose player 2 instead called with a 5 with probability 0.01 and called with an 8 with probability 1, while keeping all other probabilities the same. Denote this new strategy by s^*. Clearly s^* weakly dominates s' (it performs strictly better if player 1 decides to bet with some probability with a 5, 6, 7, or 8). It is also clear that s^* also constitutes an equilibrium strategy for player 2.

We can see that the strategies for player 1 are also dominated, as he calls with a 4 vs. a bet with positive probability, while he also folds with an 8 with positive probability (player 2 never bets with a 4–8).

While we did not compute the initial equilibrium strategies s', we are the first to observe that they are dominated.

It turns out that we can compute a UNE by the procedure described in Algorithm 18. The algorithm follows pretty straightforwardly from a previously-known result that any strategy that is a best response to a fully mixed strategy of the opponent is undominated [111]; however, we are unable to find the algorithm described explicitly in any published work. Furthermore, we are the first to investigate its usefulness in practice in large games.

Algorithm 18 Algorithm for computing an undominated equilibrium strategy for player i.

```
Input: game $G$, fully mixed strategy for the opponent $y$ (e.g., the uniform random strategy)
  1. Compute the value, $v_i$, of $G$ for player $i$ by computing a Nash equilibrium.
  2. Compute the best response of player $i$ to $y$ subject to the constraint that the worst-case
     expected payoff of agent $i$ is at least $v_i$.
```

Proposition 14. *Algorithm 18 computes an undominated equilibrium strategy for player i.*

Proof. Suppose the algorithm outputs a mixed strategy σ_i that is weakly dominated by another mixed strategy σ'_i. Then $u_i(\sigma'_i, s_{-i}) \geq u_i(\sigma_i, s_{-i})$ for all pure strategies s_{-i} for the opponent, where the inequality is strict for at least one s_{-i}. Let σ^m_{-i} be the fully mixed strategy for the opponent, and let $\sigma_{-i}^m(s_{-i})$ denote the probability that it selects pure strategy s_{-i}. Then

$$\begin{aligned} u_i(\sigma_i, \sigma_{-i}^m) &= \sum_{s_{-i}} \left[\sigma_{-i}^m(s_{-i}) u_i(\sigma_i, s_{-i})\right] \\ &< \sum_{s_{-i}} \left[\sigma_{-i}^m(s_{-i}) u_i(\sigma_i', s_{-i})\right] \\ &= u_i(\sigma_i', \sigma_{-i}^m). \end{aligned}$$

This contradicts the fact that σ_i is a best response to σ_{-i}^m. Thus σ_i is not weakly dominated by any strategy. □

The first step of Algorithm 18 can be accomplished using the LP formulation [72], while the second step can be accomplished by solving the LP formulation for computing a best response [72] with additional constraints ensuring that player i guarantees a worst-case expected payoff of at least v_i. To show how the second step works[8], we first present the LP formulation for computing a best response for player 1 to a given strategy y of player 2 in a two-player zero-sum extensive-form game of imperfect information, which is described in [72]:

$$\begin{aligned} \text{maximize}_x \quad & x^T A y \\ \text{subject to} \quad & x^T E^T = e^T \\ & x \geq 0 \end{aligned}$$

We modify this procedure as follows to compute a best response for player 1 to strategy y of player 2 subject to the additional constraint that the worst-case expected payoff is at least v_1, where v_1 is the value of the game to player 1 (and all matrices and vectors are as defined in [72]). We have already computed v_1 in Step 1 of Algorithm 18, and we assume it is an input to Step 2. The vector y is any fully-mixed strategy for player 2 (i.e., any strategy that puts positive weight on each action at each information set), and in our experiments we will be assuming it is the strategy that plays uniformly at random at each information set. The formulation for player 2 is analogous.

$$\begin{aligned} \text{maximize}_x \quad & x^T A y \\ \text{subject to} \quad & x^T E^T = e^T \\ & x \geq 0 \\ & x^T A \geq -qF \\ & q[0] = -v_1 \end{aligned}$$

[8]Note that the second step does not compute a full best response to y, but only a best response out of strategies that have worse-case expected payoff at least v_i (i.e., a best response to y out of equilibrium strategies).

Thus, overall this approach takes about twice as long as the standard approach for computing an NE, since it involves solving two LPs of approximately the same size instead of one. If we wanted a UNE strategy for both players, we could solve a third LP to obtain the strategy for the other player. In our experiments, we used the uniform random strategy as the fully mixed strategy of the opponent. Algorithm 18 scales to games with around 10^8 nodes in the game tree, since it is based on the LP formulation; however—unlike for finding just any NE—there are no known algorithms for UNE that would scale to larger games, such as the abstractions for full no-limit Texas hold 'em used by the strongest agents.

10.7.2 ϵ-Quasi-perfect equilibrium

Another solution concept that we considered is the ϵ-quasi-perfect equilibrium (ϵ-QPE). Informally, a player following a QPE takes observed as well as potential future mistakes of his opponents into account but assumes that he himself will not make a mistake in the future, even if he observes that he has done so in the past. The formal definitions of QPE and ϵ-QPE are fairly elaborate, and we omit them for brevity. But we note that a Nash equilibrium of a modified game $G(\epsilon)$ where we require the realization weight of any sequence σ of actions that can be played to be at least $\epsilon^{|\sigma|}$, where $|\sigma|$ is the number of actions in the sequence σ, is guaranteed to be an ϵ-QPE. This gives rise to an algorithm consisting of a single LP solve for computing an ϵ-QPE [84], making it potentially useful in practice. A full QPE can be computed in polynomial time as well, though the algorithm involves solving linear programs containing coefficients with a very large number of digits, and it is not practical [84]. Note that while a QPE is guaranteed to be a Nash equilibrium (in fact, it is a refinement of UNE), an ϵ-QPE is not, and technically it is not an equilibrium refinement.

ϵ-QPE has several potential advantages over UNE: it can be computed in one LP solve (rather than two), and QPE is a stricter solution concept than UNE. However, it also has several disadvantages. First, it uses an additional parameter ϵ, and it is not clear how that should be chosen. We can show that an ϵ-QPE is guaranteed to have exploitability at most $\epsilon|A||M|$, where $|A|$ is the maximum number of actions at an information set, and $|M|$ is the maximum absolute value of a payoff. However, this bound is not necessarily useful, as for any $\epsilon > 0$ and $\delta > 0$, we can construct a game that has an ϵ-QPE with exploitability exceeding δ (by increasing $|M|$ and/or $|A|$). This analysis indicates that ϵ may be required to be quite small in order to achieve a low exploitability in a given game. Another issue with ϵ-QPE is that the algorithm for computing it runs into numerical stability issues when ϵ is too small, since the $\epsilon^{|\sigma|}$ terms are extremely close to zero. This can result in standard LP solvers outputting solutions that are infeasible and/or suboptimal. We actually obtained worse performance when we tried solving for an ϵ-QPE within the endgames in preliminary experiments than when we solved for an NE or UNE. This suggests that the disadvantages outweigh the advantages.

10.7.3 One-card poker

One-card poker is a two-person zero-sum poker game, consisting of a thirteen-card deck and a single round of betting. In Section 10.7.1, we showed that the (exact) equilibrium strategies that have previously been computed for it using the standard LP approach are (weakly) dominated.

Thus, this game provides a demonstration that the undominated Nash equilibrium is a useful solution concept in realistic large imperfect-information games.

Here are the full rules of one-card poker:

- Two players: P1 and P2
- Both players ante \$1
- Deck containing thirteen cards: 2–A
- Each player is dealt one card uniformly at random
- P1 acts first and can either bet \$1 or check
    - If P1 bets, P2 can call or fold
        - If P1 bets and P2 calls, then whoever has the higher card wins the \$4 pot
        - If P1 bets and P2 folds, then P1 wins the entire \$3 pot
    - If P1 checks, P2 can bet \$1 or check.
        - If P1 checks and P2 bets, then P1 can call or fold.
            - If P1 checks, P2 bets, and P1 calls, then whoever has the higher card wins the \$4 pot
            - If P1 checks, P2 bets, and P1 folds, then P2 wins the \$3 pot
        - If P1 checks and P2 checks, then whoever has the higher card wins the \$2 pot

The equilibrium strategies computed by Gordon using the LP formulation are available at

`http://www.cs.cmu.edu/~ggordon/poker/`,

and are replicated in Tables 10.2 and 10.3.

Holding	2	3	4	5	6	7	8	9	T	J	Q	K	A
Bet initially	0.454	0.443	0.254	0.000	0.000	0.000	0.000	0.422	0.549	0.598	0.615	0.628	0.641
Call after check–bet	0.000	0.000	0.169	0.269	0.429	0.610	0.760	1.000	1.000	1.000	1.000	1.000	1.000

Table 10.2: Equilibrium strategy for player 1 in one-card poker, computed using the LP formulation. The first row is the probability of betting with each hand in the initial betting round, and the second row is the probability of calling after player 1 checks initially and player 2 bets.

Holding	2	3	4	5	6	7	8	9	T	J	Q	K	A
Bet vs. check	1.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000
Call vs. bet	0.000	0.000	0.000	0.251	0.408	0.583	0.759	1.000	1.000	1.000	1.000	1.000	1.000

Table 10.3: Equilibrium strategy for player 2 in one-card poker. The first row is the probability of betting with each hand after player 1 checks, and the second row is the probability of calling after player 1 bets.

10.7.4 Prior experiments against 2012 poker competition agents

We ran additional experiments on two-player no-limit Texas hold ’em using an agent that performed well in the 2012 poker competition as the base agent. These experiments used a prior version of our endgame solving algorithm that did not implement several of the features of our new algorithm (we will not describe full details of the prior approach here). Also note that the base agent for these experiments was different from the base agent in the experiments described in the main submission, and that our new variance-reduction technique was not used. We include these older results primarily because they present preliminary evidence that the undominated Nash equilibrium solution concept is useful in this domain, though results are not statistically significant.

The results (shown in Table 10.4) indicate that solving endgames led to improved performance against each of the strongest opponents from the 2012 computer poker competition. In some cases this improvement was quite dramatic; against the base agent and O2, our win rate improved by over 100 mbb/h. Furthermore, against each opponent, computing an undominated equilibrium for the endgames outperformed the approach of computing a potentially dominated one.

	Base	Vanilla endgame solver	Undominated endgame solver
Base	—	115 $\pm$ 35	120 $\pm$ 42
O1	-161 $\pm$ 36	-124 $\pm$ 44	-105 $\pm$ 59
O2	56 $\pm$ 38	214 $\pm$ 56	238 $\pm$ 76
O3	-102 $\pm$ 30	-48 $\pm$ 43	-39 $\pm$ 63
O4	165 $\pm$ 63	204 $\pm$ 58	273 $\pm$ 84

Table 10.4: No-limit Texas hold ’em results against the strongest competitors from the 2012 Annual Computer Poker Competition. The payoffs are given in milli big blinds per hand for the agent listed in the column, and the 95% confidence intervals are reported.

10.8 Summary and extensions

We demonstrated that endgame solving can be successful in practice in large imperfect-information games despite the fact that the strategies it computes is not guaranteed to constitute an equilibrium in the full game (which we showed). We also showed that endgame solving guarantees a low exploitability in certain games, and presented a framework that can be used to evaluate its applicability more broadly. We described several benefits of endgame solving in large imperfect-information games, including exact computation of Nash equilibria in abstracted endgames, the ability to compute certain equilibrium refinements, the ability to compute finer-grained, history-aware, and strategy-biased abstractions in endgames, and a solution to the off-tree problem. We presented an efficient algorithm for performing endgame solving in very large imperfect-information games and showed that it led to a significantly stronger performance against the strongest no-limit Texas hold ’em agents from the 2013 computer poker competition, utilizing a new variance-reduction technique that we described.

This work opens many interesting avenues for future research. We showed that endgame solving can produce strategies with high exploitability in certain games, while it guarantees low exploitability in others. It would be interesting to study where different game classes fall on this spectrum. It is possible that for interesting classes of games—perhaps even classes that include variants of poker—endgame solving is guaranteed to produce strategies with low exploitability. We would also like to study various subdivisions of a game into a trunk and endgames, to experiment on additional game classes, to experiment with the refinements we described, and to develop improved variance-reduction techniques.

Chapter 11

Computing Equilibria by Incorporating Qualitative Models

In this chapter we develop an additional new paradigm for solving large games. Rather than construct a smaller game using an abstraction algorithm, we propose solving an infinite approximation of the original game, then mapping the equilibrium of the infinite game to a strategy profile in the original game. Perhaps counterintuitively, it is often the case that the infinite approximation can be solved much more easily than the finite game. We show that sometimes very fine abstractions would be needed to match the solution quality of our approach.

Our main algorithmic innovation takes advantage of the fact that in many multiagent settings it is significantly easier to infer qualitative models of the structure of equilibrium strategies than it is to actually compute an equilibrium. For example, in (sequences of) take-it-or-leave-it offers, equilibria involve accepting offers above a certain threshold and rejecting offers below it [100]. Threshold strategies are also common in auctions (e.g., [11]) and in deciding when to make and break partnerships and contracts (e.g., [101]). In poker the hole cards are private signals, and in equilibrium, often the same action is taken in continuous regions of the signal space (e.g., [4]).

We develop an approach for exploiting such qualitative models in equilibrium finding. We study a broad class of imperfect-information games where players are given private signals at the start. We first consider the two-player (general-sum) case in which private signals are drawn independently from finite sets. For this case, we develop an algorithm based on a mixed integer linear feasibility program (MILFP) formulation that provably computes an equilibrium assuming we are given a "correct" qualitative model as input. The size of the program is polynomial in the parameters of the problem and the constraints are very sparse, suggesting that it can be solved quickly in practice. Our experiments confirm that the algorithm runs very fast on a simplified endgame of limit Texas hold 'em, leading to a significant performance improvement.

Next, we generalize our algorithm to computing equilibria in the following important extensions: many players, continuous private signal distributions, dependent private signal distributions, and multiple candidate qualitative models that satisfy a certain technical property (some of which can be incorrect). In most of these cases, we present the first algorithm that provably solves the class of games. We also develop new mixed-integer programming based algorithms for computing equilibria in general multiplayer normal and extensive-form games based on the extension of our initial algorithm to the multiplayer setting, which may be of independent interest.

The remainder of this chapter is organized as follows. In Section 11.1, we introduce continuous games and present relevant definitions and results. In Section 11.3, we present a continuous two-player game that is used to motivate the use of qualitative models and the setting of the remainder of the paper. Section 11.4 presents the main setting of our paper, and Section 11.5 introduces parametric (i.e., qualitative) models. Section 11.6 presents our main algorithm for solving large two-player games given a parametric model, as well as a proof of correctness of the algorithm. Section 11.7 presents several extensions of our main algorithm. Finally, Section 11.8 describes our experimental results: in Section 11.8.1 we present evidence that approximating large finite games with infinite games can outperform abstraction-based approaches; in Section 11.8.2 we demonstrate that our main algorithm leads to improved play in two-player limit Texas hold 'em; and in Section 11.8.3 we demonstrate the effectiveness of our approach in the multiplayer setting.

11.1 Continuous Games

Continuous games generalize finite strategic-form games to the case of (uncountably) infinite strategy spaces. Many natural games have an uncountable number of actions; for example, games in which strategies correspond to an amount of time, money, or space. One example of a game that has recently been modeled as a continuous game in the AI literature is computational billiards, in which the strategies are vectors of real numbers corresponding to the orientation, location, and velocity at which to hit the ball [5].

Definition 3. *A* continuous game *is a tuple $G = (N, S, U)$ where*

- *$N = \{1, 2, 3, \ldots, n\}$ is the set of players,*
- *$S = (S_1, \ldots, S_n)$ where each S_i is a metric space corresponding to the set of strategies of player i, and*
- *$U = (u_1, \ldots, u_n)$ where $u_i : S \to \mathbb{R}$ is the utility function of player i.*

The main result regarding the existence of a Nash equilibrium in continuous games is the following [28]:

Theorem 4. *Consider a strategic-form game whose strategy spaces S_i are nonempty compact subsets of a metric space. If the payoff functions u_i are continuous, there exists a (mixed strategy) Nash equilibrium.*

While this existence result has been around for a long time, there has been very little work on practical algorithms for computing equilibria in continuous games. One interesting class of continuous games for which algorithms have been developed is *separable games* [109]:

Definition 4. *A* separable game *is a continuous game with utility functions $u_i : S \to \mathbb{R}$ taking the form*

$$u_i(s) = \sum_{j_1=1}^{m_1} \cdots \sum_{j_n=1}^{m_n} a_i^{j_1 \ldots j_n} f_1^{j_1}(s_1) \ldots f_n^{j_n}(s_n),$$

where $a_i^{j_1 \ldots j_n} \in \mathbb{R}$ and the $f_i^j : S_i \to \mathbb{R}$ are continuous.

As we will see, this is a significant restriction on the utility functions, and many interesting continuous games are not separable. Additionally, algorithms for computing approximate equilibria have been developed for several other classes of infinite games, including simulation-based games [115] and graphical tree-games [106].

For a broad class of games, we will show that the equilibrium existence theorem above does not hold directly, but we can nevertheless prove the existence of an equilibrium by incorporating a qualitative equilibrium model. However, we show that these games are not separable, so the prior algorithm does not apply. These are the topics of the next two sections. After that, we will develop new algorithms for solving these games.

11.2 Measure theory background

Some additional math background is needed to prove some theoretical results in this chapter. Most of the definitions presented here are taken or adapted from [2].

Definition 5. *A nonempty family A of subsets of a set T is an* algebra *of sets if it is closed under finite unions and complementation. That is,*

$$G, H \in A \rightarrow \left[G \cup H \in A \textit{ and } G^C = T \backslash G \in A\right].$$

*A σ-*algebra *is an algebra that is also closed under countable unions. That is,*

$$\{G_i\} \subset A \text{ implies } \cup_{i=1}^{\infty} G_i \in A.$$

Definition 6. *A* measurable space *is a pair (T, Σ), where T is a set and Σ is a σ-algebra of subsets of T.*

Definition 7. *Let A_T and A_Y be nonempty families of subsets of T and Y, respectively. A function $f : T \rightarrow Y$ is (A_T, A_Y)-*measurable *if $f^{-1}(A)$ belongs to A_T for each A in A_Y. We say that f is A_T-*measurable *when A_Y is clearly understood, and we say that f is* measurable *when both A_Y and A_T are clearly understood.*

Definition 8. *Let Σ be a σ-algebra over a set T. A function μ from Σ to the extended real number line is called a* measure *if it satisfies the following properties:*

1.
$$\mu(E) \geq 0 \textit{ for all } E \in \Sigma.$$

2.
$$\mu(\emptyset) = 0.$$

3. For all countable collections $\{E_i\}$ of pairwise disjoint sets in Σ:

$$\mu\left(\cup_i E_i\right) = \sum_i \mu(E_i).$$

Definition 9. *A* measure space *is a triplet (T, Σ, μ), where Σ is a σ-algebra of subsets of T and $\mu : \Sigma \rightarrow [0, \infty]$ is a measure. If $\mu(T) = 1$, then μ is a* probability measure.

Definition 10. *Let (T, Σ) and (Y, Θ) be measurable spaces. A* Markov kernel *is a mapping $k : T \times \Theta \rightarrow [0, 1]$ satisfying the following two properties.*

1. *For each $t \in T$, the set function $k(t, \cdot) : \Theta \to [0, 1]$ is a probability measure.*
2. *For each $\theta \in \Theta$, the mapping $k(\cdot, \theta) : T \to [0, 1]$ is Σ-measurable.*

Definition 11. *Let (T, Σ) be a measurable space and let Y be a finite set. A* discrete Markov kernel *is a mapping $k : T \times Y \to [0, 1]$ satisfying the following two properties.*

1. *For each $t \in T$, the set function $k(t, \cdot) : Y \to [0, 1]$ is a probability mass function.*
2. *For each $y \in Y$, the mapping $k(\cdot, y) : T \to [0, 1]$ is Σ-measurable.*

Definition 12. *Let Σ_i be the set of all intervals of the form $(a, b), (a, b], [a, b), [a, b]$ for $a \leq b$ that are completely contained in X_i. (Note that X_i is defined in Definition 19). Then Σ_i is a σ-algebra, and (X_i, Σ_i) is a measurable space.*

Definition 13. *Let Λ_i denote the set of all discrete Markov kernels from (X_i, Σ_i) to C_i (note that we assume C_i is finite). We call Λ_i the* mixed strategy space *of player i.*

The following definitions and theorem were presented and proved in [109], and are used for the proof of Proposition 19 in Section 11.3.

Let Δ_i denote the space of Borel probability measures over $\hat{S}_i$, which we call the set of *mixed strategies* of player i. Let V_i denote the space of all finite-valued signed measures on $\hat{S}_i$.

Definition 14. *Two measures $\sigma_i, \tau_i \in V_i$ are* almost payoff equivalent *if $u_j(\sigma_i, s_{-i}) = u_j(\tau_i, s_{-i})$ for all $j \neq i$ and all $s_{-i} \in \hat{S_{-i}}$.*

Let 0 denote the zero measure in V_i, and define

$$Y_i = \{\text{measures almost payoff equivalent to 0}\}.$$

Definition 15. *The* rank *of a continuous game is the n-tuple $\rho = (\rho_1, \ldots, \rho_n)$ where $\rho_i = \frac{dim\ \Delta_i}{Y_i}$. A game has* finite rank *if $\rho_i < \infty$ for all i.*

Theorem 5. *A continuous game is separable iff it has finite rank.*

11.3 Motivating example

Consider the following simplified poker game [4]. Suppose two players are given private signals, x_1 and x_2, independently and uniformly at random from [0,1]. Suppose the pot initially has size ρ (one can think of this as both players having put in an ante of $\frac{\rho}{2}$, or that we are at the final betting round—aka final *street*—of a multi-street game). Player 1 is allowed to bet or check. If player 1 checks, the game is over and the player with the *lower* private signal wins the pot (following the convention of [4]). If player 1 bets, then player 2 can call or fold. If player 2 folds, then player 1 wins the pot. If player 2 calls, then whoever has the lower private signal wins $\rho + 1$, while the other player loses 1. This situation can be thought of as an abstraction of the final street of a hand of limit Texas hold 'em where raising is not allowed and player 2 has already checked.

It seems natural to define the strategy space S_1 of player 1 as the set of functions from $[0, 1]$ to $\{\text{check}, \text{bet}\}$ (i.e., to $\{0, 1\}$), and to define S_2 for player 2 as the set of functions from $[0, 1]$ to $\{\text{fold}, \text{call}\}$ (i.e., to $\{0, 1\}$). Let $p_i(a_1, a_2, x_1, x_2)$ denote the payoff of player i when the players play actions a_i and are given private signals x_i. Formally, p_i is defined as follows:

$$\begin{aligned}
p_1(1,0,x_1,x_2) &= \rho \\
p_1(0,a_2,x_1,x_2) &= \begin{cases} 0 & : \ x_1 > x_2 \\ \rho & : \ x_1 < x_2 \\ \frac{\rho}{2} & : \ x_1 = x_2 \end{cases} \\
p_1(1,1,x_1,x_2) &= \begin{cases} -1 & : \ x_1 > x_2 \\ \rho + 1 & : \ x_1 < x_2 \\ \frac{\rho}{2} & : \ x_1 = x_2 \end{cases} \\
p_2(a_1,a_2,x_1,x_2) &= \rho - p_1(a_1,a_2,x_1,x_2)
\end{aligned}$$

Given this definition of p_i, the the utility of player i under the strategy profile $s = (s_1, s_2)$ is defined as

$$u_i(s) = \int_{x_1=0}^{1} \int_{x_2=0}^{1} p_i(s_1(x_1), s_2(x_2), x_1, x_2) dx_2 dx_1.$$

We would like to represent each player's strategy set as a compact metric space, so that we can apply Theorem 4. Unfortunately, the naive representation does not yield compact metric spaces; so, we need to go through a number of transformations to achieve this goal. In particular, by iteratively eliminating dominated strategies, we arrive at a representation where each player's strategy space is isomorphic to a compact subset of a Euclidean space.

In order for $u_i(s)$ to be defined, we must restrict the strategy spaces S_i to consist of only the measurable functions. In addition, if we want to turn S_i into a metric space, we need to define a distance function. A natural distance function to use is the L_1 distance function: $d_i(s_i, s'_i) = \int_{X_i} |s_i(x_i) - s'_i(x_i)| dx_i$. Note that (S_i, d_i) does not quite define a metric space because the condition $d_i(s,t) = 0$ iff $s = t$ is not satisfied. To turn it into a metric space, we can let $\sim$ be the equivalence relation defined by $s \sim_i s'$ iff $d_i(s, s') = 0$. If we then let $\overline{S_i}$ equal the set of equivalence classes with respect to $\sim_i$, then $(\overline{S_i}, d_i)$ is a metric space.

Unfortunately, the metric space $(\overline{S_i}, d_i)$ is not compact, and we cannot apply Theorem 4 to guarantee the existence of an equilibrium.

Proposition 15. *The metric space $(\overline{S_i}, d_i)$ is not compact.*

Proof. We prove this by showing that the space is not totally bounded; that is, that we cannot cover the space with a finite number of ϵ-balls for at least one ϵ. Let $\epsilon = 0.5$, and let $\{f_1, \ldots, f_k\}$ be a finite set of elements of $\overline{S_i}$ which is claimed to be a cover. For each $j \in \{1, \ldots, k\}$, let $g_j = \{x : f_j(x) = 0\}$, and let $h_j = \{x : f_j(x) = 1\}$. By the measurability of f_j, both g_j and h_j must consist of the union of intervals of $[0, 1]$ of the following form: (a, b), $(a, b]$, $[a, b)$, $[a, b]$. Note that we can partition the interval $[0, 1]$ by $0 = y_0 < y_1 < \ldots < y_{m-1} < 1 = y_m$ such that $(y_{i'}, y_{i'+1})$ is completely contained in one of the intervals of either g_j or h_j for all i', j. Now consider the function f^* defined as follows: for each interval $[y_{i'}, y_{i'+1}]$, f^* equals 0 for the first half of the interval and equals 1 for the second half. Thenf^* will agree with each f_j for exactly half of the length of each interval. So $d_i(f^*, f_j) = 0.5 = \epsilon$ for all j; thus the set of ϵ-balls centered at $\{f_1, \ldots, f_k\}$ is not a cover and we are done. $\square$

Similarly, the space fails to be compact if we use other common distance functions, such as the discrete metric, any L_p metric, or L_∞. So we can not simply use one of those distance functions instead of L_1 to get around Proposition 15.

However, the following observations allow us to restrict our attention to a much smaller set of strategies.

Definition 16. *Let S_i be the set of pure strategies of player i. Then $s_i \in S_i$ is* weakly dominated *for player i if there exists a strategy $s_i^* \in S_i$ such that for all strategy profiles $s_{-i} \in S_{-i}$ for the other players, we have $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$.*

Definition 17. *Let $\overline{s_i}$ be an equivalence class of player i's pure strategies with respect to $\sim_i$. Then $\overline{s_i}$ is* weakly dominated *if s_i is weakly dominated for all $s_i \in \overline{s_i}$.*

Definition 18. *The equivalence class $\overline{s_i}$ is* undominated *if it is not weakly dominated.*

Proposition 16. *The equivalence class of strategies $\overline{s_2} \in \overline{S_2}$ of player 2 is undominated only if it contains a unique strategy of the following form: call if $x_2 \leq x_2^*$ and fold otherwise, for some x_2^*.*

Proof. First note that any equivalence class that contains a strategy of the desired form can only contain a single such strategy (since otherwise it would contain two strategies that do not have zero distance from each other). Now suppose $\overline{s_2}$ is undominated but does not contain any strategies of the desired form. Let $s_2 \in \overline{s_2}$ be arbitrary. Then there must exist an interval α_1 of one of the following forms with $a < b$: $(a,b), (a,b], [a,b), [a,b]$, and an interval α_2 of one of the following forms with $c < d$: $(c,d), (c,d], [c,d), [c,d]$, with $b < c$, where s_2 calls with all private signals in α_2 and folds with all private signals in α_1. Now define s^* to be the strategy that calls with all signals in α_1, folds with all signals in α_2, and agrees with s_2 otherwise. It is clear that s^* (weakly)-dominates s_2, and we have a contradiction. So all equivalence classes of strategies that are undominated must be of the desired form. □

We can remove all of the strategies from $\overline{S_2}$ that are dominated according to Proposition 16 from our consideration, forming a much smaller strategy space. In addition, we can remove the strategies not satisfying the threshold property given in the proposition from each equivalence class $\overline{s_2} \in \overline{S_2}$, thus turning the equivalence classes into singletons. In the remainder of our discussion, we will let $\overline{S_2}$ denote this smaller set.

We can now iteratively remove many strategies from S_1 by the following observation.

Proposition 17. *The equivalence class of strategies $\overline{s_1} \in \overline{S_1}$ of player 1 is a best response to some element of $\overline{S_2}$ only if it contains a unique strategy of the following form: bet if $x_1 \leq \underline{x_1^*}$, check if $\underline{x_1^*} \leq x_1 \leq \overline{x_1^*}$ and bet if $\overline{x_1^*} \leq x_1 \leq 1$, for some $\underline{x_1^*} \leq \overline{x_1^*}$.*[1]

Proof. The uniqueness part follows from the same reasoning as the proof of Proposition 16. Now suppose player 2 is playing the following strategy: call if $x_2 \leq x_2^*$ and fold otherwise. Let x_1 denote player 1's private signal.

- Case 1: $0 \leq x_1 \leq \frac{x_2^*}{2}$.
  Then the expected payoff of betting is

$$(1 - x_2^*)\rho + (x_2^* - x_1)(\rho + 1) - x_1 = (1 - x_1)\rho + x_2^* - 2x_1,$$

  and the expected payoff of checking is $(1 - x_1)\rho$. So player 1's best response is to bet.

[1] One can think of $[0, \underline{x_1^*}]$ as player 1's "value betting" range—where he bets hoping to get called by worse hands—and $[\overline{x_1^*}, 1]$ as his "bluffing" range—where he bets hoping to get better hands to fold.

- Case 2: $\frac{x_2^*}{2} \leq x_1 \leq x_2^*$.
  Then the formulas for the expected payoffs of betting and checking are the same as in Case 1. However, now the expected payoff of checking is larger, so player 1's best response is to check.
- Case 3: $x_2^* \leq x_1 \leq x_2^* \cdot \frac{\rho+1}{\rho}$.
  The expected payoff of betting is

$$(1 - x_2^*)\rho - x_2^* = \rho - \rho x_2^* - x_2^*,$$

  while the expected payoff of checking is $(1 - x_1)\rho$. So player 1's best response is to check.
- Case 4: $x_2^* \cdot \frac{\rho+1}{\rho} \leq x_1 \leq 1$.
  The formulas for the expected payoffs are the same as in case 3, but now player 1's best response is to bet.

□

After removing the dominated strategies, the new strategy space for player 1 becomes isomorphic to a compact subset of $\mathbb{R}^2$, and the new strategy space for player 2 becomes isomorphic to a compact subset of $\mathbb{R}$. Let $\hat{S}_1$ and $\hat{S}_2$ now refer to these new strategy spaces.

It turns out that the functions u_i are continuous in s for both players using the new strategy spaces, if we define the distance between two strategy profiles $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$ as $d(s, s') = d_1(s_1, s'_1) + d_2(s_2, s'_2)$.

Proposition 18. *For both players, the utility functions u_i are continuous in s.*

Proof. Let $\epsilon > 0$ be arbitrary, and let $\delta = \frac{\epsilon}{2\rho+2}$. Let $s, s' \in S$ be arbitrary, with $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$, and suppose that $||s - s'|| \leq \delta$. Note that

$$||s - s'|| = \int_{x=0}^{1} |s_1(x) - s'_1(x)|dx + \int_{x=0}^{1} |s_2(x) - s'_2(x)|dx$$

and $||u_1(s) - u_1(s')||$ equals

$$\left| \int_X \int_Y [u_1((s_1(x), s_2(y)), (x, y)) - u_1((s'_1(x), s'_2(y)), (x, y))]\, dydx \right|.$$

Note that the maximum possible value of the integrand is $2\rho + 2$. So

$$||u_1(s) - u_1(s')|| \leq$$

$$(2\rho + 2) \int_{x=0}^{1} \int_{y=0}^{1} I_1(s_1(x), s_2(y), s'_1(x), s'_2(y))dydx,$$

where $I_1(s_1(x), s_2(y), s'_1(x), s'_2(y))$ equals 0 iff $s_1(x) = s'_1(x)$ and $s_2(y) = s'_2(y)$ and equals 1 otherwise. With this definition, we have

$$I_1(s_1(x), s_2(y), s'_1(x), s'_2(y)) \leq |s_1(x) - s'_x(s)| + |s_2(y) - s'_2(y)|.$$

So we have

$$||u_1(s) - u_1(s')|| \le$$

$$(2\rho + 2)\int_X \int_Y [|s_1(x) - s_1'(x)| + |s_2(y) - s_2'(y)|]\, dydx$$

$$= (2\rho + 2)\left[\int_X |s_1(x) - s_1'(x)|dx + \int_Y |s_2(y) - s_2'(y)|dy\right]$$

$$= (2\rho + 2)||s - s'|| \le 2(\rho + 2)\delta = \epsilon.$$

The continuity of u_2 in s follows from the same argument, and we are done. □

It follows from Theorem 4 that the game has a Nash equilibrium using the new strategy spaces $\hat{S}_1$ and $\hat{S}_2$.

Now that the existence of an equilibrium is guaranteed, we must figure out how to compute one. It turns out that the game is not separable, so one cannot apply the algorithm from prior work [109].

Proposition 19. *This game is not separable.*

Proof. Suppose there exists a measure (not equal to 0) for player 2 that is almost payoff equivalent to 0; call this τ_2. (Note that 0 refers to always folding.) Let

$$k = \frac{3\int_0^1 x d\tau_2(x)}{4}.$$

Now let $s_1 \in \hat{S}_1$ be the strategy of always raising in $[k, 1]$ and always folding in $[0, k)$. Then when player 1 has private signal in $[k, 1]$, he will have a negative profit since he will raise and will win less than $\frac{1}{4}$ of the time when he is called. When player 1 has private signal in $[0, k)$, his payoff will be 0 since he will fold. So $u_1(s_1, \tau_2) < 0$. However, if player 2 plays 0, then player 1 will have positive profit in $[k, 1]$ since he will raise and player 2 will fold, and will have zero profit in $[0, k)$, since he folds. So $u_1(s_1, 0) > 0$. Therefore, $u_1(s_1, \tau_2) \neq u_1(s_1, 0)$, and τ_2 is not almost payoff equivalent to 0. So we have a contradiction, and must have $Y_2 = \emptyset$. So $\rho_2 = \infty$, and the game does not have finite rank. Therefore, by Theorem 5, it is not separable. □

However, it turns out that we can still solve this game quite easily if we notice that every equilibrium will have $\underline{x_1^*} \le x_2^* \le \overline{x_1^*}$. Given this guess of the form of an equilibrium, it is easy to compute an equilibrium by noting that a player must be indifferent between two actions at each threshold. For example, at $\underline{x_1^*}$ player 1 must be indifferent between betting and checking. His expected payoff of betting is $(1 - x_2^*)\rho + (x_2^* - \underline{x_1^*})(\rho + 1) - \underline{x_1^*}$ and his expected payoff of checking is $\rho(1 - \underline{x_1^*})$. Setting these two expressions equal to each other yields $x_2^* = 2\underline{x_1^*}$. We can create a linear equation similarly for the other two thresholds. Thus we have a system of n linear equations with n unknowns (where n is the total number of thresholds), which can be solved efficiently by matrix inversion.

11.4 Our setting

The main setting of the rest of the paper will be a generalization of the setting of the above example along several dimensions.

Definition 19. *A* continuous Bayesian game *is a tuple $G = (N, X, C, U, F)$ where*

- $N = \{1, 2, 3, \ldots, n\}$ *is the set of players,*
- $X = (X_1, \ldots, X_n)$ *where each X_i is a compact subset of $\mathbb{R}$ corresponding to the set of private signals of player i,*
- $C = (C_1, \ldots, C_n)$ *where each C_i is a compact subset of $\mathbb{R}$ corresponding to the set of actions of player i,*
- $U = (u_1, \ldots, u_n)$ *where $u_i : C \times X \to \mathbb{R}$ is the measurable utility function of player i, and*
- $F = (F_1, \ldots, F_n)$ *where $F_i : X_i \to [0, 1]$ is the cumulative distribution function (CDF) of the private signal distribution of player i (i.e., $F_i(x_i) = Pr(X_i \le x_i)$).*

The strategy space S_i of player i is the set of all measurable functions from X_i to C_i. In this paper we will assume that the sets C_i are finite.

Let Λ_i be the mixed strategy space of player i defined according to Definition 13. Let $\Lambda = \times_i \Lambda_i$. A *(mixed) strategy profile* is $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$, where $\sigma_i \in \Lambda_i$. Recall that $\sigma_i(x_i, c_i)$ denotes the probability that player i will take action c_i given private signal x_i.

Then define

$$\hat{u_i}(\boldsymbol{\sigma}, \mathbf{x}) = \sum_{c_1 \in C_1} \cdots \sum_{c_n \in C_n} \left[u_i(\boldsymbol{c}, \boldsymbol{x}) \prod_{j=1}^{n} \sigma_j(x_j, c_j) \right].$$

Define $u_{i|x_i}(\boldsymbol{\sigma})$, where $\boldsymbol{\sigma} \in \Lambda$, $x_i \in X_i$, as follows:

$$u_{i|x_i}(\boldsymbol{\sigma}) = \int_{X_1} \cdots \int_{X_{i-1}} \int_{X_{i+1}} \cdots \int_{X_n} \hat{u_i}(\boldsymbol{\sigma}, \boldsymbol{x}) f_n(x_n) \ldots f_{i+1}(x_{i+1}) f_{i-1}(x_{i-1}) \ldots f_1(x_1)$$

$$dx_n \ldots dx_{i+1} dx_{i-1} \ldots dx_1,$$

where $f_i(x_i) = \frac{d}{dx_i} F(x_i)$ is the probability density function of X_i. This denotes the expected utility of player i given that he has received private signal x_i when strategy profile $\boldsymbol{\sigma}$ is played.

For $i \in N$, $x_i \in X_i$, $\sigma_i \in \Lambda_i$ and $\sigma_{-i} \in \times_{j \neq i} \Lambda_j$, define $u_{i|x_i}(\sigma_i, \sigma_{-i})$ as the expected utility of player i given private signal x_i when he plays σ_i and the other players play σ_{-i}.

According to the definition of Nash equilibrium, player i can play arbitrarily in cases where he has a private signal that has zero measure in the signal distribution F_i. Such behavior can result in equilibria that violate our qualitative models (discussed later) even when "equivalent" equilibria exist that satisfy the models. Thus, we define a slightly stronger notion of equilibrium in order to rule out arbitrary behavior in these regions of measure zero. In other words, we require an agent to play rationally even if he gets a private signal that has zero probability. (This strengthening of the equilibrium concept is analogous to *perfect Bayesian equilibrium*, where each agent has to act rationally even in information sets that are reached with zero probability due to the strategies of the players. In our case the reaching with zero probability is due to nature's action, i.e., giving the agents types.)

Definition 20. *Strategy profile $\sigma \in \Lambda$ is an* every-private-signal Bayesian (EPSB) equilibrium *of G if for all $i \in N$, for all $x_i \in X_i$, and for all $\tau_i \in \Lambda_i$, we have $u_{i|x_i}(\sigma_i, \sigma_{-i}) \geq u_{i|x_i}(\tau_i, \sigma_{-i})$.*

Proposition 20. *Let G be a game of the form given in Definition 19. Then G has an EPSB equilibrium if and only if it has an equilibrium.*

Proof. Consider some equilibrium σ. For $i \in N$, let

$$Z_i = \{x_i \in X_i : u_{i|x_i}(\sigma_i, \sigma_{-i}) \neq \arg\max_{\tau_i \in \Lambda_i} u_{i|x_i}(\tau_i, \sigma_{-i})\}.$$

Since σ is an equilibrium, Z_i must have measure zero for all i. Now suppose for all $i \in N$ and all $x_i \in Z_i$, player i plays $\sigma'_i \in \arg\max_{\tau_i \in \Lambda_i} u_{i|x_i}(\tau_i, \sigma_{-i})$ and plays σ_i otherwise. Call the new profile γ. Then γ and σ differ only on a set of measure zero, and thus γ is an equilibrium. Since each player is playing a best response given each possible private signal under γ, γ is an EPSB equilibrium.

The other direction is trivial because every EPSB equilibrium is also an equilibrium. □

We now strengthen the notions of best response and dominated strategies analogously. These will be useful when we analyze our algorithms.

Definition 21. *Strategy σ_i is an EPSB best response for player $i \in N$ to profile σ_{-i} if for all $x_i \in X_i$ and for all $\tau_i \in \Lambda_i$, we have $u_{i|x_i}(\sigma_i, \sigma_{-i}) \geq u_{i|x_i}(\tau_i, \sigma_{-i})$.*

Definition 22. *Strategy σ_i is an EPSB ϵ-best response for player $i \in N$ to profile σ_{-i} if for all $x_i \in X_i$ and for all $\tau_i \in \Lambda_i$, we have $u_{i|x_i}(\sigma_i, \sigma_{-i}) \geq u_{i|x_i}(\tau_i, \sigma_{-i}) - \epsilon$.*

Definition 23. *Strategy σ_i is EPSB-undominated if for all $\tau_i \in \Sigma_i$ there exist $x_i \in X_i, \sigma_{-i} \in \Sigma_{-i}$ such that $u_{i|x_i}(\sigma_i, \sigma_{-i}) > u_{i|x_i}(\tau_i, \sigma_{-i})$.*

11.5 Parametric models

In many multiagent settings, it is significantly easier to infer qualitative models of the structure of equilibrium strategies than it is to compute an equilibrium. The introduction gives several examples, including sequences of take-it-or-leave-it offers, certain auctions, and making or breaking partnerships and contracts. In general, we call the values that divide the different strategic regions *thresholds* (e.g., $\underline{x_1^*}$, $\overline{x_1^*}$, and x_2^* in the example above), and refer to the guess of the structure of an equilibrium defined by these thresholds a *parametric model*. Many additional examples of games that are solved by the procedure used in the above example appear in [4].

Definition 24. *A* parametric model *of game $G = (N, X, C, U, F)$ is a tuple $P = (T, Q, \prec)$ where*

- *$T = (T_1, \ldots, T_n)$, where T_i denotes the number of regions for player i,*
- *$Q = (Q_1, \ldots, Q_n)$, where Q_i is a sequence $\{q_{ij} : 1 \leq j \leq T_i\}$, where $q_{ij} \in C_i$ denotes the action taken by player i in his j'th region of the model (at a boundary the lower region's action is taken), and*
- *$\prec$ is a partial ordering over the set of tuples $(y_{ij}, y_{i'j'})$, where $y_{ij} \prec y_{i'j'}$ if we require that the lower threshold of player i's j'th region is less than or equal to the lower threshold of player i''s j''th region.*

We saw in Section 11.3 that restricting the strategy spaces of a game by forcing all strategies to conform to a specified parametric model can allow us to both guarantee the existence of an equilibrium and to actually compute one when neither of these could be accomplished in the original game by previously known techniques.

11.6 Our main algorithm

In this section we present our algorithm for computing an equilibrium given a parametric model. While parametric models associate a pure action for each interval of signals, this can be problematic when the probability of obtaining individual private signals is nonzero. In this case, our algorithm will actually output mixed strategies.

For now we assume the game is finite, has two players, and a single parametric model is specified. We will extend the algorithm to more general settings along each of these dimensions in Section 11.7.

11.6.1 Constructing a MILFP

Given a problem instance $G = (N, X, C, U, F)$ and a parametric model P, we first construct a mixed integer linear feasibility program (MILFP) that contains both integer and continuous variables. Since X_i is finite for all players, we assume without loss of generality that it is the set of integers from 1 to $\overline{n}$. Let $\{t_i\}$ denote the union of the sets of thresholds for both players under P. For each threshold t_i, we introduce two real-valued variables, x_i and y_i, where x_i corresponds to $F_1(t_i)$ and y_i corresponds to $F_2(t_i)$. For each threshold t_i and each integer $j \in [1, \overline{n}]$, we introduce an indicator (0–1 integer) variable, $z_{i,j}$, such that $z_{i,j} = 1$ implies $j - 1 \leq t_i \leq j$. So, overall we have $2|T| + |T||S|$ variables, where $|T|$ is the number of thresholds in P and $|S|$ is the number of private signals.

Indifference constraints As the example in Section 11.3 demonstrates, we want to obtain indifference between two actions at each threshold. Thus, we have $|T|$ equality constraints where each is of the form $f(x, y) = 0$ where f is linear in the x_i and y_i.

Threshold ordering constraints In order to guarantee that the solution conforms to the parametric model, we must add inequality constraints corresponding to the partial ordering $\prec$. If $t_i \prec t_j$, then we add the constraints $x_i \leq x_j$, $y_i \leq y_j$.

Consistency constraints Next, we require that for each i, x_i and y_i are consistent in the sense that there exists some value for t_i such that $F_1(t_i)$ corresponds to x_i and $F_2(t_i)$ corresponds to y_i. To accomplish this, we include indicator constraints of the following form for each i, j:

$$(z_{i,j} = 1) \Rightarrow (F_1(j-1) \leq x_i \leq F_1(j)) \tag{11.1}$$

$$(z_{i,j} = 1) \Rightarrow (F_2(j-1) \leq y_i \leq F_2(j)), \tag{11.2}$$

where we define $F_1(-1) = F_2(-1) = 0$. Stated explicitly, we add the following 4 linear inequalities for each $i \in [1, |T|], j \in [0, \overline{n}]$:

$$x_i - F_1(j-1)z_{i,j} \geq 0 \qquad x_i - z_{i,j}(F_1(j) - 1) \leq 1$$

$$y_i - F_2(j-1)z_{i,j} \geq 0 \qquad y_i - z_{i,j}(F_2(j) - 1) \leq 1$$

Finally, we must ensure that for each i, $z_{i,j} = 1$ for precisely one j (i.e., $t_i \in [j-1, j]$ for some j). We accomplish this by adding the equality constraint $\sum_{j=0}^{\overline{n}} z_{i,j} = 1$ for each i.

Thus, there are $O(|T||S|)$ consistency constraints, where $|S|$ is the number of private signals. There are more consistency constraints than constraints of any other type, and thus the MILFP has $O(|T||S|)$ total constraints.

11.6.2 Obtaining mixed strategies from the MILFP solution

Once we obtain the x_i and y_i by solving the MILFP, we must map them into mixed strategies of the game. Suppose player 1 is dealt private signal $z \in [1, \overline{n}]$ and consider the interval $I = [F_1(z-1), F_1(z)]$. Now define the intervals $J_i = [x_{i-1}, x_i]$ where we define $x_{-1} = 0$. Let O_i denote the overlap between sets I and J_i. Then player 1 will play the strategy defined by region i with probability $\frac{O_i}{\sum_i O_i}$. The strategy for player 2 is determined similarly, using the y_i and F_2.

11.6.3 Algorithm correctness

We are now ready to prove that our algorithm indeed yields an equilibrium.

Lemma 1. *Suppose* $0 \leq x_1 \leq \ldots \leq x_n \leq 1$ *and* $0 \leq y_1 \leq \ldots \leq y_n \leq 1$ *and there exists a* $k \in [1, n]$ *s.t.* $x_k \neq y_k$. *Then there exists a* $j \in [1, n]$ *s.t.* $x_j \neq y_j$ *and* $y_{j-1} \leq x_j \leq y_{j+1}$ *(where we define* $x_0 = y_0 = 0$ *and* $x_{n+1} = y_{n+1} = 1$*).*

Proof. Let

$$i' = \min_i \text{ s.t. } x_i \neq y_i.$$

- Case 1: $x_{i'} < y_{i'}$
  Suppose $x_{i'} < y_{i'-1}$. Then $x_{i'-1} < y_{i'-1}$ which contradicts the definition of i'. So $y_{i'-1} \leq x_{i'} < y_{i'}$ and we are done.
- Case 2: $x_{i'} > y_{i'}$
  Let

  $$z(i) = \max_j \text{ s.t. } x_j \leq y_i$$

  and let

  $$i^* = \min_{i > i'} \text{ s.t. } z(i) \geq i.$$

  Clearly i^* exists since $z(n+1) = n+1$.
  - Subcase 2.1: $i^* = i' + 1$.
    By assumption we have $x_{i^*-1} > y_{i^*-1}$. Suppose $x_{i^*-1} > y_{i^*}$. By definition of i^* we have $z(i^*) \geq i^*$ and therefore $x_{i^*} \leq y_{i^*}$. Since $x_{i^*-1} \leq x_{i^*}$, we have a contradiction. Thus $y_{i^*-1} < x_{i^*-1} \leq y_{i^*}$ and we are done.

- Subcase 2.2: $i^* > i' + 1$.
  Suppose that $x_{i^*-1} \leq y_{i^*-1}$. By definition of i^* we have $z(i^* - 1) < i^* - 1$. This implies that $x_{i^*-1} > y_{i^*-1}$ which gives a contradiction. Now suppose that $x_{i^*-1} > y_{i^*}$. We have $z(i^*) \geq i^*$ and therefore $x_{i^*} \leq y_{i^*}$ which gives a contradiction since $x_{i^*-1} \leq x_{i^*}$. So $y_{i^*-1} < x_{i^*-1} \leq y_{i^*}$ and we are done.

□

Theorem 6. *Suppose that for all $i \in N$ and for all $\sigma_{-i} \in \Lambda_{-i}$, all pure-strategy EPSB best responses of player i to σ_{-i} satisfy the given parametric model. Then our algorithm outputs an equilibrium.*

Proof. Suppose our algorithm outputs a non-equilibrium, s. Then there exists a player i and an EPSB best response σ_i to s_{-i} such that not all of the thresholds of σ_i and s_i are equal. Let $\{u_j\}$ denote the thresholds of σ_i and $\{v_j\}$ denote the thresholds of s_i. By Lemma 1 there exists a j such that $u_j \neq v_j$ and $u_{j-1} \leq v_j \leq u_{j+1}$. Denote by a_1 and a_2 the two actions that player i is indifferent between at v_j. Suppose a_1 is played at v_j under σ_i. Now define the strategy profile s'_i as follows: s'_i is identical to σ_i except with private signal v_j action a_2 is played. Clearly player i is playing an EPSB best response since σ_i is an EPSB best response and he is indifferent between a_1 and a_2 against s_{-i} at v_j. So s'_i is an EPSB best response of player i to s_{-i}; however, s'_i violates the parametric model. This contradicts the premise. □

The theorem directly implies the following corollary. In some settings it may be easier to check the premise of the corollary than the premise of the theorem.

Corollary 1. *Suppose all EPSB-undominated strategies follow the given parametric model. Then our algorithm outputs an equilibrium.*

11.7 Extensions to more general settings

In this section we describe several important extensions of our approach to more general settings.

11.7.1 Continuous private signal distributions

In this subsection we generalize the approach to continuous private signal distributions. If the CDFs F_i are continuous and piecewise linear, we only need to alter the consistency constraints. Suppose that $\{c_i\}$ denotes the union of the breakpoints of the CDFs of the F_i, and suppose $F_i(x) = a_{i,j}x + b_{i,j}$ for $c_{j-1} \leq x \leq c_j$ (where $c_{-1} = 0$).

Recall that for a given threshold t_k, we have introduced variables x_k and y_k in the MILFP such that t_k corresponds to $F_1(x_k)$ and $F_2(y_k)$. In the case of continuous CDFs, we actually have $t_k = F_1(x_k) = F_2(y_k)$. To achieve this, we will introduce new variables t_k corresponding to the actual thresholds, and include the following linear constraints for all k, j:

$$(z_{k,j} = 1) \to \left(t_k = \frac{x_k - b_{1,j}}{a_{1,j}} \right)$$

$$(z_{k,j} = 1) \to \left(t_k = \frac{y_k - b_{2,j}}{a_{2,j}} \right).$$

These constraints replace Constraints 11.1 and 11.2 from the discrete case.

Using the technique described in Section 11.7.3, we can also approximately solve the problem for continuous CDFs that are not piecewise linear by approximating them with piecewise linear functions.

11.7.2 Dependent private signals

Many common real-world situations have dependent private signal distributions. For example, in card games such as Texas hold 'em, the cards dealt to each player depend on what was dealt to the other players (if one player is dealt the ace of spades, another player cannot also have it).

We will assume the private signals are given by a joint probability density function (PDF) $h(x, y)$ (rather than independent CDFs F_1 and F_2 as before). Instead of having $|T|$ variables x_i and y_i corresponding to both players' CDFs at the different thresholds, if the private signals are dependent, we will use $|T|^2$ variables x_{ij}, y_{ij} where x_{ij} corresponds to $Pr(X \leq t_i | Y = t_j)$ and y_{ij} corresponds to $Pr(Y \leq t_j | X = t_i)$. We also need to add $|T|^2|S|^2$ indicator variables (where $|S|$ is the number of private signals) and corresponding consistency constraints using the given joint PDF $h(x, y)$. Thus, the overall size of the CSP is larger by approximately a factor of $|T||S|$.

To construct the new consistency constraints, we define the following functions q_1 and q_2:

$$q_1(x, y) \equiv Pr(X \leq x | Y = y) = \frac{\int_{-\infty}^{x} h(v, y) dv}{\int_{-\infty}^{\infty} h(v, y) dv}.$$

$$q_2(x, y) \equiv Pr(Y \leq y | X = x) = \frac{\int_{-\infty}^{y} h(x, v) dv}{\int_{-\infty}^{\infty} h(x, v) dv}.$$

We will replace Constraints 11.1 and 11.2 from the independent case with the following constraints:

$$(z_{i,j,k,m} = 1) \Rightarrow (q_1(k-1, m) \leq x_{i,j} \leq q_1(k, m))$$

$$(z_{i,j,k,m} = 1) \Rightarrow (q_2(k, m-1) \leq y_{i,j} \leq q_2(k, m)).$$

11.7.3 Many players

In games with more than two players, the indifference constraints are no longer linear functions of the variables (while all other constraints remain linear). With n players the indifference constraints are degree $n - 1$ polynomials. Therefore, there is a need to represent products of continuous variables, $x_i x_j$, using linear constraints only since we wish to model the problem as a MILFP.

Approximating products of continuous variables using linear constraints

In this subsection we describe how a modeling technique [10] can be applied to approximate the nonlinear functions by piecewise linear functions. First we define two new variables

$$\beta_1 = \frac{1}{2}(x_i + x_j)$$

$$\beta_2 = \frac{1}{2}(x_i - x_j),$$

noting that $\beta_1^2 - \beta_2^2 = x_i x_j$. To approximate $w_1 = \beta_1^2$, we select k breakpoints from the interval [0,1]—in our experiments we will use $q_i = \frac{i-1}{k-1}$, where $k(\epsilon)$ is a function of an input parameter ϵ. Next, we add the constraint

$$\sum_{i=1}^{k} \lambda_{1i} q_i^2 = w_1,$$

where the λ_{1i} are continuous variables. Next we add the constraint

$$\sum_{i=1}^{k} \lambda_{1i} q_i = \beta_1.$$

We also add the constraint

$$\sum_{i=1}^{k} \lambda_{1i} = 1,$$

where we also require that at most two adjacent $\lambda_{1i}'s$ are greater than zero (we accomplish this in the standard way of adding a binary indicator variable per segment and the appropriate constraints, called SOS2 constraints). Then if we define the variable u_{ij} to represent the product $x_i x_j$, we just need to add the constraint

$$u_{ij} = w_1 - w_2,$$

where w_2 and its additional constraints are defined analogously to w_1.

Finally, we replace each indifference equation $f(x) = 0$ with the inequalities $f(x) \leq \frac{\epsilon}{2}$ and $f(x) \geq -\frac{\epsilon}{2}$ where ϵ is an approximation parameter given as input to the algorithm.

Tying the accuracy of the approximation to the accuracy of the equilibrium

Suppose we select $k + 1$ breakpoints per piecewise linear curve, with

$$k \geq \sqrt{\frac{(\overline{T} + 2)^{(n-1)} M(n-1)}{\epsilon}}, \tag{11.3}$$

where $\overline{T}$ is the maximal number of thresholds of one of the parametric models for a player, M is the difference between the maximum and minimum possible payoff of the game, n is the number of players, and ϵ is an approximation parameter given as input to the algorithm.

Lemma 2. *Suppose we obtain piecewise linear approximation $\hat{f}(x)$ of $f(x) = x^2$ over $[a, b]$ using $k+1$ breakpoints $q_0, \ldots, q_k$ with $q_i = a + (b-a)\frac{i}{k}$. If $k \geq \frac{b-a}{2\sqrt{\epsilon}}$, then $|\hat{f}(x) - f(x)| \leq \epsilon$ for all $x \in [a, b]$.*

Proof. Let $x \in [a, b]$ be arbitrary and suppose $q_i \leq x \leq q_{i+1}$. Then we have

$$\hat{f}(x) = \hat{f}(q_i) + \frac{x - q_i}{q_{i+1} - q_i}(\hat{f}(q_{i+1}) - \hat{f}(q_i)).$$

It is clear that $\hat{f}(x) \geq x^2$, so consider the difference $g(x) = \hat{f}(x) - x^2$. Since $g(q_i) = g(q_{i+1}) = 0$, the derivative of g must be zero at its maximum over the range $[q_i, q_{i+1}]$. Note that

$$g'(x) = \frac{\hat{f}(q_{i+1}) - \hat{f}(q_i)}{q_{i+1} - q_i} - 2x.$$

Setting this equal to 0 yields

$$x^* = \frac{\hat{f}(q_{i+1}) - \hat{f}(q_i)}{2(q_{i+1} - q_i)} = \frac{q_{i+1}^2 - q_i^2}{2(q_{i+1} - q_i)} = \frac{q_{i+1} + q_i}{2}.$$

Thus the maximal value of g is

$$\begin{aligned} g(x^*) &= \hat{f}(q_i) + \frac{x^* - q_i}{q_{i+1} - q_i}(\hat{f}(q_{i+1}) - \hat{f}(q_i)) - (x^*)^2 \\ &= \left(\frac{b-a}{2k}\right)^2 \leq \epsilon. \end{aligned}$$

□

Theorem 7. *Suppose that for all $i \in N$ and for all $\sigma_{-i} \in \Lambda_{-i}$, all pure-strategy EPSB ϵ-best responses of player i to σ_{-i} satisfy the given parametric model. Furthermore suppose that the number of breakpoints satisfies Equation 11.3. Then our algorithm outputs an ϵ-equilibrium.*

Proof. By Lemma 2 we have

$$|\hat{f}(x) - f(x)| \leq \frac{\epsilon}{4(T+2)^{(n-1)}M(n-1)} \text{ for all } x \in [0, 1].$$

Suppose the indifference inequalities have the form $\hat{h}(x) \leq \hat{g}(x) + \frac{\epsilon}{2}$ and $\hat{g}(x) \leq \hat{h}(x) + \frac{\epsilon}{2}$. Note that each term of $g(x)$ and $h(x)$ is a polynomial of degree $n-1$ with coefficients of absolute value at most M. Also, note that there are at most $(T+2)^{(n-1)}$ such terms.

So $|\hat{g}(x) - g(x)| \leq \frac{\epsilon}{4}$ and $|\hat{h}(x) - h(x)| \leq \frac{\epsilon}{4}$. So by the triangle inequality,

$$|g(x) - h(x)| \leq |g(x) - \hat{g}(x)| + |\hat{g}(x) - \hat{h}(x)| + |h(x) - \hat{h}(x)| = \epsilon.$$

Suppose our algorithm outputs a non-equilibrium, s. Then there exists a player i and an EPSB best response σ_i to s_{-i} such that not all of the thresholds of σ_i and s_i are equal. Let $\{u_j\}$

denote the thresholds of σ_i and $\{v_j\}$ denote the thresholds of s_i. By Lemma 1 there exists a j such that $u_j \neq v_j$ and $u_{j-1} \leq v_j \leq u_{j+1}$. Denote by a_1 and a_2 the two actions that player i is ϵ-indifferent between at v_j. Suppose a_1 is played at v_j under σ_i. Now define the strategy profile s'_i as follows: s'_i is identical to σ_i except with private signal v_j action a_2 is played. Then player i is playing an EPSB ϵ-best response since σ_i is an EPSB best response and he is ϵ-indifferent between a_1 and a_2 against s_{-i} at v_j. So s'_i is a pure strategy EPSB ϵ-best response of player i to s_{-i}; however, s'_i violates the parametric model. This contradicts the premise. $\square$

For particular games, the number of breakpoints needed to obtain a desired ϵ can actually be far smaller. For example, if each indifference equation consists of the sum of at most T^* expressions, for $T^* < (\overline{T}+2)^{(n-1)}$, then we can replace $(\overline{T}+2)^{(n-1)}$ with T^* to create a tighter upper bound. Additional constant-factor improvements to Equation 11.3 will be described in detail in Section 11.8.3.

Additionally, even though the number of breakpoints in Equation 11.3 is exponential in the number of players, we can actually model the problem as a MILFP using a polynomial number (in n) of constraints and variables (i.e., using a number of constraints and variables that is logarithmic in the number of breakpoints). This is accomplished by a recently published way of modeling piecewise linear functions in a MIP [112]. (It uses a binary rather than unary encoding to refer to the pieces via indicator variables.)

New MIP algorithms for computing equilibria in normal and extensive-form games

It is worth noting that the modeling approach of Section 11.7.3 can be used to develop new algorithms for computing an ϵ-equilibrium in general-sum games with two or more players in both normal and extensive form. In particular, the *MIP Nash* algorithm for computing an equilibrium in two-player general-sum normal-form games [102] can be directly extended to a MIP formulation of multiplayer normal-form games which contains some nonlinear constraints (corresponding to the expected utility constraints). If we apply our approach using sufficiently many breakpoints, we can obtain an ϵ-equilibrium for arbitrary ϵ by approximating the nonlinear constraints by piecewise linear constraints. Additionally, we can represent the equilibrium-computation problem in multiplayer extensive-form games as a MIP if we write out the expected utility constraints separately on a per-information-set basis. This leads to a new algorithm for computing equilibria in multiplayer extensive-form games, an important class of games for which no algorithms for computing a Nash equilibrium with solution guarantees were known.

11.7.4 Multiple parametric models

Quite often it is prohibitively difficult to come up with one parametric model, P, that is correct, but one can construct several parametric models, P_i, and know that at least one of them is correct. This is the case for our experiments on Texas hold 'em in Section 11.8.2. This scenario could arise for several reasons; for example, often we can immediately rule out many parametric models because all strategies that satisfy them are dominated. We now generalize our approach to this situation.

We define the notion of model refinement in a natural way:

Definition 25. *$P = (T, Q, \prec)$ is a* refinement *of $P' = (T', Q', \prec')$ if for each $i \in N$, Q'_i is a (not necessarily contiguous) subsequence of Q_i.*

Definition 26. *P is a* US-refinement *of P' if Q'_i corresponds to a unique subsequence of Q_i for each i.*

For example, if $N = \{1\}$ and $Q'_1 = \{1, 2\}$ while $Q_1 = \{1,2,3,2\}$, then P is a refinement of P', but is not a US-refinement.

We now generalize our algorithm to the setting where the P_i have a common US-refinement P'. We first define an indicator variable ζ_i corresponding to each model. Next we replace each indifference constraint $f(x) = 0$ corresponding to model P_i by the following two inequalities, where K is a sufficiently large constant: $f(x) - K\zeta_i \geq -K$ and $f(x) + K\zeta_i \leq K$.

Next we add certain inequalities corresponding to the models P_i that differ from P'. For simplicity, we will demonstrate these by example. Suppose that, under P', player 1 plays action a_1 in his first region, a_2 in his second region, and a_3 in his third region. Suppose that in P_1 he plays a_1 in his first region and a_3 in his second region (recall that P' is a refinement of P_1). Then we must add two constraints that ensure that at the first threshold of P_1, both a_1 and a_3 are (weakly) preferred to a_2. In general, whenever actions of P' are omitted by a P_i, we must add constraints to the neighboring actions at their intersection ensuring that they are preferred to the omitted actions.

We also replace each order constraint $x_j - x_{j'} \leq 0$ corresponding to model P_i by $x_j - x_{j'} + K\zeta_i \leq K$. Finally, we add the equality $\sum_i \zeta_i = 1$ to ensure that only the constraints corresponding to one of the candidate parametric models are used in the solution.

The following theorem states that our approach is correct even in this setting where there are multiple parametric models, assuming they have a common US-refinement.

Theorem 8. *Let there be two players. Let $\{P_i\}$ be a set of parametric models with a common US-refinement. Suppose that for all $i \in N$ and for all $\sigma_{-i} \in \Lambda_{-i}$, all pure-strategy EPSB best responses of player i to σ_{-i} satisfy at least one of the P_i (not necessarily the same P_i). Then our algorithm outputs an equilibrium.*

Proof. Suppose P' is the common US-refinement of the P_i. Suppose our algorithm outputs a non-equilibrium, s. Then there exists a player i and an EPSB best response σ_i to s_{-i} such that not all of the thresholds of σ_i and s_i are equal. Since P' is a refinement of each P_i, σ_i must satisfy P'. Let $\{u_j\}$ denote the thresholds of σ_i and $\{v_j\}$ denote the thresholds of s_i.

- Case 1: There exists a v_j that differs from all of the u_j.
  Then v_j must lie in the interior of some region of σ_i—suppose action a_1 is played in this region. Suppose v_j separates actions a_2 and a_3. If $a_1 \neq a_2, a_3$, then our algorithm requires that a_2 and a_3 are both (weakly) preferred to a_1 at v_j. Suppose that player i plays σ_i except with private signal v_j plays a_2. Then this must also be a pure-strategy EPSB best response to s_{-i}. However, it contradicts the premise of the theorem because no P_i can be a strict refinement of P'.
- Case 2: For all v_j, there exists a u_k such that $u_k = v_j$.
  - Subcase 2.1: There exists a v_j such that $u_k = v_j$ and the action-regions that v_j separates are not both equal to the action regions that u_k separates.
    Suppose v_j separates a_1, a_2 while u_k separates b_1, b_2, and that $a_1 \neq b_1, b_2$. Then player

i can play σ_i except with private signal u_k play a_1. This must also be a pure-strategy EPSB best response to s_{-i}, since our algorithm ensures that a_1 is (weakly) preferred to all other actions at v_j. However, it contradicts the premise of the theorem because no P_i can be a strict refinement of P'.

- Subcase 2.2: For all v_j, there exists a u_k such that $u_k = v_j$ and v_j, u_k separate the same action-regions.
  Since not all of the thresholds of σ_i and s_i are equal, there must exist an additional u_k that does not equal any of the v_j. Suppose that u_k lies between v_j and v'_j, and suppose that action a is played in this region under s_i. By the premise of this subcase, a cannot be the action of either of the regions separated at u_k. Also by the premise of this subcase, a must be played both below and above u_k. However, this would mean that σ_i has two regions where a is played corresponding to the single region of s_i, which contradicts the fact that P' is a US-refinement of the P_i.

□

We can also obtain a result with an ϵ guarantee similar to Theorem 7 for the case of more than two players.

It is worth noting that the number of variables and constraints in the new MILFP formulation is still $O(|S||T|)$ (assuming a constant number of parametric models). Alternatively, we could have solved several MILFP's—one for each parametric model. While each MILFP would be smaller than the one we are solving, each would still have $O(|S||T|)$ variables and $O(|S||T|)$ constraints, and thus have the same size asymptotically as our formulation. This alternative formulation is also potentially effective, assuming we have access to several processors to run the threads in parallel.

11.8 Experiments

We now present results from several experiments that investigate the practical applicability of our algorithm and extensions, as well as the overall procedure of solving large finite games by approximating them by continuous games.

11.8.1 Approximating large finite games by continuous games

In Section 11.3 we saw an example of a game with infinite strategy spaces that could be solved by an extremely simple procedure (once we guessed a correct parametric model). If instead the set of private signals were the finite set $\{1, \dots, n\}$, then it is clear that as n gets large, the running time of computing an exact equilibrium of the game will get arbitrarily large; on the other hand, solving the infinite approximation as n goes to infinity will still take essentially no time at all, and we would expect the solution to the infinite game to be very close to the solution of the finite game. In this section we will consider a similar game and show that very fine-grained abstractions would be needed to match the solution quality of our approach.

Kuhn poker is a simplified version of poker that was one of the first games studied by game theorists [76]. It works as follows. There are two players and a deck containing three cards: 1,

n	50	100	150	200	250
$v(G_n)$	-0.0576	-0.0566	-0.0563	-0.0561	-0.0560
$\pi(\sigma_n)$	-0.0624	-0.0612	-0.0579	-0.0583	-0.0560

Figure 11.1: Worst-case payoff of playing the projection of the equilibrium of G_∞ ($\pi(\sigma_n)$) versus the value of G_n ($v(G_n)$).

2, and 3. Each player is dealt one card at random, and both players ante \$1. Player 1 acts first and can either check or raise by \$1. If player 1 raises, player 2 can either call—in which case whoever has the higher card wins the \$4 pot—or fold—in which case player 1 wins the entire \$3 pot. If player 1 checks, player 2 can check—in which case whoever has the higher card wins the \$2 pot—or bet. If player 1 checks and player 2 bets, player 1 can call—in which case whoever has the higher card wins the \$4 pot—or fold, in which case player 2 wins the \$3 pot.

Generalized Kuhn poker, G_n, has the same rules as Kuhn poker except that the deck contains n cards instead of 3. Define G_∞ to be the same as G_n except the players are both dealt a real number drawn uniformly at random from the unit interval $[0, 1]$. Informally, G_∞ is like the limit as n approaches infinity of G_n. It turns out that G_∞ has a relatively simple pure strategy Nash equilibrium that is derived in [4]. It can be computed by solving a system of six linearly independent indifference equations with six unknowns.

Once the infinite game, G_∞, has been solved, its solution can be projected down to a corresponding strategy profile in G_n: call this profile σ_n. We ran experiments for several settings of n. We compared the performance of σ_n against its nemesis to the value of the game to player 1 (i.e., how an optimal strategy performs): the results are summarized in Figure 11.1. The payoffs agree to four decimal points when $n = 250$.

Next, we considered different abstractions of G_{250} obtained by grouping consecutive private signals together. For each abstraction, we computed an equilibrium in the corresponding abstracted game, then determined the payoff of player 1's component of that equilibrium against its nemesis in the full game G_{250}. As Figure 11.2 shows, 125 buckets are needed to obtain

# buckets	2	5	10	25	50	125
payoff	-0.2305	-0.0667	-0.0593	-0.0569	-0.0562	-0.0560

Figure 11.2: Experiments for G_{250}.

agreement with the value of the game to four decimal places—something that σ_{250} accomplishes as we showed above. As n gets even larger, we would expect to require even more buckets in our abstraction to obtain a strategy with exploitability as low as that of σ_n. Thus we can potentially obtain a given level of exploitability with a much lower runtime with our projection approach, since the computation required by abstraction-based approaches increases dramatically as n increases, while solving G_∞ and projecting its solution down to G_n requires very little computation.

To put these results in perspective, the game tree for two-player limit Texas hold 'em has approximately 9.17×10^{17} states, while recent solution techniques can compute approximate equilibria for abstractions with up to 10^{10} game states (e.g., [46]). Thus the ratio of the number of states in the largest solvable abstraction to the number of states in the full game is approximately 10^{-8}. On the other hand, we saw in G_{250} that we require at least half of the number of states in the full game in our abstraction to compete with the solution generated by the infinite game (assuming we are restricting ourselves to uniform abstractions). Thus, it is conceivable that one can come up with an infinite approximation of Texas hold 'em (or one of its subgames) that results in less exploitable strategies than the current strategies obtained by abstraction-based approaches.

In the next section we conduct an investigation of the feasibility of applying the algorithm and extensions developed in this paper to large real-world games, using Texas hold 'em as a benchmark.

11.8.2 Two-player limit Texas hold 'em

We ran our algorithm on a game similar to the river endgame of a hand of limit Texas hold 'em. In this game, there is an initial pot of ρ, and both players are dealt a private signal from [0,1] according to piecewise linear CDFs F_1 and F_2. Player 1 acts first and can either check or bet \$1. If player 1 checks, then player 2 can check or bet; if he checks the game is over, and if he bets then player 1 can call or fold. If player 1 bets, then player 2 can fold, call, or raise (by 1). If player 1 bets and player 2 raises, then player 1 can either call or fold. Thus, our game is similar to Game 10 in [4], except that we do not assume uniform private signal distributions.

To obtain a wide range of interesting prior distributions, we decided to use the actual prior distributions generated by a high caliber limit Texas hold 'em player. Once the river card is dealt in Texas hold 'em, there is no more information to be revealed and the game becomes a 1-street game like the game described above (except that in limit Texas hold 'em the private signals are dependent[2], and up to three raises are allowed in each betting round). If we assume that the full strategies of both players are known in advance, then when the river card is dealt we can create a distribution over the possible 5-card hand rankings each player could have, given the betting history so far (e.g., the probability he has a royal flush, a full house with 9's over 7's, etc.).

In particular, we obtained the strategies from GS4—a bot that performed competitively in the 2008 AAAI computer poker competition. To test our algorithm, we created a new bot GS4-MIP that plays identically to GS4 on the first three streets, and on the river plays according to our algorithm. Specifically, we assume that both players' hand rankings on the river are distributed assuming they had been following the strategy of GS4 until that point; these determine the private signal distributions.

Given this game model, we developed three different parametric models that we expected equilibria to follow (depending on the private signal distributions at the given hand). This is noteworthy since [4] only considers a single parametric model for their game, and our experiments revealed that if we did not include all three models, our MILFP would sometimes have

[2]We do not expect the dependence to have a large effect in practice for this game due to the large number of possible private signals. In addition, we have developed an efficient extension of our MILFP to deal with the case of dependent private signals (see Section 11.7.2), which can be used if we expect dependence to have a significant effect.

no solution, demonstrating that all three models are necessary. The models are given in Figures 11.3– 11.5. It is easy to see that the first model is a US-refinement of the other two. To solve the MILFP, we used CPLEX's MIP solver on a single machine.

The first parametric model, shown in Figure 11.3, is identical to the model presented in [4] (with the thresholds renamed). For player 1, the action before the hyphen specifies the first action taken in the betting round, and the action after the hyphen specifies the next action taken if the betting gets back to player 1. For example, between thresholds b and c player 1 will check first, and if player 2 bets he will call. A *bluff* denotes a bet with a bad hand (where the player betting is hoping the other player folds). So with a private signal between d and 1 player 1 will bet, and he will fold if player 2 raises. For player 2, the first action listed denotes the action taken when player 1 bets, and the second action (after the slash) denotes the action taken when player 1 checks. For example, between f and g player 2 will call if player 1 bets and check if player 1 checks. The second parametric model, shown in Figure 11.4, is identical to the first model except that threshold i is shifted above threshold d. In the third parametric model (Figure 11.5), player 1 only checks when he is first to act (and never bets).

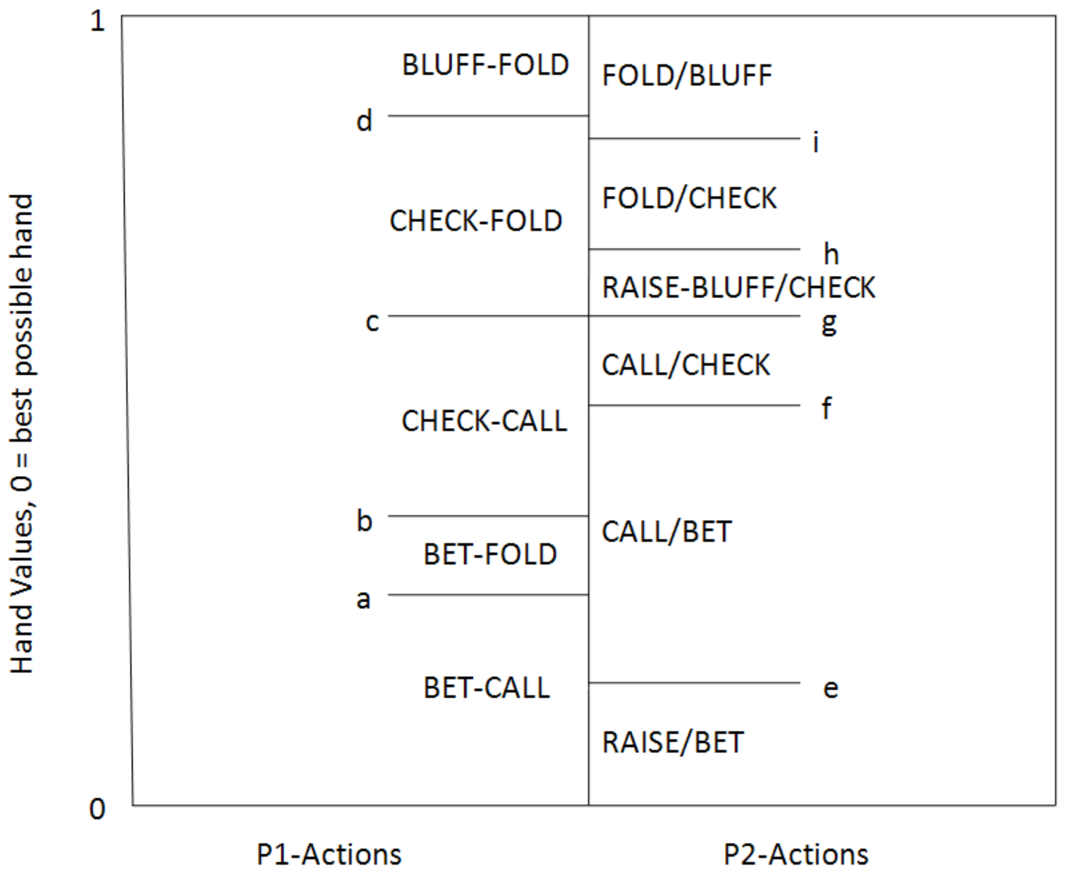


Figure 11.3: First parametric model.

Once we solved this simplified game, we used a very naive mapping to transform it to a

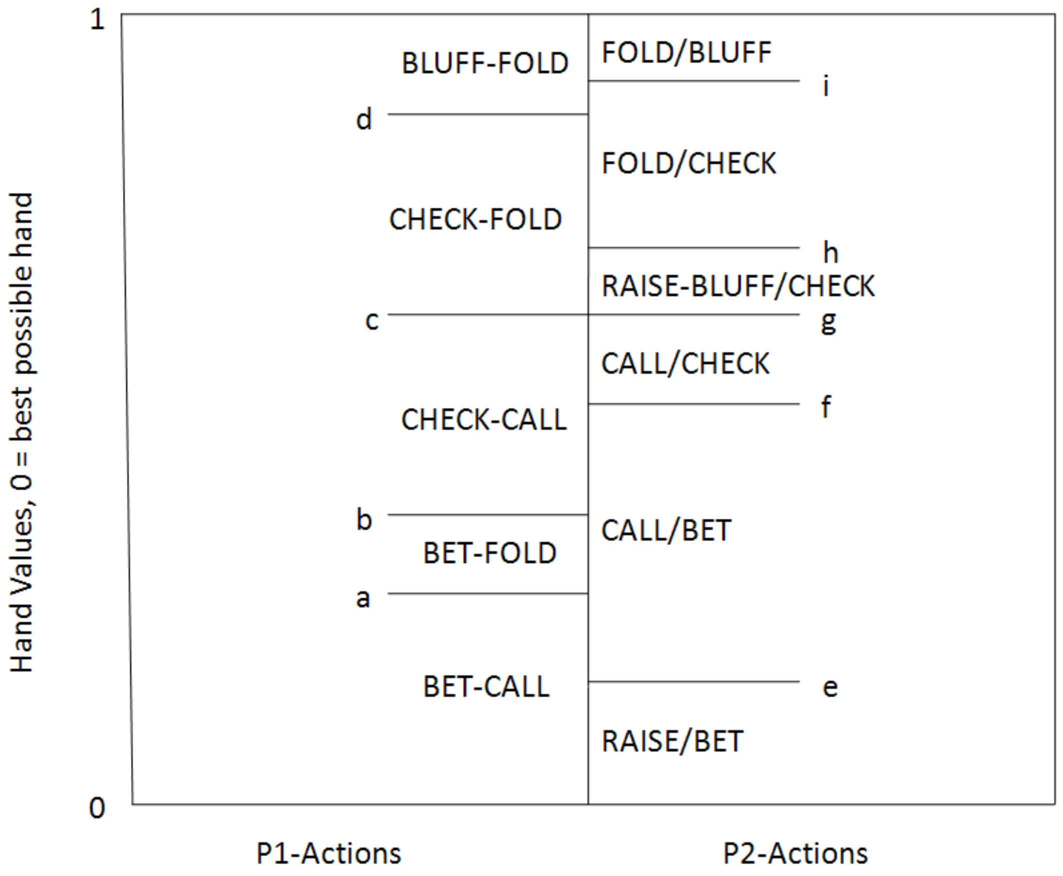


Figure 11.4: Second parametric model.

strategy in the full 3-raise game[3]. Since this mapping was so simple, we suspect that most of the success of the strategy was due to the solution computed by our algorithm.

We ran GS4-MIP against the top five entrants of the 2008 AAAI computer poker competition, which includes GS4. For each pairing, we used 20,000 duplicate hands to reduce the variance. GS4-MIP performed better against 4 of the 5 competitors than GS4 did. In the match between GS4-MIP and GS4, GS4-MIP won at a rate of 0.018 small bets per hand. This is quite significant since most of the top bots in the competition were separated by just 0.02–0.03 small bets per hand overall, and the only difference between GS4 and GS4-MIP is on the river street, which is reached only on some hands. Additionally, GS4-MIP averaged only 0.49 seconds of computation time per hand on hands that went to the river (and 0.25 seconds of computation per hand overall) even though it was solving a reasonably large MIP at runtime[4] (1,000–2,000 rows and several hundred

[3]For example, we assumed player 1 will put in a second raise with hands in the top half of player 2's raise range. We omit a full discussion of our transformation to the 3-raise game, since it is fairly tangential.

[4]The reason we need to solve the MIP at runtime is that we have to solve a different MIP for each betting sequence up until the river and each set of community cards (in the full game, not in the abstract game). Since there is such a large number of such subgames, it is much easier to just solve them quickly at runtime than to solve them

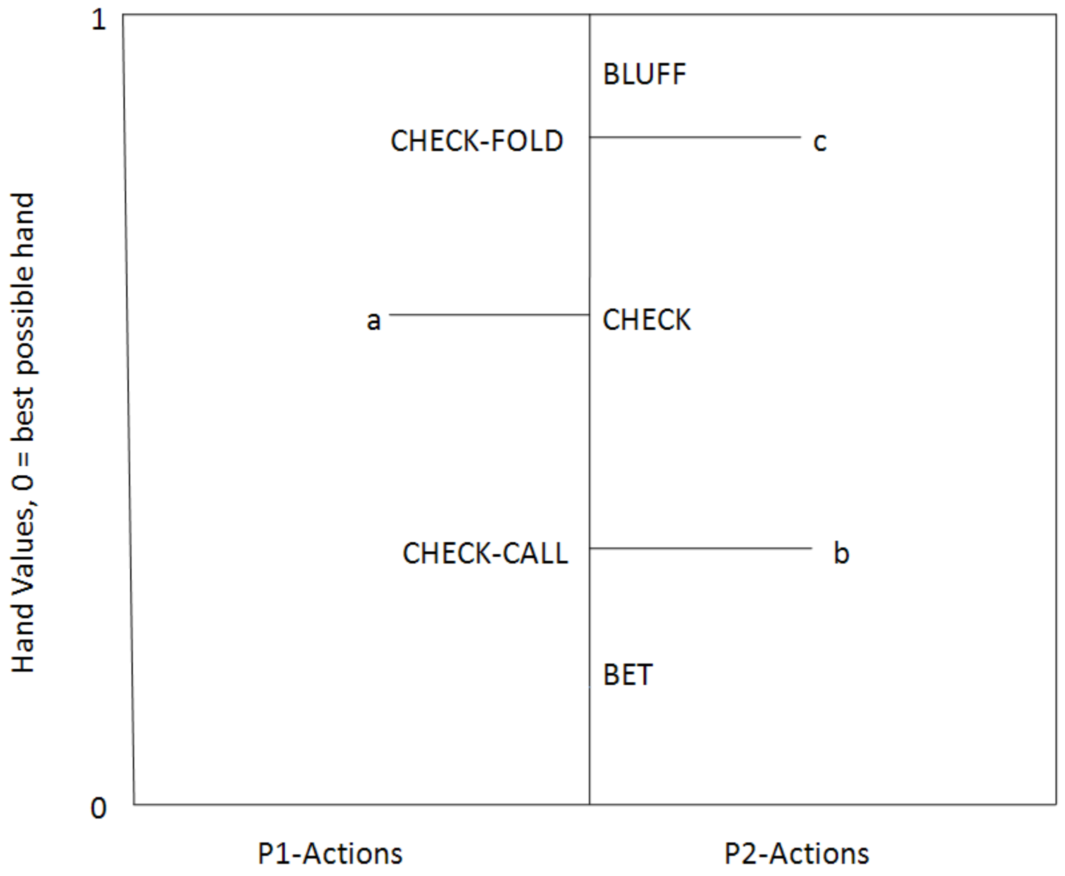


Figure 11.5: Third parametric model.

columns). The actual competition allows an average of 7 seconds per hand, so our algorithm was well within the time limit. Perhaps it is the sparsity of the constraints that enabled CPLEX to solve the problems so quickly, as the majority of the constraints are indicator constraints which only have a few non-zero entries.

It is worth noting that our algorithm is perhaps not the most effective algorithm for solving this particular problem; in the discrete case of actual Texas hold 'em, the river subgame can be formulated as a linear program (which can probably be solved faster than our MILFP). On the other hand, continuous games, two player general-sum games, and multiplayer games cannot be modeled as linear programs while they can be solved with our approach. Furthermore, the results in the previous subsection show that large finite two-player zero-sum games can sometimes be solved more effectively (both according to runtimes and quality of solutions) by approximating them by a continuous game that is easier to solve, than by abstracting the game and solving a smaller finite game.

all in advance.

11.8.3 Multiplayer experiments

To test the extensions of our algorithm to continuous distributions and multiple players, we ran our algorithm on the following simplified three-player poker game. The game has one betting round, and all players are given a private signal in $[0, 1]$. Player 2 initially has \$1 invested in the pot, while player 3 has \$2 invested in the pot (i.e., the small and big blinds). Player 1 is first to act and he can either fold or raise to \$4. If player 1 folds, then player 2 can fold or raise to \$4. If a player is facing a raise in front of him, then he can either call or fold. Thus, this game is an extension of the game in Section 11.3 to multiple players.

It is easy to see that all EPSB-undominated strategies will have the following form. If a player is first to enter the pot, he will raise with his better hands and fold with his worse hands (and never bluff). If a player is facing a raise ahead of him, he will call with his better hands and fold with his worse hands. Thus given each betting history, the parametric model for each player will just have a single threshold. The full parametric model is shown in Figure 11.6. For player 2, the first action listed denotes the action taken when player 1 raises, and the second action (after the slash) denotes the action taken when player 1 folds. For player 3, the first action listed denotes the action taken when player 1 raises and player 2 calls, the second action denotes the action taken when player 1 raises and player 2 folds, and the third action denotes the action taken when player 1 folds and player 2 raises.

We ran our algorithm on this game using a variety of continuous piecewise-linear cumulative distribution functions, and obtained rapid convergence to an ϵ-equilibrium for tiny ϵ for all games on which we experimented. In the remainder of this section, we will describe our results with uniform CDFs (i.e., each player is given his private signal uniformly at random from $[0, 1]$) in detail.

Figure 11.7 shows our experimentally-obtained values of ϵ, as well as the worst-case theoretical values of ϵ according to Equation 11.3. As noted in Section 11.7.3, the worst-case bound for k as a function of ϵ is very loose, and we expect to require a much smaller value of k to obtain a given ϵ in practice. Our results confirm this conjecture on this game. Figure 11.7(a) shows our experimental values of ϵ as a function of the number of breakpoints. We obtained an ϵ of 0.01 using just 5 breakpoints, and observed a rapid decrease of ϵ to about 10^{-5} as we increased the number of breakpoints to 50. Figure 11.7(b) shows the ϵ guaranteed according to Equation 11.3. In sharp contrast, an ϵ of almost 25 is guaranteed using 5 breakpoints, which is meaningless since the difference between the best and worst-case payoffs of the game is only 12. Even using 50 breakpoints only guarantees an ϵ of 0.24, which is also essentially meaningless for practical purposes. So our results confirm that we are in fact able to obtain good performance results in practice despite the fact that our worst-case theoretical bound is not very meaningful for such small numbers of breakpoints.

Conditional parametric model representation

In some cases, it can be beneficial to use an alternate, but equivalent, representation of parametric models. For example, in this game, rather than use a model for player 3 with three different thresholds, we could instead use three parametric models for player 3—where each one corresponds to a different nonterminal betting sequence of the other two players (e.g., raise/call,

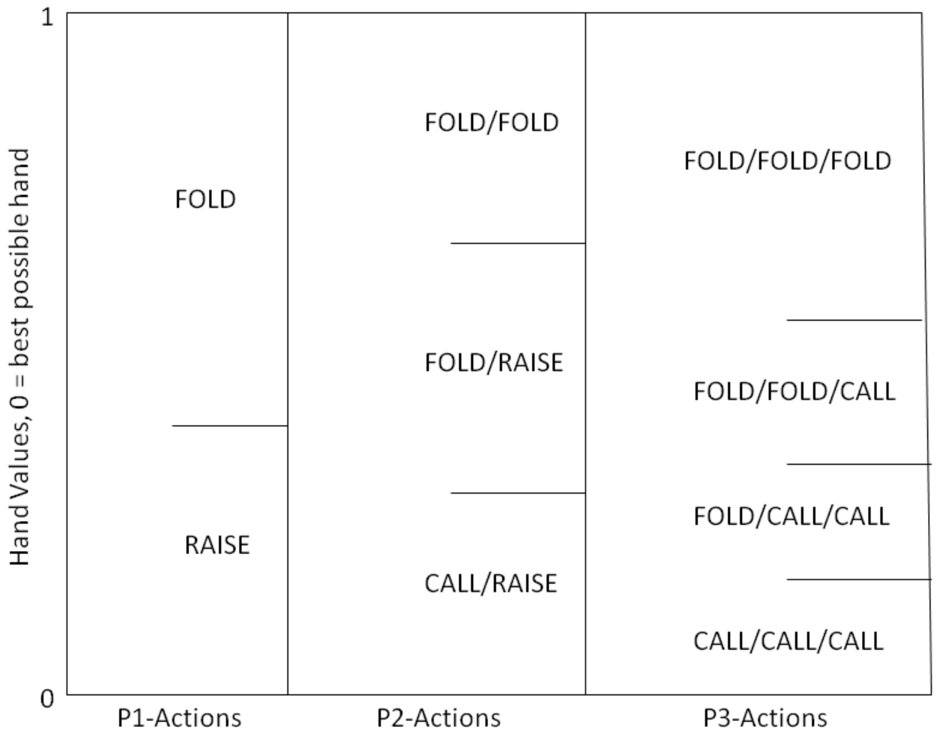


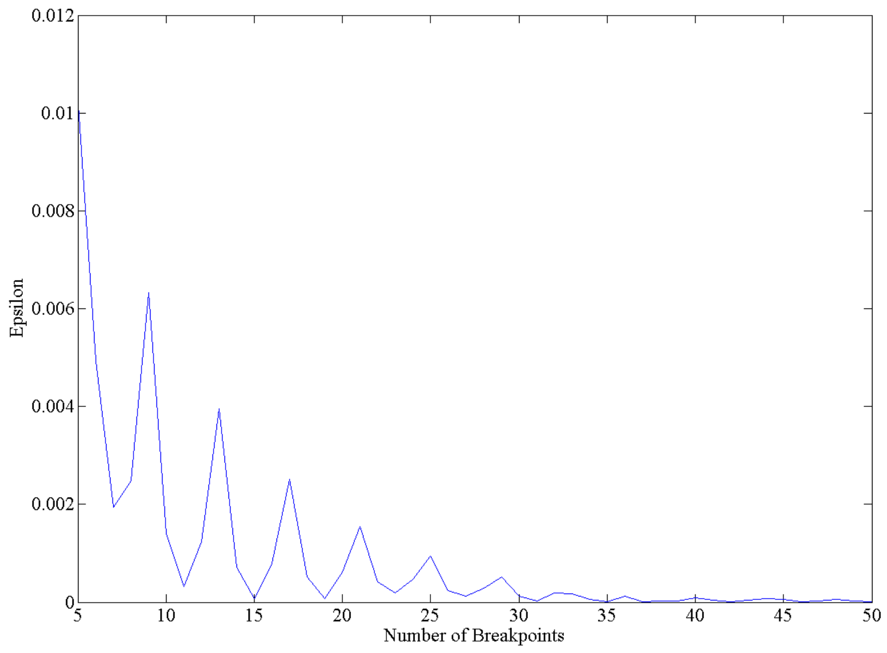
Figure 11.6: Parametric model for the three-player poker game.

raise/fold, or fold/raise)—see Figure 11.9. Then each of these parametric models would only have a single threshold, thus simplifying the representation of each model (though there are more of them). We call such a representation a *conditional parametric model*, due to the fact that a player's model is conditional on the action sequences of the other players.

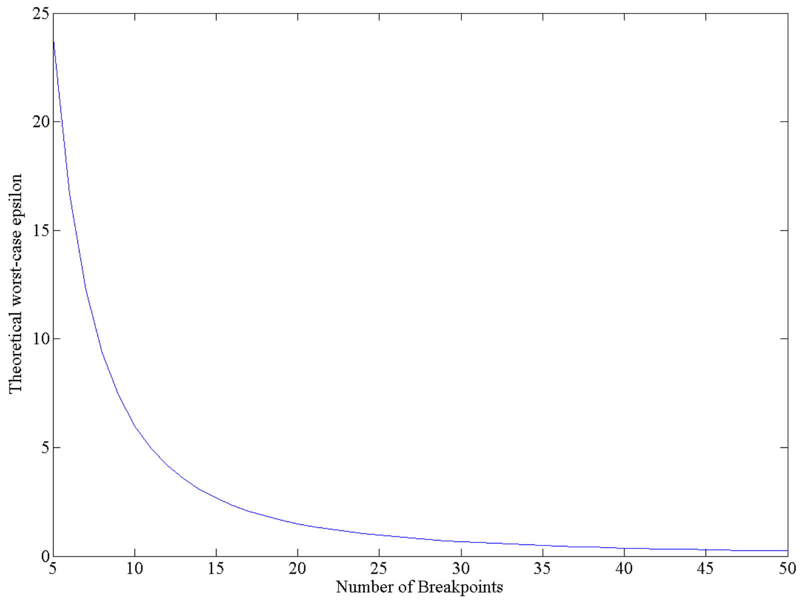
The equivalent *conditional* parametric model representation for the model presented in Figure 11.6 is given in Figure 11.9. The first column denotes player 1's action, the second column denotes player 2's action when player 1 raises, the third column denotes player 2's action when player 1 folds, the fourth column denotes player 3's action when player 1 raises and player 2 calls, the fifth column denotes player 3's action when player 1 raises and player 2 folds, and the sixth column denotes player 3's action when player 1 folds and player 2 raises.

These two representations are equivalent in terms of their expressive power, the representation sizes of the action spaces, and the MILFP program that gets generated. For example, while only one column corresponds to player 3's action space in Figure 11.6 and three columns do in Figure 11.9, the length of the size of player 3's action (e.g., FOLD/FOLD/CALL) is three times larger in Figure 11.6.

It is easy to see that conditional parametric models are equivalent to our standard parametric models both in terms of representation power and size, and that they will create the exact same MILFP. However, they often lead to a simpler visual representation, making them more useful



(a) Empirical solution quality as a function of the number of breakpoints.



(b) Solution quality guaranteed by our theory as a function of the number of breakpoints.

Figure 11.7: Experimental values of, and the theoretical bound on, ϵ.

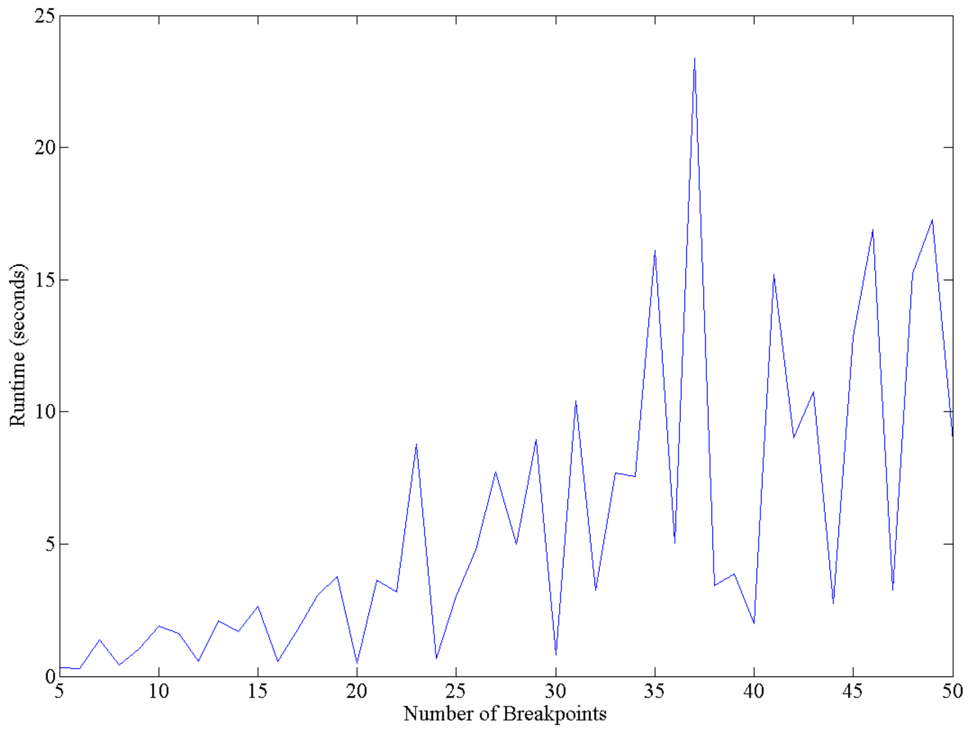


Figure 11.8: Running time (in seconds) as a function of the number of breakpoints.

in certain situations. In addition, $\overline{T}$ in Equation 11.3 (recall this refers to the maximum number of thresholds in a parametric model of any player) can now be replaced by $\hat{T}$ which denotes the maximum number of thresholds in a conditional parametric model of any player. In our example, $\overline{T}$ is three while $\hat{T}$ is only one. This actually gives an exponential improvement with respect to the number of players in the worst-case number of breakpoints needed according to Equation 11.3 in cases where $\hat{T} < \overline{T}$.

Computing best responses

To determine the ϵ's in the experiments, we need to be able to compute the best response for each player to a given strategy profile of the other players. This is relatively easy if we are sure in advance that for each strategy profile of the other players, there exists a best response that conforms to the given parametric model (e.g., as in the premise of Theorem 6). However, often we are not sure in advance that this is the case, and might only be able to come up with a set of parametric models such that a best response satisfies at least one of them.

In the game considered in this section, it is not the case that every best response satisfies the given parametric model. For example, suppose that if player 1 raises, then players 2 and 3 will call with every hand. Then player 1 will only want to raise some of his hands, and fold his bad hands. Thus his threshold will be below the calling thresholds of the other players, which differs

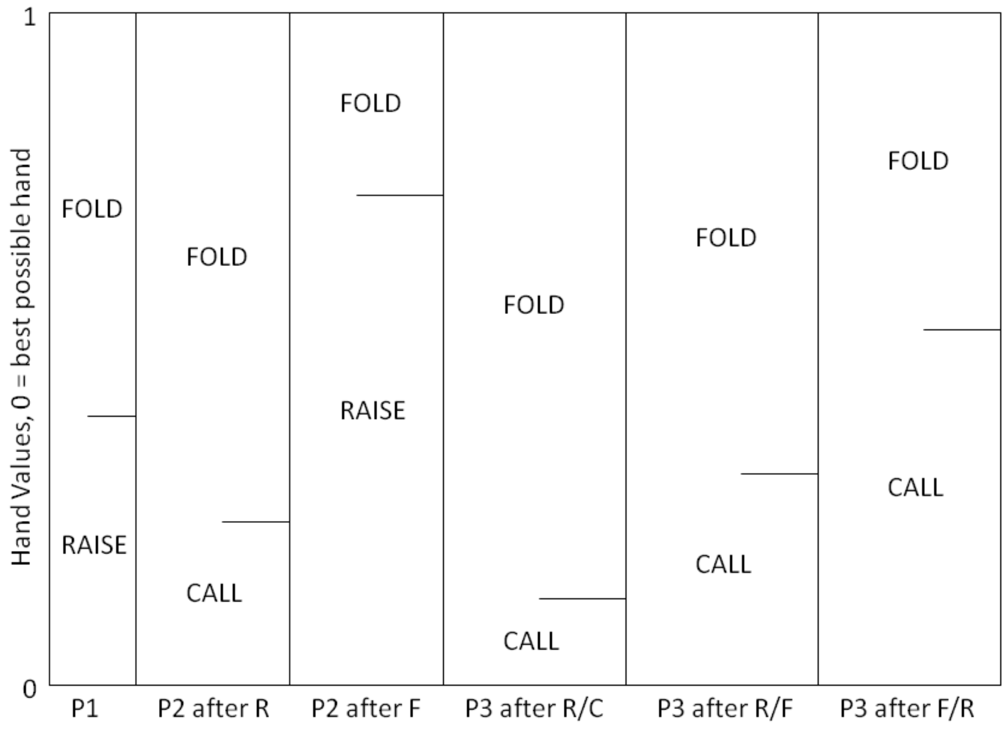


Figure 11.9: Conditional parametric model representation of the model given in Figure 11.6.

from the equilibrium parametric model given above.

We now present an algorithm for computing a best response in the setting where we are able to construct a set of parametric models for which we can prove that for each player and a given strategy profile of the other players, there exists a best response that satisfies at least one of the models. It is easy to see that our game satisfies this property. First, notice that every EPSB-undominated strategy for each player must satisfy the given threshold structure. However, it is not clear how the thresholds for the different players will relate to each other. But note that given strategies of the other players, there are at most 4 possible parametric models consistent with the threshold structure (i.e., the relevant threshold of the player in question must lie somewhere with respect to the other thresholds).

Our algorithm is the following. For each player, we fix the given strategies σ_{-i} of the other players and iterate over all the possible parametric models. Then we compute the best response for the given player i using each fixed model. This can be accomplished by treating the values $x_i = F_i(t_i)$ of player i's CDF evaluated at the thresholds as variable, and writing the formula for the expected profit of player i given σ_{-i} in terms of the x_i's. This yields a polynomial function of degree at most n in terms of the x_i's, where n is the number of players. This constrained optimization problem can be solved efficiently by standard techniques (e.g., using Matlab which presumably uses Newton's method), to determine the expected profit of the best response satisfying the given parametric model. We do this for each model, and take the highest value—call it π^*. The difference between π^* and the expected payoff of player i under σ yields ϵ_i—the difference between the payoff of his best response and his actual payoff. We do this for each player, and set ϵ equal to the maximum of the ϵ_i's.

This algorithm can be used as an *ex-post* checking procedure even if the premises of Theorem 6 (i.e., every EPSB-best response satisfies the given parametric model) or of Theorem 8 are not satisfied. As long as we can construct a set S of parametric models such that there (provably) exists a best response of each player i to the strategy profile σ_{-i} of the other players output by our algorithm that is consistent with a model in S, then we have computed an ϵ-equilibrium of the game, where ϵ is determined by the above procedure. Thus, the results of this section show that our algorithm can still be successful even in cases for which the premises of our theorems are not met. This is important, especially in light of the relatively strong premises of the theorems. Future work could look into relaxing the premise of the theorems, and proving the correctness of our algorithm in a wider range of settings.

Algorithm performance

Figure 11.8 shows the running times of our experiments, as a function of the number of breakpoints used. As shown in the figure, runtimes increased steadily from 0.3 seconds with 5 breakpoints to 8.9 seconds with 50 breakpoints.

Despite a clear positive correlation, the runtimes do not increase monotonically with the number of breakpoints. This is due to the fact that CPLEX is solving a fundamentally different MILFP for each number of breakpoints, and the runtimes of CPLEX's MIP solver are notoriously unpredictable (even on inputs that are seemingly quite similar). We saw a similar deviation from monotonicity of ϵ as a function of the number of breakpoints in Figure 11.7(a). Therefore, for large problems one may want to try several different numbers of breakpoints, since even consecutive values can lead to drastically different runtimes and values of ϵ. The optimal number to use will clearly depend on the desired ϵ and running time limitations. However, one implication of our results is that often far fewer breakpoints are needed to obtain a given ϵ than one might expect based on our theoretical bound in Equation 11.3. So a reasonable algorithm to use in practice might be to start by running our algorithm with some small number of breakpoints (such as 5), then increment the number of breakpoints by 1 and repeat until a desired ϵ or time limit is reached. This procedure could be easily parallelized by running our algorithm with different numbers of breakpoints on different cores, since the computations do not depend on each other.

11.9 Summary and extensions

We presented a new approach for solving large (even infinite) multiplayer games of imperfect information. The key idea behind our approach is that we include additional inputs in the form of qualitative models of equilibrium strategies (how the signal space should be qualitatively partitioned into action regions). In addition, we showed that our approach can lead to strong strategies in large finite games that we approximate with infinite games. We proved that our main algorithm is correct even if given a set of qualitative models (with a common US-refinement) of which only some are accurate.

In two player settings, our algorithm finds an exact equilibrium. The solution technique uses a mixed integer linear feasibility program. With more than two players, the models include nonlinear elements, which we approximate with piecewise linear functions. We showed how the accuracy of ϵ-equilibrium depends on the number of those pieces—both with a worst-case theorem and experiments that show that significantly fewer pieces are needed in practice.

For settings where our algorithm outputs a solution but we do not know that even one of the qualitative models is correct, we developed an *ex post* procedure for checking whether the solution is an equilibrium or an ϵ-equilibrium. The *ex post* check works under a significantly weaker assumption than our theorems, namely that we use qualitative models for which we can prove that for each player and the given strategy profile of the other players, there exists a best response that satisfies at least one of the models.

Experiments suggest that approximating a finite game with an infinite one can outperform abstraction-based approaches on some games. We constructed a game in which only a tiny amount of abstraction can be performed while obtaining strategies that are no more exploitable than the equilibrium strategies of our infinite approximation. Thus our approach presents a viable alternative to abstraction-based approaches. This is particularly promising in light of the recently uncovered abstraction pathologies.

We also showed how to extend our algorithm to the cases of more than two players, continuous private signal distributions, and dependent private signal distributions. In most of these cases, we presented the first algorithm that provably solves the class of games. Our experiments show that the algorithm runs efficiently in practice in both two-player and multi-player settings. It leads to a significant performance improvement in two-player limit Texas hold 'em poker—the most studied imperfect-information game in computer science—by solving endgames.

While in this paper we inferred the infinite approximations of finite games and the parametric models manually, future research could attempt to develop methods for generating them systematically and automatically. It is also possible that in future research one could prove that our approach works under less restrictive premises than are currently used in the main theorems.

Part IV

Exploiting Suboptimal Opponents

Chapter 12

DBBR: A Scalable, Domain-Independent Opponent Exploitation Algorithm

The previous portions of the thesis have focused on the problem of approximating Nash equilibrium strategies in large games. While equilibrium strategies guarantee a good performance in the worst case (at least in two-player zero-sum games), they potentially fail to take advantages of opponents' mistakes. I now turn to the problem of exploiting the mistakes of suboptimal opponents.

The natural approach for opponent exploitation is to try to learn a model of the opponent's strategy (based on prior game iterations and possibly additional historical data about the given opponent or other agents), then to maximally exploit this model. However, this approach has several drawbacks. First, it is incredibly difficult to learn an accurate model of the opponent's strategy quickly in large games. For example, no-limit Texas hold 'em has approximately 10^{165} states in its tree, while typical matches in the poker competition consist of just 3000 hands. So we only observe the opponent's actions at a minuscule portion of the information sets. The guarantees of no-regret learning algorithms are meaningless in such situations. This is further complicated by the fact that we may only see the opponent's private information after some hands and not others (e.g., in poker we only get to see the opponent's hand if no player folded in any betting round). For these reasons, opponent exploitation has been largely abandoned in recent years as a viable approach for strong agents in large imperfect-information games, in favor of the game-solving approach previously described.

An additional problem with this approach to opponent exploitation is that it can lead the exploiter to play a highly exploitable strategy himself. In general, a full best response will be a deterministic strategy that can be arbitrarily exploitable. This is problematic for several reasons. First, our model of the opponent will not be exact. If our model is wrong, we would like to not perform too poorly. Second, the opponent may not simply be playing a static strategy, and may try to deceive us by playing one way at the start of a match, then altering his strategy to exploit us once we start to exploit his initial strategy; this is known as the "get taught and exploited problem" [98]. A third reason is that if we play a maximal best response, it may be easier for the opponent to recognize his mistakes and fix his strategy (possibly exploiting us along the way).

Therefore, we would like to perform exploitation in a way that is robust to deviations of the opponent's strategy from our model. One approach that has been proposed is the *ϵ-safe best* *response* [64, 65]. In this approach, we assume the opponent plays according to our opponent model σ^* with some probability p, and that he can play arbitrarily with probability $1-p$. Our strategy will be the solution to this new game. The parameter p is adjusted so that some desired level of exploitability ϵ for our own strategy is obtained. These techniques find optimal tradeoffs between exploitation and exploitability; the construction of an opponent model is a subproblem of these techniques, and DBBR can be integrated within their framework. Our new approach DBBR relaxes the need of opponent private observations when building a model, which is a precursor step to BR, RNR, and DBR.

Many multiagent-learning and opponent exploitation algorithms make very strong assumptions about access to data and observability of play. It is extremely rare to have a large amount of labelled data on the specific opponent at hand. Often we have unlabelled, or semi-labelled, data from the play of some pool of agents, which may or may not resemble the given opponent in any meaningful way. In many situations there may be no historical data available at all, and if we hope to exploit mistakes of an opponent, we must learn to do so in real time solely based on our observations of his play in prior game iterations.

A recent approach precomputes several exploitative strategies in advance using the procedure described above, then decides dynamically between then in real time using a no-regret algorithm [8]. However, this approach is also limited by the issues discussed above; specifically, it relies on having access to massive amounts of labelled data, and also relies on the fact that the given opponent will play similarly to the opponents represented in the dataset (and that we can determine which expert to play very quickly with partial observability of the opponent's private information).

We present new approaches for opponent exploitation that do not rely on such strong (and often unrealistic) assumptions. They are able to effectively learn to exploit opponents in real time in large imperfect-information games after only a small number of interactions, without access to any historical data, and with partial or no observability of the opponent's private information in past game iterations. In addition, we show that in some games, it is actually possible to exploit suboptimal opponents significantly more than playing a static Nash equilibrium strategy while still guaranteeing zero exploitability in the worst case.

12.1 Introduction

While much work has been done in recent years on abstracting and computing equilibria in large extensive-form games, relatively less work has been done on exploiting suboptimal opponents (aka *opponent modeling*), in large part because the problem is significantly harder. While playing an equilibrium guarantees at least the value of the game in a two-player zero-sum game, often much higher payoffs can be obtained by deviating from equilibrium to exploit opponents who make significant mistakes. For example, against a poker opponent who always folds, the strategy of always raising will perform far better than any equilibrium strategy (which will sometimes fold with bad hands).

Texas hold 'em poker has emerged as the main testbed for evaluating algorithms in extensive-form games. In addition to its tremendous popularity, it also contains enormous strategy spaces, imperfect information, and stochastic events; such elements also characterize most of the challenging problems in computational game theory and multiagent systems. In light of these factors and the AAAI annual computer poker competition, poker has emerged as an important, visible challenge problem for AI as a whole, and multiagent systems in particular.

It is worth noting, however, that a fair amount of prior work has been done on opponent exploitation in significantly smaller games. For example, Hoehn et al. [56] ran experiments on Kuhn poker, a small two-player poker variant with about 20 states in its game tree. Recent work has also been done on opponent exploitation in rock-paper-scissors [80] and the repeated prisoners' dilemma [20]. However, these algorithms do not scale to large games. In contrast, the game tree of limit Texas hold 'em has about 10^{18} states.

A potential drawback of evaluating algorithms on one specific problem is that we run the risk of developing algorithms that are so game specific that they will not generalize to other settings. Heeding this risk, in this work we abandon many of the game-specific assumptions taken by prior approaches. Rather than relying on massive databases of human poker play [24, 91] and expert-generated features or prior distributions [56, 108], we will instead rely on game-theoretic concepts such as Nash equilibrium and best response, which apply to all games.

In addition, we require our algorithms to operate efficiently in real time (online), as opposed to algorithms that perform offline computations assuming they have access to a large number of samples of the opponent's strategy in advance [65, 92]. That prior work also assumed access to historical data which included the private information of the opponents (i.e., their hole cards) even when such information was only observed by the opponent. In many multiagent settings, an agent must play against opponents about whom he has little to no information in advance, and must learn to exploit weaknesses in a small number of interactions. Thus, we assume we have no prior information on our opponent's strategy in advance, and our algorithms will operate online.

Our main algorithm, called *Deviation-Based Best Response (DBBR)*, works by noting deviations between the opponent's strategy and that of a precomputed approximate equilibrium strategy, and constructing a model of the opponent based on these deviations. Then it computes and plays a best response to this opponent model (in real time). Both the construction of the opponent model and the computation of a best response take time linear in the size of the game tree and can be performed quickly in practice. As discussed above, we evaluate our algorithm empirically on limit Texas hold 'em; it achieves significantly higher win rates against several opponents — including competitors from recent AAAI computer poker competitions — than an approximate equilibrium strategy does.

We will assume that the moves of all players other than chance are observed by all players; for example, in poker all moves other than the initial dealing of the cards are publicly observed. In this setting, we can partition all game states into *public history sets*, PH_i, where states in the same public history set correspond to the same history of publicly observed actions. Note that each public history set must consist of a set of information sets of player i. For public history set $n \in PH_i$, let A_n denote the set of actions of player i at n. In general when we omit subscripts, player i will be implied.

Recent solution techniques can compute equilibria for games with up to 10^{17} states [15]. Such algorithms typically take several weeks or months to compute an ϵ-equilibrium for small ϵ. On the other hand, best responses can be computed much faster, and in abstractions with 10^{12} states they can be computed in about an hour. If coarser abstractions are used, best responses can be computed in minutes or even seconds, and can potentially be used as a subroutine in adaptive real-time algorithms.

12.2 DBBR: an efficient real-time opponent modeling algorithm

In this section we present our algorithm, *Deviation-Based Best Response (DBBR)*. It works by observing the opponent's action frequencies over the course of game, then using these observations to construct a model of the opponent's strategy. Essentially, we would like to conservatively assume that the opponent is playing the best (i.e., least exploitable) strategy that is consistent with our observations of his play. The obvious way to accomplish this would be to add linear constraints to the LP for finding an equilibrium [73] that force the opponent model to conform with our observations. However, such a computation could take several weeks, and would not be practical for real-time play in large games.

To obtain a more practical algorithm, we must find a faster way of constructing an opponent model from our observations. DBBR constructs the model by noting deviations of our opponent's observed action frequencies from equilibrium frequencies. For example, in poker suppose an equilibrium strategy raises 50% of the time when first to act, while the opponent raises only 30% of the time. While the opponent might be raising any 30% of hands, a safe guess might be to assume that he is raising his 'best' 30% of hands; we can construct such a strategy by starting with the equilibrium strategy, then removing the 'worst' 20% of hands from the raising range. Our algorithm is based on this intuition.

12.2.1 Overview of the algorithm

Pseudocode for a high-level overview of DBBR is given in Algorithm 19. In the first step, an approximate equilibrium σ^* of the game is precomputed offline. Next, when the game begins, the frequencies of the opponent's actions at different public history sets are recorded. These are used to compute the opponent's *posterior action probabilities*: the probabilities with which he chooses each action at each public history set $n \in PH_{-i}$. (We say that the elements of PH_{-i} are numbered according to breadth-first-search (BFS) traversal order.) Next, we compute the probability the opponent is in each bucket at n given our model of his play so far; we refer to these probabilities as the *posterior bucket probabilities*. We then compute a full model of the opponent's strategy by considering the deviations between the opponent's posterior action probabilities and those of σ^* at n. Based on these deviations, we iterate over all buckets and shift weight away from the action probabilities in σ^* until we obtain a strategy consistent with our model of the opponent's action probabilities. Finally, after we have iterated over all public history sets, we compute a best response to the opponent model. The next subsections will discuss the different components of the algorithm in detail.

12.2.2 Computing posterior action probabilities

In the course of our play against the opponent, we observe how often he chooses each action a at each public history set n; we denote this quantity by $c_{n,a}$. One idea would be to assume the

Algorithm 19 High-level overview of *DBBR*

```
Compute an approximate equilibrium of the game.
Maintain counters from observing opponent's play throughout the match.
for $n = 1$ to $|PH_{-i}|$ do
    Compute posterior action probabilities at $n$.
    Compute posterior bucket probabilities at $n$.
    Compute full model of opponent's strategy at $n$.
end for
return Best response to the opponent model.
```

opponent will play action a with probability

$$\frac{c_{n,a}}{\sum_{a'} c_{n,a'}}.$$

However, doing this could be problematic for a few reasons. First, we might not have any observations at a given set n, in which case this quantity would not even be defined. More generally, the quality of our observations might vary dramatically between public history sets; for example, we have a lot more confidence in sets for which we have 1000 observations than sets for which we have just 1 or 2, and we would like our algorithm to reflect this. A similar observation was the motivation behind a recent paper [64], though that work assumed that the opponent's private information was observable.

Our algorithm works by choosing a combination of the observed probability and the probability under the equilibrium strategy σ^*, where the weight on the observed frequencies is higher at public history sets for which we have more observations. Specifically, we use a Dirichlet prior distribution, where we assume we have seen N_{prior} fictitious hands at the given public history set for which the opponent played according to σ^*. Let $p^*_{n,a}$ denote the probability that σ^* plays action a at public history set n. We compute the posterior action probabilities, $\alpha_{n,a}$, as follows:

$$\alpha_{n,a} = \frac{p^*_{n,a} \cdot N_{prior} + c_{n,a}}{N_{prior} + \sum_{a'} c_{n,a'}}. \tag{12.1}$$

12.2.3 Computing posterior bucket probabilities

Since we are constructing the model of the opponent's strategy using a BFS ordering of the public history sets, we assume that we have already set his strategy for all ancestors of the current set n (including the parent n'). Let $s_{n',b,a}$ denote our model of the probability that the opponent plays his portion of the strategy sequence leading to n', then chooses action a in bucket b at state n'; this quantity has already been computed by the time we get to n in the algorithm. We can use these probabilities to construct the posterior probability, $\beta_{n,b}$, that the opponent is in bucket b (i.e., in poker, the opponent has those private cards) at public history set n. Pseudocode for this procedure is given in Algorithm 20, where h_b denotes the probability that chance makes the moves needed to put the opponent in bucket b.

Algorithm 20 ComputeBucketProbs(n)

```
for $b = 1$ to $|B_n|$ do
    $n' \leftarrow parent(n)$
    $a \leftarrow$ action taken to get from $n'$ to $n$.
    $\beta_{n,b} \leftarrow h_b \cdot s_{n',b,a}$
end for
Normalize the values $\beta_n$ so they sum up to 1.
```

12.2.4 Computing the opponent model

In this section we will present three different techniques for computing the opponent model. Recall that our high-level goal is to compute the 'best' (i.e., least exploitable) strategy for the opponent that is consistent with our observations of his behavior. We could accomplish this by performing an equilibrium-like computation; however, such a computation is too challenging to be performed in real time.

Rather than find the strategy consistent with our observations that is least exploitable, we will instead find the strategy that is 'closest' to the precomputed equilibrium. It turns out that this can be accomplished efficiently in practice, and intuitively we would expect strategies closer to equilibrium to be less exploitable.

Weighted L_1-distance minimization

Recall that the L_1 distance between two vectors x and y is defined as

$$||x - y||_1 = \sum_{i=1}^{k} |x_i - y_i|. \tag{12.2}$$

While this function treats all indices of the vector equally, in some cases we might want to put more weight on some components than on others. If p is a probability distribution over the integers from 1 to k, we define the *weighted L_1 distance* between x and y as

$$\sum_{i=1}^{k} p_i \cdot |x_i - y_i|. \tag{12.3}$$

Now, suppose we are at public history set n, where $\beta_{n,b}$ denotes the posterior probability that we are in bucket b, as computed by Algorithm 20. If we let the y_i's in Equation 12.3 correspond to the equilibrium probabilities of taking each action, and let the p_i's correspond to the $\beta_{n,b}$'s, then we can formulate the problem of finding the strategy closest to the precomputed equilibrium, subject to the posterior action probabilities $\alpha_{n,a}$, as an L_1-distance minimization problem.

Formally, we can formulate the optimization problem as follows, for a given public history set n:

$$\begin{aligned}
\text{minimize} \quad & \sum_{b \in B_n} \sum_{a \in A_n} \left[\beta_{n,b} \cdot |x_{n,b,a} - \sigma^*_{n,b,a}|\right] \\
\text{subject to} \quad & \sum_{b \in B_n} \left[\beta_{n,b} \cdot x_{n,b,a}\right] = \alpha_{n,a} \text{ for all } a \in A_n \\
& \sum_{a \in A_n} x_{n,b,a} = 1 \text{ for all } b \in B_n \\
& 0 \leq x_{n,b,a} \leq 1 \text{ for all } a \in A_n, b \in B_n
\end{aligned} \tag{12.4}$$

Recall that B_n denotes the set of all buckets we could be in at public history set n, while A_n denotes the set of actions at n. The variables $x_{n,b,a}$ correspond to the model of the opponent's strategy that we are trying to compute. Note that we can do this optimization separately for each public history set n; it makes more sense to do many smaller optimizations than to do a huge one for all public history sets at once, since the computations of the actions taken at different states do not depend on each other.

So as discussed above, we will perform a separate optimization at each n according to the program of Equation 12.4. It turns out that this can be cast as a linear program (LP) and solved efficiently using CPLEX's dual simplex algorithm for solving LPs. Doing this for each public history set n yields the opponent model x. Note that the program could have many solutions, and that CPLEX will just output the first solution it encounters (and not necessarily the solution that performs best in practice). This means that there might actually exist a strategy that minimizes L_1 distance from equilibrium that performs better in practice than the strategy output by CPLEX.

Weighted L_2-distance minimization

While Section 12.2.4 uses the weighted L_1 distance to measure the proximity of two strategies, we could also use other distance metrics. In this section we will consider another common distance function: the weighted L_2 distance.

Similarly to Equation 12.2, the L_2 distance between x and y is defined as

$$||x - y||_2 = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}. \tag{12.5}$$

Analogously to the L_1 case, we define the *weighted L_2 distance* between x and y as

$$\sqrt{\sum_{i=1}^{k} p_i \cdot (x_i - y_i)^2}. \tag{12.6}$$

The new program for computing the opponent model at n is the following:

$$\begin{aligned} \text{minimize} \quad & \sum_{b \in B_n} \sum_{a \in A_n} \left[\beta_{n,b} \cdot (x_{n,b,a} - \sigma^*_{n,b,a})^2\right] \\ \text{subject to} \quad & \sum_{b \in B_n} \left[\beta_{n,b} \cdot x_{n,b,a}\right] = \alpha_{n,a} \text{ for all } a \in A_n \\ & \sum_{a \in A_n} x_{n,b,a} = 1 \text{ for all } b \in B_n \\ & 0 \leq x_{n,b,a} \leq 1 \text{ for all } a \in A_n, b \in B_n \end{aligned} \tag{12.7}$$

Note that we can omit the square root, since it is a monotonic operator. The resulting formulation in Equation 12.7 is a quadratic program (QP), which can also be solved efficiently in practice using CPLEX. As in the L_1 case, we can formulate and solve a separate optimization problem for each public history set n to compute the opponent model x.

Our custom weight-shifting algorithm

While the previous two sections described how to compute an opponent model using two popular distance functions, perhaps we can do even better by designing our own custom algorithm that takes into account the conservative reasoning about the opponent that we discussed earlier. In this section we will describe such an algorithm. In particular, it takes into account the fact that we already know an approximate ranking of the buckets at each public history set from the approximate equilibrium σ^*.

For example, suppose the opponent is only raising 30% of the time when first to act, while σ^* raises 50% of the time in that situation (as given in the example at the beginning of this section). Instead of doing a full L_1 or L_2-minimization explicitly, we could use the following heuristic algorithm: sort all buckets by how often the opponent raises with them under σ^*, then greedily keep removing buckets from his raising range until the weighted sum (using the $\beta_{n,b}$'s as weights) equals 30%. This is a simple greedy algorithm, which can be run significantly more efficiently in practice than the L_1 and L_2-minimization procedures described in the last two subsections, which must repeatedly use CPLEX at runtime.

For simplicity, we present our algorithm for the case of three actions, although it extends naturally to any number of actions. First we initialize the opponent's strategy at n, σ_n, to the equilibrium σ^*. We also initialize our current model of his action probabilities γ_n to $p^*_{n,a}$, the equilibrium action probabilities.

Next, we check whether the opponent is taking action 3 more often than he should at n by comparing $\alpha_{n,3}$ to $\gamma_{n,3}$. If he is, we are going to want to increase the probabilities he plays action 3 in various buckets; otherwise, we will decrease these probabilities. For now, we will assume that $\alpha_{n,3} > \gamma_{n,3}$ (the other case is handled analogously).

We start by adding weight to the bucket that plays action 3 with the highest probability at n; denote this bucket by $\hat{b}$. If

$$\gamma_{n,3} + \beta_{n,\hat{b}} \cdot (1 - \sigma_{n,\hat{b},3}) < \alpha_{n,3}, \tag{12.8}$$

we set $\sigma_{n,\hat{b},3} = 1$, since that will not cause $\gamma_{n,3}$ to exceed $\alpha_{n,3}$ once it is adjusted. Otherwise, we increase $\sigma_{n,\hat{b},3}$ by $\frac{(\alpha_{n,3} - \gamma_{n,3})}{\beta_{n,\hat{b}}}$. (Recall that $\beta_{n,\hat{b}}$ denotes the posterior probability that the opponent holds bucket $\hat{b}$ at n, as computed in Algorithm 20.) Let Δ denote the amount by which we increase $\sigma_{n,\hat{b},3}$. We will also increase the action probability $\gamma_{n,3}$ by $\beta_{\hat{b}} \cdot \Delta$.

Next we must compensate for this increase of the probability of playing action 3 in bucket $\hat{b}$ by decreasing the probabilities of playing actions 1 and/or 2. Let $\underline{a}$ denote the action (1 or 2) played with lower probability in σ_n in bucket $\hat{b}$, and let $\overline{a}$ denote the other action. If $\sigma_{n,\hat{b},\underline{a}} \geq \Delta$, then we set $\sigma_{n,\hat{b},\underline{a}} = \sigma_{n,\hat{b},\underline{a}} - \Delta$ and update $\gamma_{n,\underline{a}}$ accordingly. Otherwise, we set $\sigma_{n,\hat{b},\underline{a}} = 0$ and remove the remaining probability $\Delta - \sigma_{n,\hat{b},\underline{a}}$ from $\sigma_{n,\hat{b},\overline{a}}$.

If the inequality of Equation 12.8 held above, then our opponent model probabilities still do not agree with the posterior action probabilities, and thus we must continue shifting probability mass; we continue by setting $\hat{b}$ to the bucket that plays action 3 with the second highest probability at n, and repeating the above procedure. Otherwise, we are done setting the probabilities for action 3, and we perform a similar procedure to shift weight between the probabilities that he plays actions 1 and 2 until they agree with α_n.

We have now constructed an opponent model that agrees with our posterior action probabilities. Note that we had to iterate over possibly all of the buckets at public history set n. Since each bucket is contained in only one public history set, the algorithm's run time is linear in the size of the game tree.

Additionally, although we presented this algorithm for the case of three actions at n, it easily generalizes to more actions. Rather than just designating $\overline{a}$ and $\underline{a}$, we will sort all actions in the order of how often they are played in bucket $\hat{b}$, and proceed through this list adjusting probabilities as in the three-action case.

12.2.5 Full algorithm

In practice, constructing an opponent model and computing a best response at each repetition of the game (e.g., hand in poker) might be too slow. This can be mitigated by doing so only every k repetitions. In addition, we may want to start off playing the equilibrium σ^* for several repetitions so that we can obtain a reasonable number of samples of the opponent's play, rather than trying to exploit him immediately. Overall, our full algorithm will have three parameters: T denotes how many repetitions to first play the equilibrium σ^* before starting to exploit, k denotes how often to recompute an opponent model and best response, and N_{prior} from Equation 12.1 is the parameter of the action probability prior distributions. Pseudocode for the algorithm is given in Algorithm 21, where M is the number of repetitions in the match.

12.3 Experiments and discussion

We used two-player limit Texas hold 'em as our experimental domain. It is a large-scale game with 10^{18} states in the game tree. It is the most-studied full-scale poker game in computer science, and is also played by human professionals. The rules are described in Chapter 3.

Algorithm 21 DBBR(T,k,N_{prior})

```
for $iter = 1$ to $T$ do
    Play according to the precomputed equilibrium strategy $\sigma^*$
end for
$opponent\_model = ComputeOppModel(N_{prior})$
$\sigma_{BR} = ComputeBestResponse(opponent\_model)$
for $iter = T + 1$ to $M$ do
    if $iter$ is a multiple of $k$ then
        $opponent\_model = ComputeOppModel(N_{prior})$
        $\sigma_{BR} = ComputeBestResponse(opponent\_model)$
    end if
    Play according to $\sigma_{BR}$
end for
```

12.3.1 Experimental results

We ran our algorithm against several opponents; the results are shown in Table 12.1. The first four opponents—Random, AlwaysFold, AlwaysCall, and AlwaysRaise—play naïvely as their names suggest. GUS2 and Dr. Sahbak were entrants in the 2008 AAAI computer poker competition, and Tommybot was an entrant in the 2009 competition; we selected these bots to experiment against because they had the worst performances in the competitions, and we expect opponent modeling to provide the biggest improvement against weak opponents. Against stronger opponents one might prefer to always play the precomputed equilibrium rather than turning on the exploitation. This can be accomplished by periodically looking at the win rate, and only attempting to exploit the opponent if a win rate above some threshold is attained.

GS5 is a bot we entered in the 2009 AAAI computer poker competition that plays an approximate-equilibrium strategy. It was computed using an abstraction which had branching factors of 15, 40, 6, and 6 respectively in the four betting rounds. The parameter values we used in DBBR (as described in Section 12.2.5) were $T = 1000$, $k = 50$, $N_{prior} = 5$, with GS5 playing the role of the initial approximate-equilibrium strategy (i.e., we ran GS5 for the first 1000 hands of each match and recomputed an opponent model and best response every 50 hands subsequently). Since each match consists of 3000 duplicate hands, this means that GS5 and DBBR play the same strategy for the first third of each match.

We set $T = 1000$ since it is essential that our algorithm obtains a reasonable number of samples of the opponent's play (in different parts of the game tree) before attempting to exploit. As discussed in the next paragraph, our main motivation in setting k was to allow us to update the opponent model as frequently as we could while remaining under the competition time limit. For N_{prior}, we wanted to choose a small number so that our observations would quickly trump the prior for common public history sets, but so that the prior would have more weight if we had just one or two observations. Note that setting $N_{prior} = 5$ means that our prior and our observations will have equal weight in our model when we have observed the opponent's action 5 times at the given public history set. Changing the parameter values could certainly have a large effect on the results, and should be studied further.

Unfortunately GS5 was too large to use as the approximate-equilibrium strategy in our real-time opponent modeling updates. Therefore, we also precomputed an approximate-equilibrium σ^* that used a much smaller abstraction than GS5: the branching factors of its abstraction were 8, 12, 4, and 4. While σ^* is clearly an inferior strategy to GS5, it was small enough to allow us to construct opponent models and compute best responses in just a few seconds, keeping us within the time limit of the AAAI competition.

We experimented with all three of the approaches for computing the opponent model described in Section 12.2.4: the three algorithms DBBR-L_1, DBBR-L_2, and DBBR-WS (i.e., 'Weight-Shifting') correspond to the three different algorithms in that section. We ran all three of these algorithms against each of the opponents described above (with the exception of Tommybot, which we were not able to play against DBBR-L_1 and DBBR-L_2 due to technical issues).

As shown in Table 12.1, DBBR-WS performed significantly better against all of the opponents than GS5 did (in one case, the win rate was over twice as high). Furthermore, DBBR-WS beat GUS2 by more than any other bot in the 2008 competition did, and its win rates against Dr. Sahbak and Tommybot were surpassed by the win rate of just a single bot.

	Random	AlwaysFold	AlwaysCall	AlwaysRaise	GUS2	Dr. Sahbak	Tommybot
GS5	0.854 ± 0.008	0.646 ± 0.0009	0.582 ± 0.005	0.791 ± 0.009	0.636 ± 0.004	0.665 ± 0.027	0.552 ± 0.008
DBBR-WS	1.769 ± 0.025	0.719 ± 0.002	0.930 ± 0.014	1.391 ± 0.034	0.807 ± 0.011	1.156 ± 0.043	1.054 ± 0.044
DBBR-L_1	2.164 ± 0.036	0.717 ± 0.002	0.935 ± 0.017	0.878 ± 0.032	0.609 ± 0.054	1.153 ± 0.074	
DBBR-L_2	2.287 ± 0.046	0.716 ± 0.002	0.931 ± 0.026	1.143 ± 0.084	0.721 ± 0.050	1.027 ± 0.072	

Table 12.1: Win rate in small bets/hand of the bot listed in the row. The $\pm$ given is the standard error (standard deviation divided by the square root of the number of hands).

12.3.2 Comparing the opponent modeling algorithms

It is not totally clear from the results in Figure 12.1 which of the three algorithms for constructing the opponent model — L_1, L_2, or our weight-shifting algorithm — is best. For example, DBBR-WS obtains a win rate of 1.391 sb/h against AlwaysRaise while DBBR-L_1 obtains a win rate of 0.878 sb/h, but DBBR-L_1 obtains a win rate of 2.164 sb/h against Random while DBBR-WS obtains only 1.769 sb/h. Similarly, for all other pairings there exist opponents such that one bot achieves a higher win rate against one opponent, but not against the other opponent. So there is no clear total ordering of the three algorithms.

That being said, DBBR-L_2 does at least as well (or essentially the same) against all of the opponents as DBBR-L_1, except for Dr. Sahbak; this suggests that DBBR-L_2 is a stronger program. As between DBBR-L_2 and DBBR-WS, it really seems to depend on the opponent. DBBR-WS performs significantly better against AlwaysRaise, GUS2, and Dr. Sahbak and slightly better against AlwaysFold than DBBR-L_2; however, DBBR-L_2 performs significantly better against Random and slightly better against AlwaysCall than DBBR-WS. So DBBR-WS performs significantly better against three of the six opponents than DBBR-L_2 (and essentially the same against two opponents), suggesting that it is a better algorithm.

In addition, DBBR-WS performs significantly better against both of the actual opponents from the AAAI competition (GUS2 and Dr. Sahbak) than DBBR-L_2, which suggests that it might perform better in practice against realistic opponents. This fact, combined with the fact that DBBR-WS is more efficient than the other algorithms, which have to perform many optimizations using CPLEX at runtime, suggest that DBBR-WS is a better algorithm to use in practice.

Note that this does not imply that the weighted L_1 and L_2 distance functions are poor distance metrics; it just means that the particular solution output by CPLEX does not do as well as the solution output by DBBR-WS. It is very possible that if CPLEX used different LP/QP algorithms, it might find a solution that does significantly better. This would certainly be a worthwhile avenue for future work.

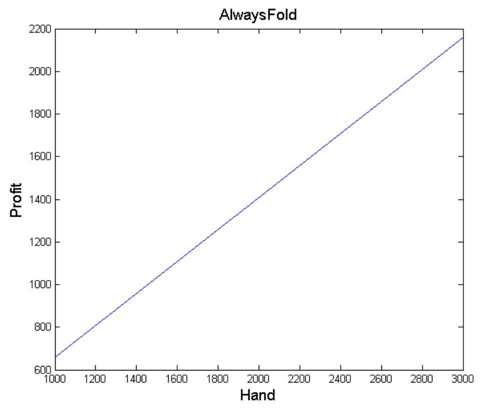
12.3.3 Win rates over time

One might expect that DBBR[1] would immediately begin exploiting the opponents at hand 1001—when it switches from playing an approximate equilibrium to opponent modeling—and that the win rate would increase steadily. In fact, this happened in the matches against most of the bots. For example, Figure 12.1(a) shows that DBBR's profits against AlwaysFold increase linearly over time, and Figure 12.1(b) shows that DBBR's win rate increases in a concave fashion.

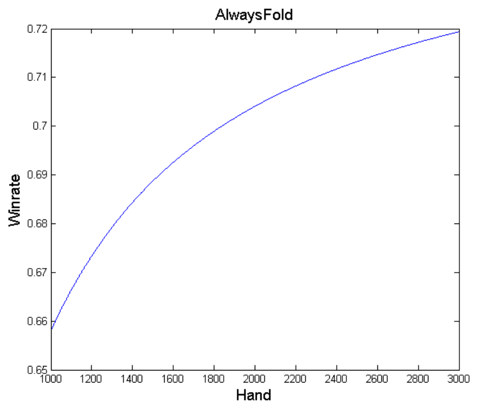
Surprisingly, we observed a different behavior in the matches against several other opponents. In the matches against AlwaysRaise and GUS2, the win rate decreases significantly for the first several hundred hands before it starts to increase, as shown in Figures 12.2(b) and 12.3(b). This happens because the approximate-equilibrium strategy plays some action sequences with very low probability, leading it to not explore the opponent's full strategy space in the 1000 hands. This will lead to a significant disparity between the prior and actual strategies of the opponent at hand 1001 if the opponent's strategy differs significantly from the approximate equilibrium in those unexplored regions. This in turn may cause DBBR to think it can immediately exploit the opponent in certain ways, which turn out to be unsuccessful; but eventually as DBBR explores these sequences further and gathers more observations, it figures out successful exploitations.

The following hand from our experiments between DBBR and AlwaysRaise exemplifies this phenomenon. The hand was the 1006'th hand of the match. There were many raises and reraises during the preflop, flop, and turn betting rounds. When the river card came, DBBR had only a ten high (a very weak hand in this situation). However, based on its observations during the first 1005 hands, it knew that AlwaysRaise had a very wide range of hands given this betting sequence, many of which were also weak hands (though probably still stronger than ten high). On the other hand, DBBR had very few observations of how AlwaysRaise responds to a series of raises on the river, since GS5 made those plays very rarely during the first 1000 hands; hence, DBBR resorted to the prior to model the opponent, which had the opponent folding all of his weak hands to a raise (since GS5 would do this). So DBBR thought that raising would get the opponent to fold most of his hands, while in reality AlwaysRaise continues to raise with all of his hands. In this particular hand, DBBR lost a significant amount of money due to the additional raises he made on the river with a very weak hand.

[1]The results in this section refer to our main algorithm, DBBR-WS.

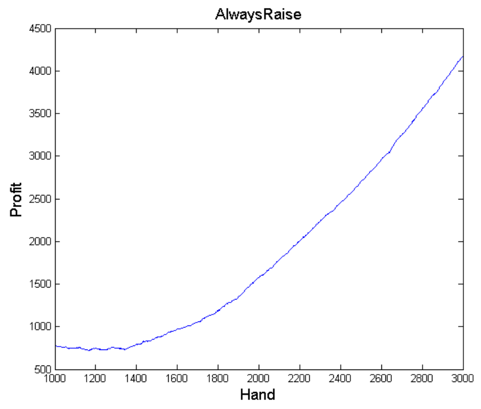


(a)

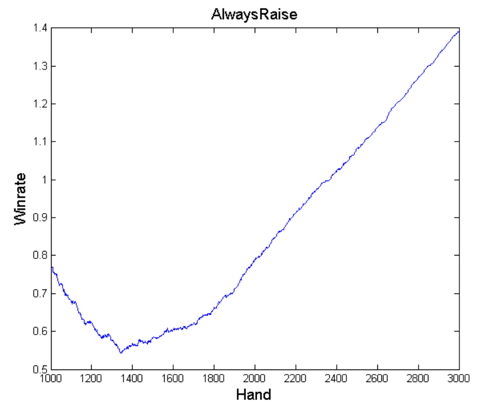


(b)

Figure 12.1: Profits and win rates over time of DBBR-EM against AlwaysFold.

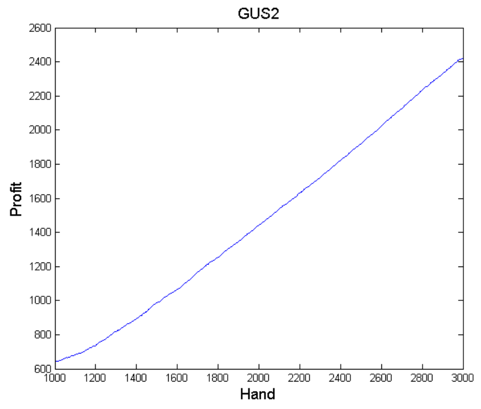


(a)

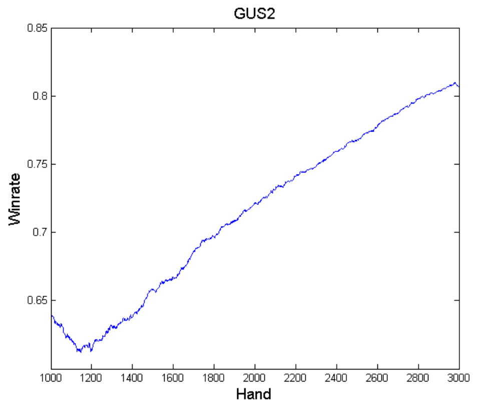


(b)

Figure 12.2: Profits and win rates over time of DBBR-EM against AlwaysRaise.



(a)



(b)

Figure 12.3: Profits and win rates over time of DBBR-EM against competition agent GUS2.

12.4 Summary and extensions

We presented DBBR, an efficient real-time algorithm for opponent modeling and exploitation in large extensive-form games. It works by observing the opponent's action frequencies and building an opponent model by combining information from a precomputed equilibrium strategy with the observations. This enables the algorithm to combine game-theoretic reasoning and pure opponent modeling, yielding a hybrid that can effectively exploit opponents after a small number of interactions.

Our experiments in full-scale two-player limit Texas hold'em poker show that DBBR is effective in practice against a variety of opponents, including several entrants from recent AAAI computer poker competitions. DBBR achieved a significantly higher win rate than an approximate-equilibrium strategy against all of the opponents in our experiments. Furthermore, it achieved a higher win rate against the opponents from previous competitions than all of the entrants from that year's competition achieved (except for at most one). We compared three different algorithms for constructing the opponent model, and conclude that our custom weight-shifting algorithm outperforms algorithms that employ weighted L_1 and L_2-distance minimization.

While DBBR is able to effectively exploit weak opponents, it might actually become significantly exploitable to strong opponents (e.g., opponents who operate in a finer-grained abstraction). Thus, we would like to only attempt to exploit weak opponents, while playing the equilibrium against strong opponents. This can be accomplished by periodically looking at the win rate, and only attempting to exploit the opponent if a win rate above some threshold is attained. Our current work involves developing automated schemes that alternate between DBBR and equilibrium play based on the specific opponent at hand. In addition, DBBR could be extended to the setting where the opponent's private information from the previous game iteration is sometimes observed. Finally, future work could look at more robust versions of DBBR, where the opponent model allows the opponent to sometimes deviate from his observed action probabilities, or a safer strategy than the actual best response is used.

Chapter 13

Safe Opponent Exploitation

In repeated interactions against an opponent, an agent must determine how to balance between *exploitation* (maximally taking advantage of weak opponents) and *exploitability* (making sure that he himself does not perform too poorly against strong opponents). In two-player zero-sum games, an agent can play a minimax strategy, which guarantees at least the value of the game in expectation against any opponent. However, doing so could potentially forego significant profits against suboptimal opponents. Thus, an equilibrium strategy has low (zero) exploitability, but achieves low exploitation. On the other end of the spectrum, agents could attempt to learn the opponent's strategy and maximally exploit it (using algorithms such as DBBR, which was described in the preceding chapter); however, doing so runs the risk of being exploited in turn by a deceptive opponent. This is known as the "get taught and exploited problem" [98]. Such deception is common in games such as poker; for example, a player may play very aggressively initially, then suddenly switch to a more conservative strategy to capitalize on the fact that the opponent tries to take advantage of his aggressive "image," which he now leaves behind. Thus, pure opponent exploitation potentially leads to a high level of exploitation, but at the expense of exploitability. Respectively, the game-solving community has, by and large, taken two radically different approaches: finding game-theoretic solutions and opponent exploitation.

We are interested in answering a fundamental question that helps shed some light on this tradeoff:

Is it possible to exploit the opponent more than any equilibrium strategy of a stage game would, while simultaneously guaranteeing at least the value of the full game in expectation in the worst case?

If the answer is no, then fully safe exploitation is not possible, and we must be willing to accept some increase in worst-case exploitability if we wish to deviate from equilibrium in order to exploit suboptimal opponents. However, if the answer is yes, then safe opponent exploitation would indeed be possible.

One may think (and this author did) that safe opponent exploitation is not possible [32]. The intuition for that argument was that the opponent could have been playing an equilibrium all along, and when we deviate from equilibrium to attempt to exploit him, then we run the risk of being exploitable ourselves. However, that argument is incorrect. It does not take into account the fact that our opponent may give us a *gift* by playing an identifiably suboptimal strategy, such as one that is strictly dominated.[1] If such gift strategies are present in a game, then it turns out that safe exploitation can be achieved; specifically, we can deviate from equilibrium to exploit the opponent provided that our worst-case exploitability remains below the total amount of profit won through gifts (in expectation).

Is it possible to obtain such gifts that do not correspond to strictly-dominated strategies? What about other forms of dominance, such as weak, iterated, and dominance by mixed strategies? Recently it was claimed that all non-iteratively-weakly-dominated strategies are best responses to each equilibrium strategy of the other player [117]. This would suggest that such undominated strategies cannot be gifts, and that gift strategies must therefore be dominated according to some form of dominance. We disprove this claim and present a game in which a non-iteratively-weakly-dominated strategy is not a best response to an equilibrium strategy of the other player. Safe exploitation is possible in the game by taking advantage of that particular strategy. We define a formal notion of gifts, which is more general than iteratively-weakly-dominated strategies, and show that safe opponent exploitation is possible specifically in games in which such gifts exist [37].

Next, we provide a full characterization of the set of safe exploitation strategies, and we present several efficient algorithms for converting any opponent exploitation architecture (that is arbitrarily exploitable) into a fully safe opponent exploitation procedure. One of our algorithms is similar to a procedure that guarantees safety in the limit as the number of iterations goes to infinity [80]; however, the algorithms in that paper can be arbitrarily exploitable in the finitely-repeated game setting, which is what we are interested in. The main idea of our algorithm is to play an ϵ-safe best response (a best response subject to the constraint of having exploitability at most ϵ) at each time step rather than a full best response, where ϵ is determined by the total amount of gifts obtained thus far from the opponent. Safe best responses have also been studied in the context of Texas hold 'em poker [65], though that work did not use them for online opponent exploitation. We also present several other safe algorithms which alternate between playing an equilibrium and a best response depending on how much has been won so far in expectation. Algorithms have been developed which guarantee ϵ-safety against specific classes of opponents (stationary opponents and opponents with bounded memory) [94]; by contrast, our algorithms achieve full safety against all opponents.

It turns out that safe opponent exploitation is also possible in extensive-form games, though we must redefine what strategies constitute gifts and must make pessimistic assumptions about the opponent's play in game states off the path of play. We present efficient algorithms for safe exploitation in games of both perfect and imperfect information, and fully characterize the space of safe strategies in these game models. We also show when safe exploitation can be performed in the middle of a single iteration of an extensive-form game. This may be useful when a mistake is observed early on.

We compare our algorithms experimentally on Kuhn poker [76], a simplified form of poker which is a canonical problem for testing game-solving algorithms and has been used as a test problem for opponent-exploitation algorithms [56]. We observe that our algorithms obtain a significant improvement over the best equilibrium strategy, while also guaranteeing safety in the worst case. Thus, in addition to providing theoretical advantages over both minimax and

[1]We thank Vince Conitzer for pointing this out to us.

fully-exploitative strategies, safe opponent exploitation can be effective in practice.

13.1 Uses, applicability, and generality of the approach

In this section we suggest two alternative uses of the approach, as well as discuss its applicability and generality.

13.1.1 Two alternative uses of the methodology

We can view safe exploitation as a meta-algorithm that enforces the safety of *any* opponent exploitation procedure by ensuring that it does not risk too much at any point. An opponent exploitation architecture consists of two components: 1) an opponent modeling algorithm, which takes as input the observations of both players' actions (to the extent that they are observable) and constructs a model of the opponent's strategy, and 2) a strategy selection algorithm, which takes the opponent model and the observations as input and outputs an exploitative strategy. This strategy may not be safe in general.

The first way to use our safe exploitation methodology is to obtain safety by curtailing the strategies that the architecture may propose. This is depicted in Figure 13.1.

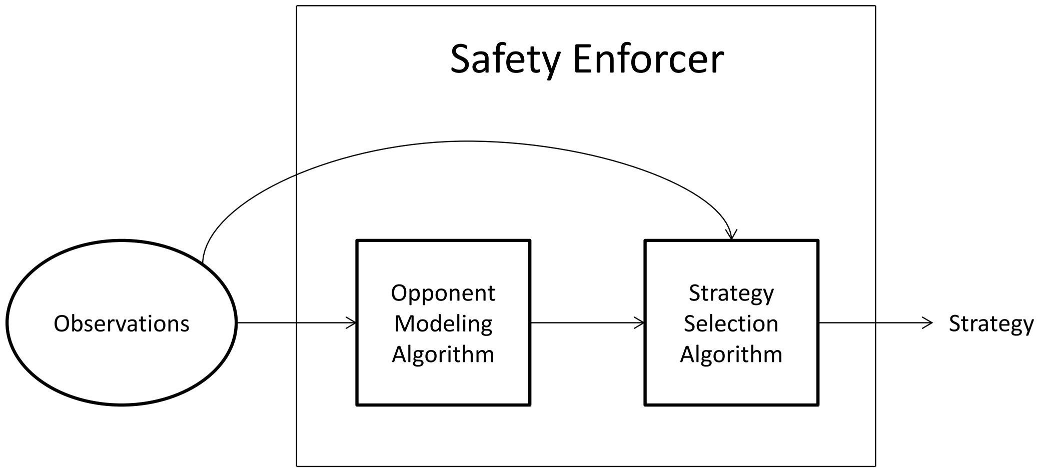


Figure 13.1: Our safe exploitation methodology used as a meta-algorithm which makes any opponent exploitation architecture safe. An opponent exploitation architecture consists of two components: an opponent modeling algorithm and a strategy selection algorithm.

The second way to use the methodology is to view our safe exploitation algorithms as alternatives to standard exploitation algorithms within the opponent exploitation paradigm. Our safe algorithms still work with any opponent modeling algorithm to construct an opponent model, but replace a potentially unsafe strategy selection algorithm with a new algorithm that guarantees safety. This is depicted in Figure 13.2.

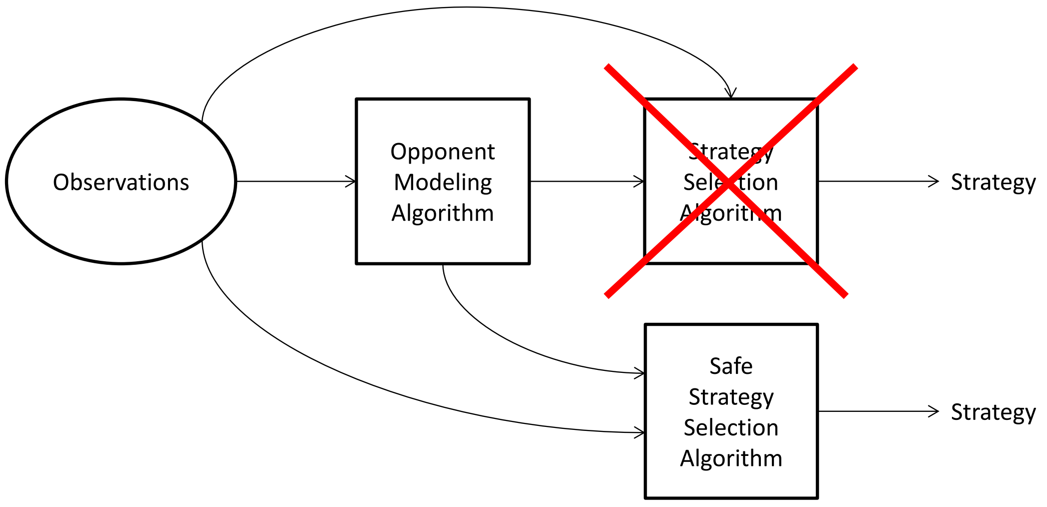

Observations
Opponent Modeling Algorithm
Strategy Selection Algorithm
Strategy
Safe Strategy Selection Algorithm
Strategy

Figure 13.2: Our safe exploitation methodology used to replace the strategy selection component while retaining the opponent modeling component of any opponent exploitation architecture.

13.1.2 Bounds suffice for using the methodology

We expect our algorithms to be useful in practice in many real-world domains, for example, in (cyber)security games. It has been observed that human adversaries in such domains often behave irrationally, and there can be significant benefits to exploiting their mistakes [12, 89, 90]. However, the cost of making a mistake ourselves is extremely high in such domains, for example, since human lives could be at stake. Algorithms that can exploit irrational opponents while still guaranteeing safety would be very desirable.

Furthermore, perhaps the main criticism of security games to date is that the numeric payoffs for the attacker and defender are questionable. Our approach does not require an exact model of the game. We only need a lower bound on the gifts (mistakes) that the opponent has given us and an upper bound on the loss from our exploitation. This would be especially useful in security games, since it guarantees robustness even when the game models are not accurate. (Another advantage is that our approach applies also to multi-step games, which are a richer, more powerful framework than the security game models used to date—Stackelberg games—where the defender moves once and then the attacker moves once.)

13.1.3 The methodology also applies to infinitely repeated, general-sum, and multiplayer games

Our methodology also applies to two-player zero-sum infinitely repeated games. While some of our algorithms specifically depend on the finite time horizon and will not extend to the infinite setting, several of them do not, and will apply straightforwardly. In particular, the algorithm that is most aggressive (among safe algorithms) and performed best in the experiments does not rely on a finite horizon.

For general-sum and multiplayer games, our methodology applies straightforwardly if we replace the minimax value with the *maximin value* (i.e., maximizing our expected payoff minimized over the others' strategies) in our algorithms. In two-player zero-sum games, these two values coincide, and any equilibrium strategy guarantees at least this value in expectation in the worst case. In general-sum and multiplayer games, these properties do not hold; however, in many settings it could be very desirable to exploit opponents' mistakes while still guaranteeing the maximin value. For example, this could be extremely useful in security domains, which are often modeled as non-zero-sum games [74]—since safety is of high importance.

13.1.4 Safe exploitation can be viewed as selection among equilibria of the repeated game

As we discuss in Section 13.2, in repeated games, the set of safe strategies is the same as the set of maximin strategies *in the repeated game* (and therefore, the set of Nash equilibria in the case where the repeated game is a two-player zero-sum game). Thus, one can view our safe exploitation algorithms as procedures for selecting among equilibria *of the repeated game*. In the context of non-repeated games, our work can be viewed as equilibrium selection in the non-repeated game. However, in both repeated and non-repeated games, as we will discuss in Section 13.6.3, our equilibrium refinement differs from subgame perfection [104], and thus also from all the usual equilibrium refinements, which are further refinements of subgame perfection.

13.2 Safety

One desirable property of a strategy for a repeated game is that it is *safe*:

Definition 27. *A* safe *strategy for a repeated game is a strategy that guarantees a worst-case payoff of at least v_i per period in expectation.*

The set of safe strategies is the same as the set of minimax strategies in the full repeated game. Clearly playing a (stage-game) minimax strategy at each iteration is safe, since it guarantees at least v_i in each iteration. However, a minimax strategy may fail to maximally exploit a suboptimal opponent. On the other hand, deviating from stage-game equilibrium in an attempt to exploit a suboptimal opponent could lose the guarantee of safety and may result in an expected payoff below the value of the game against a deceptive opponent (or if the opponent model is incorrect). Thus, a natural question to consider is whether there exist strategies that are safe, yet deviate from stage-game equilibrium strategies (in order to exploit an opponent's mistakes).

13.2.1 A game in which safe exploitation is not possible

Consider the classic game of Rock-Paper-Scissors (RPS), whose payoff matrix is depicted in Figure 13.3. The unique equilibrium σ^* is for each player to randomize equally among all three pure strategies.

Now suppose that our opponent has played Rock in each of the first 10 iterations (while we have played according to σ^*). We may be tempted to try to exploit him by playing the pure strategy Paper at the 11th iteration. However, this would not be safe; it is possible that he has in fact been playing his equilibrium strategy all along, and that he just played Rock each time by chance (this will happen with probability $\frac{1}{3^{10}}$). It is also possible that he will play Scissors in

	R	P	S
R	0	-1	1
P	1	0	-1
S	-1	1	0

Figure 13.3: Payoff matrix of Rock-Paper-Scissors.

the next round (perhaps to exploit the fact that he thinks we are more likely to play Paper having observed his actions). Against such a strategy, we would actually have a negative expected total profit—0 in the first 10 rounds and -1 in the 11th. Thus, our strategy would not be safe. By similar reasoning, it is easy to see that any deviation from σ^* will not be safe, and that safe exploitation is not possible in RPS.

13.2.2 A game in which safe exploitation is possible

Now consider a variant of RPS in which player 2 has an additional pure strategy T. If he plays T, then we get a payoff of 4 if we play R, and 3 if we play P or S. The payoff matrix of this new game RPST is given in Figure 13.4. Clearly the unique equilibrium is still for both players to randomize equally between R, P, and S. Now suppose we play our equilibrium strategy in the first game iteration, and the opponent plays T; no matter what action we played, we receive a payoff of at least 3. Suppose we play the pure strategy R in the second round in an attempt to exploit him (since R is our best response to T). In the worst case, our opponent will exploit us in the second round by playing P, and we will obtain payoff -1. But combined over both time steps, our payoff will be positive no matter what the opponent does at the second iteration. Thus, our strategy constituted a safe deviation from equilibrium. This was possible because of the existence of a ‘gift’ strategy for the opponent; no such gift strategy is present in standard RPS.

	R	P	S	T
R	0	-1	1	4
P	1	0	-1	3
S	-1	1	0	3

Figure 13.4: Payoff matrix of RPST.

13.3 Characterizing gifts

What exactly constitutes a gift? Does it have to be a strictly-dominated pure strategy, like T in the preceding example? What about weakly-dominated strategies? What about iterated dominance, or dominated mixed strategies? In this section we first provide some negative results which show that several natural candidate definitions of gifts strategies are not appropriate. Then we provide a formal definition of gifts and show that safe exploitation is possible if and only if such gift strategies exist.

Recent work has asserted the following:[2]

Assertion 1. *[117] An equilibrium strategy makes an opponent indifferent to all non-[weakly]-iteratively-dominated strategies. That is, to tie an equilibrium strategy in expectation, all one must do is play a non-[weakly]-iteratively-dominated strategy.*

This assertion would seem to imply that gifts correspond to strategies that put weight on pure strategies that are weakly iteratively dominated. However, consider the game shown in Figure 13.5.

	L	M	R
U	3	2	10
D	2	3	0

Figure 13.5: A game with a gift strategy that is not weakly iteratively dominated.

It can easily be shown that this game has a unique equilibrium, in which P1 plays U and D with probability $\frac{1}{2}$, and P2 plays L and M with probability $\frac{1}{2}$. The value of the game to player 1 is 2.5. If player 1 plays his equilibrium strategy and player 2 plays R, player 1 gets expected payoff of 5, which exceeds his equilibrium payoff; thus R constitutes a gift, and player 1 can safely deviate from equilibrium to try to exploit him. But R is not dominated under any form of dominance. This disproves the assertion, and causes us to rethink our notion of gifts.

Proposition 21. *It is possible for a strategy that survives iterated weak dominance to obtain expected payoff worse than the value of the game against an equilibrium strategy.*

We might now be tempted to define a gift as a strategy that is not in the support of any equilibrium strategy.

	L	R
U	0	0
D	-2	1

Figure 13.6: Strategy R is not in the support of an equilibrium for player 2, but is also not a gift.

However, the game in Figure 13.6 shows that it is possible for a strategy to not be in the support of an equilibrium and also not be a gift (since if P1 plays his only equilibrium strategy U, he obtains 0 against R, which is the value of the game).

Now that we have ruled out several candidate definitions of gift strategies, we now present our new definition, which we relate formally to safe exploitation in Proposition 22.

Definition 28. *A strategy σ_{-i} is a* gift strategy *if there exists an equilibrium strategy σ_i^* for the other player such that σ_{-i} is not a best response to σ_i^*.*[3]

[2]This is made as a statement of fact in prior work [117], and not in the form of an assertion.

[3]This definition of gift strategies coincides with the strategies for the opponent specified by the third step of a procedure for selecting a particular equilibrium of a (one-shot) two-player zero-sum game, known as Dresher's procedure [25, 111]. The procedure assumes the opponent will make a mistake (i.e., by playing a gift strategy), then selects a strategy that maximizes the minimum gain resulting from a possible mistake of the opponent. It has been shown that the strategies selected by this procedure coincide with the *proper equilibria* of the game [111],

When such a strategy σ_{-i} exists, player i can win an immediate profit beyond v_i against an opponent who plays σ_{-i} by simply playing the safe strategy σ_i^*; then he can play a potentially unsafe strategy (that has exploitability below some limit) in future iterations in an attempt to exploit perceived weaknesses of the opponent. Using this definition, RPS and the game depicted in Figure 13.6 have no gift strategies for either player, while T is a gift for player 2 in RPST, and R is a gift for player 2 in the game depicted in Figure 13.5.

Proposition 22. *Assuming we are not in a trivial game in which all of player i's strategies are minimax strategies, then non-stage-game-equilibrium safe strategies exist if and only if there exists at least one gift strategy for the opponent.*

Proof. Suppose some gift strategy σ_{-i} exists for the opponent. Then there exists an equilibrium strategy σ_i^* such that $u_i(\sigma_i^*, \sigma_{-i}) > v_i$. Let $\epsilon = u_i(\sigma_i^*, \sigma_{-i}) - v_i$. Let s_i' be a non-equilibrium strategy for player i. Suppose player i plays σ_i^* in the first round, and in the second round does the following: if the opponent did not play σ_{-i} in the first round, he plays σ_i^* in all subsequent rounds. If the opponent did play σ_{-i} in the first round, then in the second round he plays $\hat{\sigma}_i$, where $\hat{\sigma}_i$ is a mixture between s_i' and σ_i^* that has exploitability in $(0, \epsilon)$ (we can always obtain such a mixture by putting sufficiently much weight on σ_i^*), and he plays σ_i^* in all subsequent rounds. Such a strategy constitutes a safe strategy that deviates from stage-game equilibrium.

Now suppose no gift strategy exists for the opponent, and suppose we deviate from equilibrium for the first time in some iteration t'. Suppose the opponent plays a nemesis strategy at time step t' (to the strategy we are playing at time step t'), and plays an equilibrium strategy at all future time steps. Then we will win less than v^* in expectation against his strategy. Therefore, we cannot safely deviate from equilibrium. □

The following procedure gives an efficient algorithm, consisting of solving two linear programs (LPs), to determine whether a gift strategy for the opponent exists in a two-player zero-sum strategic-form game (and therefore whether safe exploitation is possible).

an equilibrium refinement concept defined by Myerson 1978. Thus, proper equilibrium strategies exploit all gift strategies, and one could equivalently define gift strategies as strategies that are not a best response to a proper equilibrium strategy of the opponent. One could view proper equilibria, as well as some other equilibrium refinement concepts (e.g., trembling-hand perfect equilibrium) as approaches for exploiting mistakes of the opponent in (non-repeated) games—although they are typically thought of as means to prescribe action probabilities for information sets that are reached with zero probability in equilibrium. In contrast, our main focus is on repeated games, although our techniques apply to single-shot games as well. Furthermore, we will show in Section 13.6.3 that even in single-shot games, our safe exploitative strategies differ from the strategies prescribed by subgame perfection [104], and thus our approach differs from all prior refinements that are further refinements of subgame perfection. So, our work can be viewed as providing novel equilibrium selection concepts and procedures. In broad strokes, at every point in the game, prior refinements try to play as well as possible against an (almost) rational opponent (e.g., one who "trembles" with small probability), while ours exploits an opponent model (which does not have to be rational in any way) as much as possible subject to safety. So, our approach can exploit the opponent significantly more than prior equilibrium refinements. (Some of the prior refinements also assume that we will "tremble" with small probability ourselves; this is not motivated by exploitation, but rather so that we know how to respond to actions further down the tree at information sets that would otherwise be reached with probability zero.) Another difference is that in our technique, a safe, maximally exploitative strategy can be computed in polynomial time both in theory and practice. In contrast, while proper equilibrium strategies can be computed in polynomial time in theory for both strategic-form and extensive-form games, those polynomial-time algorithms are numerically unstable in practice [81, 83].

1. Compute an equilibrium by solving the LP; this determines the value of the game to player i, v_i.
2. Solve the LP that maximizes the expected payoff of player i against the uniform random strategy of the opponent, subject to the constraints that player i's strategy is an equilibrium (these constraints will use v_i). Let $\hat{v}$ denote the optimal objective value of this LP.
3. If $\hat{v} > v_i$, then at least one gift strategy for the opponent exists; otherwise no gift strategies exist.

Proposition 23. *The above procedure determines in polynomial time whether a gift strategy for the opponent exists in a given two-player zero-sum game.*

Proof. Suppose a gift strategy s_{-i} for the opponent exists. Then there exists an equilibrium strategy σ_i^* such that $u_i(\sigma_i^*, s_{-i}) > v_i$. For every other strategy t_{-i} for the opponent, we have $u_i(\sigma_i^*, t_{-i}) \geq v_i$. Thus, player i's expected payoff of playing σ_i^* against the uniform random strategy will strictly exceed v_i, and $\hat{v} > v_i$.

Now suppose no gift strategies exist. Then for all equilibrium strategies σ_i^* and all strategies s_{-i} for the opponent, we have $u_i(\sigma_i^*, s_{-i}) = v_i$. Thus, all equilibrium strategies will obtain expected payoff v_i against the uniform random strategy, and we have $\hat{v} = v_i$.

The procedure is polynomial time since it consists of solving LPs of polynomial size (the LP formulations for computing a best response as well as the equilibrium constraints are described by, for example, Koller et al. [72]). □

13.4 Safety analysis of some natural exploitation algorithms

Now that we know it is possible to safely deviate from equilibrium in certain games, can we construct efficient procedures for implementing such safe exploitative strategies? In this section we analyze the safety of several natural exploitation algorithms. In short, we will show that all prior algorithms and natural other candidate algorithms are either unsafe or unexploitative. We introduce algorithms that are safe and exploitative.

13.4.1 Risk What You've Won (RWYW)

The "Risk What You've Won" algorithm (RWYW) is quite simple and natural; essentially, at each iteration it risks only the amount of profit won so far. More specifically, at each iteration t, RWYW plays an ϵ-safe best response to a model of the opponent's strategy (according to some opponent modeling algorithm M), where ϵ is our current cumulative payoff minus $(t-1)v^*$. Pseudocode is given in Algorithm 22.

Proposition 24. *RWYW is not safe.*

Proof. Consider RPS, and assume our opponent modeling algorithm M says that the opponent will play according to his distribution of actions observed so far. Since initially $k^1 = 0$, we must play our equilibrium strategy σ^* at the first iteration, since it is the only strategy with exploitability of 0. Without loss of generality, assume the opponent plays R in the first iteration. Our expected payoff in the first iteration is 0, since σ^* has expected payoff of 0 against R (or

Algorithm 22 Risk What You've Won (RWYW)

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
for $t = 1$ to $T$ do
    $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(\max\{k^t,0\})} M(\pi)$
    Play action $a_i^t$ according to $\pi^t$
    Update $M$ with opponent's actions, $a_{-i}^t$
    $k^{t+1} \leftarrow k^t + u_i(a_i^t, a_{-i}^t) - v^*$
end for
```

any strategy). Suppose we had played R ourselves in the first iteration. Then we would have obtained an actual payoff of 0, and would set $k^2 = 0$. Thus we will be forced to play σ^* at the second iteration as well. If we had played P in the first round, we would have obtained a payoff of 1, and set $k^2 = 1$. We would then set π^2 to be the pure strategy P, since our opponent model dictates the opponent will play R again, and P is the unique k^2-safe best response to R. Finally, if we had played S in the first round, we would have obtained an actual payoff of -1, and would set $k^2 = -1$; this would require us to set π^2 equal to σ^*.

Now, suppose the opponent had actually played according to his equilibrium strategy in iteration 1, plays the pure strategy S in the second round, then plays the equilibrium in all subsequent rounds. As discussed above, our expected payoff at the first iteration is zero. Against this strategy, we will actually obtain an expected payoff of -1 in the second iteration if the opponent happened to play R in the first round, while we will obtain an expected of 0 in the second round otherwise. So our expected payoff in the second round will be $\frac{1}{3} \cdot (-1) + \frac{2}{3} \cdot 0 = -\frac{1}{3}$. In all subsequent rounds our expected payoff will be zero. Thus our overall expected payoff will be $-\frac{1}{3}$, which is less than the value of the game; so RWYW is not safe. $\square$

RWYW is not safe because it does not adequately differentiate between whether profits were due to skill (i.e., from gifts) or to luck.

13.4.2 Risk What You've Won in Expectation (RWYWE)

A better approach than RWYW would be to risk the amount won so far *in expectation*. Ideally we would like to do the expectation over both our randomization and our opponent's, but this is not possible in general since we only observe the opponent's action, not his full strategy. However, it would be possible to do the expectation only over our randomization. For example, suppose we play according to the equilibrium σ^* at one iteration of RPS, and end up selecting action R, while the opponent selects action P; then our actual payoff is -1, but our expected payoff (over our own randomization) is 0. It turns out that we can indeed achieve safety using this procedure, which we call RWYWE. Pseudocode is given in Algorithm 23. Here $u_i(\pi_i^t, a_{-i}^t)$ denotes our expected payoff of playing our mixed strategy π_i^t against the opponent's observed action a_{-i}^t. The difference between RWYWE and RWYW is in the step for updating k^t: RWYW uses $u_i(a_i^t, a_{-i}^t)$ while RWYWE uses $u_i(\pi_i^t, a_{-i}^t)$. As the latter quantity will generally have lower variance than the former, one can view RWYWE as a variance-reduction procedure over RWYW.

Algorithm 23 Risk What You've Won in Expectation (RWYWE)

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
for $t = 1$ to $T$ do
    $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(k^t)} M(\pi)$
    Play action $a_i^t$ according to $\pi^t$
    The opponent plays action $a_{-i}^t$ according to unobserved distribution $\pi_{-i}^t$
    Update $M$ with opponent's actions, $a_{-i}^t$
    $k^{t+1} \leftarrow k^t + u_i(\pi_i^t, a_{-i}^t) - v^*$
end for
```

Lemma 3. *Let π be updated according to RWYWE, and suppose the opponent plays according to π_{-i}. Then for all $n \geq 0$,*

$$E[k^{n+1}] = \sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*.$$

Proof. Since $k^1 = 0$, the statement holds for $n = 0$. Now suppose the statement holds for all $t \leq n$, for some $n \geq 0$. Then

$$\begin{aligned}
E[k^{n+2}] &= E[k^{n+1} + u_i(\pi_i^{n+1}, a_{-i}^{n+1}) - v^*] \\
&= E[k^{n+1}] + E[u_i(\pi_i^{n+1}, a_{-i}^{n+1})] - E[v^*] \\
&= \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + E[u_i(\pi_i^{n+1}, a_{-i}^{n+1})] - v^* \\
&= \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + u_i(\pi_i^{n+1}, \pi_{-i}^{n+1}) - v^* \\
&= \sum_{t=1}^{n+1} u_i(\pi_i^t, \pi_{-i}^t) - (n+1)v^*
\end{aligned}$$

□

Lemma 4. *Let π be updated according to RWYWE. Then for all* $t \geq 1$, $k^t \geq 0$.

Proof. By definition, $k^1 = 0$. Now suppose $k^t \geq 0$ for some $t \geq 1$. By construction, π^t has exploitability at most k^t. Thus, we must have

$$u_i(\pi_i^t, a_{-i}^t) \geq v^* - k^t.$$

Thus $k^{t+1} \geq 0$ and we are done. □

Proposition 25. *RWYWE is safe.*

Proof. By Lemma 3,

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) = E[k^{T+1}] + Tv^*.$$

By Lemma 4, $k^{T+1} \geq 0$, and therefore $E[k^{T+1}] \geq 0$. So

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) \geq Tv^*,$$

and RWYWE is safe. □

RWYWE is similar to the Safe Policy Selection Algorithm (SPS) [80]. The main difference is that SPS uses an additional decay function $f : \mathbf{N} \to \mathbf{R}$ setting $k^1 \leftarrow f(1)$ and using the update step

$$k^{t+1} \leftarrow k^t + f(t+1) + u_i(\pi^t, a_{-i}^t) - v^*.$$

They require f to satisfy the following properties

1. $f(t) > 0$ for all t
2. $\lim_{T\to\infty} \frac{\sum_{t=1}^{T} f(t)}{T} = 0$

In particular, they obtained good experimental results using $f(t) = \frac{\beta}{t}$. They are able to show that SPS is safe in the limit as $T \to \infty$;[4] however SPS is arbitrarily exploitable in finitely repeated games. Furthermore, even in infinitely repeated games, SPS can lose a significant amount; it is merely the average loss that approaches zero. We can think of RWYWE as SPS but using $f(t) = 0$ for all t.

13.4.3 Best equilibrium strategy

Given an opponent modeling algorithm M, we could play the best Nash equilibrium according to M at each time step:

$$\pi^t = \text{argmax}_{\pi \in \text{SAFE}(0)} M(\pi).$$

This would clearly be safe, but can only exploit the opponent as much as the best equilibrium can, and potentially leaves a lot of exploitation on the table.

13.4.4 Regret minimization between an equilibrium and an opponent exploitation algorithm

We could use a no-regret algorithm (e.g., Exp3 [7]) to select between an equilibrium and an (unsafe) opponent exploitation algorithm at each iteration. As prior work has pointed out [80], this would be safe in the limit as $T \to \infty$. However, this would not be safe in finitely-repeated games. Even in the infinitely-repeated case, no-regret algorithms only guarantee that average regret goes to 0 in the limit; in fact, total regret can still grow arbitrarily large.

[4]We recently discovered a mistake in their proof of safety in the limit, though this does not affect the correctness of the result. A corrected proof is available at `http://webdocs.cs.ualberta.ca/~bowling/papers/04aaai-fallsymp-errata.pdf`.

13.4.5 Regret minimization in the space of equilibria

Regret minimization in the space of equilibria is safe, but again would potentially miss out on a lot of exploitation against suboptimal opponents. This procedure was previously used to exploit opponents in Kuhn poker [56].

13.4.6 Best equilibrium followed by full exploitation (BEFFE)

The BEFFE algorithm works as follows. We start off playing the best equilibrium strategy according to some opponent model M. Then we switch to playing a full best response for all future iterations if we know that doing so will keep our strategy safe in the full game (in other words, if we know we have accrued enough gifts to support full exploitation in the remaining iterations). Specifically, we play a full best response at time step t if the amount of gifts we have accumulated, k^t, is at least $(T - t + 1)(v^* - \epsilon)$, where ϵ is the exploitability of a full best response. Otherwise, we play the best equilibrium. Pseudocode is given in Algorithm 24.

Algorithm 24 Best Equilibrium Followed by Full Exploitation (BEFFE)

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
**for** $t = 1$ to $T$ **do**
    $\pi^t_{BR} \leftarrow \text{argmax}_\pi M(\pi)$
    $\epsilon \leftarrow v^* - \text{min}_{\pi_{-i}} u_i(\pi^t_{BR}, \pi_{-i})$
    **if** $k^t >= (T - t + 1)(v^* - \epsilon)$ **then**
        $\pi^t \leftarrow \pi^t_{BR}$
    **else**
        $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(0)} M(\pi)$
    **end if**
    Play action $a^t_i$ according to $\pi^t$
    The opponent plays action $a^t_{-i}$ according to unobserved distribution $\pi^t_{-i}$
    Update $M$ with opponent's actions, $a^t_{-i}$
    $k^{t+1} \leftarrow k^t + u_i(\pi^t_i, a^t_{-i}) - v^*$
**end for**
```

This algorithm is similar to the DBBR algorithm [32], which plays an equilibrium for some fixed number of iterations, then switches to full exploitation. However, BEFFE automatically detects when this switch should occur, which has several advantages. First, it is one fewer parameter required by the algorithm. More importantly, it enables the algorithm to guarantee safety.

Proposition 26. *BEFFE is safe.*

Proof. Follows by same reasoning as proof of safety of RWYWE, since we are playing a strategy with exploitability at most k^t at each iteration. □

One possible advantage of BEFFE over RWYWE is that it potentially saves up exploitability until the end of the game, when it has the most accurate information on the opponent's strategy (while RWYWE does exploitation from the start when the opponent model has noisier data). On the other hand, BEFFE possibly misses out on additional rounds of exploitation by waiting until the end, since it may accumulate additional gifts in the exploitation phase that it did not take into account. Furthermore, by waiting longer before turning on exploitation, one's experience of the opponent can be from the wrong part of the space; that is, the space that is reached when playing equilibrium but not when exploiting. Consequently, the exploitation might not be as effective because it may be based on less data about the opponent in the pertinent part of the space. This issue has been observed in opponent exploitation in Heads-Up Texas hold 'em poker [32].

13.4.7 Best equilibrium and full exploitation when possible (BEFEWP)

BEFEWP is similar to BEFFE, but rather than waiting until the end of the game, we play a full best response at each iteration where its exploitability is below k^t; otherwise we play the best equilibrium. Pseudocode is given in Algorithm 25.

Algorithm 25 Best Equilibrium and Full Exploitation When Possible (BEFEWP)

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
**for** $t = 1$ to $T$ **do**
    $\pi^t_{BR} \leftarrow \text{argmax}_\pi M(\pi)$
    $\epsilon \leftarrow v^* - \min_{\pi_{-i}} u_i(\pi^t_{BR}, \pi_{-i})$
    **if** $\epsilon <= k^t$ **then**
        $\pi^t \leftarrow \pi^t_{BR}$
    **else**
        $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(0)} M(\pi)$
    **end if**
    Play action $a^t_i$ according to $\pi^t$
    The opponent plays action $a^t_{-i}$ according to unobserved distribution $\pi^t_{-i}$
    Update $M$ with opponent's actions, $a^t_{-i}$
    $k^{t+1} \leftarrow k^t + u_i(\pi^t_i, a^t_{-i}) - v^*$
**end for**
```

Like RWYWE, BEFEWP will continue to exploit a suboptimal opponent throughout the match provided the opponent keeps giving us gifts. It also guarantees safety, since we are still playing a strategy with exploitability at most k^t at each iteration. However, playing a full best response rather than a safe best response early in the match may not be the greatest idea, since our data on the opponent is still quite noisy.

Proposition 27. *BEFEWP is safe.*

13.5 A full characterization of safe strategies in strategic-form games

In the previous section we saw a variety of opponent exploitation algorithms, some which are safe and some which are unsafe. In this section, we fully characterize the space of safe algorithms.

Informally, it turns out that an algorithm will be safe if at each time step it selects a strategy with exploitability at most k^t, where k is updated according to the RWYWE procedure. This does not mean that RWYWE is the only safe algorithm, or that safe algorithms must explicitly use the given update rule for k^t; it just means that the exploitability at each time step must be bounded by the particular value k^t, assuming that k had hypothetically been updated according to the RWYWE rule.[5]

Definition 29. *An algorithm for selecting strategies is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in \textit{SAFE}(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the rule

$$k^{t+1} \leftarrow k^t + u_i(\pi^t, a^t_{-i}) - v^*.$$

Proposition 28. *A strategy π (for the full game, not the stage game) is safe if and only if it is expected-profit-safe.*

Proof. If π is expected-profit-safe, then it follows that π is safe by similar reasoning to the proof of Proposition 25.

Now suppose π is safe, but at some iteration t' selects $\pi^{t'}$ with exploitability exceeding $k^{t'}$, as defined in Definition 29 (assume t' is the first such iteration); let e' denote the exploitability of $\pi^{t'}$. Suppose the opponent had been playing the pure strategy that selects action a^t_{-i} with probability 1 at each iteration t for all $t < t'$, and suppose he plays his nemesis strategy to $\pi^{t'}$ at time step t' (and follows a minimax strategy at all future iterations). Then our expected payoff in the first t' iterations is

$$\begin{aligned} & \sum_{t=1}^{t'-1} u_i(\pi^t, a^t_{-i}) + v^* - e' \\ < \; & \sum_{t=1}^{t'-1} u_i(\pi^t, a^t_{-i}) + v^* - k^{t'} \\ = \; & \sum_{t=1}^{t'-1} u_i(\pi^t, a^t_{-i}) + v^* - \left(\sum_{t=1}^{t'-1} u_i(\pi^t, a^t_{-i}) - (t'-1)v^* \right) \qquad (13.1) \\ = \; & t'v^*. \end{aligned}$$

In Equation 13.1, we use Lemma 3 and the fact that $E[k^{t'}] = k^{t'}$, since the opponent played a deterministic strategy in the first $t' - 1$ rounds. We will obtain payoff at most v^* at each future iteration, since the opponent is playing a minimax strategy. So π is not safe and we have a contradiction; therefore π must be expected-profit-safe, and we are done. □

[5]We could generalize the approaches to play strategies in SAFE($f(k^t)$) at each time step rather than SAFE(k^t), where $f(k^t) \leq k^t$ is an arbitrary function that is a potentially lower upper bound on the exploitability. This would result in a larger worst-case payoff guarantee when $f(k^t) < k^t$, but potentially at the expense of exploitation (since we are now restricting our space of strategies to a smaller set). In the opposite direction, we could also select strategies in SAFE($k^t + \delta$) for $\delta > 0$; this would lead to strategies that are approximately safe (within an additive factor δ), and potentially achieve higher levels of exploitation.

13.6 Safe exploitation in extensive-form games

In extensive-form games, we cannot immediately apply RWYWE (or the other safe algorithms that deviate from equilibrium), since we do not know what the opponent would have done at game states off the path of play (and thus cannot evaluate the expected payoff of our mixed strategy).

13.6.1 Extensive-form games of perfect information

In extensive-form games of perfect information, it turns out that to guarantee safety we must assume pessimistically that the opponent is playing a nemesis off the path of play (while playing his observed action on the path of play). This pessimism potentially limits our amount of exploitation when the opponent is not playing a nemesis, but is needed to guarantee safety. We present an extensive-form version of RWYWE below as Algorithm 26. As in the strategic-form case, the time step t refers to the iteration of the repeated game (not to the depth of the tree within a single iteration); the strategies refer to behavioral strategies for a single iteration of the full extensive-form game.

Algorithm 26 Extensive-Form RWYWE

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
for $t = 1$ to $T$ do
    $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(k^t)} M(\pi)$
    Play action $a_i^t$ according to $\pi^t$
    The opponent plays action $a_{-i}^t$ according to unobserved distribution $\pi_{-i}^t$
    Update $M$ with opponent's actions, $a_{-i}^t$
    $\tau_{-i}^t \leftarrow$ strategy for the opponent that plays $a_{-i}^t$ on the path of play, and plays a best response to $\pi^t$ off the path of play
    $k^{t+1} \leftarrow k^t + u_i(\pi_i^t, \tau_{-i}^t) - v^*$
end for
```

Lemma 5. *Let π be updated according to Extensive-Form RWYWE, and suppose the opponent plays according to π_{-i}. Then for all $n \geq 0$,*

$$E[k^{n+1}] \leq \sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*.$$

Proof. Since $k^1 = 0$, the statement holds for $t = 0$. Now suppose the statement holds for all

$t \leq n$, for some $n \geq 0$. Then

$$\begin{aligned}
E[k^{n+2}] &= E[k^{n+1} + u_i(\pi_i^{n+1}, \tau_{-i}^{n+1}) - v^*] \\
&= E[k^{n+1}] + E[u_i(\pi_i^{n+1}, \tau_{-i}^{n+1})] - E[v^*] \\
&\leq \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + E[u_i(\pi_i^{n+1}, \tau_{-i}^{n+1})] - v^* \\
&\leq \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + u_i(\pi_i^{n+1}, \pi_{-i}^{n+1}) - v^* \\
&= \sum_{t=1}^{n+1} u_i(\pi_i^t, \pi_{-i}^t) - (n+1)v^*
\end{aligned}$$

□

Lemma 6. *Let π be updated according to Extensive-Form RWYWE. Then for all $t \geq 1$, $k^t \geq 0$.*

Proof. By definition, $k^1 = 0$. Now suppose $k^t \geq 0$ for some $t \geq 1$. By construction, π^t has exploitability at most k^t. Thus, we must have

$$u_i(\pi_i^t, \tau_{-i}^t) \geq v^* - k^t.$$

Thus $k^{t+1} \geq 0$ and we are done. □

Proposition 29. *Extensive-Form RWYWE is safe.*

Proof. By Lemma 5,

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) \geq E[k^{T+1}] + Tv^*.$$

By Lemma 6, $k^{T+1} \geq 0$, and therefore $E[k^{T+1}] \geq 0$. So

$$\sum_{t=1}^{T} u_i(\pi_i^t, \pi_{-i}^t) \geq Tv^*,$$

and Extensive-Form RWYWE is safe. □

We now provide a full characterization of safe exploitation algorithms in extensive-form games—similarly to what we did for strategic-form games earlier in the paper.

Definition 30. *An algorithm for selecting strategies in extensive-form games of perfect information is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in \textit{SAFE}(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the same rule as Extensive-Form RWYWE.

Lemma 7. *Let π be updated according to Extensive-Form RWYWE, and suppose the opponent plays according to $\pi_{-i} = \tau_{-i}$, where τ_{-i} is defined in Algorithm 26. Then for all $n \geq 0$,*

$$E[k^{n+1}] = \sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*.$$

Proof. Since $k^1 = 0$, the statement holds for $t = 0$. Now suppose the statement holds for all $t \leq n$, for some $n \geq 0$. Then

$$\begin{aligned}
E[k^{n+2}] &= E[k^{n+1} + u_i(\pi_i^{n+1}, \tau_{-i}^{n+1}) - v^*] \\
&= E[k^{n+1}] + E[u_i(\pi_i^{n+1}, \tau_{-i}^{n+1})] - E[v^*] \\
&= \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + E[u_i(\pi_i^{n+1}, \tau_{-i}^{n+1})] - v^* \\
&= \left[\sum_{t=1}^{n} u_i(\pi_i^t, \pi_{-i}^t) - nv^*\right] + u_i(\pi_i^{n+1}, \pi_{-i}^{n+1}) - v^* \\
&= \sum_{t=1}^{n+1} u_i(\pi_i^t, \pi_{-i}^t) - (n+1)v^*
\end{aligned}$$

□

Proposition 30. *A strategy π in an extensive-form game of perfect information is safe if and only if it is expected-profit-safe.*

Proof. If π is expected-profit-safe, then it follows that π is safe by similar reasoning to the proof of Proposition 29.

Now suppose π is safe, but at some iteration t' selects $\pi^{t'}$ with exploitability exceeding $k^{t'}$, as defined in Definition 30; let e' denote the exploitability of $\pi^{t'}$. Suppose the opponent had been playing the pure strategy that selects action a_{-i}^t with probability 1 at each iteration t for all $t < t'$, and suppose he plays his nemesis strategy at time step t' (and follows a minimax strategy at all future iterations). Then our expected payoff is

$$\begin{aligned}
&\sum_{t=1}^{t'-1} u_i(\pi^t, a_{-i}^t) + v^* - e' \\
<\; &\sum_{t=1}^{t'-1} u_i(\pi^t, a_{-i}^t) + v^* - k^{t'} \\
=\; &\sum_{t=1}^{t'-1} u_i(\pi^t, a_{-i}^t) + v^* - \left(\sum_{t=1}^{t'-1} u_i(\pi^t, a_{-i}^t) - (t'-1)v^*\right) \\
=\; &t'v^*.
\end{aligned}$$

In Equation 13.2, we use Lemma 7 and the fact that $E[k^{t'}] = k^{t'}$, since the opponent played a deterministic strategy in the first $t' - 1$ rounds. We will obtain payoff at most v^* at each future iteration, since the opponent is playing a minimax strategy. So π is not safe and we have a contradiction; therefore π must be profit-safe, and we are done. $\square$

13.6.2 Extensive-form games of imperfect information

In extensive-form games of imperfect information, not only do we not see the opponent's action off of the path of play, but sometimes we do not even see his private information. For example, in an auction we may not see the opponent's valuation, and in a poker hand we will not see the opponent's private cards if he folds (while we will see them if neither player folds during the hand). The extent to which his private information is revealed will in general depend on the rules and information structure of the game. We consider the two cases—when his private information is observed and unobserved—separately.

Setting where the opponent's private information is observed at the end of the game

When the opponent's private information is observed at the end of each game iteration, we can play a procedure similar to Extensive-Form RWYWE. Here, we must pessimistically assume that the opponent would have played a nemesis at every information set off of the path of play (though we do not make any assumptions regarding his play along the path of play other than that he played action a^t_{-i} with observed private information θ^t_{-i}). Pseudocode for this procedure is given in Algorithm 27.

Algorithm 27 Safe exploitation algorithm for extensive-form games of imperfect information where opponent's private information is observed at the end of the game

```
$v^* \leftarrow$ value of the game to player $i$
$k^1 \leftarrow 0$
for $t = 1$ to $T$ do
    $\pi^t \leftarrow \text{argmax}_{\pi \in \text{SAFE}(k^t)} M(\pi)$
    Play action $a^t_i$ according to $\pi^t$
    The opponent plays action $a^t_{-i}$ with observed private information $\theta^t_{-i}$, according to unobserved distribution $\pi^t_{-i}$
    Update $M$ with opponent's actions, $a^t_{-i}$, and his private information, $\theta^t_{-i}$
    $\tau^t_{-i} \leftarrow$ strategy for the opponent that plays a best response to $\pi^t$ subject to the constraint that it plays $a^t_{-i}$ on the path of play with private information $\theta^t_{-i}$
    $k^{t+1} \leftarrow k^t + u_i(\pi^t_i, \tau^t_{-i}) - v^*$
end for
```

Proposition 31. *Algorithm 27 is safe.*

Proof. Follows by identical reasoning to the proof of Proposition 29, using the new definition of τ. $\square$

Definition 31. *An algorithm for selecting strategies in extensive-form games of imperfect information is* expected-profit-safe *if it satisfies the rule*

$$\pi^t \in SAFE(k^t)$$

at each time step t from 1 to T, where initially $k^1 = 0$ and k is updated using the same rule as Algorithm 27.

Proposition 32. *A strategy π in an extensive-form game of imperfect information is safe if and only if it is expected-profit-safe.*

Proof. Follows by similar reasoning to the proof of Proposition 30, using the new definition of τ. □

Setting where the opponent's private information is not observed

Unfortunately we must be extremely pessimistic if the opponent's private information is not observed, though it can still be possible to detect gifts in some cases. We can only be sure we have received a gift if the opponent's observed action would have been a gift for any possible private information he may have. Thus we can run an algorithm similar to Algorithm 27, where we redefine τ^t_{-i} to be the opponent's best response subject to the constraint that he plays a^t_{-i} with *some* private information.

The approaches from this subsection and the previous subsection can be combined if we observe some of the opponent's private information afterwards but not all. Again, we must be pessimistic an assume he plays a nemesis subject to the restriction that we plays the observed actions with the observed part of his private information.

13.6.3 Gift detection and exploitation within a game iteration

In some situations, we can detect gift actions early in the game that enable us to do safe exploitation even in the middle of a single game iteration. For example, an opponent may make a bet size known to be suboptimal early in a poker hand.

As a second, concrete example, consider the extensive-form game where players play the game depicted in Figure 13.7 followed by the game depicted in Figure 13.8. (The second game is the first game with all payoffs doubled, so the extensive-form game is not quite the same as just repeating the first stage game twice.) The unique stage-game equilibrium for both rounds is for P1 to play up (U and u) and for P2 to play left (L and ℓ). Down is strictly dominated for P1, and is therefore a gift. P2 can exploit this gift by playing r in the second round if he observes that player 1 has played D in the first round (since r outperforms ℓ against d). If P1 does in fact play D in the first round, P2 gains at least 3, and P2 will risk at most 2 by playing r in the second round; so this exploitation would be safe. The extensive-form representation is given in Figure 13.9. All subgame perfect equilibrium strategies [104] for P2 involve him playing ℓ (ℓ1/ℓ2/ℓ3/ℓ4), while there exist safe exploitative strategies that put positive weight on r3 and r4. Since subgame perfect equilibrium is the coarsest of the traditional equilibrium refinements, this example demonstrates that our approach provides a new equilibrium refinement that differs from all the traditional ones.

In general, one can use a variant of the Extensive-Form RWYWE update rule to detect gifts during a game iteration, where we redefine τ^t_{-i} to be the opponent's best response to π^t_i subject to the constraint that he has taken the observed actions along the path of play thus far. This allows us to safely deviate from equilibrium to exploit him even during a game iteration.

	L	R
U	4	5
D	1	0

Figure 13.7: Payoff matrix for first stage game of extensive-form game where we can detect and exploit a gift within a game iteration.

	ℓ	r
u	8	10
d	2	0

Figure 13.8: Payoff matrix for second stage game of extensive-form game where we can detect and exploit a gift within a game iteration.

13.7 Experiments

We ran experiments using the extensive-form imperfect-information variants of several of the safe algorithms presented in Section 13.4. The domain we consider is Kuhn poker [76], a simplified form of poker which has been frequently used as a test problem for game-theoretic algorithms [31, 51, 53, 56, 71].

13.7.1 Kuhn poker

Kuhn poker is a two-person zero-sum poker game, consisting of a three-card deck and a single round of betting. Here are the full rules:

- Two players: P1 and P2
- Both players ante \$1
- Deck containing three cards: K, Q, and J
- Each player is dealt one card uniformly at random
- P1 acts first and can either bet \$1 or check
    - If P1 bets, P2 can call or fold
        - If P1 bets and P2 calls, then whoever has the higher card wins the \$4 pot
        - If P1 bets and P2 folds, then P1 wins the entire \$3 pot
    - If P1 checks, P2 can bet \$1 or check.
        - If P1 checks and P2 bets, then P1 can call or fold.
            - If P1 checks, P2 bets, and P1 calls, then whoever has the higher card wins the \$4 pot
            - If P1 checks, P2 bets, and P1 folds, then B wins the \$3 pot
        - If P1 checks and P2 checks, then whoever has the higher card wins the \$2 pot

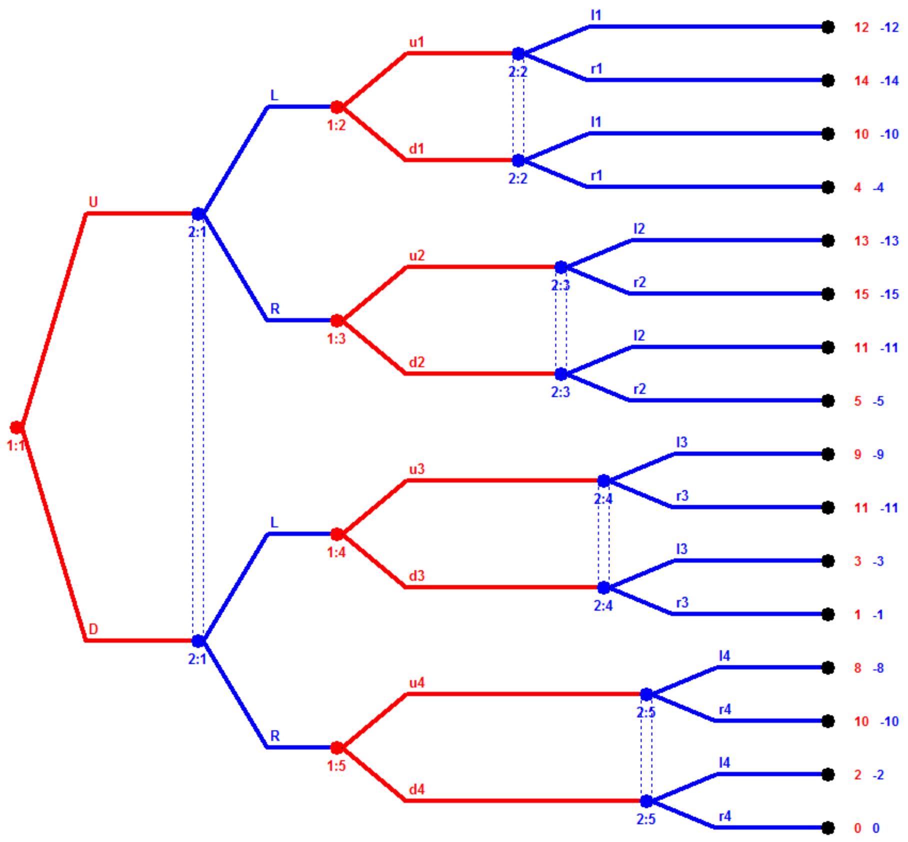


Figure 13.9: Extensive-form representation of a game where we can detect and exploit a gift within a game iteration. *X*:*Y* denotes the *Y*th information set of Player *X*. Dotted lines tie together nodes within an information set. At leaves of the game tree, the payoff of Player 1 is listed first, followed by the payoff of Player 2.

The value of the game to player 1 is $-\frac{1}{18} \approx -0.0556$. For any $0 \leq \alpha \leq 1$ the following strategy profile is an equilibrium (and these are all the equilibria) [76].

- P1 bets with a J in the first round with probability $\frac{\alpha}{3}$
- P1 always checks with a Q in the first round
- P1 bets with a K in the first round with probability α
- If P1 bets in the first round, then:
  - P2 always folds with a J
  - P2 calls with a Q with probability $\frac{1}{3}$
  - P2 always calls with a K
- If P1 checks in the first round, then:

    - P2 bets with a J with probability $\frac{1}{3}$
    - P2 always checks with a Q
    - P2 always bets with a K
- If P1 checks and P2 bets, then:
    - P1 always folds with a J
    - P1 calls with a Q with probability $\frac{\alpha}{3} + \frac{1}{3}$
    - P1 always calls with a K

Note that player 2 has a unique equilibrium strategy, while player 1 has infinitely many equilibrium strategies parameterized by a single value (α). In our experiments, we play the role of player 1 while the opponent plays the role of player 2.

Player 2 has four actions that are played with probability zero in his equilibrium strategy. These actions are 1) calling a bet with a J, 2) folding to a bet with a K, 3) checking a K if player 1 checks, and 4) betting a Q if player 1 checks. The first three of these are dominated, while the fourth is iteratively dominated. In this game, it turns out that the gift strategies for player 2 are exactly the strategies that play at least one of these four actions with positive probability.

13.7.2 Experimental setup

We experimented using several of the safe strategies described in Section 13.4—RWYWE, Best Equilibrium, BEFFE, and BEFEWP. For all algorithms, we used a natural opponent modeling algorithm similar to prior work [32, 56]. We also compare our algorithms to a full best response using the same opponent modeling algorithm. This strategy is not safe and is highly exploitable in the worst case, but it provides a useful metric for comparison.

Our opponent model assumes the opponent plays according to his observed frequencies so far, where we assume that we observe his hand at the end of each game iteration as prior work on exploitation in Kuhn poker has done [56]. We initialize our model by assuming a Dirichlet prior of 5 fictitious hands at each information set at which the opponent has played according to his unique equilibrium strategy, as prior work in Texas hold 'em has done [32].

We adapted all five algorithms to the imperfect-information setting by using the pessimistic update rule described in Algorithm 27. To compute ϵ-safe best responses, which is a subroutine in several of the algorithms, we used the procedure described in Section 13.7.3. We ran the algorithms against four general classes of opponents.

- The first class of opponent chooses a mixed strategy in advance that selects an action uniformly at random at each information set, then follows this strategy for all game iterations. (Similar random opponents were used also in prior work when experimenting on Kuhn poker [56]).
- The second opponent class is also static but more sophisticated. At each information set the opponent selects each action with probability chosen uniformly randomly within 0.2 of the equilibrium probability (recall that player 2 has a unique equilibrium strategy). Thus, these opponents play relatively close to optimally, and are perhaps more indicative of realistic suboptimal opponents. As in the first class, the strategy is chosen in advance, and played

in all iterations.

- The third class of opponents is dynamic. Opponents in this class play the first 100 hands according to a uniform random mixed strategy that is chosen in advance, then play a true best response (i.e., nemesis strategy) to our player's strategy for the remainder of the match. So, after the first 100 hands, we make the opponent more powerful than any real opponent could be in practice, by assuming that the opponent knows our mixed strategy for that iteration.
- Finally, the fourth class is the static unique Nash equilibrium strategy of player 2.

We ran all five algorithms against the same 40,000 opponents from each class. (For the dynamic opponents, this means that we selected 40,000 different choices of the mixed strategy σ' that is played for the initial 100 iterations; for each of these choices, we ran each of the five algorithms against an opponent algorithm that uses σ' for the first 100 iterations, followed by a best response to our strategy for the next 900 iterations.) Each match against a single opponent consisted of 1,000 hands, and we assume that the hands for both players were dealt identically for each of the algorithms against a given opponent (to reduce variance). For example, suppose algorithm A1 is dealt a K and opponent O is dealt a Q in the first hand of the match. Then in the runs of all other algorithms A against O, A is dealt a K and O is dealt a Q in the first hand. The 95% confidence intervals are reported for all experiments.

13.7.3 Algorithm for computing safe responses in extensive-form games

The following LP [72] efficiently computes a best response for player 1 to a given strategy y of player 2 in a two-player zero-sum extensive-form game of imperfect information. This algorithm utilizes the sequence form representation of strategies and runs in polynomial time.

$$\begin{array}{rl} \text{maximize}_x & x^T A y \\ \text{subject to} & x^T E^T = e^T \\ & x \geq 0 \end{array}$$

We modify this procedure as follows to compute an ϵ-safe best response for player 1 to strategy y of player 2, where v_1 is the value of the game to player 1 (and all matrices and vectors are as defined by Koller et al. [72]). This new formulation is used as a subroutine in several of the algorithms in the experiments.

$$\begin{array}{rl} \text{maximize}_x & x^T A y \\ \text{subject to} & x^T E^T = e^T \\ & x \geq 0 \\ & x^T A \geq -qF \\ & q[0] = \epsilon - v_1 \end{array}$$

13.7.4 Experimental results

The results from our experiments are given in Table 13.1. Against random opponents, the ordering of the performances of the safe algorithms was RWYWE, BEFEWP, BEFFE, Best Equilibrium (and all of the individual rankings are statistically significant using 95% confidence intervals). Against sophisticated static opponents the rankings of the algorithms' performances were identical, and all results are statistically significant except for the difference between RWYWE and BEFEWP. (Recall that the value of the game to player 1 is $-\frac{1}{18} \approx -0.0556$, so a negative win rate is not necessarily indicative of losing). In summary, against static opponents, our most aggressive safe exploitation algorithm outperforms the other safe exploitation algorithms that either stay within equilibrium strategies or use exploitation only when enough gifts have been accrued to use full exploitation, and furthermore all of our new algorithms outperform Best Equilibrium (which plays the best stage game equilibrium strategy at each iteration). Against the dynamic opponents, our algorithms are indeed safe as the theory predicts, while the best response algorithm does very poorly (and much worse than the value of the game). As a sanity check, the experiments show that against the equilibrium opponent, all the algorithms obtain approximately the value of the game as they should.

Table 13.1: Win rate in \$/hand of the five algorithms against opponents from each class. The $\pm$ given is the 95% confidence interval.

	Opponent			
	Random	Sophisticated static	Dynamic	Equilibrium
RWYWE	0.3636 ± 0.0004	-0.0110 ± 0.0004	-0.02043 ± 0.00044	-0.0556 ± 0.0004
BEFEWP	0.3553 ± 0.0004	-0.0115 ± 0.0004	-0.02138 ± 0.00045	-0.0556 ± 0.0004
BEFFE	0.1995 ± 0.0004	-0.0131 ± 0.0004	-0.03972 ± 0.00044	-0.0556 ± 0.0004
Best Equilibrium	0.1450 ± 0.0004	-0.0148 ± 0.0004	-0.03522 ± 0.00044	-0.0556 ± 0.0004
Best response	**0.4700 ± 0.0004**	**0.0548 ± 0.0004**	**-0.12094 ± 0.00039**	**-0.0556 ± 0.0004**

In some matches, RWYWE accumulates gifts throughout the match, and k^t increases steadily. An example of the graph of profit and k^t for one such opponent is given in Figure 13.10. In this situation, the opponent is frequently giving us gifts, and we quickly start playing (and continue to play) a full best response according to our opponent model.

In other matches, k^t remains very close to 0 throughout the match, despite the fact that profits are steadily increasing; one such example is given in Figure 13.11. Against this opponent, we are frequently playing an equilibrium or an ϵ-safe best response for some small ϵ, and only occasionally playing a full best response. Note that k^t falling to 0 does not necessarily mean that we are losing or giving gifts to the opponent; it just means that we are not completely sure about our worst-case exploitability, and are erring on the side of caution to ensure safety.

13.8 Summary and extensions

We showed that safe opponent exploitation is possible in certain games, disproving a recent (incorrect) statement. Specifically, profitable deviations from stage-game equilibrium are possible

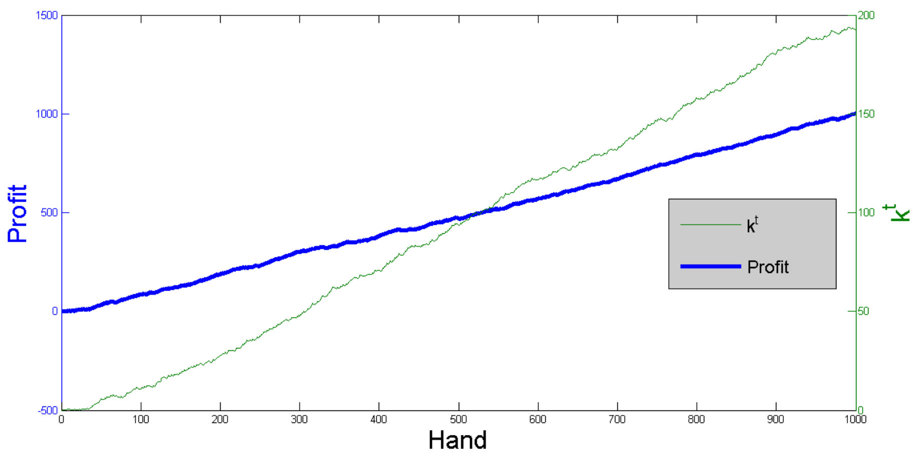


Figure 13.10: Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, both k^t and profits steadily increase.

in games where 'gift' strategies exist for the opponent, which we defined formally and fully characterized. We considered several natural opponent exploitation algorithms and showed that some guarantee safety while others do not; for example, risking the amount of profit won so far is not safe in general, while risking the amount won so far *in expectation* is safe. We described how some of these algorithms can be used to convert *any* opponent exploitation architecture into a safe one. Next we provided a full characterization of safe algorithms for strategic-form games, which corresponds to precisely the algorithms that are expected-profit safe. We also provided algorithms and full characterizations of safe strategies in extensive-form games of perfect and imperfect information.

In our experiments against static opponents, several safe exploitation algorithms significantly outperformed an algorithm that selects the best Nash equilibrium strategy; thus we conclude that safe exploitation is feasible and potentially effective in realistic settings. Our most aggressive safe exploitation algorithm outperformed the other safe exploitation algorithms that use exploitation only when enough gifts have been accrued to use full exploitation. In experiments against an overly strong dynamic opponent that plays a nemesis strategy after 100 iterations, our algorithms are indeed safe as the theory predicts, while the best response algorithm does very poorly (and much worse than the value of the game).

The approach can also be used in settings where we do not have an exact game model—such as in (cyber)security games—because we only need to lower bound the gifts that the opponent has given us and upper bound the maximum expected loss from the exploitative action we are planning to take currently. It can also be extended easily to the setting where we are able to lose some amount $\delta > 0$, as opposed to guaranteeing total safety.

Several challenges must be confronted before applying safe exploitation algorithms to larger extensive-form games of imperfect information, such as Texas hold 'em poker. First, the best known technique for computing ϵ-safe best responses involves solving a linear program on par with performing a full equilibrium computation; performing such computations in real time, even

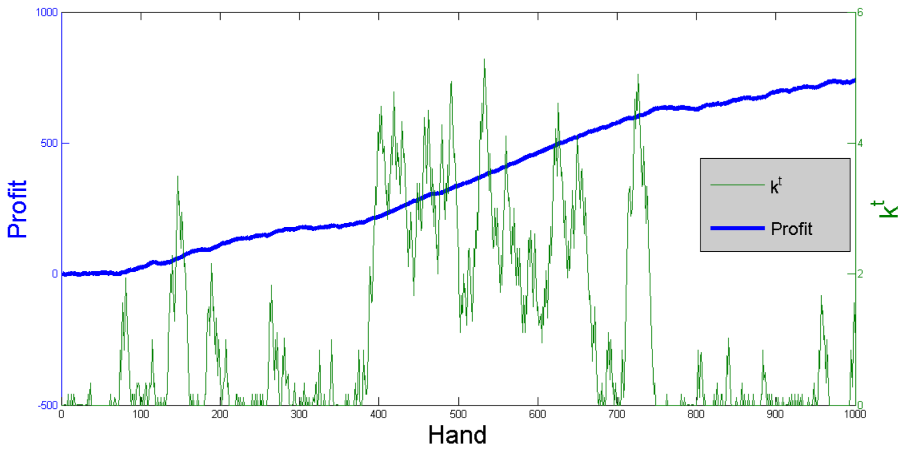


Figure 13.11: Profit and k^t over the course of a match of RWYWE against a random opponent. Profits are denoted by the thick blue line using the left Y axis, while k^t is denoted by the thin green line and the right Y axis. Against this opponent, k^t stays relatively close to 0 throughout the match, while profit steadily increases.

in a medium-sized abstracted game, is not feasible in Texas hold 'em. Perhaps the approaches of BEFEWP and BWFEE, which alternate between equilibrium and full best response, would be preferable to RWYWE in such games, since a full best response can be computed much more efficiently in practice than an ϵ-safe best response. In addition, perhaps performance can be improved if we integrate our algorithms with lower-variance estimators of our winnings due to the opponent's mistakes [14, 122, 124].

Part V

Conclusion

Chapter 14

Reiteration of Game-Solving Challenges

Designing strong agents in multiagent strategic settings is a challenging problem for several reasons. First, there is a fundamental conceptual challenge: in games with more than two players and games that are not zero sum, there is no theoretical justification for why an agent should follow a Nash equilibrium strategy. There may exist many Nash equilibria, each yielding different payoffs to the different agents; if we follow one equilibrium while the opponents follow a different one, we can perform arbitrarily poorly; and even if all agents follow the same equilibrium as we have computed, one agent's deviation could reduce our payoff in addition to his own. Even in two-player zero-sum games, refinements of the Nash equilibrium concept have been proposed that are desirable in various ways, and it is not clear that simply following an arbitrary Nash equilibrium is the best thing to do, even ignoring the aspect of opponent exploitation.

Second, even if we decided that a Nash equilibrium was the correct solution to compute, computing one is hard from a complexity-theoretic perspective in many important game classes. It is PPAD-hard in games with more than two players and non zero-sum games, even for strategic-form games, and therefore it is widely conjectured that no polynomial-time algorithms exist.

Third, even for classes where computing a Nash equilibrium can be done theoretically in polynomial time, such as two-player zero-sum extensive-form games of imperfect information, it can be extremely difficult to compute, or even approximate, one in practice for many interesting games, due to their massive state spaces. For example, a popular variant of poker—two-player no-limit Texas hold 'em—has around 10^{165} nodes in its game tree, while the best exact algorithm (linear program) scales to approximately 10^8 states, and the best algorithms for approximating an equilibrium scale to approximately 10^{17} states.

And fourth, even if we were able to compute a Nash equilibrium for game classes where it is a compelling solution concept—e.g., two-player zero-sum games—we could potentially achieve a significantly higher payoff by learning to exploit weaknesses of opponents than by following equilibrium strategies in each round. Ideally we would like to perform such opponent exploitation in a *robust* way that performs well even against strong, potentially deceptive opponents.

Chapter 15

Contributions

I have presented new approaches and paradigms that address these challenges. The leading paradigm for solving large imperfect-information games consists of three main components—abstraction, equilibrium computation, and reverse mapping. The approaches within this paradigm are geared at addressing the third main game-solving challenge: that of approximating equilibrium strategies in settings that are too large to apply polynomial-time algorithms.

I have presented new approaches for each step of the paradigm. I presented the first algorithm for performing potential-aware imperfect-recall abstraction using the earth-mover's distance metric. I presented a new hierarchical abstraction algorithm that enables massive distributed equilibrium in large imperfect-information games. I presented a new algorithm for translating actions for the opponent that have been removed from the action abstraction. I presented new post-processing techniques that improve performance in a variety of domains. And I presented algorithms for approximating equilibrium strategies in multiplayer stochastic imperfect-information games. Many of these approaches were utilized by our two-player no-limit Texas hold 'em agent that won the 2014 AAAI Annual Computer Poker Competition.

I have also presented two new game-solving paradigms. The first involves solving relevant portions of the game that we have actually reached in real time to a greater degree of accuracy than the abstract approximate equilibrium strategies that have been computed offline. I presented theoretical analysis that shows that this approach can produce highly exploitable strategies in certain games, while it guarantees low exploitability in others. I described several clear benefits provided by endgame solving in large imperfect-information games, and presented an efficient algorithm for conducting it that is significantly more efficient than the naïve approach (that approach would require $O(n^2)$ lookups in the strategy table, while our approach requires $O(n)$). I showed that utilizing this paradigm leads to significantly stronger performance against the strongest prior no-limit Texas hold 'em agents.

The second new paradigm leverages qualitative representations of the structure of equilibrium strategies to improve the speed of equilibrium finding. I presented a complete algorithm that guarantees computing an equilibrium assuming one of a set of qualitative representations correctly describes the structure of equilibrium strategies. I presented a set of three qualitative strategy models that describe equilibrium strategies for the final round of limit Texas hold 'em given any distribution of private information for the agents as input. In addition to leading to faster equilibrium-finding algorithms, this paradigm can be used to represent and extract knowledge from strategies that is human understandable. The approaches in the leading paradigm typically produce strategies that are represented as massive binary files and unintelligible humans. Simple qualitative model representations of these complex strategies can lead to improved human understanding and decision making.

To address the fourth game-solving challenge, I presented a new algorithm for opponent modeling in large imperfect-information games that successfully learns to exploit a variety of opponents very quickly. I also presented new algorithms for performing opponent exploitation *safely* that guarantee strong performance even against strong dynamic opponents. These algorithms can be viewed as meta-procedures that take any opponent modeling algorithm as an input, and ensure safety by restricting the amount we risk attempting to do the exploitation.

The approaches are domain independent. Many of them have been benchmarked in well-studied challenge problems, such as no-limit Texas hold ’em, for concreteness; however they apply to very broad classes, which include many important situations beyond those considered. For example, most of the approaches apply to any extensive-form imperfect-information game.

While most of the approaches are presented in the context of two-player zero-sum games, they are also applicable to multiplayer games and games that are not zero sum, which addresses the first and second challenges. Many of the approaches are applicable to these games directly with no modification of the analysis needed: for example, the abstraction, translation, and post-processing algorithms. The qualitative model approach is also applicable with a modification of analysis, which is presented. The opponent exploitation algorithm would be applicable if we store observations of joint opponent action profiles, since best responses to given strategies can be computed efficiently in these game classes. The endgame solving approach would also be directly applicable to games with more than two agents if the resulting endgames contain only two agents: this is often the case, for example, in many popular poker variants, where there may be 6 or 9 players at the table to start the hand, but only 2 players remain in the hand after the first or second round of betting. Endgame solving could take any distribution of private information as inputs for the two-player endgame, regardless of the number of agents who were present initially. I expect this approach to be extremely useful for future research in games with more than two agents, as it is more difficult to compute strong game-theoretic strategies offline, and real-time solving of specific portions of the tree will be important. Endgame solving would also be applicable to endgames containing more than two agents if an equilibrium-finding algorithm is available that solves such games effectively. Furthermore, for any of these game sizes, endgame solving can take arbitrary distributions of private information for the agents as inputs; in particular, one need not input the distributions induced from the precomputed approximate equilibrium strategies using Bayes’ rule, and could input an opponent model that differs significantly from equilibrium. Thus, endgame solving can be an extremely useful tool for opponent exploitation, in addition to the initial use of approximating equilibrium strategies.

The equilibrium-finding algorithms presented also apply directly to games with more than two agents, though they do not have guarantees of convergence to equilibrium. Both the distributed equilibrium-solving algorithm for imperfect-information games and the algorithms for stochastic imperfect-information games can be applied directly to games with more than two agents: for the latter, we show that the algorithms can never converge to a non-equilibrium (though they are not guaranteed to converge at all), and we experimented on a three-player game. Variants of counterfactual-regret minimization similar to the distributed algorithm have been demonstrated to be very successful in large imperfect-information games, though the main theoretical result thus far is relatively modest: that the computed strategies are guaranteed to not put weight on dominated actions or strategies [39]. I also presented experimental results for the qualitative model algorithm on a three-player imperfect-information game.

The safe exploitation algorithms can also be applied to games with more than two agents; however, they would have to substitute the *maximin* value for the game value in all of the algorithms and theoretical guarantees, since the game value is not defined for these games. The maximin value is the payoff that can be guaranteed in the event of worst-case opponents. In general playing a *maximin strategy* that guarantees this payoff would be very conservative, and therefore the new theoretical guarantees would be significantly more modest than in the two-player zero-sum setting.

Chapter 16

Future Research

I conclude by outlining several key problems for future study that this thesis paves the road for.

1. **What is the "right" game-theoretic solution concept for multiplayer and non zero-sum games, and to what extent will this depend on the particular game or game class?**

   Nash equilibrium has no theoretical justification in these game classes, and playing one could perform arbitrarily poorly at least in theory against certain opponents in certain games. Empirically, agents that are computed using algorithms that aim to approximate Nash equilibrium strategies have performed well in the 3-player limit Texas hold 'em poker competition [39], though there have not been many participants traditionally in that division. I personally believe that an agent that followed Nash equilibrium strategies would perform extremely well in variants of poker with more than two players: this opinion is largely based on my experience as a professional high-stakes poker player who has specialized in variants with more than two players. There has been significant recent interest in game-theoretic analysis within the poker community for multiplayer variants of poker, e.g., [4, 57].

2. **What is the "right" refinement of Nash equilibrium for both two-player zero-sum games and for multiplayer/non zero-sum games, and to what extent will this depend on the particular game or game class?**

   I have shown that undominated Nash equilibrium is a useful concept in certain interesting games, that both it and ϵ-quasi perfect equilibrium can be computed efficiently in practice in large imperfect-information games, that both can be efficiently integrated with endgame solving, and that preliminary experiments indicate that undominated equilibrium improves performance in practice for endgame solving.

   It would be interesting to run more experiments with these, both in poker and in other domains, to understand better when they improve performance over computing just a Nash equilibrium. It would also be interesting to examine whether algorithms can be devised for other refinements (e.g., proper equilibrium) that will scale to large games.

   I think that we are very far from understanding the correct equilibrium refinement concept in even the simplest possible no-limit poker game. I have described the Clairvoyance game in Section 7.2.1, and presented the infinite set of equilibrium strategies in Section 7.5. I argued that a particular equilibrium is compelling: the one where we call a bet of size

x with probability $\frac{1}{1+x}$ for all x (this is the same equilibrium that has previously been computed by Ankenman and Chen for this game [4]). The intuition for why I believe this is compelling is that if the opponent bets x, as opposed to the "optimal" size n that he should bet in equilibrium, then a reasonable deduction is that he isn't even aware that n would have been the optimal size, and believes that x is optimal. Therefore, it would make sense to play a strategy that is an equilibrium in the game where the opponent is restricted to only betting x (or to betting 0, i.e., checking). Doing so would correspond to the particular equilibrium that I have prescribed. The other equilibria pay more heed to the concern that the opponent could exploit us by deviating to bet x instead of n; but in fact, I argue that we need not be as concerned about this possibility, since a rational opponent who knew to bet n would not be betting x.

Interestingly, the equilibrium I have singled out as being compelling does not coincide to any traditional Nash equilibrium refinements. One popular refinement is the normal-form proper equilibrium (NFPE). Based on personal communication with Troels Sørensen and Jiří Čermák, we have computed the unique NFPE for the clairvoyance game where player 1 is allowed to bet 0, 1, or 2. It differs from the equilibrium I propose; in the NFPE P2 calls vs. a bet of 1 with probability $\frac{5}{9}$, while in the one I prescribe above he calls with probability $\frac{1}{2}$.

I believe that all of the (infinitely many) equilibria for this game satisfy each of the popular refinement concepts besides NFPE. So, the one I believe is compelling does not coincide with traditional solution concepts, and perhaps this will motivate a new refinement concept that can be computed efficiently.

3. **Can we develop approaches for approximating Nash equilibrium in multiplayer and non zero-sum games that work well in practice?**

   I have presented algorithms that provably compute ϵ-equilibrium strategies in a 3-player poker tournament endgame. However, I cannot prove that these algorithms will always converge, and I have not experimented with them on other games. Other algorithms based on counterfactual regret minimization have been successfully applied to other variants of poker, e.g., [1, 39]. However, it is not clear whether the strategies that they compute are close to Nash equilibrium. Counterfactual regret minimization has been shown to converge to equilibrium in some games, but to not converge in others.

4. **Can we develop improved theoretical guarantees for existing algorithms for approximating Nash equilibrium in multiplayer non zero-sum games?**

   For the algorithms that I have presented for multiplayer stochastic imperfect-information games, I prove that if they converge, then the resulting strategies are guaranteed to be an equilibrium. Recent work shows that the strategies output by counterfactual regret minimization are guaranteed to not contain any dominated actions [39]. It would be nice to prove stronger theoretical results for these algorithms, at least for certain interesting game classes.

5. **Can we develop practical abstraction algorithms with theoretical guarantees or provide theoretical guarantees for existing algorithms?**

   The main abstraction algorithms that have been successful in practice for agents in large imperfect-information games are heuristic and have no theoretical guarantees. It is extremely difficult to prove meaningful theoretical guarantees when performing such a large degree of abstraction, e.g., approximating a game with 10^{165} states by one with 10^{14} states.

There has been some recent work done on abstraction algorithms with theoretical guarantees, though that work does not scale to games nearly as large as no-limit Texas hold 'em. One line of work performs lossless abstraction, that guarantees that the abstract game is exactly isomorphic to the original game [44]. This work has been applied to compute equilibrium strategies in Rhode Island hold 'em, a medium-sized (3.1 billion nodes) variant of poker.

Recent work has also presented the first lossy abstraction algorithms with bounds on the solution quality [75]. However, the algorithms are based on integer programming formulations, and only scale to a tiny poker game with a 5 card deck.

It would be very interesting to bridge this gap between heuristics that work well in practice for large games with no theoretical guarantees, and the approaches with theoretical guarantees that have more modest scalability.

6. **Can we provide a theoretical proof that purification leads to improved performance in 4x4 matrix games with uniform [-1,1] payoffs, and can we generalize this result?**

   In Section 8.4 I demonstrated, via simulation, that applying purification to the equilibrium of a random 3×3 abstraction improves performance with statistical significance against the full equilibrium strategy for the 4×4 game. It would be interesting to prove this formally and to generalize the result to games of arbitrary dimension.

   In Observation 1 I described an observation on the specific structure of the games where purification improves performance. Again this observation was based purely on simulations, which were statistically significant, and I would like to prove this, and generalized versions, theoretically.

7. **Why do post-processing approaches such as purification and thresholding improve performance in practice?**

   While experiments show that the new post-processing techniques I have presented consistently improve performance across several domains and against many different opponents, the theoretical reason for this observed improvement is unclear. There are two main hypotheses that come to mind. First, as described in Chapter 8 and depicted in Figure 8.1, there is an overfitting phenomenon, as the full-game exploitability of strategies computed by an equilibrium-finding algorithm within an abstraction may increase, despite the fact that the exploitability within the abstraction continues to decrease. The post-processing techniques can help mitigate this overfitting by adding preference for higher-probability actions since it may be better to generally extrapolate that the actions are "good" than to try to infer the specific probabilities from the abstract equilibrium.

   A second hypothesis is that the techniques compensate for the failure of iterative approximate equilibrium-finding algorithms to fully converge within given amounts of time. For example, as described in Section 10.2.1, our 2012 no-limit Texas hold 'em agent had an exploitability of 800 milli big blinds per hand (mbb/h) even within the abstract game [33] (by comparison, an agent that folds every hand would only have an exploitability of 750 mbb/h). When the equilibrium-finding algorithms are far from convergence, they may still place significant probability mass on "bad" actions. For instance, if the computed strategies say to take an action with probability 0.08, it is very reasonable to assume (though of course not certain) that the probability would have fallen to 0 had we run the algorithm longer.

It would be interesting to distill out which of these factors is most influential behind the successful performance of the approaches. My personal belief is that the latter issue, on the convergence of equilibrium-finding algorithms, plays a bigger role. This is based on empirical observations that the algorithms take an extremely long time to converge on the games of the magnitude currently being solved, and often produce strategies with high exploitability even within the abstraction, as described above.

However, the fact that the techniques improve performance even in small matrix games and the smaller Leduc hold 'em poker variant, where the equilibrium finding is exact and therefore the above issue does not arise, implies that the convergence issue is not the only explanation for the observed improvement, and overfitting is certainly likely to play at least some role.

Geoff Gordon has recently proposed a third hypothesis related to covariances and strategy divergence that could also help explain the observed performance improvement; this merits further study as well.

8. **Is our action translation mapping "optimal?"**

   The new action translation mapping I presented in Chapter 7, the pseudo-harmonic mapping, produced the lowest exploitability in all of the games considered. Not surprisingly it produced zero exploitability in the small game from which it was derived (the clairvoyance game). It also produced by far the lowest exploitability in Kuhn poker for a variety of stack sizes and action abstractions, and for no-limit Leduc hold 'em for a popular action abstraction. Alarmingly however, the exploitability was still relatively far from zero for the latter game, despite being lower than the exploitabilities of the other approaches.

   There are three possible explanations for this. First, it is possible that better translation mappings exist that would produce lower exploitability given an action abstraction. Second, it is possible that no better translation mappings exist, and that no mapping can produce lower exploitability using the particular action abstraction in Leduc hold 'em, and that the high exploitability is simply due to the abstraction being very coarse. And third, it is possible that no mappings within our framework of action translation can produce lower exploitability when paired with this action abstraction; however, different frameworks of approaches might improve performance. For example, our framework only allows the translation mapping to consider the two neighboring action sizes in the action abstraction, and not any additional game state information. Furthermore, for poker, it restricts the agent to always check or call in response to a bet that is mapped down to zero, to always go all-in ourselves in response to a bet that is mapped to an all-in, and to get confused sometimes when we get "off-tree" after translating opponent actions, which causes us to have a misperception of the pot size. While our solutions for these issues are natural, it is possible that more complex frameworks can address them even better, preferably in a theoretically-principled way.

Regarding the first point, I would be very surprised if a better mapping within the framework described than the pseudo-harmonic mapping existed, due to the theoretical results and derivation from analytical solutions of fundamental poker games. This could be assessed conclusively for the no-limit Leduc issue posed above. I have formulated a linear program that computes the "optimal" translation mapping, given a particular action abstraction and an abstract equilibrium. Implementing and applying this would determine whether our approach is optimal within the class of translation mappings, or whether better mappings exist.

Regarding the second point, I think that the action abstraction of fold-pot-call-allin in this game is extremely coarse, and am not surprised if even the strategy with lowest exploitability within that abstraction still has high full-game exploitability.

Regarding the third point, I think addressing these other fundamental limitations of the action translation framework is a very interesting and important direction for future study. While the approaches which I have described above have worked well so far in practice for no-limit Texas hold 'em, they are somewhat game-specific and not theoretically principled, and it would be nice to explore other domains and gain a better theoretical perspective of these issues.

It would also be very interesting to consider additional action translation axioms; perhaps there exist further natural axioms that only the pseudoHarmonic mapping satisfies (currently the randomized arithmetic mapping also satisfies all the axioms, though it has extremely high exploitability in certain games, which we show). Separately, it would be interesting to consider whether this mapping is optimal for certain general game classes.

9. **Solving the "off-tree problem."**

   The "off-tree problem," which is discussed in Section 10.2.4, occurs when the opponent has taken an action outside our action abstraction which we have had to translate back to an action in the abstraction. Even if we are using a very sophisticated action translation mapping, such a mapping will still abstract away relevant game state information. For example, in poker, the agent's perception of the pot size will be incorrect after translating an action for the opponent outside the betting tree. See Section 10.2.4 for a specific example of this phenomenon. Some agents, such as prior versions of the University of Alberta's Hyperborean agent, attempt to mitigate this problem by specifically taking actions aimed to get us back on the tree (e.g., making a bet that we would not ordinarily make to correct for the pot size disparity described above). However, this is problematic too, as it requires us to take an undesirable action.

   The endgame solving approach I have presented provides a solution to this problem by inputting the correct pot size to the endgame solving algorithm in the case of poker, even if this differs from what our perception of it was at that point due to the opponent taking an action outside of the action abstraction. In general, real-time endgame solving could correct for any misperceptions in game state information that have been accumulated along the course of game play.

   However, this solution would only apply to the endgames. This would not apply to the trunk portion of the game prior to the endgames, or if we are not applying endgame solving at all. It would be interesting to explore additional approaches geared specifically at addressing the off-tree problem independently of endgame solving.

10. **Does endgame solving decrease exploitability in no-limit Texas hold 'em endgames?**

    As I pointed out, endgame solving has theoretical limitations and can significantly increase exploitability in certain games, even in games with a unique equilibrium and single endgame. However, I showed that it produces low exploitability in other games, and I described several further theoretical benefits in Section 10.2. While it improved performance empirically in no limit Texas hold 'em endgames, it is not clear if it reduced exploitability.

    I personally believe that our endgame solving algorithm leads to a significant decrease in exploitability in the no-limit Texas hold 'em endgames. The main intuition the fact that we compute an exact equilibrium (within an extremely good and fine-grained abstraction, both for states of information and actions), which would guarantee that our strategies would be well-balanced for all action sizes; i.e., our ratios of "value bets" with strong hands to "bluffs" with weak hands would be appropriate for all bet sizes, and our calling frequencies with mediocre hands would be similarly appropriate (e.g., as indicated by reasoning along similar lines to the analysis presented in Section 7.5). By contrast, currently the strategies output by counterfactual regret minimization can be arbitrarily "unbalanced" for particular bet sizes, though they are of course guaranteed to be an exact equilibrium in the limit as the algorithm converges. The opponent can significantly exploit us in this case by, e.g., betting the specific size at which our strategies are most "unbalanced."

    It seems difficult to assess the exploitability of the approach in no-limit Texas hold 'em endgames, as a full-game best response calculation is infeasible. One idea would be to compute the exploitability of the original strategies (not involving endgame solving) just within the abstraction in the endgames. Endgame solving guarantees an exact equilibrium within the abstract endgame, since it is using an exact equilibrium solver. If the original strategies have extremely high exploitability within the abstract endgames, that would indicate that they likely have extremely high exploitability in the full game as well (if the original strategies constituted an exact equilibrium, then they would be guaranteed to have exploitability zero in the endgames induced by following the original strategies for the trunk). Though it would not be clear what threshold to use for determining whether the exploitability within the abstract endgames is high or low. This comparison would not conclusively indicate whether endgame solving improves exploitability, but an extremely high value for this quantity would provide compelling evidence that it does.

    Many strong human poker players use algorithms akin to our endgame solving algorithm to improve their abilities. For example, a software program calls GTORangeBuilder solves endgames determined by distributions of private information for both agents specified by the user [52]. This program was created subsequently to the first version of our work on endgame solving, and is significantly less sophisticated; e.g., it restricts the agents to follow a much coarser action abstraction algorithm than our approach scales to. This tool illustrates several of the key extensions of endgame solving that I have described in Chapter 15: that it can apply to games with more than two agents as long as only two remain in the endgame, and that it can take any distribution of private information as input, and can therefore be utilized for opponent exploitation in addition to the initial goal of approximating equilibrium strategies.

11. **Techniques for computing best responses and exploitability in imperfect-recall games**

    As described above, it is not feasible to compute best responses in extremely large imperfect-information games, and it is particularly challenging in games with imperfect recall (the abstraction algorithms used by the strongest agents have imperfect recall). This makes it extremely difficult to assess the strength of strategies we have computed other than by just examining their performance against other agents. A recent breakthrough allowed for best response computation in the limit Texas hold 'em variant [66] (though the need for such computation has been diminished due to recent computation of near exact equilibrium strategies for that variant [15]), though approaches that are scalable to no-limit Texas hold 'em even using coarse action abstractions have been evasive.

12. **For what games and decompositions does endgame solving improve performance?**

    I have presented an example game in which endgame solving leads to a significantly high exploitability, an example game where it leads to zero exploitability, and demonstrated its usefulness in practice in no-limit Texas hold 'em using endgames consisting of the final betting round. I have presented a general framework for assessing its usefulness in general games given a decomposition of the game into a trunk and endgames in Section 10.1. It would be interesting to apply this framework to different domains and subdivisions to understand better when and why it is helpful.

13. **Developing improved variance-reduction techniques**

    One major challenge for large-scale game solving is developing efficient procedures for evaluating agent performance. No-limit Texas hold 'em in particular has an extremely large variance, and it often requires many samples of play to evaluate performance with statistical significance. This is particularly challenging for agents that take significant amounts of time by performing real-time computation (e.g., by applying endgame solving or real-time opponent exploitation algorithms).

    We have developed a new variance reduction procedure, which is described in Section 10.6, that is specifically geared towards reducing variance in the evaluation of an endgame solving algorithm. It works by comparing the performance between the base agent and endgame solving agent only over the sample of hands where both make it to an endgame (we show that only considering the hands where they make it to the same endgame would be biased). We showed that we were able to obtain statistical significance for the performance improvement of our endgame solving algorithm over the base strategy using this new variance-reduction technique, while we would not have using the prior approach over that sample, even utilizing the traditional "duplicate" variance-reduction procedure (where all hands are duplicated with the cards reversed). It would be interesting to explore improved variance-reduction procedures specifically geared towards endgame solving; for example, I strongly suspect that the new approach I presented can be integrated with a variant of duplicate scoring to reduce variance further in this setting. Perhaps further improvement can be achieved by integrating our technique with other more sophisticated approaches, such as DIVAT and MIVAT [122, 124]. Without such approaches, it is extremely difficult to assess endgame solving approaches with statistical significance (for instance, I was not able to obtain statistical significance when evaluating the integration of equilibrium refinement techniques with endgame solving described in Section 10.7.4 for this reason), and it will be very difficult to make research advances on this front.

It would also be extremely useful to develop improved variance-reduction techniques that are not specific to endgame solving. For example, there has been discussion recently of adding a new 6-player no-limit Texas hold ’em variant to the AAAI Annual Computer Poker Competition. However, the main concern is that it would be extremely difficult to obtain statistical significance. I think there are many interesting research challenges in this area. For instance, how to select optimal permutations of the agents, how many to select, and whether duplicate scoring or existing variance-reduction approaches can be integrated.

14. **Is our opponent modeling algorithm DBBR effective in other domains?**

    I presented a new domain-independent algorithm that effectively models and exploits opponents in extremely large imperfect-information games. The algorithm computes an approximate equilibrium strategy that is used as a prior, and constructs an opponent model by computing the “closest” strategy for the opponent to this prior subject to additional constraints that his strategy conforms to our observations of his play thus far (i.e., the frequencies with which he has taken each action at every history of publicly observable actions), using an appropriate distance metric.

    This seems like a natural approach to apply to many domains, particularly those where there is limited historical data available for the set of opponents we wish to play against, and we must utilize a prior model that does not have access to such data.

    An alternative approach for this setting would be to model the opponent as playing the strategy with maximum entropy that matches our observations of his play thus far. This approach could be preferable to DBBR if we expect the opponent strategy to be extremely far from the prior strategy we have precomputed, or if we are unable to compute a good prior strategy. It would be interesting to explore connections between these approaches and experiment in different settings.

    It would also be interesting to consider extensions of DBBR to the setting where we can precompute several approximate equilibrium strategies in advance, as opposed to just one. We would then need to develop a technique to compute the strategy for the opponent that is closest to this set of strategies subject to our observations (as opposed to minimizing distance from just a single strategy). We could even further extend this approach by allowing for sets of prior strategies that are not all necessarily approximate equilibrium strategies (for instance, we could have a set of opponent models that has been precomputed from historical data).

15. **Can we design robust exploitation approaches that perform well against a population consisting of a mixture of weak opponents and strong dynamic ones?**

    Our algorithm DBBR was designed to specifically exploit static weak opponents, and admittedly could perform extremely poorly against strong dynamic opponents. This is true of many pure exploitation algorithms. For example, a recent effective approach for online exploitation that uses regret minimization between several precomputed strategies often does extremely well at exploiting weak opponents in the competition, but is not competitive against the strong agents [8]. Our safe exploitation approaches are not scalable yet to games of this size, primarily because of the challenge of computing ϵ-safe best responses efficiently in real time.

Developing scalable algorithms that exploit mistakes of weak opponents and also perform well against strong, potentially dynamic opponents is a very important problem. As described above, endgame solving can be utilized for exploitation in addition to its initially proposed use for approximating equilibrium, and it performs such exploitation in an inherently robust way, as it is computing equilibrium strategies in an induced endgame given strategies for both agents (that can possibly be computed via the application of an opponent modeling algorithm) as inputs. I think that the integration of ideas from the approaches of endgame solving, DBBR, and the algorithms I have presented for safe exploitation can lead to significant progress on this problem.

16. **Can we develop a principled approach for generating strong game-theoretic strategies that are human understandable?**

    The strategies typically computed for large games are represented as massive binary files and totally unintelligible to humans. One important by-product of the work I presented in Chapter 11 is that often the structure of equilibrium strategies can be represented in a very compact, human-understandable format. I presented three qualitative models that, together, describe the equilibrium strategies for final-round endgames for the limit variant of Texas hold 'em for any input card distributions that were encountered in practice (where the input card distributions were computed by applying Bayes' rule to the approximate equilibrium strategies that were precomputed in advance). Such representations can make equilibrium strategies more accessible to humans, thereby leading to improved human decision making.

    I would like to explore more general principled approaches for computing strategies that are human understandable that extend far beyond poker. One simple approach I investigated for matrix games was to compute an equilibrium refinement corresponding to the Nash equilibrium with minimal support; intuitively we would expect strategies with small support to be easier to represent and understand. However, even this problem proved to be very difficult, and the integer-program formulation I devised and implemented had an extremely high running time even in very small games. I suspect that even this seemingly simple problem is computationally intractable even for two-player zero-sum strategic-form games.

17. **Can we develop a small set of qualitative models for no-limit Texas hold 'em like we did for limit Texas hold 'em endgames?**

    I developed a set of qualitative models that describe equilibrium strategy in limit Texas hold 'em endgames. It would be very interesting to see if similar analysis could devise models for no-limit Texas hold 'em, based on the strategies that were computed by our champion agent from the 2014 competition. No-limit Texas hold 'em is a much more popular variant for humans than limit, and this would potentially have significant impact and interest within the human poker-playing community. However, it is much more challenging, because we would need to incorporate different betting sizes into the model (limit just has a single fixed size), represent our action facing all possible bet sizes for the opponent, etc. However, I still suspect that good compact representations exist for this problem.

18. **Can we develop improved opponent modeling and exploitation algorithms that leverage large amounts of historical data?**

    Our exploitation DBBR was specifically designed for the setting where no historical data was available for the class of opponents we expected to encounter. However, in many cases, some amount of historical data is available, and we would like to devise approaches that are able to capitalize on it. As one obvious modification, we could simply revise DBBR to utilize any prior strategy for the opponent, not necessarily an approximate-equilibrium strategy. For example, if we are able to compute a "population average" strategy for the pool of opponents we expect to encounter based on historical data, we could use this as the prior. More sophisticated approaches could use a full distribution of strategies as the prior and posterior opponent models as opposed to single point estimates.

    The related work of Bard et al. described above [8] takes into account historical data to an extent: it constructs a small set of exploitative strategies from computing responses to specific opponent strategies based on this data, which it selects between in real time. However, the historical data is not utilized in real time in terms of Bayesian reasoning in constructing the opponent model.

    It would very be interesting to consider approaches that are able to incorporate historical data effectively online within the Bayesian setting.

19. **Can effective opponent modeling algorithms be developed for the settings where the opponent's private information is never, sometimes, or always revealed after a game iteration?**

    Various research on imperfect-information games has made various assumptions about the revelation of private information about the opponent. For example, our exploitation algorithm DBBR assumed that we never observed the opponent's private information after a hand, while for the experiments for our safe exploitation algorithms in Kuhn poker we assumed the opponent's private information was always revealed. This variability in modeling has made the literature on this problem somewhat disorganized and complicates the problem of comparing different approaches.

    Future research should develop approaches that apply to all settings. For example, in poker, the opponent's private information is actually revealed if both agents make it to the "showdown" (which occurs if neither player has folded in any betting round), while the opponent's private information is not revealed if a player has folded. (In the historical log files from previous competitions however, the private information of all agents is listed regardless of the betting sequence, and therefore exploitation algorithms that utilize historical data would be able to leverage data labels for this domain.) Thus, effective approaches should be developed that apply if the opponent' private information is not revealed, but are able to capitalize on it further when it is revealed. While neither our implementation of DBBR nor the safe exploitation algorithms apply to this setting, the approaches could be modified for this setting in various ways. The safe exploitation algorithms would need to make extremely pessimistic assumptions about the opponent in this setting (which are described in Section 13.6.2), which would likely make them far too conservative in practice. One approach that would enable DBBR to apply to this setting would be to set the opponent model to be the deterministic action that was observed with the given private information if it was revealed (and to randomize according to the frequencies of observation if multiple actions have been observed), and if it was not revealed to follow the existing approach. Improved algorithms can also integrate existing approaches that have been developed for evaluating strategies using importance sampling [14].

Bibliography

[1] Nick Abou Risk and Duane Szafron. Using counterfactual regret minimization to create competitive multiplayer poker agents. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[2] Charalambos D. Aliprantis and Kim C. Border. *Infinite Dimensional Analysis*. Springer-Verlag, third edition, 2006.

[3] Rickard Andersson. Pseudo-optimal strategies in no-limit poker. Master's thesis, Umeå University, May 2006.

[4] Jerrod Ankenman and Bill Chen. *The Mathematics of Poker*. ConJelCo LLC, 2006.

[5] C. Archibald and Y. Shoham. Modeling billiards games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Budapest, Hungary, 2009.

[6] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.

[7] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32:48–77, 2002.

[8] Nolan Bard, Michael Johanson, Neil Burch, and Michael Bowling. Online implicit agent modelling. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[9] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[10] Johannes Bisschop. *AIMMS—Optimization Modeling*. Paragon Decision Technology, 2006.

[11] Liad Blumrosen, Noam Nisan, and Ilya Segal. Auctions with severely bounded communication. *Journal of Artificial Intelligence Research*, 28:233–266, 2007.

[12] Jim Blythe, Aaron Botello, Joseph Sutton, David Mazzaco, Jerry Lin, Marc Spraragen, and Mike Zyda. Testing cyber security with simulated humans. In *Innovative Applications of Artificial Intelligence (IAAI)*, pages 1622–1627, 2011.

[13] Béla Bollobás. *Combinatorics*. Cambridge University Press, Cambridge, 1986.

[14] Michael Bowling, Michael Johanson, Neil Burch, and Duane Szafron. Strategy evalua-

tion in extensive games with importance sampling. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 72–79, 2008.

[15] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, January 2015.

[16] Ronen Brafman and Moshe Tennenholtz. Learning to coordinate efficiently: A model-based approach. *Journal of Artificial Intelligence Research*, 19:11–23, 2003.

[17] Noam Brown and Tuomas Sandholm. Regret transfer and parameter optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[18] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas Hold'em agent. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[19] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[20] Doran Chakraborty and Peter Stone. Convergence, targeted optimality, and safety in multiagent learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[21] Xi Chen and Xiaotie Deng. Settling the complexity of 2-player Nash equilibrium. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2006.

[22] George Dantzig. A proof of the equivalence of the programming problem and the game problem. In Tjalling Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 330–335. John Wiley & Sons, 1951.

[23] Constantinos Daskalakis, Paul Goldberg, and Christos Papadimitriou. The complexity of computing a Nash equilibrium. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2006.

[24] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'2000)*, pages 1467–1473, 2000.

[25] Melvin Dresher. *Games of Strategy: Theory and Applications*. Prentice Hall, 1961.

[26] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

[27] Drew Fudenberg and David Levine. *The Theory of Learning in Games*. MIT Press, 1998.

[28] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, 1991.

[29] Sam Ganzfried and Tuomas Sandholm. Computing an approximate jam/fold equilibrium for 3-player no-limit Texas Hold'em tournaments. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2008.

[30] Sam Ganzfried and Tuomas Sandholm. Computing equilibria in multiplayer stochastic games of imperfect information. In *Proceedings of the 21st International Joint Conference*

on Artificial Intelligence (IJCAI), 2009.

[31] Sam Ganzfried and Tuomas Sandholm. Computing equilibria by incorporating qualitative models. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[32] Sam Ganzfried and Tuomas Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2011.

[33] Sam Ganzfried and Tuomas Sandholm. Tartanian5: A heads-up no-limit Texas Hold'em poker-playing program. In *Computer Poker Symposium at the National Conference on Artificial Intelligence (AAAI)*, 2012.

[34] Sam Ganzfried and Tuomas Sandholm. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[35] Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[36] Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[37] Sam Ganzfried and Tuomas Sandholm. Safe opponent exploitation. *ACM Transactions on Economics and Computation (TEAC)*, 2015. Special issue on selected papers from EC-12. Early version appeared in EC-12.

[38] Sam Ganzfried, Tuomas Sandholm, and Kevin Waugh. Strategy purification and thresholding: Effective non-equilibrium approaches for playing large games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[39] Richard Gibson. *Regret Minimization in Games and the Development of Champion Multiplayer Computer Poker-Playing Agents*. PhD thesis, University of Alberta, 2014.

[40] Richard Gibson, Neil Burch, Marc Lanctot, and Duane Szafron. Efficient Monte Carlo counterfactual regret minimization in games with many player actions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

[41] Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, and Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[42] Andrew Gilpin and Tuomas Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

[43] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*,

2007.

[44] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM*, 54(5), 2007.

[45] Andrew Gilpin and Tuomas Sandholm. Expectation-based versus potential-aware automated abstraction in imperfect information games: An experimental comparison using poker. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2008. Short paper.

[46] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding Nash equilibria in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE)*, San Diego, CA, 2007.

[47] 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 AAAI Conference on Artificial Intelligence (AAAI)*, 2007.

[48] 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 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2008.

[49] Andrew Gilpin, Javier Peña, and Tuomas Sandholm. First-order algorithm with $\mathcal{O}(\ln(1/\epsilon))$ convergence for ϵ-equilibrium in two-person zero-sum games. *Mathematical Programming*, 133(1–2):279–298, 2012. Conference version appeared in AAAI-08.

[50] Geoff Gordon. One-card poker. `http://www.cs.cmu.edu/~ggordon/poker/`, 2013.

[51] Geoffrey J. Gordon. No-regret algorithms for structured prediction problems. Technical Report CMU-CALD-05-112, Carnegie Mellon University, 2005.

[52] GTORangeBuilder. GTO RangeBuilder, 2014. http://gtorangebuilder.com/#home.

[53] John Hawkin, Robert Holte, and Duane Szafron. Automated action abstraction of imperfect information extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 681–687, 2011.

[54] John Hawkin, Robert Holte, and Duane Szafron. Using sliding windows to generate action abstractions in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[55] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[56] Bret Hoehn, Finnegan Southey, Robert C. Holte, and Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 783–788, 2005.

[57] Hold'emResources. Hold'emResources.net, 2008. http://www.holdemresources.net/h/web-calculators/nashicm.html.

[58] Junling Hu and Michael P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[59] Gurobi Optimization, Inc. Gurobi optimizer reference manual version 6.0, 2014. URL `http://www.gurobi.com`.

[60] Eric Jackson. Slumbot NL: Solving large games with counterfactual regret minimization using sampling and distributed processing. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2013.

[61] Eric Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2014.

[62] Michael Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[63] Michael Johanson. Measuring the size of large no-limit poker games. Technical report, University of Alberta, 2013.

[64] Michael Johanson and Michael Bowling. Data biased robust counter strategies. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[65] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1128–1135, 2007.

[66] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[67] 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 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[68] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[69] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71:291–307, 2005.

[70] Michael Kearns, Yishay Mansour, and Satinder Singh. Fast planning is stochastic games. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[71] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, July 1997.

[72] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC)*, pages 750–760, 1994.

[73] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of

equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[74] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, 41:297–327, 2011.

[75] Christian Kroer and Tuomas Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2014.

[76] H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

[77] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2009.

[78] Marc Lanctot, Richard Gibson, Neil Burch, Martin Zinkevich, and Michael Bowling. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

[79] Michael Littman. Friend-or-foe Q-learning in general-sum Markov games. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 322–328, 2001.

[80] Peter McCracken and Michael Bowling. Safe strategies for agent modelling in games. In *AAAI Fall Symposium on Artificial Multi-agent Learning*, October 2004.

[81] Peter Bro Miltersen and Troels Bjerre Sørensen. Computing proper equilibria of zero-sum games. In *Computers and Games*, pages 200–211, 2006.

[82] Peter Bro Miltersen and Troels Bjerre Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.

[83] Peter Bro Miltersen and Troels Bjerre Sørensen. Fast algorithms for finding proper strategies in game ttrees. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 874–883, 2008.

[84] Peter Bro Miltersen and Troels Bjerre Sørensen. Computing a quasi-perfect equilibrium of a two-player game. *Economic Theory*, 42(1):175–192, 2010.

[85] Roger B. Myerson. Refinements of the Nash equilibrium concept. *International Journal of Game Theory*, 15:133–154, 1978.

[86] John Nash. Non-cooperative games. *Annals of Mathematics*, 54:289–295, 1951.

[87] Ofir Pele and Michael Werman. A linear time histogram metric for improved SIFT matching. In *Proceedings of the European Conference on Computer Vision*, 2008.

[88] Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *Proceedings of the International Conference on Computer Vision*, 2009.

[89] James Pita, Manish Jain, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. Robust

solutions to Stackelberg games: Addressing bounded rationality and limited observations in human cognition. *Artificial Intelligence Journal*, 174(15):1142–1171, 2010.

[90] James Pita, Richard John, Rajiv Maheswaran, Milind Tambe, and Sarit Kraus. A robust approach to addressing human adversaries in security games. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 660–665, 2012.

[91] Marc Ponsen, Jan Ramon, Tom Croonenborghs, Kurt Driessens, and Karl Tuyls. Bayes-relational learning of opponent models from incomplete information in no-limit poker. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

[92] Marc Ponsen, Marc Lanctot, and Steven de Jong. Mcrnr: Fast computing of restricted Nash responses by means of sampling. In *AAAI workshop on Interactive Decision Theory and Game Theory Workshop*, 2010.

[93] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

[94] Rob Powers, Yoav Shoham, and Thuc Vu. A general criterion and an algorithmic framework for learning in multi-agent systems. *Machine Learning*, 67(1-2):45–76, 2007.

[95] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2005.

[96] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175 (5–6):958–987, 2011.

[97] Jonathan Rubin and Ian Watson. Case-based strategies in computer poker. *AI Communications*, 25(1):19–48, 2012.

[98] Tuomas Sandholm. Perspectives on multiagent learning. *Artificial Intelligence*, 171:382–391, 2007.

[99] Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Magazine*, pages 13–32, Winter 2010. Special issue on Algorithmic Game Theory.

[100] Tuomas Sandholm and Andrew Gilpin. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006.

[101] Tuomas Sandholm and Victor R Lesser. Leveled commitment contracts and strategic breach. *Games and Economic Behavior*, 35:212–270, 2001.

[102] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 495–501, 2005.

[103] David Schnizlein, Michael Bowling, and Duane Szafron. Probabilistic state translation in extensive games with large action sets. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[104] Reinhard Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 12:301–324, 1965.

[105] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, London, UK, 2002. Springer-Verlag.

[106] Satinder P Singh, Vishal Soni, and Michael P Wellman. Computing approximate Bayes-Nash equilibria in tree-games of incomplete information. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 81–90, New York, NY, 2004. ACM.

[107] Sit-n-Go Power Tools, 2005. http://www.sitngo-analyzer.com/.

[108] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, July 2005.

[109] Noah D. Stein, Asuman Ozdaglar, and Pablo A. Parillo. Separable and low-rank continuous games. *International Journal of Game Theory*, 37(4):475–504, 2008.

[110] Two Plus Two Forums. Single table tournament forum FAQ, 2007. http://forumserver.twoplustwo.com/.

[111] Eric van Damme. *Stability and Perfection of Nash Equilibrium*. Springer-Verlag, 1987.

[112] Juan Pablo Vielma and George L Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. In *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2008.

[113] John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100: 295–320, 1928.

[114] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[115] Yevgeniy Vorobeychik and Michael Wellman. Stochastic search methods for Nash equilibrium approximation in simulation-based games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Estoril, Portugal, 2008.

[116] Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2002.

[117] Kevin Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

[118] Kevin Waugh. A fast and optimal hand isomorphism algorithm. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2013.

[119] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2009.

[120] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael Bowling. A practical use of imperfect recall. In *Proceedings of the Symposium*

on Abstraction, Reformulation and Approximation (SARA), 2009.

[121] Michael P. Wellman, Daniel M. Reeves, Kevin M. Lochner, Shih-Fen Cheng, and Rahul Suri. Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *Proceedings of AAAI'05*, 2005.

[122] Martha White and Michael Bowling. Learning a value analysis tool for agent evaluation. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1976–1981, 2009.

[123] Bryce Wiedenbeck and Michael P. Wellman. Scaling simulation-based game analysis through deviation-preserving reduction. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[124] Martin Zinkevich, Michael Bowling, Nolan Bard, Morgan Kan, and Darse Billings. Optimal unbiased estimators for evaluating agent performance. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 573–578, 2006.

[125] Martin Zinkevich, Michael Bowling, Michael Johanson, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.