Algorithms for abstracting and solving imperfect information games

Andrew Gilpin

May 2009

CMU-CS-09-127

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

Thesis Committee:
Tuomas Sandholm, Chair
Avrim Blum
Geoff Gordon
Javier Peña
Bernhard von Stengel, London School of Economics

This research was supported in part by the National Science Foundation under ITR grants IIS-0121678 and IIS-0427858, a Sloan Fellowship, and a Siebel Scholarship. We also acknowledge Intel Corporation and IBM for their gifts.

Keywords: Computational game theory, artificial intelligence, equilibrium computation, automated abstraction, nonsmooth convex optimization, sequential games, repeated games, imperfect information, poker AI.

To my parents.

Abstract

Game theory is the mathematical study of rational behavior in strategic environments. In many settings, most notably two-person zero-sum games, game theory provides particularly strong and appealing solution concepts. Furthermore, these solutions are efficiently computable in the complexity-theory sense. However, in most interesting potential applications in artificial intelligence, the solutions are difficult to compute using current techniques due primarily to the extremely large state-spaces of the environments.

In this thesis, we propose new algorithms for tackling these computational difficulties. In one stream of research, we introduce *automated abstraction algorithms for sequential games of imperfect information*. These algorithms take as input a description of a game and produce a description of a strategically similar, but smaller, game as output. We present algorithms that are lossless (*i.e.,* equilibrium-preserving), as well as algorithms that are lossy, but which can yield much smaller games while still retaining the most important features of the original game.

In a second stream of research, we develop *specialized optimization algorithms for finding ϵ-equilibria in sequential games of imperfect information*. The algorithms are based on recent advances in non-smooth convex optimization and provide significant improvements over previous algorithms for finding ϵ-equilibria.

Combining these two streams, we enable the application of game theory to games much larger than was previously possible. As an illustrative example, we find near-optimal solutions for four-round models of limit and no-limit Texas Hold'em poker, and report that the resulting players won the most chips overall at the 2008 AAAI Computer Poker Competition.

Acknowledgments

It would be a difficult task to overstate how influential my advisor, Tuomas Sandholm, has been on my research career and life. I have had the singular privilege of working with Tuomas for nine years, both in industry and in academia. It has been an amazingly exhilarating and rewarding experience. Tuomas always demonstrated a strong patience when communicating difficult concepts to me. When I interrupted him to confess that I did not understand a particular topic (a common occurence), Tuomas was always able to find an alternative explanation that somehow managed to transfer his knowledge to me. There is zero probability that I would be earning a Ph.D., let alone writing this thesis, had it not been for his influence.

Javier Peña has also been extremely influential on my research trajectory. Indeed, without my collaboration with Javier, the second half of this thesis would not have happened. Javier's knowledge of continuous optimization is inspiring, and I am lucky that I was able to learn from such an accomplished and talented mathematician.

Special thanks also goes to the rest of my committee: Avrim Blum, Geoff Gordon, and Bernhard von Stengel. This thesis is undoubtably better as a result of each of your insights and comments.

While at CMU I have benefited greatly from my interactions with the graduate students and visitors in the Agent-Mediated Electronic Marketplaces lab. In particular, I would like to thank David Abraham, Pranjal Awasthi, Mike Benisch, Felix Brandt, Vincent Conitzer, Sam Ganzfried, Benoît Hudson, Kate Larson, Alex Nareyek, Abe Othman, Paolo Santi, Rob Shields, Troels Bjerre Sørensen, and Erik Zawadski. Elsewhere at CMU, I was lucky to have had many enjoyable conversations, both social and research-oriented, with Nina Balcan, George Davis, Paul Enders, Sam Hoda, Ryan Kelly, Ben Peterson, Peter Richter, Aaron Roth, Mugizi Rwebangira, and John Turner.

Extra-special acknowledgments go to my friends and officemates for 7 years, Nina and Doru Balcan. I will always fondly remember traveling with you and meeting your families in Romania. Nina: *Eşti o profesoara excelenta!* Doru: *Eşti o profesoara extraordinara!*

The highlights of my time in graduate school mostly revolve around the many conferences I attended, and the wonderful people that I have met from around the globe. In particular, I want to acknowledge Moshe Babaioff, Navin Bhat, Liad Blumrosen, Ruggiero Cavallo, Viet Dung Dang, Raj Dash, Felix Fischer, Kobi Gal, Enrico Gerding, Rica Gonen, Christian Guttman, Jason Hartline, Nick Jennings, Sébastien Lahie, Ron Lavi, Kevin Leyton-Brown, Peter Bro Miltersen, Ahuva Mu'alem, Jigar Patel, Iyad Rahwan, Talal Rahwan, Gopal Ramchurn, Alex Rogers, Rahul Savani, and Troels Bjerre Sørensen.

I have benefited greatly from having had the opportunity to TA under many talented professors. At Washington University, this includes Christopher Gill, Will Gillett, David Levine, Ron Loui, and Michael Plezbert. At CMU, this includes Anupam Gupta, Mike Lewicki, Steven Rudich, and Tuomas Sandholm. Assisting with the teaching of your courses was a fantastic learning experience.

I would like to thank the many professors who have had a particularly inspiring influence on my education. At Washington University, this includes Christopher Gill, Sally Goldman, David Levine, Ron Loui, Robert Pless, Catalin Roman, Doug Schmidt, Subhash Suri, Jonathan Turner, George Varghese, and Weixiong Zhang. At CMU, this includes Egon Balas, Christos Faloutsos, Peter Lee, Francois Margot, Tom Mitchell, Andrew Moore, Javier Peña, Tuomas Sandholm, and Danny Sleator.

My research direction was greatly influenced by the R&D team at CombineNet. In particular, I gratefully acknowledge the many interesting discussions I had at CombineNet with Bryan Bailey, Egon Balas, Craig Boutilier, Rob Clarke, Andrew Fuqua, Sam Hoda, Tom Kuhn, David Levine, George Nemhauser, David Parkes, Tuomas Sandholm, Rob Shields, Yuri Smirnov, Subhash Suri, and Bill Walsh.

Equally important in helping me make it through graduate school are the world-class administrators that work in the Computer Science Department at CMU. In particular, I want to thank Sharon Burks, Deb Cavlovich, and Marilyn Walgora for doing an amazing job in helping make the difficult journey through graduate school much smoother and more enjoyable than it could have been. Also deserving of mention is Dave Livingston with SCS Facilities for going out of his way to help make the construction of our computing infrastructure much smoother.

Switching gears slightly, it would be impolite of me to neglect to mention a few other groups of individuals who have done more than they will ever know in making my graduate school experience more enjoyable. Although they have no idea who I am, I would like to thank the 2007 West Virginia University football team, the 2007-08 Huntington High School basketball team, and the 2005 and 2008 Super Bowl Champion Pittsburgh Steelers for providing me with greatly appreciated entertaining and inspiring distractions. My memories of your achievements will always be intertwined with those of my own graduate school achievements.

Helping me keep a normal life away from the computer were my many wonderful friends: Bryan, Candace, Chris, Emily, George, Jennie, John, Lawrence, Lisa, Mary, Mike B., Mike J., Mike M., Mike P., Paul, Randy, Rob, and Sam. For those whom I forgot, my apologies. You know who you are.

I am extraordinarily lucky to have such a wonderful friendship with one particularly remarkable person. Lauren, there is no doubt that your support, care, and patience have had a major influence on my life, career, and this thesis. Thank you.

Finally, I am blessed with a wonderful family. Ann, Kate, Marty, Caroline, and Charlie: Thank you for your constant love and support. Mom: thank you for (literally) showing me how to earn a Ph.D. at CMU. Mom and Dad: Thank you for teaching me the important lessons that are not covered in the classroom. This thesis is dedicated to you.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

Chapter 1

Overview

1.1 Introduction

In settings with multiple, self-interested agents, the outcome for each individual agent depends on the actions of the other agents in the system. Consequently, *rational* and *optimal* strategies for each agent also depend on the other agents' actions. In order for an agent to achieve their best possible outcome it is necessary to take the other agents' preferences and strategies into account during the deliberation process.

Game theory is the mathematical framework that enables the study of rational behavior in competitive multiagent environments. *Inter alia*, game theory defines *solution concepts* that provide prescriptive behavior (*i.e.,* strategies) for each agent. The *Nash equilibrium* [112] is the most prevalent such solution concept. In a Nash equilibrium, no agent has any incentive to deviate to any other strategy.

In some settings, the algorithms for finding (or approximating) Nash equilibria are straightforward. In fact, there has been an enormous amount of research on algorithms for two-person perfect information games (sometimes called *combinatorial games* [9]). In these games, applying minimax search (possibly with α-β-pruning) actually yields a Nash equilibrium (assuming that the game tree is completely searched and no internal nodes are replaced by leaves according to some evaluation function) [131]. Perhaps without realizing it, the researchers that developed these algorithms were motivated by a line of reasoning that is analogous to the reasoning behind the Nash equilibrium: the best action for one agent largely depends on the best actions for the other agents. However, as we will discuss below, these algorithms are not applicable to many interesting, real-world games.

Developing expert-level game-playing computer agents has long been a major focus of the artificial intelligence (AI) community. The most notable successes include *Chinook*, which defeated the checkers world champion Dr. Marion Tinsley in 1992 [139]; *Deep Blue*, which defeated Garry Kasparov, the chess world champion in 1997; and *TD-Gammon*, the best backgammon-playing program in the world [156]. (See [140] for a survey of other AI success stories in game playing.) These are all very impressive applications of AI techniques and have done much to advance the standing of AI in the wider science community. However, each of these three games possess the property known as *perfect information*, *i.e.*, at any point in time, both players are fully informed about the state of the world. In contrast, most interesting potential application areas of game theory have the property of *imperfect information*: at most stages of the game, the players are only partially informed about the state of the world. Examples include poker and most other card games (in which each player does not know the cards held by the other players), economic environments (in which each player does not know the other players' preferences), and adversarial robot environments (in which each robot does not know the locations and goals of the other robots). Due to this informational difference, algorithms for perfect information games are unhelpful when designing agents for games with imperfect information.

In the last 15 years, there has been a surge of research with the goal of developing the theory and algorithms for finding equilibria in sequential games with imperfect information [84, 160, 86, 85, 88, 87]. Among other breakthroughs, it is now well-known that one can compute a Nash equilibrium in a two-person zero-sum sequential game with imperfect information in time polynomial in the size of the game tree. The prescribed method for solving this problem is to model the game as a linear program and solve for the equilibrium using general-purpose linear programming tools. However, for most interesting applications in AI, these tools do not scale. In this thesis we present two complementary streams of research to tackle this problem.

1. We introduce *automated abstraction algorithms for sequential games of imperfect information* as a method for finding (nearly) equivalent, smaller representations of games on which the equilibrium analysis may be carried out.

2. We improve the equilibrium finding algorithms themselves via the development of *specialized optimization algorithms for finding approximate equilibria in sequential games of imperfect information*.

Combining these approaches enables the application of game theory to games many orders of magnitude larger than previously possible. In the remainder of this section, we motivate our main application area (poker), give the thesis statement, and summarize the research presented in the remainder of this document.

1.2 Poker and artificial intelligence

Poker is an enormously popular card game played around the world. The strategies employed by expert players can be extremely sophisticated [148]. A poker player cannot succeed with just the ability to compute odds and probabilities; they also need to utilize randomized (information-hiding) strategies that attempt to deceive the opponent. When performing actions, successful poker players need to consider not only what possible private information their opponent knows, but also what their own action reveals about their own private information. Thus, players must speculate, counter-speculate, counter-counter-speculate, *etc.*, about what their actions are achieving and what they are revealing. Game theory, via its various equilibrium concepts, is particularly well-suited to providing definitive answers in these types of situations.

In addition to the challenging research issues presented by poker, it is a particularly attractive testbed for computational game theory research. Unlike many other important games with imperfect information (*e.g.* financial markets, business-to-business interactions, political negotiations, legal disputes), there are no issues with the game model. Game theory is capable of modeling the game *precisely* as it is played. Another motivation for studying poker is that it represents a frontier of machine intelligence: while artificially intelligent agents have surpassed the skill level of humans in games such as chess, checkers, and backgammon, poker remains a game where humans are superior.

For the above reasons, as well as many others, poker has been identified as an important area of research for AI [15], and it is with these challenges in mind that we present this thesis.

1.3 Thesis statement

> *Automated abstraction in conjunction with specialized equilibrium-finding algorithms enables the construction of agents capable of intelligently performing in large-scale sequential competitive environments of imperfect information.*

1.4 Summary of thesis contribution

- We developed a provably lossless automated abstraction algorithm, *GameShrink*, for sequential games of imperfect information. It enabled the computation of optimal strategies for Rhode Island Hold'em poker, which at the time was the largest sequential game of imperfect information solved by over four orders of magnitude [58, 61].

- We developed approximation versions of *GameShrink* for performing information-based abstraction in even larger games, as well as algorithms for action abstraction and stage-based abstraction. These new algorithms have been used to develop a series of players for heads-up limit and no-limit Texas Hold'em poker, the latest of which are competitive with all known poker-playing programs [59, 60, 64, 65, 62].

- We developed specialized equilibrium-finding algorithms based on recent techniques developed for non-smooth convex optimization. Our contribution includes both theoretical improvements and practical improvements. These algorithms have enabled the solution of games four orders of magnitude larger than was previously possible using state-of-the-art linear programming solvers [74, 56, 57].

- In a slightly tangential line of research, we investigate *repeated* games of imperfect information. For two-person zero-sum games with lack of information on one side (*i.e.* only one player is informed), we develop an algorithm for finding optimal strategies [63].

1.5 Organization

The thesis is organized into four parts. The remainder of Part I presents the game theory background used in this thesis (Chapter 2) and, since the primary application studied in this thesis is poker, provides the relevant information about the game (Chapter 3).

Parts II and III discuss the thesis contribution. Part II discusses a range of automated abstraction algorithms, including lossless information abstraction (Chapter 5), lossy information abstraction (Chapter 6), and stage and action abstraction (Chapter 7).

Part III develops algorithms for equilibrium finding. In Chapter 8 we discuss gradient-based equilibrium-finding algorithms, and show that our most advanced algorithm finds an ϵ-equilibrium in $O(\log 1/\epsilon)$ iterations, an exponential improvement over the prior best algorithms in terms of the dependence on ϵ. Chapter 9 discusses a randomized sampling methodology for speeding up the gradient-based algorithms, and Chapter 10 discusses a number of systems-level techniques for further improving performance, including an implementation designed for a particular supercomputing architecture. In Chapter 11 we shift gears to study *repeated* games of imperfect information and discuss an equilibrium-finding algorithm suitable for that class of games.

Part IV summarizes the thesis.

Chapter 2

Game Theory

In this section we review relevant definitions and algorithms from game theory. Game theory is the mathematical study of decision-making in interactive, competitive environments. The most basic assumption underlying the (classical) theory involves the embodiment of rationality in individual agents via utility-function-inducing preferences. Further, the players are assumed to act in such as a way as to maximize their utility based on their knowledge about the game. We do not further discuss these basic assumptions, nor detail the many objections raised against them. The assumptions are standard. We simply present the basic game theory that is used in this thesis. The models and solution concepts discussed in this chapter mirrors the development of any standard game theory text (*e.g.*, [120, 111]).

2.1 Extensive form games and perfect recall

Normal form games (often called matrix (resp. bimatrix) games in two-person zero-sum (resp. non-zero-sum) games) are games in which each player simultaneously chooses an action in their action set, and these choices deterministically determine the outcome of the game. Although there is a deep theory for these games, and many interesting games that naturally fit this model, they are not our main interest.

In this thesis, we are primarily interested in *sequential games*, in which players may take moves after observing moves of chance (*e.g.*, a roll of a die) and moves of the other players. This model is much more powerful in terms of modeling capability as many real-world games can be concisely represented in this model.[1] This class of games is referred

[1]In principle, any finite sequential game can be represented in the normal form by considering cross products of all possible contingency plans [91]. However, such representations lead to exponential increases

to as *extensive form games* and our definition of this class is standard:

Definition 1 *An n-*person game in extensive form *is a tuple* $\Gamma = \langle I, V, E, P, H, A, u, p\rangle$ *satisfying the following conditions:*

1. *$I = \{0, 1, \ldots, n\}$ is a finite set of players. By convention, Player 0 is the* chance *player.*

2. *The pair (V, E) is a finite directed tree with nodes V and edges E. Z denotes the leaves of the tree, called* terminal nodes. *$V \setminus Z$ are* decision nodes. *$N(x)$ denotes x's children and $N^*(x)$ denotes x's descendants.*

3. *$P : V \setminus Z \to I$ determines which player moves at each decision node. P induces a partition of $V \setminus Z$ and we define $P_i = \{x \in V \setminus Z \,|\, P(x) = i\}$.*

4. *$H = \{H_0, \ldots, H_n\}$ where each H_i is a partition of P_i. For each of Player i's* information sets *$h \in H_i$ and for $x, y \in h$, we have $|N(x)| = |N(y)|$. We denote the information set of a node x as $h(x)$ and the player who controls h is $i(h)$.*

5. *$A = \{A_0, \ldots, A_n\}$, $A_i : H_i \to 2^E$ where for each $h \in H_i$, $A_i(h)$ is a partition of the set of edges $\{(x, y) \in E \,|\, x \in h\}$ leaving the information set h such that the cardinalities of the sets in $A_i(h)$ are the same and the edges are disjoint. Each $a \in A_i(h)$ is called an* action *at h.*

6. *$u : Z \to \mathbb{R}^N$ is the* payoff *function. For $x \in Z$, $u_i(x)$ is the payoff to Player i in the event that the game ends at node x.*

7. *$p : H_0 \times \{a \in A_0(h) \,|\, h \in H_0\} \to [0, 1]$ where*

$$\sum_{a \in A_0(h)} p(h, a) = 1$$

   for all $h \in H_0$ is the transition probability for chance nodes.

In this thesis we restrict our attention to games with *perfect recall* [93], which means that players never forget information:

Definition 2 *An n-person game in extensive form satisfies* perfect recall *if the following two constraints hold:*

in the size of the game and are not at all suitable for computational purposes [160].

1. *Every path in (V, E) intersects h at most once.*

2. *If v and w are nodes in the same information set and there is a node u that precedes v and $P(u) = P(v)$, then there must be some node x that is in the same information set as u and precedes v and the path taken from u to v is the same as from x to w.*

A straightforward representation for strategies in extensive form games is the *behavior strategy* representation. This is without loss of generality since Kuhn's Theorem [93] states that for any mixed strategy there is a payoff-equivalent behavioral strategy in games with perfect recall. For each information set $h \in H_i$, a behavior strategy is $\sigma_i(h) \in \Delta(A_i(h))$ where $\Delta(A_i(h))$ is the set of all probability distributions over actions available at information set h. A group of strategies $\sigma = (\sigma_1, \ldots, \sigma_n)$ consisting of strategies for each player is a *strategy profile*. We sometimes write $\sigma_{-i} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$ and $(\sigma'_i, \sigma_{-i}) = (\sigma_1, \ldots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \ldots, \sigma_n)$. By an abuse of notation, we will say Player i receives an expected payoff of $u_i(\sigma)$ when all players are playing the strategy profile σ.

2.2 Solution concepts

Having defined the model of games we wish to consider, we now define the various solutions of interest.

Definition 3 *A strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ for a game $\Gamma = \langle I, V, E, P, H, A, u, p\rangle$ is a* Nash equilibrium *if $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma'_i, \sigma_{-i})$ for all $i \in I$ and all σ'_i.*

If the game happens to be two-person zero-sum, then a Nash equilibrium may be called a *minimax solution* and satisfies the following additional properties.

1. If (σ_1, σ_2) is a minimax solution to a two-person zero-sum game Γ, and (σ'_1, σ'_2) is also a minimax solution to Γ, then (σ_1, σ'_2) and (σ'_1, σ_2) are also minimax solutions to Γ.

2. If (σ_1, σ_2) is a minimax solution to a two-person zero-sum game Γ, then $u_1(\sigma_1, \sigma_2) \geq u_1(\sigma_1, \sigma'_2)$ for all σ'_2.

3. All convex combinations of minimax solutions for two-person zero-sum games are also minimax solutions. (The set of minimax solutions forms a convex set.)

The first property means that there is no equilibrium selection problem. In many games there are multiple equilibria, and there is no *a priori* reason to expect that an agent would prefer to play one equilibrium over another. This issue weakens the strength of the predictive power of game theory in some settings. The second property means that equilibria solutions in two-person zero-sum games are robust in that they don't depend on what strategy the opponent employs. The third property (convexity of the set of equilibria) will be of importance when we design equilibrium-finding algorithms.

Due to computational limitations (and in particular the inherent finiteness of floating-point arithmetic), we are often interested in the following slightly relaxed version of Nash equilibrium:

Definition 4 *A strategy profile* $\sigma = (\sigma_1, \ldots, \sigma_n)$ *for a game* $\Gamma = \langle I, V, E, P, H, A, u, p \rangle$ *is an* ϵ-equilibrium *if* $u_i(\sigma_i', \sigma_{-i}) - u_i(\sigma_i, \sigma_{-i}) \leq \epsilon$ *for all* $i \in I$ *and all* σ_i'.

In many algorithms, the parameter ϵ is specified as an input parameter and the algorithm guarantees finding such an ϵ-equilibrium. For a small enough ϵ, these solutions are acceptable in many domains.

2.3 Standard algorithms for finding equilibria

In this section we describe existing algorithms for finding Nash equilibria and ϵ-equilibria in both normal form and extensive form games.

2.3.1 Algorithms for finding equilibria in normal form games

The Nash equilibrium problem for two-person zero-sum (matrix) games can be modeled and solved as a linear program [36, 102, 32]. Linear programs are typically solved via the simplex algorithm or interior-point methods. The simplex algorithm has exponential worst-case complexity, but runs efficiently in practice. Interior-point methods run in polynomial time, and, increasingly, are also fast in practice. Other solution techniques include learning-based approaches, such as fictitious play [25, 128] and experts-based approaches [50]. These approaches are more interested in the learning process itself rather than in arriving at an equilibrium, and generally do not provide very good convergence bounds. Most recently, the *excessive gap technique* was proposed as a method for solving certain non-smooth convex optimization problems, and it has been applied to the problem of finding ϵ-equilibria in matrix games [118, 117]. Finally, bundle-based methods have recently been shown to be effective on some large poker games, including Rhode Island Hold'em [106]. One drawback to those algorithms is that the memory usage increases every iteration, and the time to solve each iteration increases with every iteration (although there are heuristics that mitigate this).

There has been significant recent work on Nash equilibrium finding for two-person non-zero-sum normal form games. Most interesting questions about optimal (for many definitions of "optimal") equilibria are NP-complete [55, 33]. An ϵ-equilibrium in a normal form game with any constant number of agents can be constructed in quasi-polynomial time [98, 97], but finding an exact equilibrium is PPAD-complete even in a two-person game [30], and even if the payoffs are in $\{0, 1\}$ [1]. The most prevalent algorithm for finding an equilibrium in a two-person bimatrix game is *Lemke-Howson* [95], but it takes exponentially many steps in the worst case [137]. For a survey of equilibrium computation in two-person games, see [161]. Recently, equilibrium-finding algorithms that enumerate supports (*i.e.*, sets of pure strategies that are played with positive probability) have been shown efficient on many games [126], and efficient mixed integer programming algorithms that search in the space of supports have been developed [136]. For more than two players, many algorithms have been proposed, but they currently only scale to very small games [70, 105, 126]. Progress has also been made on algorithms for finding equilibria in restricted and/or structured games (*e.g.*, [121, 10, 96, 21, 147]), as well as for finding market equilibria (*e.g.*, [43, 44, 78, 138]).

2.3.2 Algorithms for finding equilibria in extensive form games

Nash equilibria of two-person sequential games with perfect information can be found by simply searching over the game tree.[2] In computer science terms, this is done using *minimax search* (often in conjunction with *α-β-pruning* to reduce the search tree size and thus enhance speed). Minimax search runs in linear time in the size of the game tree.[3]

The differentiating feature of games of imperfect information, such as poker, is that they are not fully observable: when it is an agent's turn to move, she does not have access

[2]This actually yields a solution that satisfies not only the Nash equilibrium solution concept, but a stronger solution concept called *subgame perfect Nash equilibrium* [143].

[3]This type of algorithm has its limits, of course, particularly when the game tree is huge, but extremely effective game-playing agents can be developed, even when the size of the game tree prohibits complete search, by evaluating intermediate nodes using a heuristic evaluation and then treating those nodes as leaves of the tree.

to all of the information about the world. In such games, the decision of what to do at a point in time cannot generally be optimally made without considering decisions at all other points in time (including ones on other paths of play) because those other decisions affect the probabilities of being at different states at the current point in time. Thus the algorithms for perfect information games do not solve games of imperfect information.

As discussed previously, one could try to find an equilibrium of a sequential game by converting the normal form, but this is computationally intractable. However, by observing that one needs to consider only sequences of moves rather than pure strategies, one arrives at a more compact representation, the *sequence form*, which is linear in the size of the game tree [130, 144, 84, 160]. For two-person zero-sum games, there is a polynomial-sized (in the size of the game tree) linear programming formulation (linear complementarity in the non-zero-sum case) based on the sequence form such that strategies for Players 1 and 2 correspond to primal and dual variables. Thus, the equilibria of reasonable-sized two-person2- games can be computed using this method [160, 86, 88].[4] However, this approach still yields enormous (unsolvable) optimization problems for many real-world games, such as poker.

The Nash equilibrium problem for two-person zero-sum sequential games of imperfect information can be formulated using the sequence form representation [130, 84, 160] as the following saddle-point problem:

$$\max_{\mathbf{x}\in Q_1} \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}\rangle = \min_{\mathbf{y}\in Q_2} \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}, \mathbf{x}\rangle. \tag{2.1}$$

In this formulation, $\mathbf{x}$ is Player 1's strategy and $\mathbf{y}$ is Player 2's strategy. The bilinear term $\langle A\mathbf{y}, \mathbf{x}\rangle$ is the payoff that Player 1 receives from Player 2 when the players play the strategies $\mathbf{x}$ and $\mathbf{y}$. The strategy spaces are represented by $Q_i \subseteq \mathbb{R}^{|S_i|}$, where S_i is the set of sequences of moves of Player i, and Q_i is the *set of realization plans* of Player i. Thus $\mathbf{x}$ ($\mathbf{y}$) encodes probability distributions over actions at each point in the game where Player 1 (2) acts. The set Q_i has an explicit linear description of the form $\{z \geq 0 : Ez = \mathbf{e}\}$. Consequently, as mentioned above, problem (2.1) can be modeled as a linear program (see [160] for details). We will return to this formulation in Chapter 8 when we discuss algorithms for solving it.

[4]Recently this approach was extended to handle computing *sequential equilibria* [90] as well [109].

2.3.3 Algorithmic approximations

As discussed above, the equilibrium problem for two-person zero-sum games can be modeled as a linear program (LP), which can in turn be solved using the simplex method. This approach has inherent features which we can leverage into desirable properties in the context of solving games.

In the LP, primal solutions correspond to strategies of Player 2, and dual solutions correspond to strategies of Player 1. There are two versions of the simplex method: the primal simplex and the dual simplex. The primal simplex maintains primal feasibility and proceeds by finding better and better primal solutions until the dual solution vector is feasible, at which point optimality has been reached. Analogously, the dual simplex maintains dual feasibility and proceeds by finding increasingly better dual solutions until the primal solution vector is feasible. (The dual simplex method can be thought of as running the primal simplex method on the dual problem.) Thus, the primal and dual simplex methods serve as *anytime* algorithms (for a given abstraction) for Players 2 and 1, respectively. At any point in time, they can output the best strategies found so far.

Also, for any feasible solution to the LP, we can get bounds on the quality of the strategies by examining the primal and dual solutions. (When using the primal simplex method, dual solutions may be read off of the LP tableau.) Every feasible solution of the dual yields an upper bound on the optimal value of the primal, and vice versa [32, p. 57]. Thus, without requiring further computation, we get lower bounds on the expected utility of each agent's strategy against that agent's worst-case opponent.

One problem with the simplex method is that it is not a primal-dual algorithm, that is, it does not maintain both primal and dual feasibility throughout its execution. (In fact, it only obtains primal and dual feasibility at the very end of execution.) In contrast, there are interior-point methods for linear programming that maintain primal and dual feasibility throughout the execution. For example, many interior-point path-following algorithms have this property [163, Ch. 5]. We observe that running such a linear programming method yields a method for finding ϵ-equilibria (*i.e.*, strategy profiles in which no agent can increase her expected utility by more than ϵ by deviating). A threshold on ϵ can also be used as a termination criterion for using the method as an anytime algorithm. Furthermore, interior-point methods in this class have polynomial-time worst-case run time, as opposed to the simplex algorithm, which takes exponentially many steps in the worst case.

Chapter 3

Poker

Poker is an enormously popular card game. Every year, hundreds of millions of dollars change hands at poker tables in casinos around the world. Increasingly, poker players compete in online casinos, and television stations regularly broadcast poker tournaments. Poker has been identified as an important research area in AI due to the uncertainty stemming from opponents' cards, opponents' future actions, and chance moves, among other reasons [15].

3.1 Poker rules and variants

There are many variations of poker. In this thesis, we work with two particular versions: Texas Hold'em and Rhode Island Hold'em. We discuss each of these in the following two subsections.

3.1.1 Texas Hold'em

Texas Hold'em itself has many variations. The main feature differentiating variations is the betting structure. One type of betting structure is *limit*, in which players are only allowed to bet a single fixed amount at each betting opportunity, and *no-limit*, in which players may bet any amount up to their current bankroll.[1] We first describe the rules of Texas Hold'em in the context of a limit betting structure, and then we describe the no-limit betting structure.

[1]The third-most popular variant is *pot-limit*, in which players may bet any amount up to the current size of the pot. We do not discuss pot-limit poker in this thesis.

The basic rules of two-player limit Texas Hold'em are as follows.[2]

Blinds Before any cards are dealt, the first player, called the *small blind*, contributes one chip to the pot; the second player (*big blind*) contributes two chips.[3]

Deal Each player is dealt two *hole cards* from a randomly shuffled standard deck of 52 cards.

Pre-flop Next, the players participate in the first of four betting rounds, called the *pre-flop*. The small blind acts first; she may either call the big blind (contribute one chip), raise (three chips), or fold (zero chips). The players then alternate either calling the current bet (contributing two chips), raising the bet (four chips), or folding (zero chips). In the event of a fold, the folding player forfeits the game and the other player wins all of the chips in the pot. Once a player calls a bet, the betting round finishes. The number of raises allowed is typically limited to four in each round.

Flop The second round is called the *flop*. Three *community cards* are dealt face-up, and a betting round takes place with bets equal to two chips. The big blind player is the first to act, and there are no blind bets placed in this round.

Turn and River The third and fourth rounds are called the *turn* and the *river*. In each round, a single card is dealt face-up, and a betting round similar to the flop betting round takes place, but with bets equal to four chips (twice as much as in the flop betting round).

Showdown The *showdown* occurs when the last betting round ends with neither player folding. Each player uses the seven cards available (their two hole cards along with the five community cards) to form the best five-card poker hand, where the hands are ranked in the usual order. The player with the best hand wins the pot; in the event of a tie, the players split the pot.

No-limit Texas Hold'em is played as above, but there is no limit on the number of chips that may be bet as long as the bettor has that many chips available. In particular, a player may go "all-in" and bet all of their remaining chips. In the event that a player is facing a bet or a raise, if the player chooses to re-raise they must do so in an amount at least the size of the existing bet or raise.

[2]It is straightforward to modify the rules of two-player Texas Hold'em to allow for multiple players. As most prior work on poker, in our application of our abstraction algorithms we focus on the setting with two players, called *heads-up*. However, our abstraction algorithms are applicable to the multiple player case.

[3]The monetary value of a chip is irrelevant. Thus, we refer only to the quantity of chips.

3.1.2 Rhode Island Hold'em

Rhode Island Hold'em poker was invented as a testbed for computational game playing [146]. Its game tree has 3.1 billion nodes, and it shares many of the interesting complications present in Texas Hold'em. Although it has been solved optimally [61], it remains useful in its intended role as a testbed. The rules of the game are as follows:

Ante Each player puts an *ante* of 500 chips into the *pot*.

Pre-flop Both players receive a single, face-down *hole card*. The players participate in one betting round. The possible actions are *check* (not placing any money in the pot) or *bet* (placing 1000 chips into the pot) if no bets have been placed. If a bet has been placed, then the player may *fold* (thus forfeiting the game along with the chips in the pot), *call* (adding chips to the pot equal to the last bet), or *raise* (calling the current bet and making an additional bet). In Rhode Island Hold'em, the players are limited to three bets each per betting round. (A raise equals two bets.) In the this betting round, the bets are equal to 1000 chips.

Flop A *community card* is dealt face up. A betting round similar to the Pre-flop takes place, but with bets equal to 2000 chips.

Turn A second *community card* is dealt face up. A betting round identical to the Flop takes place.

Showdown If neither player folds, both players turn over their cards. The player with the best 3-card poker hand wins the chips in the pot. The pot is split evenly if the players have the same hand rank.

Table 3.1 contains the rankings of three-card poker hands, which are different than the normal rankings of five-card poker hands.

3.2 Related research on poker

Almost since the field's founding, game theory has been used to analyze different aspects of poker [23, 159, 8, 92, 113, 7, 77, 80, 119, 69, 51, 35, 132, 133]. That early work was limited to tiny games that could be solved by hand. In fact, most of that work was focused on computing *analytical solutions* for various stylized versions of poker. For example, one typical assumption in that line of work is that the cards are drawn uniformly at random

Hand	**Prob.**	**Description**	**Example**
Straight flush	0.00217	3 cards w/ consecutive rank & same suit	K♠, Q♠, J♠
Three of a kind	0.00235	3 cards of the same rank	Q♠, Q♡, Q♣
Straight	0.03258	3 cards w/ consecutive rank	3♣, 4♠, 5♡
Flush	0.04959	3 cards of the same suit	2♢, 5♢, 8♢
Pair	0.16941	2 cards of the same rank	2♢, 2♠, 3♡
High card	0.74389	None of the above	J♣, 9♡, 2♠

Table 3.1: Ranking of three-card poker hands, from highest to lowest.

from the unit interval, followed by a simple betting protocol. Although this approach seems likely to be of use only for extremely small games, recently there has been renewed interest in extending some of these models and determing analytical solutions from them with the goal of applying them to certain situations in real poker games [46, 47, 4]. Simplified versions of poker have also been developed for illustrative examples in education [127]. From a cognitive modeling and analysis perspective, poker has proved to be a fertile environment for research [48, 26, 27].

Subsequent to the development of the sequence form and linear programming-based solution methodology methods [84, 130, 160], larger-scale game-theoretic approaches to poker AI began to appear. Koller and Pfeffer [88] determined solutions to poker games with up to 140,000 nodes using the sequence form and linear programming. The use of abstraction and game-theoretic solutions as applied to Texas Hold'em appeared as early as 2000 in the undergraduate thesis of Takusagawa [154]. For Rhode Island Hold'em, game theory-based solutions were developed using a lossy abstraction followed by linear programming [146]. Another notable game-theoretic approach to Texas Hold'em was *SparBot* [14]. These early approaches used either expert-designed abstractions or simple forms of heuristic abstraction.

More recently, other poker-playing programs that use automated abstraction followed by equilibrium-finding have been developed [3, 79, 166, 165]. Those other approaches differ from ours primarily in the abstraction and equilibrium-finding algorithms used.

In addition to game theory-based research, there has also been recent work in the area of *opponent modeling* [18, 17, 19, 38, 15, 37, 13, 75, 152, 153, 141] in which a poker-playing program attempts to identify and exploit weaknesses in the opponents. The opponent modeling-based poker-playing programs typically combine opponent modeling with simulation-based tree search [18, 17, 19, 38, 15], and do not try to approximate game-theoretic solutions. Another non-game-theoretic approach employs Bayesian techniques to model inference and uncertainty in the game [89].

A *no-limit poker tournament* is a variant of poker in which play is repeated until all but one of the players have lost all of their chips. Approaches to this problem have used action abstraction (the only possible actions being to fold or go all-in) and card abstraction (suit isomorphisms are applied to the pre-flop hands). Both two-player tournaments [110] and three-player tournaments [53] have been studied. The focus of that research has been on identifying near-optimal solutions to the final stages of the tournament when the blinds are high relative to the number of chips that a player has. In contrast, our research is geared towards a game when more complicated betting structures are needed and when the blinds are low relative the number of chips (which is the typical situation encountered).

Part II

Automated Abstraction

Chapter 4

Automated Abstraction Overview

Game theory-based approaches to constructing agents for competitive environments have emerged as a dominant methodology in many settings, such as poker. Determining how to play game-theoretically requires solving for the equilibrium of the game. In two-person zero-sum sequential games of imperfect information, this can be done in polynomial time using linear programming [130, 86, 160], although general-purpose linear programming solvers have limited scalability [61]. Recent research in equilibrium-finding technology for two-person zero-sum games has led to dramatic increases in the scalability of equilibrium-finding algorithms [74, 56, 165, 166, 106, 57], but it is still infeasible to scale to the size of games that are encountered in practice. For example, the game tree of heads-up (*i.e.*, two-player) limit Texas Hold'em poker has 10^{18} nodes making it far beyond the scalability of the state-of-the-art equilibrium-finding algorithms. For two-person general-sum games or games with more than two players, the need for abstraction is even more critical since finding an equilibrium in those games is PPAD-complete [30].

To handle such massive game trees, a recent practical approach has emerged: automated abstraction algorithms that take the rules of the game as input and generate a game that is much smaller but still strategically similar (or, in some cases equivalent [61]) to the original game. After finding an equilibrium for the abstracted game, one can map those strategies back into a strategy for the original game. For example, since 2003 there has been tremendous progress on developing computer programs for playing (both limit and no-limit) heads-up Texas Hold'em poker, and all the leading programs are nowadays developed using automated abstraction followed by equilibrium finding in the abstracted game [165, 166, 64, 65].

We have identified three high-level classes of abstraction that can be performed on an imperfect-information sequential game. In the remainder of Part II of this thesis, we develop algorithms and present experiments on each of these different classes.

1. **Information abstraction.** This type of abstraction filters the information signaled to players by nature. For example, in poker this type of abstraction groups different poker hands into the same "bucket". Chapter 5 discusses our *GameShrink* algorithm which performs abstraction in a way that allows one to find an equilibrium in the original game. Chapter 6 describes four of our algorithms for information abstraction, and presents experimental results comparing the performance of each of the algorithms. We present both controlled experiments measuring the relative performance of the algorithms as well as experiments comparing how well our poker-playing programs built using these algorithms compare against other programs built using a wide variety of approaches.

2. **Stage abstraction.** Many games feature distinct stages. In poker, these stages corresponded to betting rounds. Stage abstraction breaks the game into separate *phases*, where each phase contains some number of the stages from the game. These phases are solved separately and then "glued" back together. This process can have ill effects on the quality of the solution if it is not done carefully. In Section 7.2 we describe our techniques for mitigating some of the problems encountered when using stage abstraction, and present experimental results demonstrating the value of the techniques.

3. **Action abstraction.** This type of abstraction treats several distinct actions as the same action. This is particularly useful in games where each player has a large—or even infinite—number of actions at each information set. For example, in no-limit Texas Hold'em poker, even after discretizing the betting actions to integral amounts, each player can have as many as 1000 actions in certain information sets in a restricted version where each player has 1000 chips. Sections 7.3 and 7.4 present our techniques for action abstraction in imperfect-information sequential games, and includes experiments measuring the performance impact of certain types of action abstraction.

Chapter 5

Lossless Information Abstraction

5.1 Introduction

In this chapter, we begin our investigation of abstraction algorithms with the development of an information-based lossless abstraction algorithm. Instead of developing an equilibrium-finding method *per se*, we develop a methodology for automatically abstracting games in such a way that any equilibrium in the smaller (abstracted) game corresponds directly to an equilibrium in the original game. Thus, by computing an equilibrium in the smaller game (using any available equilibrium-finding algorithm), we are able to construct an equilibrium in the original game. The motivation is that an equilibrium for the smaller game can be computed drastically faster than for the original game.

To this end, we introduce *games with ordered signals* (Section 5.3), a broad class of games that has enough structure which we can exploit for abstraction purposes. Instead of operating directly on the game tree (something we found to be technically challenging), we introduce the use of *information filters* (Section 5.3.2), which coarsen the information each player receives. They are used in our analysis and abstraction algorithm. By operating only in the space of filters, we are able to keep the strategic structure of the game intact, while abstracting out details of the game in a way that is lossless from the perspective of equilibrium finding. We introduce the *ordered game isomorphism* to describe strategically symmetric situations and the *ordered game isomorphic abstraction transformation* to take advantange of such symmetries (Section 5.4). As our main equilibrium result we have the following:

Theorem 2 *Let Γ be a game with ordered signals, and let F be an information filter for Γ. Let F' be an information filter constructed from F by one applica-*

tion of the ordered game isomorphic abstraction transformation, and let σ' be a Nash equilibrium strategy profile of the induced game $\Gamma_{F'}$ (i.e., the game Γ using the filter F'). If σ is constructed by using the corresponding strategies of σ', then σ is a Nash equilibrium of Γ_F.

The proof of the theorem uses an equivalent characterization of Nash equilibria: σ is a Nash equilibrium if and only if there exist beliefs μ (players' beliefs about unknown information) at all points of the game reachable by σ such that σ is sequentially rational (*i.e.*, a best response) given μ, where μ is updated using Bayes' rule. We can then use the fact that σ' is a Nash equilibrium to show that σ is a Nash equilibrium considering only local properties of the game.

We also give an algorithm, *GameShrink*, for abstracting the game using our isomorphism exhaustively (Section 5.5). Its complexity is $\tilde{O}(n^2)$, where n is the number of nodes in a structure we call the signal tree. It is no larger than the game tree, and on nontrivial games it is drastically smaller, so *GameShrink* has time and space complexity *sublinear* in the size of the game tree. We present several algorithmic and data structure related speed improvements (Section 5.5.1), and we demonstrate how a simple modification to our algorithm yields an approximation algorithm (Section 5.6).

5.2 Applications

Sequential games of imperfect information are ubiquitous, for example in negotiation and in auctions. Often aspects of a player's knowledge are not pertinent for deciding what action the player should take at a given point in the game. Although in some simple situations some aspects of a player's knowledge are never pertinent (likely indicating a poorly constructed game model), in general, some aspects can be pertinent in certain states of the game while they are not pertinent in other states, and thus cannot be left out of the model completely. Furthermore, it may be highly non-obvious which aspects are pertinent in which states of the game. Our algorithm automatically discovers which aspects are irrelevant in different states, and eliminates those aspects of the game, resulting in a more compact, equivalent game representation.

One broad class of sequential games of imperfect information in which information may or may not be pertinent in various stages of the game is sequential negotiation (potentially over multiple issues). Another broad application area is sequential auctions (potentially over multiple goods). For example, in those states of a 1-object auction where bidder A can infer that his valuation is greater than that of bidder B, bidder A can ignore all his other information about B's signals, although that information would be relevant for inferring B's exact valuation. Furthermore, in some states of the auction, a bidder might not care which exact other bidders have which valuations, but cares about which valuations are held by the other bidders in aggregate (ignoring their identities). Many open-cry sequential auction and negotiation mechanisms fall within the game model studied in this chapter (specified in detail later), as do certain other games in electronic commerce, such as sequences of take-it-or-leave-it offers [135].

Our techniques are in no way specific to an application. In fact, the main experiment that we present in this chapter is on Rhode Island Hold'em poker. We chose a particular poker game as the benchmark problem because it yields an extremely complicated and enormous game tree, it is a game of imperfect information, it is fully specified as a game (and the data is available), and it has been posed as a challenge problem by others [146] (to our knowledge no such challenge problem instances have been proposed for electronic commerce applications that require solving sequential games).

5.2.1 Application to Rhode Island Hold'em poker

As discussed in Chapter 3 Rhode Island Hold'em was invented as a testbed for computational game playing [146]. It was designed so that it was similar in style to Texas Hold'em, yet not so large that devising reasonably intelligent strategies would be impossible. (The rules of Rhode Island Hold'em were given in Section 3.1.2. In Section 5.3.1 we show how Rhode Island Hold'em can be modeled as a game with ordered signals, that is, it fits in our model.) We applied the techniques developed in this chapter to find an exact (minimax) solution to Rhode Island Hold'em, which has a game tree exceeding 3.1 billion nodes.

Applying the sequence form to Rhode Island Hold'em directly without abstraction yields a linear program with 91,224,226 rows, and the same number of columns. This is much too large for (current) linear programming algorithms to handle. We used our *GameShrink* algorithm to reduce this through lossless abstraction, and it yielded a linear program with 1,237,238 rows and columns—with 50,428,638 non-zero coefficients. We then applied iterated elimination of dominated strategies, which further reduced this to 1,190,443 rows and 1,181,084 columns. (Applying iterated elimination of dominated strategies without *GameShrink* yielded 89,471,986 rows and 89,121,538 columns, which still would have been prohibitively large to solve.) *GameShrink* required less than one second to perform the shrinking (i.e., to compute all of the ordered game isomorphic abstraction transformations). Using a 1.65GHz IBM eServer p5 570 with 64 gigabytes of RAM (the linear program solver actually needed 25 gigabytes), we solved it in 7 days and 17 hours using the interior-point barrier method of CPLEX version 9.1.2. We demonstrated our optimal Rhode Island Hold'em poker player at the AAAI-05 conference [58], and it is available for play on-line at `http://www.cs.cmu.edu/~gilpin/gsi.html`.

While others have worked on computer programs for playing Rhode Island Hold'em [146], no optimal strategy has been found before. At the time of our solution, this was the largest poker game solved to date by over four orders of magnitude. We have since solved larger games using gradient-based equilibrium-finding algorithms, as discussed in Part III of this thesis.

5.3 Games with ordered signals

We work with a slightly restricted class of games, as compared to the full generality of the extensive form. This class, which we call *games with ordered signals*, is highly structured, but still general enough to capture a wide range of strategic situations. A game with ordered signals consists of a finite number of rounds. Within a round, the players play a game on a directed tree (the tree can be different in different rounds). The only uncertainty players face stems from private signals the other players have received and from the unknown future signals. In other words, players observe each others' actions, but potentially not nature's actions. In each round, there can be public signals (announced to all players) and private signals (confidentially communicated to individual players). For simplicity, we assume—as is the case in most recreational games—that within each round, the number of private signals received is the same across players (this could quite likely be relaxed). We also assume that the legal actions that a player has are independent of the signals received. For example, in poker, the legal betting actions are independent of the cards received. Finally, the strongest assumption is that there is a partial ordering over sets of signals, and the payoffs are increasing (not necessarily strictly) in these signals. For example, in poker, this partial ordering corresponds exactly to the ranking of card hands.

Definition 5 *A* game with ordered signals *is a tuple* $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ *where:*

1. $I = \{1, \ldots, n\}$ *is a finite set of players.*
2. $G = \langle G^1, \ldots, G^r \rangle$, $G^j = (V^j, E^j)$, *is a finite collection of finite directed trees with*

nodes V^j and edges E^j. Let Z^j denote the leaf nodes of G^j and let $N^j(v)$ denote the outgoing neighbors of $v \in V^j$. G^j is the stage game *for round j.*

3. *$L = \langle L^1, \ldots, L^r \rangle$, $L^j : V^j \setminus Z^j \to I$ indicates which player acts (chooses an outgoing edge) at each internal node in round j.*

4. *Θ is a finite set of* signals.

5. *$\kappa = \langle \kappa^1, \ldots, \kappa^r \rangle$ and $\gamma = \langle \gamma^1, \ldots, \gamma^r \rangle$ are vectors of nonnegative integers, where κ^j and γ^j denote the number of public and private signals (per player), respectively, revealed in round j. Each signal $\theta \in \Theta$ may only be revealed once, and in each round every player receives the same number of private signals, so we require $\sum_{j=1}^{r} \kappa^j + n\gamma^j \le |\Theta|$. The public information revealed in round j is $\alpha^j \in \Theta^{\kappa^j}$ and the public information revealed in all rounds up through round j is $\tilde{\alpha}^j = (\alpha^1, \ldots, \alpha^j)$. The private information revealed to player $i \in I$ in round j is $\beta_i^j \in \Theta^{\gamma^j}$ and the private information revaled to player $i \in I$ in all rounds up through round j is $\tilde{\beta}_i^j = (\beta_i^1, \ldots, \beta_i^j)$. We also write $\tilde{\beta}^j = \left(\tilde{\beta}_1^j, \ldots, \tilde{\beta}_n^j\right)$ to represent all private information up through round j, and $\left(\tilde{\beta}'^j_i, \tilde{\beta}^j_{-i}\right) = \left(\tilde{\beta}_1^j, \ldots, \tilde{\beta}_{i-1}^j, \tilde{\beta}'^j_i, \tilde{\beta}_{i+1}^j, \ldots, \tilde{\beta}_n^j\right)$ is $\tilde{\beta}^j$ with $\tilde{\beta}_i^j$ replaced with $\tilde{\beta}'^j_i$. The total information revealed up through round j, $\left(\tilde{\alpha}^j, \tilde{\beta}^j\right)$, is said to be* legal *if no signals are repeated.*

6. *p is a probability distribution over Θ, with $p(\theta) > 0$ for all $\theta \in \Theta$. Signals are drawn from Θ according to p without replacement, so if X is the set of signals already revealed, then*
$$p(x \mid X) = \begin{cases} \frac{p(x)}{\sum_{y \notin X} p(y)} & \text{if} \quad x \notin X \\ 0 & \text{if} \quad x \in X. \end{cases}$$

7. *$\succeq$ is a partial ordering of subsets of Θ and is defined for at least those pairs required by u.*

8. *$\omega : \bigcup_{j=1}^{r} Z^j \to \{over, continue\}$ is a mapping of terminal nodes within a stage game to one of two values:* over, *in which case the game ends, or* continue, *in which case the game continues to the next round. Clearly, we require $\omega(z) = over$ for all $z \in Z^r$. Note that ω is independent of the signals. Let $\omega^j_{over} = \{z \in Z^j \mid \omega(z) = over\}$ and $\omega^j_{cont} = \{z \in Z^j \mid \omega(z) = continue\}$.*

9. *$u = (u^1, \ldots, u^r)$, $u^j : \bigtimes_{k=1}^{j-1} \omega^k_{cont} \times \omega^j_{over} \times \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{i=1}^{n} \bigtimes_{k=1}^{j} \Theta^{\gamma^k} \to \mathbb{R}^n$ is a utility function such that for every j, $1 \le j \le r$, for every $i \in I$, and for every $\tilde{z} \in$*

$\bigtimes_{k=1}^{j-1} \omega^k_{cont} \times \omega^j_{over}$, *at least one of the following two conditions holds:*

(a) Utility is signal independent: $u^j_i(\tilde{z}, \vartheta) = u^j_i(\tilde{z}, \vartheta')$ *for all legal* $\vartheta, \vartheta' \in \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{i=1}^{n} \bigtimes_{k=1}^{j} \Theta^{\gamma^k}$.

(b) $\succeq$ *is defined for all legal signals* $(\tilde{\alpha}^j, \tilde{\beta}^j_i)$ *and* $(\tilde{\alpha}^j, \tilde{\beta}'^j_i)$ *through round* j *and a player's utility is increasing in her private signals, everything else equal:*

$$\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right) \succeq \left(\tilde{\alpha}^j, \tilde{\beta}'^j_i\right) \Longrightarrow u_i\left(\tilde{z}, \tilde{\alpha}^j, \left(\tilde{\beta}^j_i, \tilde{\beta}^j_{-i}\right)\right) \geq u_i\left(\tilde{z}, \tilde{\alpha}^j, \left(\tilde{\beta}'^j_i, \tilde{\beta}^j_{-i}\right)\right).$$

We will use the term *game with ordered signals* and the term *ordered game* interchangeably.

5.3.1 Rhode Island Hold'em modeled as an ordered game

In this section we describe how Rhode Island Hold'em can be defined as an ordered game in accordance with Definition 5. To make the definition of ordered games concrete, here we define each of the components of the tuple $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ for Rhode Island Hold'em. There are two players so $I = \{1, 2\}$. There are three rounds, and the stage game is the same in each round so we have $G = \langle G_{RI}, G_{RI}, G_{RI} \rangle$ where G_{RI} is given in Figure 5.1, which also specifies the player label L.

Θ is the standard deck of 52 cards. The community cards are dealt in the second and third rounds, so $\kappa = \langle 0, 1, 1 \rangle$. Each player receives a single face down card in the first round only, so $\gamma = \langle 1, 0, 0 \rangle$. p is the uniform distribution over Θ. $\succeq$ is defined for three card hands and is defined using the ranking given in Table 3.1. The game-ending nodes ω are denoted in Figure 5.1 by ω. u is defined as in the above description; it is easy to verify that it satisfies the necessary conditions.

5.3.2 Information filters

In this subsection, we define an *information filter* for ordered games. Instead of completely revealing a signal (either public or private) to a player, the signal first passes through this filter, which outputs a *coarsened* signal to the player. By varying the filter applied to a game, we are able to obtain a wide variety of games while keeping the underlying action

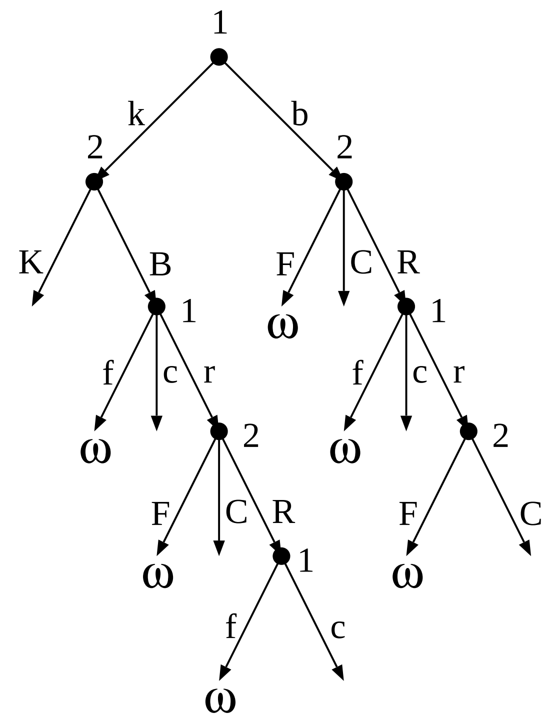

Figure 5.1: Stage game G_{RI}, player label L, and game-ending nodes ω for Rhode Island Hold'em. The action labels denote which action the player is taking: k (check), b (bet), f (fold), c (call), and r (raise). Lower case letters indicate player 1 actions and upper case letters indicate player 2 actions.

space of the game intact. We will use this when designing our abstraction techniques. Formally, an information filter is as follows.

Definition 6 *Let* $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$ *be an ordered game. Let* $S^j \subseteq \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{k=1}^{j} \Theta^{\gamma^k}$ *be the set of legal signals (i.e., no repeated signals) for one player through round* j*. An* information filter *for* Γ *is a collection* $F = \langle F^1, \ldots, F^r\rangle$ *where each* F^j *is a function* $F^j : S^j \to 2^{S^j}$ *such that each of the following conditions hold:*

1. *(Truthfulness)* $(\tilde{\alpha}^j, \tilde{\beta}^j_i) \in F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)$ *for all legal* $(\tilde{\alpha}^j, \tilde{\beta}^j_i)$.
2. *(Independence) The range of* F^j *is a partition of* S^j.
3. *(Information preservation) If two values of a signal are distinguishable in round* k*, then they are distinguishable for each round* $j > k$*. Let* $m^j = \sum_{l=1}^{j} \kappa^l + \gamma^l$*. We*

require that for all legal $(\theta_1, \ldots, \theta_{m^k}, \ldots, \theta_{m^j}) \subseteq \Theta$ *and* $(\theta'_1, \ldots, \theta'_{m^k}, \ldots, \theta'_{m^j}) \subseteq \Theta$*:*

$$(\theta'_1, \ldots, \theta'_{m^k}) \notin F^k(\theta_1, \ldots, \theta_{m^k}) \Rightarrow (\theta'_1, \ldots, \theta'_{m^k}, \ldots, \theta'_{m^j}) \notin F^j(\theta_1, \ldots, \theta_{m^k}, \ldots, \theta_{m^j}).$$

A game with ordered signals Γ and an information filter F for Γ defines a new game Γ_F. We refer to such games as *filtered ordered games*. We are left with the original game if we use the identity filter $F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right) = \left\{\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right\}$. We have the following simple (but important) result:

Proposition 1 *A filtered ordered game is an extensive form game satisfying perfect recall. (For the unfamiliar reader, the definition of games with perfect recall is given in Appendix 2.1.)*

A simple proof proceeds by constructing an extensive form game directly from the ordered game, and showing that it satisfies perfect recall. In determining the payoffs in a game with filtered signals, we take the average over all real signals in the filtered class, weighted by the probability of each real signal occurring.

5.3.3 Strategies and Nash equilibrium

We are now ready to define behavior strategies in the context of filtered ordered games.

Definition 7 *A* behavior strategy *for player* i *in round* j *of* $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$ *with information filter* F *is a probability distribution over possible actions, and is defined for each player* i*, each round* j*, and each* $v \in V^j \setminus Z^j$ *for* $L^j(v) = i$*:*

$$\sigma^j_{i,v} : \bigtimes_{k=1}^{j-1} \omega^k_{cont} \times Range\left(F^j\right) \to \Delta\left\{w \in V^j \mid (v, w) \in E^j\right\}.$$

($\Delta(X)$ *is the set of probability distributions over a finite set* X*.) A behavior strategy for player* i *in round* j *is* $\sigma^j_i = (\sigma^j_{i,v_1}, \ldots, \sigma^j_{i,v_m})$ *for each* $v_k \in V^j \setminus Z^j$ *where* $L^j(v_k) = i$*. A behavior strategy for player* i *in* Γ *is* $\sigma_i = (\sigma^1_i, \ldots, \sigma^r_i)$*. A* strategy profile *is* $\sigma = (\sigma_1, \ldots, \sigma_n)$*. A strategy profile with* σ_i *replaced by* σ'_i *is* $(\sigma'_i, \sigma_{-i}) = (\sigma_1, \ldots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \ldots, \sigma_n)$*.*

By an abuse of notation, we will say player i receives an expected payoff of $u_i(\sigma)$ when all players are playing the strategy profile σ. Strategy σ_i is said to be player i's *best* *response* to σ_{-i} if for all other strategies σ_i' for player i we have $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i', \sigma_{-i})$. σ is a *Nash equilibrium* if, for every player i, σ_i is a best response for σ_{-i}. A Nash equilibrium always exists in finite extensive form games [112], and one exists in behavior strategies for games with perfect recall [93]. Using these observations, we have the following corollary to Proposition 1:

Corollary 1 *For any filtered ordered game, a Nash equilibrium exists in behavior strateges.*

5.4 Equilibrium-preserving abstractions

In this section, we present our main technique for reducing the size of games. We begin by defining a *filtered signal tree* which represents all of the chance moves in the game. The bold edges (i.e. the first two levels of the tree) in the game trees in Figure 5.2 correspond to the filtered signal trees in each game.

Definition 8 *Associated with every ordered game $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$ and information filter F is a* filtered signal tree*, a directed tree in which each node corresponds to some revealed (filtered) signals and edges correspond to revealing specific (filtered) signals. The nodes in the filtered signal tree represent the set of all possible revealed filtered signals (public and private) at some point in time. The filtered public signals revealed in round j correspond to the nodes in the κ^j levels beginning at level $\sum_{k=1}^{j-1}\left(\kappa^k + n\gamma^k\right)$ and the private signals revealed in round j correspond to the nodes in the $n\gamma^j$ levels beginning at level $\sum_{k=1}^{j}\kappa^k + \sum_{k=1}^{j-1} n\gamma^k$. We denote children of a node x as $N(x)$. In addition, we associate weights with the edges corresponding to the probability of the particular edge being chosen given that its parent was reached.*

In many games, there are certain situations in the game that can be thought of as being strategically equivalent to other situations in the game. By melding these situations together, it is possible to arrive at a strategically equivalent smaller game. The next two definitions formalize this notion via the introduction of the *ordered game isomorphic* relation and the *ordered game isomorphic abstraction transformation*.

Definition 9 *Two subtrees beginning at internal nodes x and y of a filtered signal tree are* ordered game isomorphic *if x and y have the same parent and there is a bijection $f : N(x) \to N(y)$, such that for $w \in N(x)$ and $v \in N(y)$, $v = f(w)$ implies the weights on the edges (x, w) and (y, v) are the same and the subtrees beginning at w and v are ordered*

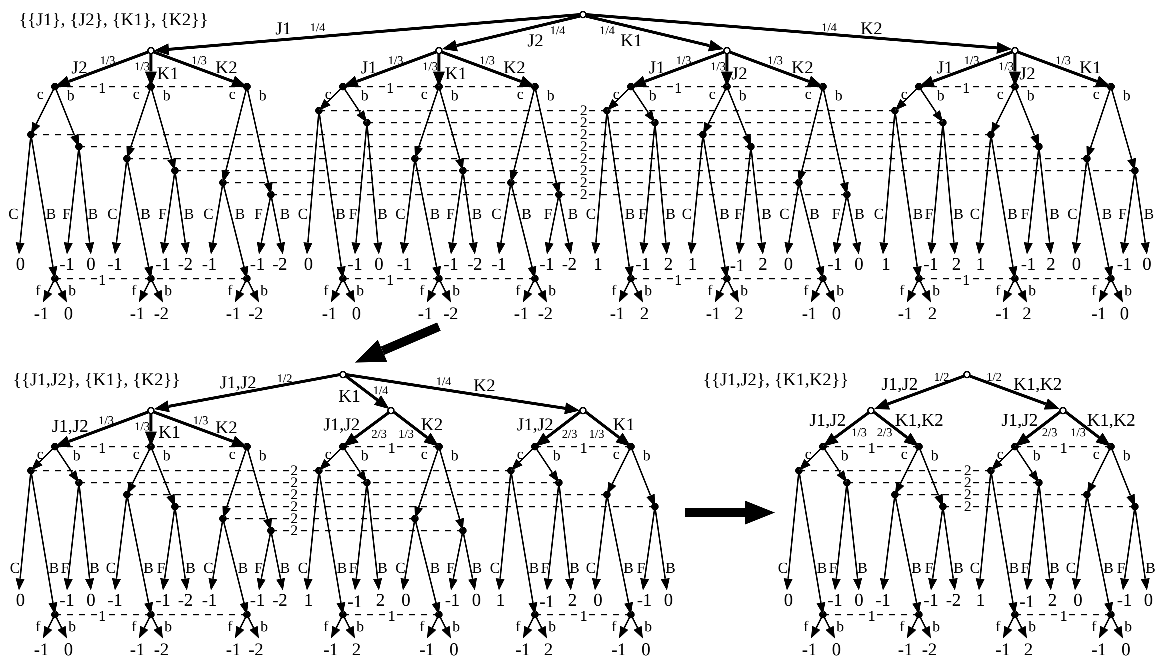

Figure 5.2: *GameShrink* applied to a tiny two-person four-card (two Jacks and two Kings) poker game. Next to each game tree is the range of the information filter F. Dotted lines denote information sets, which are labeled by the controlling player. Open circles are chance nodes with the indicated transition probabilities. The root node is the chance node for player 1's card, and the next level is for player 2's card. The payment from player 2 to player 1 is given below each leaf. In this example, the algorithm reduces the game tree from 113 nodes to 39 nodes.

game isomorphic. Two leaves (corresponding to filtered signals ϑ and ϑ' up through round r) are ordered game isomorphic if for all $\tilde{z} \in \bigtimes_{j=1}^{r-1} \omega_{cont}^{j} \times \omega_{over}^{r}$, $u^r(\tilde{z}, \vartheta) = u^r(\tilde{z}, \vartheta')$.

Definition 10 *Let $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ be an ordered game and let F be an information filter for Γ. Let ϑ and ϑ' be two information structures where the subtrees in the induced filtered signal tree corresponding to the nodes ϑ and ϑ' are ordered game isomorphic, and ϑ and ϑ' are at either level $\sum_{k=1}^{j-1}\left(\kappa^k + n\gamma^k\right)$ or $\sum_{k=1}^{j}\kappa^k + \sum_{k=1}^{j-1} n\gamma^k$ for some round j. The* ordered game isomorphic abstraction transformation *is given by creating a new information filter F':*

$$F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = \begin{cases} F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) & \text{if } \left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \notin \vartheta \cup \vartheta' \\ \vartheta \cup \vartheta' & \text{if } \left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \in \vartheta \cup \vartheta'. \end{cases}$$

Figure 5.2 shows the ordered game isomorphic abstraction transformation applied twice to a tiny poker game. Theorem 2, our main equilibrium result, shows how the ordered game isomorphic abstraction transformation can be used to compute equilibria faster.

Theorem 2 *Let $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ be an ordered game and F be an information filter for Γ. Let F' be an information filter constructed from F by one application of the ordered game isomorphic abstraction transformation. Let σ' be a Nash equilibrium of the induced game $\Gamma_{F'}$. If we take $\sigma_{i,v}^{j}\left(\tilde{z}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) = \sigma_{i,v}'^{j}\left(\tilde{z}, F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right)$, σ is a Nash equilibrium of Γ_F.*

PROOF. For an extensive form game, a *belief system* μ assigns a probability to every decision node x such that $\sum_{x \in h} \mu(x) = 1$ for every information set h. A strategy profile σ is *sequentially rational at h given belief system μ* if

$$u_i(\sigma_i, \sigma_{-i} \mid h, \mu) \geq u_i(\tau_i, \sigma_{-i} \mid h, \mu)$$

for all other strategies τ_i, where i is the player who controls h. A basic result [104, Proposition 9.C.1] characterizing Nash equilibria dictates that σ is a Nash equilibrium if and only if there is a belief system μ such that for every information set h with $\Pr(h \mid \sigma) > 0$, the following two conditions hold: (C1) σ is sequentially rational at h given μ; and (C2) $\mu(x) = \frac{\Pr(x \mid \sigma)}{\Pr(h \mid \sigma)}$ for all $x \in h$. Since σ' is a Nash equilibrium of Γ', there exists such a belief system μ' for $\Gamma_{F'}$. Using μ', we will construct a belief system μ for Γ and show that conditions C1 and C2 hold, thus supporting σ as a Nash equilibrium.

Fix some player $i \in I$. Each of i's information sets in some round j corresponds to filtered signals $F^j\left(\tilde{\alpha}^{*j}, \tilde{\beta}_i^{*j}\right)$, history in the first $j-1$ rounds $(z_1, \ldots, z_{j-1}) \in \bigtimes_{k=1}^{j-1} \omega_{cont}^k$, and history so far in round j, $v \in V^j \setminus Z^j$. Let $\tilde{z} = (z_1, \ldots, z_{j-1}, v)$ represent all of the player actions leading to this information set. Thus, we can uniquely specify this information set using the information $\left(F^j\left(\tilde{\alpha}^{*j}, \tilde{\beta}_i^{*j}\right), \tilde{z}\right)$.

Each node in an information set corresponds to the possible private signals the other players have received. Denote by $\hat{\beta}$ some legal

$$(F^j(\tilde{\alpha}^j, \tilde{\beta}_1^j), \ldots, F^j(\tilde{\alpha}^j, \tilde{\beta}_{i-1}^j), F^j(\tilde{\alpha}^j, \tilde{\beta}_{i+1}^j), \ldots, F^j(\tilde{\alpha}^j, \tilde{\beta}_n^j)).$$

In other words, there exists $(\tilde{\alpha}^j, \tilde{\beta}_1^j, \ldots, \tilde{\beta}_n^j)$ such that $(\tilde{\alpha}^j, \tilde{\beta}_i^j) \in F^j(\tilde{\alpha}^{*j}, \tilde{\beta}_i^{*j})$, $(\tilde{\alpha}^j, \tilde{\beta}_k^j) \in F^j(\tilde{\alpha}^j, \tilde{\beta}_k^j)$ for $k \neq i$, and no signals are repeated. Using such a set of signals $(\tilde{\alpha}^j, \tilde{\beta}_1^j, \ldots, \tilde{\beta}_n^j)$, let $\hat{\beta}'$ denote $(F'^j(\tilde{\alpha}^j, \tilde{\beta}_1^j), \ldots, F'^j(\tilde{\alpha}^j, \tilde{\beta}_{i-1}^j), F'^j(\tilde{\alpha}^j, \tilde{\beta}_{i+1}^j), \ldots, F'^j(\tilde{\alpha}^j, \tilde{\beta}_n^j))$. (We will abuse notation and write $F'^j_{-i}\left(\hat{\beta}\right) = \hat{\beta}'$.) We can now compute μ directly from μ':

$$\mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) = \begin{cases} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\ \qquad \text{if } F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \neq F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \text{ or } \hat{\beta} = \hat{\beta}' \\ p^*\mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\ \qquad \text{if } F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \text{ and } \hat{\beta} \neq \hat{\beta}' \end{cases}$$

where $p^* = \frac{\Pr\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right)}{\Pr\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right)}$. The following three claims show that μ as calculated above supports σ as a Nash equilibrium.

Claim 1 *μ is a valid belief system for Γ_F.*

PROOF.[of Claim 1] Let h be player i's information set after some history $\left(F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right)$. Clearly $\mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \geq 0$ for all $\hat{\beta} \in h$. We need to show

$$\sum_{\hat{\beta} \in h} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) = 1.$$

CASE 1. $F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \neq F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)$. From the construction of F', $F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)$ is ordered game isomorphic to some $F^j\left(\tilde{\alpha}'^j \tilde{\beta}_i'^j\right)$ with $F^j\left(\tilde{\alpha}'^j \tilde{\beta}_i'^j\right) \neq F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)$. Let h' be player i's information set corresponding to the history $\left(F^j\left(\tilde{\alpha}'^j, \tilde{\beta}_i'^j\right), \tilde{z}\right)$. By the definition of the ordered game isomorphism, there exists a perfect matching between the nodes in the information set h and h', where each matched pair of nodes corresponds to a pair of ordered

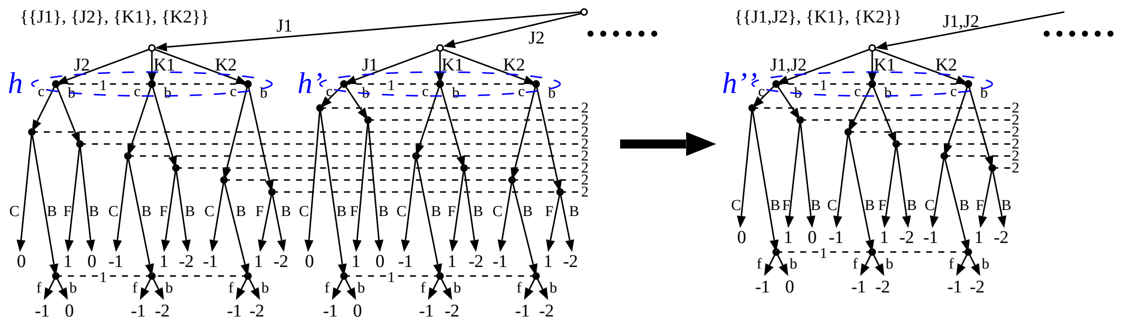
Figure 5.3: Illustration of Case 1 of Claim 1.

game isomorphic information structures. Since we have that $F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = F'^j\left(\tilde{\alpha}'^j, \tilde{\beta}_i'^j\right)$, each edge in the matching corresponds to a node in the information set corresponding to the history $\left(F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right)$ in $\Gamma_{F'}$; denote this information set by h''. (See Figure 5.3.)

Thus, there is a bijection between h and h'' defined by the perfect matching. Using this matching:

$$\begin{aligned}
\sum_{\hat{\beta}\in h} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) &= \sum_{\hat{\beta}\in h} \mu'\left(F'^j_{-i}\left(\hat{\beta}\right) \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h''} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\
&= 1.
\end{aligned}$$

Case 2. $F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)$. We need to treat members of h differently depending on if they map to the same set of signals in $\Gamma_{F'}$ or not. Let $h_1 = \left\{\hat{\beta} \in h \mid \hat{\beta} = F'^j_{-i}\left(\hat{\beta}\right)\right\}$ and let $h_2 = \left\{\hat{\beta} \in h \mid \hat{\beta} \subset F'^j_{-i}\left(\hat{\beta}\right)\right\}$. Clearly (h_1, h_2) is a partition of h. Let h' be player i's information set corresponding to the history $\left(F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right)$ in $\Gamma_{F'}$. We can create a partition of h' by letting $h_3 = \left\{F'^j_{-i}\left(\hat{\beta}\right) \mid \hat{\beta} \in h_1\right\}$ and $h_4 = \left\{F'^j_{-i}\left(\hat{\beta}\right) \mid \hat{\beta} \in h_2\right\}$. Cleary (h_3, h_4) partitions h'. (See Figure 5.4.) The rest of the proof for this case proceeds in three steps.

Step 1. In this step we show the following relationship between h_1 and h_3:

$$\begin{aligned}
\sum_{\hat{\beta}\in h_1} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) &= \sum_{\hat{\beta}\in h_1} \mu'\left(F'^j_{-i}\left(\hat{\beta}\right) \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h_3} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \qquad (5.1)
\end{aligned}$$

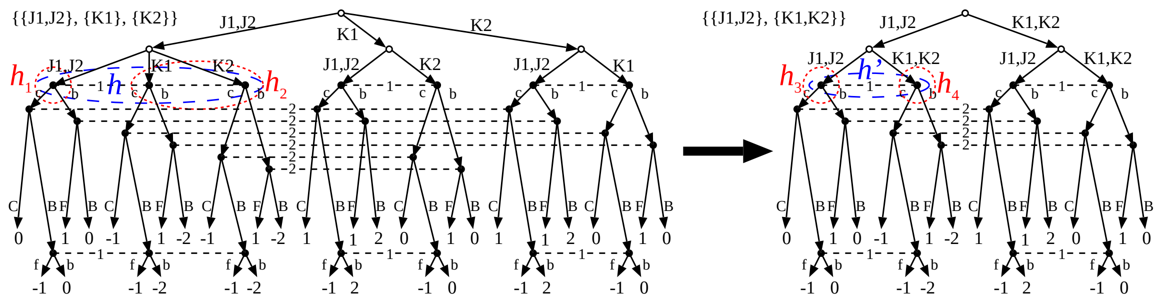

Figure 5.4: Illustration of Case 2 of Claim 1.

STEP 2. In this step we want to show a similar relationship between h_2 and h_4. In doing so, we use the following fact: $\hat{\beta} \subset \hat{\beta}' \rightarrow F'^{j}_{-i}\left(\hat{\beta}\right) = \hat{\beta}'$. With this in mind, we can write:

$$\begin{aligned}
\sum_{\hat{\beta}\in h_2} \mu\left(\hat{\beta}|F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) &= \sum_{\hat{\beta}\in h_2} \frac{\Pr\left(\hat{\beta}|F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)\right)}{\Pr\left(F'^{j}_{-i}(\hat{\beta})|F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)\right)} \mu'\left(F'^{j}_{-i}(\hat{\beta})|F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \frac{\Pr\left(\hat{\beta}|F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)\right)}{\Pr\left(F'^{j}_{-i}(\hat{\beta})|F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)\right)} \cdot \mu'\left(F'^{j}_{-i}(\hat{\beta})|F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \frac{\Pr\left(\hat{\beta}|F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\hat{\beta}'|F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \mu'\left(\hat{\beta}'|F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h_4} \mu'\left(\hat{\beta}'|F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \frac{\Pr\left(\hat{\beta}|F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\hat{\beta}'|F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \\
&= \sum_{\hat{\beta}'\in h_4} \mu'\left(\hat{\beta}'|F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \qquad (5.2)
\end{aligned}$$

STEP 3. Using (5.1) and (5.2):

$$\begin{aligned}
\sum_{\hat{\beta}\in h} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) &= \sum_{\hat{\beta}\in h_1} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) + \sum_{\hat{\beta}\in h_2} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h_3} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) + \sum_{\hat{\beta}'\in h_4} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= \sum_{\hat{\beta}'\in h'} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= 1
\end{aligned}$$

In both cases we have shown $\sum\limits_{\hat{\beta}\in h} \mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) = 1.$ □

Claim 2 *For all information sets h with $\Pr(h \mid \sigma) > 0$, $\mu(x) = \frac{\Pr(x \mid \sigma)}{\Pr(h \mid \sigma)}$ for all $x \in h$.*

PROOF.[of Claim 2] Let h be player i's information set after some history $\left(F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right)$, and fix some $\hat{\beta} \in h$. Let $\hat{\beta}' = F'^j_{-i}\left(\hat{\beta}\right)$. We need to show that $\mu(\hat{\beta}|F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i), \tilde{z}) = \frac{\Pr\left(\hat{\beta} \mid \sigma\right)}{\Pr(h \mid \sigma)}$. Let h' be player i's information set after history $(F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i), \tilde{z})$.

CASE 1. $F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i) \neq F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)$.

$$\begin{aligned}
\mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) &= \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= \frac{\Pr\left(\hat{\beta}' \mid \sigma'\right)}{\Pr\left(h' \mid \sigma'\right)} \\
&= \frac{\frac{\Pr\left(\hat{\beta}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\hat{\beta}', F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \Pr\left(\hat{\beta}' \mid \sigma'\right)}{\frac{\Pr\left(\hat{\beta}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\hat{\beta}', F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \Pr\left(h' \mid \sigma'\right)} \\
&= \frac{\Pr\left(\hat{\beta} \mid \sigma\right)}{\Pr\left(h \mid \sigma\right)}
\end{aligned}$$

CASE 2. $F^j(\tilde{\alpha}^j, \tilde{\beta}^j_i) = F'^j(\tilde{\alpha}^j, \tilde{\beta}^j_i)$ and $\hat{\beta} \neq \hat{\beta}'$.

$$\begin{aligned}
\mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) &= \frac{\Pr\left(\tilde{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\tilde{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right), \tilde{z}\right) \\
&= \frac{\Pr\left(\tilde{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\tilde{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \frac{\Pr\left(\hat{\beta}' \mid \sigma'\right)}{\Pr\left(h' \mid \sigma'\right)} \\
&= \frac{\Pr\left(\tilde{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\tilde{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \frac{\frac{\Pr\left(\tilde{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)}{\Pr\left(\tilde{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)} \Pr\left(\hat{\beta} \mid \sigma\right)}{\Pr\left(h \mid \sigma\right)} \\
&= \frac{\Pr\left(\hat{\beta} \mid \sigma\right)}{\Pr\left(h \mid \sigma\right)}
\end{aligned}$$

CASE 3. $F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j) = F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j)$ and $\hat{\beta} = \hat{\beta}'$.

$$\begin{aligned}
\mu\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) &= \mu'\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}\right) \\
&= \frac{\Pr\left(\hat{\beta}' \mid \sigma'\right)}{\Pr\left(h' \mid \sigma'\right)} \\
&= \frac{\Pr\left(\hat{\beta} \mid \sigma\right)}{\Pr\left(h \mid \sigma\right)}
\end{aligned}$$

Thus we have $\mu(x) = \frac{\Pr(x \mid \sigma)}{\Pr(h \mid \sigma)}$ for all information sets h with $\Pr(h \mid \sigma) > 0$. □

Claim 3 *For all information sets h with $\Pr(h \mid \sigma) > 0$, σ is sequentially rational at h given μ.*

PROOF.[of Claim 3] Suppose, by way of contradiction, that σ is not sequentially rational given μ. Then, there exists a strategy τ_i such that, for some $(F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z})$,

$$u_i^j(\tau_i, \sigma_{-i}|F^j\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu) > u_i^j(\sigma_i, \sigma_{-i}|F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu). \tag{5.3}$$

We will construct a strategy τ_i' for player i in $\Gamma_{F'}$ such that

$$u_i^j(\tau_i', \sigma_{-i}'|F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu') > u_i^j(\sigma_i', \sigma_{-i}'|F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu'),$$

thus contradicting the fact that σ' is a Nash equilibrium. The proof proceeds in four steps.

STEP 1. We first construct τ_i' from τ_i. For a given $F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)$, let

$$\Upsilon = \left\{F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \subseteq F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right\} \tag{5.4}$$

and let

$$\tau_{i,v}'^j(F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}) = \sum_{\vartheta \in \Upsilon} \Pr\left(\vartheta \mid F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j)\right) \tau_{i,v}^j\left(\vartheta, \tilde{z}\right).$$

In other words, the strategy τ_i' is the same as τ_i except in situations where only the filtered signal history is different, in which case τ_i' is a weighted average over the strategies at the corresponding information sets in Γ_F.

STEP 2. We need to show that $u_i^j(\tau_i', \sigma_{-i}' \mid F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu') = u_i^j(\tau_i, \sigma_{-i} \mid F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z}, \mu)$ for all histories $(F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z})$. Fix $(F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j), \tilde{z})$, and assume, without loss of generality, the equality holds for all information sets coming after this one in Γ.

CASE 1. $F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j) \neq F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j)$. Let z^j denote the current node of G^j and let Υ as in (5.4).

$$
\begin{aligned}
&u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right) = \sum_{\hat{\beta}' \in h'} \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) \\
&= \sum_{\hat{\beta} \in h} \mu'\left(F_{-i}'^j\left(\hat{\beta}\right)\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, F_{-i}'^j\left(\hat{\beta}\right)\right) \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, F_{-i}'^j\left(\hat{\beta}\right)\right) \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) \sum_{v \in N^j(z^j)} \tau_{i,v}'^j\left(\tilde{z}, F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \cdot u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), F_{-i}'^j\left(\hat{\beta}\right)\right) \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) \sum_{v \in N^j(z^j)} \sum_{\vartheta \in \Upsilon} \Pr\left(\vartheta \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \tau_{i,v}^j\left(\tilde{z}, \vartheta\right) \cdot \\
&\qquad \left[u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), F_{-i}'^j\left(\hat{\beta}\right)\right)\right] \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) \sum_{v \in N^j(z^j)} \sum_{\vartheta \in \Upsilon} \Pr\left(\vartheta \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \tau_{i,v}^j\left(\tilde{z}, \vartheta\right) \cdot \\
&\qquad \left[u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), \hat{\beta}\right)\right] \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) \sum_{v \in N^j(z^j)} u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), \hat{\beta}\right) \cdot \\
&\qquad \left[\sum_{\vartheta \in \Upsilon} \Pr\left(\vartheta \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \tau_{i,v}^j\left(\tilde{z}, \vartheta\right)\right] \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) \sum_{v \in N^j(z^j)} \tau_{i,v}^j\left(\tilde{z}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \cdot u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), \hat{\beta}\right) \\
&= \sum_{\hat{\beta} \in h} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \\
&= u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right)
\end{aligned}
$$

CASE 2. $F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j) = F'^j(\tilde{\alpha}^j, \tilde{\beta}_i^j)$. Let h_1, h_2, h_3, and h_4 as in the proof of Case 2 of Claim 1. We can show

$$\sum_{\hat{\beta}' \in h_3} \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) = \sum_{\hat{\beta} \in h_1} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \tag{5.5}$$

using a procedure similar to that in Case 1. We can show the following relationship between h_2 and h_4:

$$\begin{aligned}
&\sum_{\hat{\beta}'\in h_4} \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) \\
&\quad = \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \frac{\Pr\left(\hat{\beta} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right)}{\Pr\left(\hat{\beta}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right)} \cdot \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) \\
&\quad = \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) \\
&\quad = \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \mu\left(\hat{\beta}\right) \sum_{v\in N^j(z^j)} \tau_{i,v}'^j\left(\tilde{z}, F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \cdot u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), \hat{\beta}'\right) \\
&\quad = \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \mu\left(\hat{\beta}\right) \sum_{v\in N^j(z^j)} \tau_{i,v}^j\left(\tilde{z}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right) \cdot u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), (\tilde{z}, v), \hat{\beta}\right) \\
&\quad = \sum_{\hat{\beta}'\in h_4} \sum_{\substack{\hat{\beta}\in h_2 \\ \hat{\beta}\subset\hat{\beta}'}} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \\
&\quad = \sum_{\hat{\beta}\in h_2} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \qquad (5.6)
\end{aligned}$$

Using (5.5) and (5.6):

$$\begin{aligned}
&u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right) = \sum_{\tilde{\beta}'\in h'} \mu'\left(\tilde{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \tilde{\beta}'\right) \\
&= \sum_{\hat{\beta}'\in h_3} \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) + \sum_{\hat{\beta}'\in h_4} \mu'\left(\hat{\beta}'\right) u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}'\right) \\
&= \sum_{\hat{\beta}\in h_1} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) + \sum_{\hat{\beta}\in h_2} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \\
&= \sum_{\hat{\beta}\in h} \mu\left(\hat{\beta}\right) u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \hat{\beta}\right) \\
&= u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right)
\end{aligned}$$

In both cases we have shown:

$$u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right) = u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right). \qquad (5.7)$$

STEP 3. We can show that

$$u_i^j\left(\sigma_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right) = u_i^j\left(\sigma_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right). \qquad (5.8)$$

using a procedure similar to the previous step.

STEP 4. Combining (5.3), (5.7), and (5.8), we have:

$$\begin{aligned} u_i^j\left(\tau_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right) &= u_i^j\left(\tau_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right) \\ > \ u_i^j\left(\sigma_i, \sigma_{-i} \mid F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu\right) &= u_i^j\left(\sigma_i', \sigma_{-i}' \mid F'^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right), \tilde{z}, \mu'\right). \end{aligned}$$

Thus, σ' is not a Nash equilibrium. Therefore, by contradiction, σ is sequentially rational at all information sets h with $\Pr(h \mid \sigma) > 0$. □

We can now complete the proof of Theorem 2. By Claims 1 and 2, we know that condition C2 holds. By Claim 3, we know that condition C1 holds. Thus, σ is a Nash equilibrium. □

5.4.1 Nontriviality of generalizing beyond this model

Our model does not capture general sequential games of imperfect information because it is restricted in two ways (as discussed above): 1) there is a special structure connecting the player actions and the chance actions (for one, the players are assumed to observe each others' actions, but nature's actions might not be publicly observable), and 2) there is a common ordering of signals. In this subsection we show that removing either of these conditions can make our technique invalid.

First, we demonstrate a failure when removing the first assumption. Consider the game in Figure 5.5.[1] Nodes a and b are in the same information set, have the same parent (chance) node, have isomorphic subtrees with the same payoffs, and nodes c and d also have similar structural properties. By merging the subtrees beginning at a and b, we get the game on the right in Figure 5.5. In this game, player 1's only Nash equilibrium strategy is to play left. But in the original game, player 1 knows that node c will never be reached, and so should play right in that information set.

Removing the second assumption (that the utility functions are based on a common ordering of signals) can also cause failure. Consider a simple three-card game with a deck containing two Jacks (J1 and J2) and a King (K), where player 1's utility function is based on the ordering $\mathrm{K} \succeq \mathrm{J1} \sim \mathrm{J2}$ but player 2's utility function is based on the ordering $\mathrm{J2} \succeq \mathrm{K} \succeq \mathrm{J1}$. It is easy to check that in the abstracted game (where Player 1 treats J1 and J2 as being "equivalent") the Nash equilibrium does not correspond to a Nash equilibrium in the original game.[2]

[1]We thank Albert Xin Jiang for providing this example.

[2]We thank an anonymous person for providing this example.

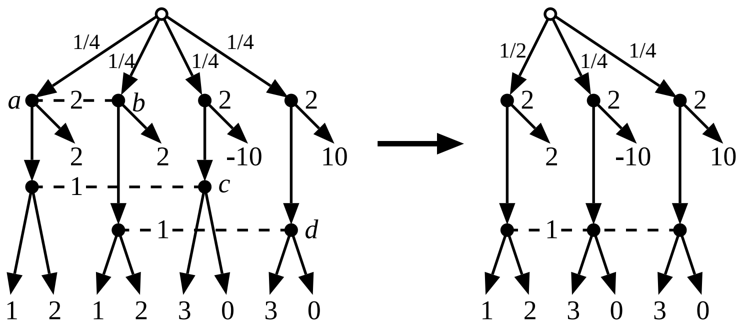

Figure 5.5: Example illustrating difficulty in developing a theory of equilibrium-preserving abstractions for general extensive form games.

5.5 GameShrink: An efficient algorithm for computing ordered game isomorphic abstraction transformations

In this section we present an algorithm, *GameShrink*, for conducting the abstractions. The algorithm only needs to analyze the signal tree discussed above, rather than the entire game tree.

We first present a subroutine that *GameShrink* uses. It is a dynamic program for computing the ordered game isomorphic relation.[3] Again, it operates on the signal tree.

Algorithm 1 *OrderedGameIsomorphic?* $(\Gamma, \vartheta, \vartheta')$

```
1. If $\vartheta$ and $\vartheta'$ are both leaves of the signal tree:
    (a) If $u^r(\vartheta \mid \tilde{z}) = u^r(\vartheta' \mid \tilde{z})$ for all $\tilde{z} \in \bigtimes_{j=1}^{r-1} \omega_{cont}^j \times \omega_{over}^r$, then return true.
    (b) Otherwise, return false.
2. Create a bipartite graph $G_{\vartheta,\vartheta'} = (V_1, V_2, E)$ with $V_1 = N(\vartheta)$ and $V_2 = N(\vartheta')$.
3. For each $v_1 \in V_1$ and $v_2 \in V_2$:
        If OrderedGameIsomorphic? $(\Gamma, v_1, v_2)$
            Create edge $(v_1, v_2)$
```

[3]Actually, this is computing a slightly relaxed notion since it allows nodes with different parents to be considered ordered game isomorphic. However, the *GameShrink* algorithm only calls it with sibling nodes as the arguments.

4. Return true if $G_{\vartheta,\vartheta'}$ has a perfect matching; otherwise, return false.

By evaluating this dynamic program from bottom to top, Algorithm 1 determines, in time polynomial in the size of the signal tree, whether or not any pair of equal depth nodes x and y are ordered game isomorphic. The test in step 1(a) can be computed in $O(1)$ time by consulting the $\succeq$ relation from the specification of the game. Each call to *OrderedGameIsomorphic?* performs at most one perfect matching computation on a bipartite graph with $O(|\Theta|)$ nodes and $O(|\Theta|^2)$ edges (recall that Θ is the set of signals). Using the Ford-Fulkerson algorithm [49] for finding a maximal matching, this takes $O(|\Theta|^3)$ time. Let S be the maximum number of signals possibly revealed in the game (e.g., in Rhode Island Hold'em, $S = 4$ because each of the two players has one card in the hand plus there are two cards on the table). The number of nodes, n, in the signal tree is $O(|\Theta|^S)$. The dynamic program visits each node in the signal tree, with each visit requiring $O(|\Theta|^2)$ calls to the *OrderedGameIsomorphic?* routine. So, it takes $O(|\Theta|^S|\Theta|^3|\Theta|^2) = O(|\Theta|^{S+5})$ time to compute the entire ordered game isomorphic relation.

While this is exponential in the number of revealed signals, we now show that it is polynomial in the size of the signal tree—and thus polynomial in the size of the game tree because the signal tree is smaller than the game tree. The number of nodes in the signal tree is

$$n = 1 + \sum_{i=1}^{S} \prod_{j=1}^{i} (|\Theta| - j + 1)$$

(Each term in the summation corresponds to the number of nodes at a specific depth of the tree.) The number of leaves is

$$\prod_{j=1}^{S} (|\Theta| - j + 1) = \binom{|\Theta|}{S} S!$$

which is a lower bound on the number of nodes.[4] For large $|\Theta|$ we can use the relation $\binom{n}{k} \sim \frac{n^k}{k!}$ to get

$$\binom{|\Theta|}{S} S! \sim \left(\frac{|\Theta|^S}{S!}\right) S! = |\Theta|^S$$

and thus the number of leaves in the signal tree is $\Omega(|\Theta|^S)$. Therefore, $O(|\Theta|^{S+5}) = O(n|\Theta|^5)$, which proves that we can indeed compute the ordered game isomorphic relation in time polynomial in the number of nodes, n, of the signal tree.

[4]Using the inequality $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$, we get the lower bound $\binom{|\Theta|}{S} S! \geq \left(\frac{|\Theta|}{S}\right)^S S! = |\Theta|^S \frac{S!}{S^S}$.

The algorithm often runs in *sublinear* time (and space) in the size of the game tree because the signal tree is significantly smaller than the game tree in most nontrivial games. (Note that the input to the algorithm is not an explicit game tree, but a specification of the rules, so the algorithm does not need to read in the game tree.) In general, if an ordered game has r rounds, and each round's stage game has at least b nonterminal leaves, then the size of the signal tree is at most $\frac{1}{b^r}$ of the size of the game tree. For example, in Rhode Island Hold'em, the game tree has 3.1 billion nodes while the signal tree only has 6,632,705.

Given the *OrderedGameIsomorphic?* routine for determining ordered game isomorphisms in an ordered game, we are ready to present the main algorithm, *GameShrink*.

Algorithm 2 *GameShrink* (Γ)

```
1. Initialize $F$ to be the identity filter for $\Gamma$.
2. For $j$ from 1 to $r$:
        For each pair of sibling nodes $\vartheta, \vartheta'$ at either level $\sum_{k=1}^{j-1}\left(\kappa^k + n\gamma^k\right)$ or $\sum_{k=1}^{j}\kappa^k + \sum_{k=1}^{j-1} n\gamma^k$ in the filtered (according to $F$) signal tree:
            If $OrderedGameIsomorphic?(\Gamma, \vartheta, \vartheta')$, then $F^j(\vartheta) \leftarrow F^j(\vartheta') \leftarrow F^j(\vartheta) \cup F^j(\vartheta')$.
3. Output $F$.
```

Given as input an ordered game $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$, *GameShrink* applies the shrinking ideas presented above as aggressively as possible. Once it finishes, there are no contractible nodes (since it compares every pair of nodes at each level of the signal tree), and it outputs the corresponding information filter F. The correctness of *GameShrink* follows by a repeated application of Theorem 2. Thus, we have the following result:

Theorem 3 GameShrink *finds all ordered game isomorphisms and applies the associated ordered game isomorphic abstraction transformations. Furthermore, for any Nash equilibrium, σ', of the abstracted game, the strategy profile constructed for the original game from σ' is a Nash equilibrium.*

The dominating factor in the run time of *GameShrink* is in the r^{th} iteration of the main for-loop. There are at most $\binom{|\Theta|}{S} S!$ nodes at this level, where we again take S to be the maximum number of signals possibly revealed in the game. Thus, the inner for-loop executes $O\left(\left(\binom{|\Theta|}{S}S!\right)^2\right)$ times. As discussed in the next subsection, we use a union-find data structure to represent the information filter F. Each iteration of the inner for-loop possibly performs a union operation on the data structure; performing M operations on a union-find data structure containing N elements takes $O(\alpha(M,N))$ amortized time per operation, where $\alpha(M,N)$ is the inverse Ackermann's function [2, 155] (which grows extremely slowly). Thus, the total time for *GameShrink* is $O\left(\left(\binom{|\Theta|}{S}S!\right)^2 \alpha\left(\left(\binom{|\Theta|}{S}S!\right)^2, |\Theta|^S\right)\right)$. By the inequality $\binom{n}{k} \leq \frac{n^k}{k!}$, this is $O\left((|\Theta|^S)^2 \,\alpha\left((|\Theta|^S)^2, |\Theta|^S\right)\right)$. Again, although this is exponential in S, it is $\tilde{O}(n^2)$, where n is the number of nodes in the signal tree. Furthermore, *GameShrink* tends to actually run in *sublinear* time and space in the size of the game tree because the signal tree is significantly smaller than the game tree in most nontrivial games, as discussed above.

5.5.1 Efficiency enhancements

We designed several speed enhancement techniques for *GameShrink*, and all of them are incorporated into our implementation. One technique is the use of the union-find data structure [34, Chapter 21] for storing the information filter F. This data structure uses time almost linear in the number of operations [155]. Initially each node in the signalling tree is its own set (this corresponds to the identity information filter); when two nodes are contracted they are joined into a new set. Upon termination, the filtered signals for the abstracted game correspond exactly to the disjoint sets in the data structure. This is an efficient method of recording contractions within the game tree, and the memory requirements are only linear in the size of the signal tree.

Determining whether two nodes are ordered game isomorphic requires us to determine if a bipartite graph has a perfect matching. We can eliminate some of these computations by using easy-to-check necessary conditions for the ordered game isomorphic relation to hold. One such condition is to check that the nodes have the same number of chances as being ranked (according to $\succeq$) higher than, lower than, and the same as the opponents. We can precompute these frequencies for every game tree node. This substantially speeds up *GameShrink*, and we can leverage this database across multiple runs of the algorithm (for example, when trying different abstraction levels; see next section). The indices for this database depend on the private and public signals, but not the *order* in which they were revealed, and thus two nodes may have the same corresponding database entry. This makes the database significantly more compact. (For example in Texas Hold'em, the database is reduced by a factor $\binom{50}{3}\binom{47}{1}\binom{46}{1}/\binom{50}{5} = 20$.) We store the histograms in a 2-dimensional database. The first dimension is indexed by the private signals, the second by the public signals. The problem of computing the index in (either) one of the dimensions is exactly the problem of computing a bijection between all subsets of size r from a set of size n and integers in $\left[0, \ldots, \binom{n}{r} - 1\right]$. We efficiently compute this using the subsets' *colexicographical ordering* [22]. Let $\{c_1, \ldots, c_r\}$, $c_i \in \{0, \ldots, n-1\}$, denote the r signals and assume that $c_i < c_{i+1}$. We compute a unique index for this set of signals as follows:

$$index(c_1, \ldots, c_r) = \sum_{i=1}^{r} \binom{c_i}{i}.$$

5.6 Approximation methods

Some games are too large to compute an exact equilibrium, even after using the presented abstraction technique. In this section we discuss general techniques for computing approximately optimal strategy profiles. For a two-player game, we can always evaluate the worst-case performance of a strategy, thus providing some objective evaluation of the strength of the strategy. To illustrate this, suppose we know player 2's planned strategy for some game. We can then fix the probabilities of player 2's actions in the game tree as if they were chance moves. Then player 1 is faced with a single-agent decision problem, which can be solved bottom-up, maximizing expected payoff at every node. Thus, we can objectively determine the expected worst-case performance of player 2's strategy. This will be most useful when we want to evaluate how well a given strategy performs when we know that it is not an equilibrium strategy. (A variation of this technique may also be applied in n-person games where only one player's strategies are held fixed.) This technique provides *ex post* guarantees about the worst-case performance of a strategy, and can be used independently of the method that is used to compute the strategies in the first place.

5.6.1 State-space approximations

By slightly modifying the *GameShrink* algorithm we can obtain an algorithm that yields even smaller game trees, at the expense of losing the equilibrium guarantees of Theorem 2. Instead of requiring the payoffs at terminal nodes to match exactly, we can compute a penalty that increases as the difference in utility between two nodes increases.

There are many ways in which the penalty function could be defined and implemented. One possibility is to create edge weights in the bipartite graphs used in Algorithm 1, and then instead of requiring perfect matchings in the unweighted graph we would require perfect matchings with low cost (i.e., only consider two nodes to be ordered game isomorphic if the corresponding bipartite graph has a perfect matching with cost below some threshold). Thus, with this threshold as a parameter, we have a knob to turn that in one extreme (threshold = 0) yields an optimal abstraction and in the other extreme (threshold = ∞) yields a highly abstracted game (this would in effect restrict players to ignoring all signals, but still observing actions). This knob also begets an *anytime* algorithm. One can solve increasingly less abstracted versions of the game, and evaluate the quality of the solution at every iteration using the *ex post* method discussed above.

We further develop state-space approximation algorithms in Chapter 6 of this thesis.

5.6.2 Algorithmic approximations

In the case of two-player zero-sum games, the equilibrium computation can be modeled as a linear program (LP), which can in turn be solved using the simplex method. This approach has inherent features which we can leverage into desirable properties in the context of solving games.

In the LP, primal solutions correspond to strategies of player 2, and dual solutions correspond to strategies of player 1. There are two versions of the simplex method: the primal simplex and the dual simplex. The primal simplex maintains primal feasibility and proceeds by finding better and better primal solutions until the dual solution vector is feasible, at which point optimality has been reached. Analogously, the dual simplex maintains dual feasibility and proceeds by finding increasingly better dual solutions until the primal solution vector is feasible. (The dual simplex method can be thought of as running the primal simplex method on the dual problem.) Thus, the primal and dual simplex methods serve as *anytime* algorithms (for a given abstraction) for players 2 and 1, respectively. At any point in time, they can output the best strategies found so far.

Also, for any feasible solution to the LP, we can get bounds on the quality of the strategies by examining the primal and dual solutions. (When using the primal simplex method, dual solutions may be read off of the LP tableau.) Every feasible solution of the dual yields an upper bound on the optimal value of the primal, and vice versa [32, p. 57]. Thus, without requiring further computation, we get lower bounds on the expected utility of each agent's strategy against that agent's worst-case opponent.

One problem with the simplex method is that it is not a primal-dual algorithm, that is, it does not maintain both primal and dual feasibility throughout its execution. (In fact, it only obtains primal and dual feasibility at the very end of execution.) In contrast, there are interior-point methods for linear programming that maintain primal and dual feasibility throughout the execution. For example, many interior-point path-following algorithms have this property [163, Ch. 5]. We observe that running such a linear programming method yields a method for finding ϵ-equilibria (i.e., strategy profiles in which no agent can increase her expected utility by more than ϵ by deviating). A threshold on ϵ can also be used as a termination criterion for using the method as an anytime algorithm. Furthermore, interior-point methods in this class have polynomial-time worst-case run time, as opposed to the simplex algorithm, which takes exponentially many steps in the worst case.

We develop approximation algorithms for finding ϵ-equilibria in Part III of this thesis.

5.7 Related research

The main technique applied in this chapter is that of transforming large extensive form games into smaller extensive form games for which an equilibrium can be computed. Then, the equilibrium strategies of the smaller game are mapped back into the original larger game. One of the first pieces of research addressing functions which transform extensive form games into other extensive form games, although not for the purpose of making the game smaller, was in an early paper [157], which was later extended [45]. In these papers, several distinct *transformations*, now known as Thompson-Elmes-Reny transformations, are defined. The main result is that one game can be derived from another game by a sequence of those transformations if and only if the games have the same *pure reduced normal form*. The pure reduced normal form is the extensive form game represented as a game in normal form where duplicates of pure strategies (i.e., ones with identical payoffs) are removed and players essentially select equivalence classes of strategies [91]. An extension to this work shows a similar result, but for slightly different transformations and *mixed reduced normal form* games [83]. Modern treatments of this previous work on game transformations have also been written [123, Ch. 6], [40].

The notion of *weak isomorphism* in extensive form games [29] is related to our notion of restricted game isomorphism. The motivation of that work was to justify solution concepts by arguing that they are invariant with respect to isomorphic transformations. Indeed, the author shows, among other things, that many solution concepts, including Nash, perfect, subgame perfect, and sequential equilibrium, are invariant with respect to weak isomorphisms. However, that definition requires that the games to be tested for weak isomorphism are of the same size. Our focus is totally different: we find strategically equivalent *smaller* games. Another difference is that their paper does not provide any algorithms.

Abstraction techniques have been used in artificial intelligence research before. In contrast to our work, most (but not all) research involving abstraction has been for single-agent problems (e.g. [82, 101]). Furthermore, the use of abstraction typically leads to sub-optimal solutions, unlike the techniques presented in this chapter, which yield optimal solutions. A notable exception is the use of abstraction to compute optimal strategies for the game of Sprouts [5]. However, a significant difference to our work is that Sprouts is a game of perfect information.

One of the first pieces of research to use abstraction in multi-agent settings was the development of *partition search*, which is the algorithm behind *GIB*, the world's first expert-level computer bridge player [66, 67]. In contrast to other game tree search algorithms which store a particular game position at each node of the search tree, partition search stores *groups* of positions that are similar. (Typically, the similarity of two game positions is computed by ignoring the less important components of each game position and then checking whether the abstracted positions are similar—in some domain-specific expert-defined sense—to each other.) Partition search can lead to substantial speed improvements over α-β-search. However, it is not game theory-based (it does not consider information sets in the game tree), and thus does not solve for the equilibrium of a game of imperfect information, such as poker.[5] Another difference is that the abstraction is defined by an expert human while our abstractions are determined automatically.

There has been some research on the use of abstraction for imperfect information games. Most notably, Billings *et al* [14] describe a manually constructed abstraction for the game of Texas Hold'em poker, and include promising results against expert players. However, this approach has significant drawbacks. First, it is highly specialized for Texas Hold'em. Second, a large amount of expert knowledge and effort was used in constructing the abstraction. Third, the abstraction does not preserve equilibrium: even if applied to a smaller game, it might not yield a game-theoretic equilibrium. Promising ideas for abstraction in the context of general extensive form games have been described in an extended abstract [124], but to our knowledge, have not been fully developed.

[5]Bridge is also a game of imperfect information, and partition search does not find the equilibrium for that game either. Instead, partition search is used in conjunction with statistical sampling to simulate the uncertainty in bridge. There are also other bridge programs that use search techniques for perfect information games in conjunction with statistical sampling and expert-defined abstraction [149]. Such (non-game-theoretic) techniques are unlikely to be competitive in poker because of the greater importance of information hiding and bluffing.

5.8 Summary

We introduced the ordered game isomorphic abstraction transformation and gave an algorithm, *GameShrink*, for abstracting the game using the isomorphism exhaustively. We proved that in games with ordered signals, any Nash equilibrium in the smaller abstracted game maps directly to a Nash equilibrium in the original game.

The complexity of *GameShrink* is $\tilde{O}(n^2)$, where n is the number of nodes in the signal tree. It is no larger than the game tree, and on nontrivial games it is drastically smaller, so *GameShrink* has time and space complexity *sublinear* in the size of the game tree. Using *GameShrink*, we found a minimax equilibrium to Rhode Island Hold'em, a poker game with 3.1 billion nodes in the game tree—over four orders of magnitude more than in the largest poker game solved previously.

To further improve scalability, we introduced an approximation variant of *GameShrink*, which can be used as an anytime algorithm by varying a parameter that controls the coarseness of abstraction. In Chapter 6 we develop several other improved approximation abstraction algorithms.

We also discussed how (in a two-player zero-sum game), linear programming can be used in an anytime manner to generate approximately optimal strategies of increasing quality. The method also yields bounds on the suboptimality of the resulting strategies. In Part III of this thesis we develop better approximation algorithms for finding ϵ-equilibria.

While our main motivation was games of private information, our abstraction method can also be used in games where there is no private information. The method can be helpful even if all signals that are revealed during the game are public (such as public cards drawn from a deck, or throws of dice). However, in such games, *expectiminimax search* [108] (possibly supplemented with α-β-pruning) can be used to solve the game in linear time in n. In contrast, solving games with private information takes significantly longer: the time to solve an $O(n) \times O(n)$ linear program in the two-person zero-sum setting, and longer in more general games. Therefore, our abstraction method will pay off as a preprocessor in games with no private information only if the signal tree of the game is significantly smaller than the game tree.

Chapter 6

Lossy Information Abstraction

6.1 Introduction

In this chapter we continue our investigation of the first of the three classes of abstraction under consideration in this chapter: *information abstraction*. In contrast to the previous chapter, we now look at algorithms that yield approximate equilibria when solving the abstracted game rather than exact equilibria. This is useful for games where the losslessly abstracted game is still too large to solve.

6.2 Stage-based sequential games

The class of extensive form games is quite flexible. However, this flexibility inhibits the development of abstractions algorithms since the structure in the games may be hard to detect. (Examples illustrating the difficulty of developing abstraction algorithms for arbitrary extensive form games were discussed in Section 5.4.1.) For this reason, we instead introduce and work with a more structured representation of games. The main benefit of our representation is that it specifically encodes stages and information relevation. The base entity in our model is an interaction, and this will be used as a component in the construction of stage-based sequential games.

Definition 11 (Interaction) *An n-*person interaction *is a tuple*

$$\Upsilon = \langle \mathcal{A}, \mathcal{O}, \Phi \rangle$$

where:

- $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ *is a collection of* action sets *for each of the n agents.*
- $\mathcal{O} = \langle O_1, \ldots, O_{|\mathcal{O}|} \rangle$ *is the finite set of* outcomes *for the interaction.*
- $\Phi : \mathcal{A} \to \Delta(\mathcal{O})$ *specifies a probability distribution over outcomes given strategies $a \in \mathcal{A}$.*

Interactions are capable of modeling arbitrarily complex strategic encounters. In particular, the strategy sets in Definition 11 may be finite or infinite. Thus, interactions can model scenarios where players employ randomized strategies. In this sense, interactions are at least as representative as both normal form and extensive form games, although the outcomes in an interaction are not associated with any payoffs to the agents.[1] We design a game model composed of interactions which we study in this chapter and the next.

Definition 12 (Stage-Based Sequential Game) *An n-*person stage-based sequential game *is a tuple*

$$\Gamma = \left\langle \Theta, p, \mathcal{T}, \left\{\left\langle \kappa^t, \gamma^t, \Upsilon^t \right\rangle\right\}_{t \in \mathcal{T}}, \Lambda, u \right\rangle$$

where:

1. Θ *is a finite set of* signals.
2. p *is a joint distribution over the signals* Θ.
3. $\mathcal{T}$ *is a finite set of stages.*
4. $\langle \kappa^t, \gamma^t, \Upsilon^t \rangle$ *is the* stage interaction *for stage $t \in \mathcal{T}$: κ^t signals are revealed publicly, γ_i^t signals are revealed privately to player i, and the players participate in the interaction Υ^t. The signals are revealed probabilistically according to p.*
5. Λ *maps outcomes from the interactions to stages $\mathcal{T} \cup \{\emptyset\}$ where $\emptyset$ denotes a terminating stage. Cycles are not permitted.*
6. $u = \langle u_1, \ldots, u_n \rangle$ *where each u_i maps the history of outcomes and signals to a real-valued* utility *for player i.*

[1]This is not to say that interactions are a superior representation for certain games. In particular, representation of an extensive form game with an interaction would lead to the same exponential increase as occurs when representing an extensive form game as a normal form game.

Behavioral strategies for stage-based sequential games are defined similarly as for extensive form games. A strategy σ_i for player i specifies, for each stage t, for each outcome in the stages up to that point, for each revelation of public information, and for each revelation of player i's private information, an action from player i's action set in the current stage's interaction. Definitions of expected utility, Nash equilibrium, and ϵ-equilibrium may be developed analagously as was done for extensive form games.

It is clear that every stage-based sequential game may be modeled as an extensive form game. In addition, every extensive form game may be modeled as a stage-based sequential game. At the very least, one may do this by defining a single-stage game where the interaction at that stage is the extensive form game. (However, such a game instance would not be very useful in conjunction with our abstraction algorithms since the algorithms rely on the structure in stage-based sequential game to operate effectively.)

The set of interactions $\{\Upsilon^t\}_{t \in \mathcal{T}}$ together with the stage transfer mapping Λ induces a rooted directed acyclic graph. The set of paths in the graph beginning at the root are the possible sequences of stages that can occur during a play of the game. We refer to this graph as the *stage graph* and will use this later when we define our abstraction algorithms. One important fact is that since the stage graph is a directed acyclic graph, it induces a partial order over the stages.

6.2.1 Abstraction representation in stage-based sequential games

We represent information abstractions using *information filters* [61]. In Chapter 5, we defined information filters in the context of *ordered games*. Here, we define information filters in the context of stage-based sequential games (Definition 12).

Definition 13 (Information Filter) *Let*

$$\Gamma = \left\langle \Theta, p, \mathcal{T}, \left\{\left\langle \kappa^t, \gamma^t, \Upsilon^t \right\rangle\right\}_{t \in \mathcal{T}}, \Lambda, u \right\rangle$$

be an n-person stage-based sequential game. Let $\bar{t} = \langle t_1, \ldots, t_k \rangle$, where $t_j \in \mathcal{T}$ for each $t_j \in \bar{t}$, be the sequence of k stages played up to one point in time. Let

$$S^{\bar{t}} \subseteq \bigtimes_{t \in \bar{t}} \Theta^{\kappa^t} \times \bigtimes_{t \in \bar{t}} \Theta^{\gamma_i^t}$$

be the set of legal signals (i.e., no repeated signals) revealed to player i through the sequence of stages $\bar{t}$. An information filter *for Γ is a collection $F = \left\langle F^{\bar{t}_1}, \ldots, F^{\bar{t}_m} \right\rangle$ containing all possible sequences of stages $\bar{t}_j$ where each $F^{\bar{t}_j}$ is a function $F^{\bar{t}_j} : S^{\bar{t}_j} \to 2^{S^{\bar{t}_j}}$ and the range of $F^{\bar{t}}$ is a partition of $S^{\bar{t}}$.*

An n-person stage-based sequential game Γ and an information filter F for Γ defines a new game Γ_F. We refer to such games as *filtered stage-based sequential games*.

Having described the needed framework for representing information abstractions, we are now ready to present our information abstraction algorithms. In the rest of this chapter we describe four of our information abstraction algorithms and present heads-up limit Texas Hold'em poker-playing programs based on each of the algorithms. In Section 6.3 we present a lossy version of our *GameShrink* lossless abstraction algorithm [61], and *GS1*, the first of our poker-playing programs. In Section 6.4 we present an information abstraction algorithm based on k-means clustering and integer programming, and describe *GS2*, the next version of our poker-playing program based on that abstraction algorithm. The similarity metric used in that algorithm is the expected probability of winning, and we refer to it as the *expectation-based* algorithm. Section 6.5 presents a *potential-aware* extension of the expectation-based algorithm, and the corresponding player, *GS3*. Section 6.6 discusses strategy-based abstraction, and the corresponding player, *GS4*. Section 6.7 contains an extensive, controlled experimental comparison of the expectation-based and potential-aware algorithms, along with a variant of the expectation-based algorithm.

6.3 Lossy version of *GameShrink*

In previous work [61] we described an algorithm, *GameShrink*, for performing lossless (*i.e. equilibrium-preserving*) abstractions in a certain class of games. In that work, we mentioned that *GameShrink* could be modified to function as a lossy version capable of finding coarser-grained lossy abstractions. Here we describe that algorithm as it applies to n-person stage-based sequential games.

The algorithm first performs a bottom-up pass on the stage graph to identify strategically-similar information states. It then performs a top-down pass to merge information states.

Algorithm 3 *GameShrink* (Γ, i)

```
// $\Gamma$ is the game, $i$ is the player for whom we are computing an abstraction

1. Let $similarity$ be a data structure mapping pairs of game states (defined by a stage
   $t \in \mathcal{T}$ and signals $\theta$ and $\theta'$ revealed to player $i$) to real values. Initialize the data
   structure to be empty.

2. For each stage $t \in \mathcal{T}$ processed in bottom-up order:
```

(a) *If t does not have outgoing edges in the stage graph, then for all pairs of signals θ, θ' that could have been revealed to player i:*

   i. *Let V_1 be the set of every possible revelation of signals θ_{-i} for the the other i players when player i has received θ.*

   ii. *Let V_2 be the set of every possible revelation of signals θ'_{-i} for the the other i players when player i has received θ'.*

   iii. *Let $E \subseteq V_1 \times V_2$ contain an edge (v_1, v_2) if v_1 and v_2 are compatible in the sense that they correspond to game states in which the other players' signals are consistent.*

   iv. *For each $(v_1, v_2) \in E$, let $w(v_1, v_2)$ be the average squared difference in utility for player i when having seen signal θ in v_1 versus θ' in v_2, where the average is taken over all possible outcomes.*

   v. *Let $similarity(t, \theta, \theta')$ be the weight of the minimum-weight perfect matching in G.*

(b) *If t has outgoing edges in the stage graph, then for all pairs of signals θ, θ' that could have been revealed to player i up through the beginning of stage $t \in \mathcal{T}$:*

   i. *Let V_1 be the set of possible pairs of game stages $\tilde{t}$ that can immediately follow t and signal revelations $\tilde{\theta}$ that can extend what has already been revealed by θ.*

   ii. *Let V_2 be the set of possible pairs of game stages $\tilde{t}'$ that can immediately follow t' and signal revelations $\tilde{\theta}'$ that can extend what has already been revealed by θ'.*

   iii. *Let $E \subseteq V_1 \times V_2$ contain an edge (v_1, v_2) if v_1 and v_2 are compatible in the sense that they correspond to pairs of game states in which the stage games are the same player i's signals are consistent.*

   iv. *For each $(v_1, v_2) \in E$, let $w(v_1, v_2)$ be equal to the corresponding value of $similarity(\tilde{t}, \tilde{\theta}, \tilde{\theta}'$.*

   v. *Let $similarity(t, \theta, \theta')$ be the weight of the minimum-weight perfect matching in G.*

3. *Initialize F to be the identity filter for Γ.*

4. *For each stage $t \in \mathcal{T}$ processed in top-down order, and for all pairs of signals θ, θ' that could have been revealed to player i, if $similarity(t, \theta, \theta') \leq \text{THRESHOLD}_t$, then*

$$F(t, \theta) \leftarrow F(t, \theta') \leftarrow F(t, \theta) \cup F(t, \theta').$$

5. Output F.

The THRESHOLD$_t$ parameter in Algorithm 3 specifies a tolerance for how strategically different two game states can be to still be considered similar. (If THRESHOLD$_t = 0$, the algorithm yields a lossless abstraction.) This must be specified by the user of the algorithm, and is how the size of the resulting abstraction is controlled. We discuss this further as we describe the construction of *GS1* using this abstraction algorithm.

6.3.1 *GS1*: Application of lossy *GameShrink* and real-time equilibrium computation for the endgame in Texas Hold'em

Using the above information abstraction as a component, we constructed *GS1*, our first Texas Hold'em poker-playing program. *GS1* solves the pre-flop and flop separately from the turn and river. We discuss these two phases in Sections 6.3.1 and 6.3.1, respectively.

Strategy computation for the pre-flop and flop

GS1 computes the strategies for the pre-flop and flop offline. There are two distinct phases to the computation: the automated abstraction and the equilibrium approximation.

For automatically computing a state-space abstraction for the first and second rounds, we use the lossy version of the *GameShrink* algorithm described above.

We control the coarseness of the abstraction that *GameShrink* computes by a set of parameters THRESHOLD$_t$, one for each $t \in \mathcal{T}$. The abstraction can range from lossless (THRESHOLD$_t$ = 0), which results in an equilibrium for the original game, to complete abstraction (THRESHOLD$_t$ = ∞), which treats all nodes of the game as the same.

In the first betting round, there are $\binom{52}{2} = 1326$ distinct possible hands. However, there are only 169 strategically different hands. For example, holding A♠A♣ is no different (in the pre-flop phase) than holding A♢A♡. Thus, any pair of Aces may be treated similarly.[2] We run *GameShrink* with a threshold of 0 for the pre-flop stage, and *GameShrink* automatically discovers these abstractions.

[2]This observation is well-known in poker, and in fact optimal strategies for pre-flop (1-round) Texas Hold'em have been computed using this observation [142].

In the second round, there are $\binom{52}{2}\binom{50}{3} = 25{,}989{,}600$ distinct possible hands. Again, many of these hands are strategically similar. However, applying *GameShrink* with the threshold set to zero results in a game which is still too large for an equilibrium-finding (LP) algorithm to handle. Thus we use a positive threshold that yields an abstraction that has 2,465 strategically different hands.

To speedup *GameShrink*, we precomputed several databases. First, a `handval` database was constructed. It has $\binom{52}{7} = 133{,}784{,}560$ entries. Each entry corresponds to seven cards and stores an encoding of the hand's rank, enabling rapid comparisons to determine which of any two hands is better (ties are also possible). These comparisons are used in many places by our algorithms.

To compute an index into the `handval` database, we need a way of mapping 7 integers between 0 and 51 to a unique integer between 0 and $\binom{52}{7} - 1$. We do this using the *colexicographical ordering* of subsets of a fixed size [22] as follows. Let $\{c_1, \ldots, c_7\}$, $c_i \in \{0, \ldots, 51\}$, denote the 7 cards and assume that $c_i < c_{i+1}$. We compute a unique index for this set of cards as follows:

$$index(c_1, \ldots, c_7) = \sum_{i=1}^{7} \binom{c_i}{i}.$$

We use similar techniques for computing unique indices in the other databases.

Another database, `db5`, stores the expected number of wins and losses (assuming a uniform distribution over remaining cards) for five-card hands (the number of draws is inferred from this). This database has $\binom{52}{2}\binom{50}{3} = 25{,}989{,}600$ entries, each corresponding to a pair of hole cards along with a triple of flop cards. In computing the `db5` database, our algorithm makes heavy use of the `handval` database. The `db5` database is used to quickly compare how strategically similar a given pair of flop hands are. This enables *GameShrink* to run much faster, which allows us to compute and evaluate several different levels of abstraction.

By using the above precomputed databases, we are able to run *GameShrink* in about four hours for a given abstraction threshold. Being able to quickly run the abstraction computation allowed us to evaluate several different abstraction levels before settling on the most accurate abstraction for which we could compute an equilibrium approximation. After evaluating several abstraction thresholds, we settled on one that yielded an abstraction that kept all the 169 pre-flop hands distinct and had 2,465 classes of flop hands.

Once we have computed an abstraction, we are ready to perform the equilibrium computation for that abstracted game. In this phase of the computation, we are only considering the game that consists of the first two betting rounds, where the payoffs for this truncated game are computed using an expectation over the possible cards for the third and fourth rounds, but ignoring any betting that might occur in those later rounds.[3]

Two-person zero-sum games can be solved via linear programming using the sequence form representation of games. Building the linear program itself, however, is a non-trivial computation. It is desirable to be able to quickly perform this operation so that we can apply it to several different abstractions (as described above) in order to evaluate the capability of each abstraction, as well as to determine how difficult each of the resulting linear programs are to solve.

The difficulty in constructing the linear program lies primarily in computing the expected payoffs at the leaf nodes. Each leaf corresponds to two pairs of hole cards, three flop cards, as well as the betting history. Considering only the card history (the betting history is irrelevant for the purposes of computing the expected number of wins and losses), there are $\binom{52}{2}\binom{50}{2}\binom{48}{3} \approx 2.8 \cdot 10^{10}$ different histories. Evaluating each leaf requires rolling out the $\binom{45}{2} = 990$ possible turn and river cards. Thus, we would have to examine about $2.7 \cdot 10^{13}$ different combinations, which would make the LP construction slow (a projected 36 days on a 1.65 GHz CPU).

To speed up this LP creation, we precomputed a database, `db223`, that stores for each pair of hole cards, and for each flop, the expected number of wins for each player (losses and draws can be inferred from this). This database thus has

$$\frac{\binom{52}{2}\binom{50}{2}}{2}\binom{48}{3} = 14{,}047{,}378{,}800$$

entries. The compressed size of `db223` is 8.4 GB and it took about a month to compute. We store the database in one file per flop combination, and we only load into memory one file at a time, as needed. By using this database, *GS1* can quickly and exactly determine the payoffs at each leaf for any abstraction. Once the abstraction is computed (as described in the previous subsection), we can build the LP itself in about an hour. This approach determines the payoffs *exactly*, and does not rely on any randomized sampling.

Using the abstraction described above yields a linear program with 243,938 rows, 244,107 columns, and 101,000,490 non-zeros. We solved the LP using the barrier method of CPLEX. This computation used 18.8 GB RAM and took 7 days, 3 hours. *GS1* uses the strategy computed in this way for the pre-flop and flop betting rounds. Because our approximation does not involve any lossy abstraction on the pre-flop cards, we expect the

[3]We used a more sophisticated technique for estimating payoffs of a truncated game in our construction of *GS2*. We describe this technique in detail in Section 7.2.

resulting pre-flop strategies to be almost optimal, and certainly a better approximation than what has been provided in previous computations that only consider pre-flop actions [142].

Strategy computation for the turn and river

Once the turn card is revealed, there are two betting rounds remaining. At this point, there are a wide number of histories that could have occurred in the first two rounds. There are 7 possible betting sequences that could have occurred in the pre-flop betting round, and 9 possible betting sequences that could have occurred in the flop betting round. In addition to the different betting histories, there are a number of different card histories that could have occurred. In particular, there are $\binom{52}{4} = 270{,}725$ different possibilities for the four community cards (three from the flop and one from the turn). The large number of histories makes computing an accurate equilibrium approximation for the final two rounds for every possible first and second round history prohibitively hard. Instead, *GS1* computes in *real-time* an equilibrium approximation for the final two rounds based on the observed history for the current hand. This enables *GS1* to perform computations that are focused on the specific remaining portion of the game tree, and thus allows more refined abstractions to be used in the later stages than if offline computation were used for the later stages (where the game tree has exploded to be enormously wide).

There are two parts to this real-time computation. First, *GS1* must compute an abstraction to be used in the equilibrium approximation. Second, *GS1* must actually compute the equilibrium approximation. These steps are similar to the two steps taken in the offline computation of the pre-flop and flop strategies, but the real-time nature of this computation poses additional challenges.

The problem of computing abstractions for each of the possible histories is made easier by the following two observations: (1) the appropriate abstraction (even a theoretical lossless one) does not depend on the betting history (but does depend on the card history, of course); and (2) many of the community card histories are equivalent due to suit isomorphisms. For example, having 2♠3♠4♠5♠ on the board is equivalent to having 2♣3♣4♣5♣ as long as we simply relabel the suits of the hole cards and the (as of yet unknown) river card. Observation 2 reduces the number of abstractions that we need to compute (in principle, one for each of the $\binom{52}{4}$ flop and turn card histories, but reduced to 135,408).

Although *GameShrink* can compute one of these abstractions *for a given abstraction threshold* in just a few seconds, we perform these abstraction computations off-line for two reasons. First, since we are going to be playing in real-time, we want the strategy computation to be as fast as possible. Given a small fixed limit on deliberation time (say, 15 seconds), saving even a few seconds could lead to a major improvement in strategy quality. Second, we can set the abstraction threshold differently for each combination of community cards in order to capitalize on the finest abstraction for which the equilibrium can still be solved within a reasonable amount of time. One abstraction threshold may lead to a very coarse abstraction for one combination of community cards, while leading to a very fine abstraction for another combination. Thus, for each of the 135,408 cases, we perform several abstraction computations with different abstraction parameters in order to find an abstraction close to a target size (which we experimentally know the real-time equilibrium solver (LP solver) can solve (exactly or approximately) within a reasonable amount of time). Specifically, our algorithm first conducts binary search on the abstraction threshold for round 3 (the turn) until *GameShrink* yields an abstracted game with about 25 distinct hands for round 3. Our algorithm then conducts binary search on the abstraction threshold for round 4 (the river) until *GameShrink* yields an abstracted game with about 125 distinct hands for round 4. Given faster hardware, or more deliberation time, we could easily increase these two targets.

Using this procedure, we computed all 135,408 abstractions in about one month using six general-purpose CPUs.

Before we can construct the linear program for the turn and river betting rounds, we need to determine the probabilities of holding certain hands. At this point in the game the players have observed each other's actions leading up to this point. Each player action reveals some information about the type of hand the player might have.

Based on the strategies computed for the pre-flop and flop rounds, and based on the observed history, we apply Bayes' rule to estimate the probabilities of the different pairs of hole cards that the players might be holding. Letting h denote the history, Θ denote the set of possible pairs of hole cards, and s denote the strategy profile for all players, we can derive the joint probability that the players hold hole cards $\theta \in \Theta$ as follows:

$$\Pr[\theta \mid h, s] = \frac{\Pr[h \mid \theta, s] \cdot \Pr[\theta]}{\Pr[h \mid s]} = \frac{\Pr[h \mid \theta, s] \cdot \Pr[\theta]}{\sum\limits_{\theta' \in \Theta} \Pr[h \mid \theta', s]}$$

Since we already know $\Pr[h \mid \theta, s]$ (we can simply look at the strategies, s_i, computed for the first two rounds), we can compute the probabilities above. Of course, the resulting probabilities might not be exact because the strategies for the pre-flop and flop rounds do not constitute an exact equilibrium since, as discussed above, they were computed without considering a fourth possible raise on the flop or any betting in rounds 3 and 4, and abstraction was used.

Once the turn card is dealt out, *GS1* creates a separate thread to construct and solve the linear problem corresponding to the abstraction of the rest of that game. When it is time for *GS1* to act, the LP solve is interrupted, and the current solution is accessed to get the strategy to use at the current time. When the algorithm is interrupted, we save the current basis which allows us to continue the LP solve from the point at which we were interrupted. The solve then continues in the separate thread (if it has not already found the optimal solution). In this way, our strategy (vector of probabilities) keeps improving in preparation for making future betting actions in rounds 3 and 4.

There are two different versions of the simplex algorithm for solving an LP: *primal simplex* and *dual simplex*. The primal simplex maintains primal feasibility, and searches for dual feasibility. (Once the primal and dual are both feasible, the solution is optimal.) Similarly, dual simplex maintains dual feasibility, and searches for primal feasibility. (Dual simplex can be thought of as running primal simplex on the dual LP.) When *GS1* is playing as player 1, the dual variables correspond to her strategies. Thus, to ensure that at any point in the execution of the algorithm we have a feasible solution, *GS1* uses dual simplex to perform the equilibrium approximation when she is player 1. Similarly, she uses the primal simplex algorithm when she is player 2. If given an arbitrarily long time to deliberate, it would not matter which algorithm was used since at optimality both primal and dual solutions are feasible. But since we are also interested in interim solutions, it is important to always have feasibility for the solution vector in which we are interested. Our conditional choice of the primal or dual simplex method ensures exactly this.

One subtle issue is that *GS1* occasionally runs off the equilibrium path. For example, suppose it is *GS1's* turn to act, and the current LP solution indicates that she should bet; thus *GS1* bets, and the LP solve continues. It is possible that as the LP solve continues, it determines that the best thing to have done would have been to check instead of betting. If the other player re-raises, then *GS1* is in a precarious situation: the current LP solution is stating that she should not have bet in the first place, and consequently is not able to offer any guidance to the player since she is in an information set that is reached with probability zero. It is also possible for *GS1* to determine during a hand whether the opponent has gone off the equilibrium path, but this rarely happens because their cards are hidden. In these situations, *GS1* simply calls the bet. (Another technique for handling the possibility of running oneself off the equilibrium path as mentioned above would be to save the previous LP solution(s) that specified a behavior to use in the information set that now has zero probability.)

We present experimental results for *GS1* in Section 6.4.2 when we compare its performance with *GS2*, discussed next.

6.4 Expectation-based automated abstraction using optimization

Algorithm 3, (the lossy version of *GameShrink*) discussed in the previous subsection, suffers from three major drawbacks. (When *GameShrink* is used in the lossless—exact rather than approximation—mode, these criticisms do not apply: it finds an equilibrium-preserving abstraction. However, if one were to apply the lossless mode to Texas Hold'em, the resulting LP would be way too large to solve.)

- The first, and most serious, is that the abstraction that *GameShrink* computes can be highly inaccurate because the grouping of states is in a sense greedy. For example, if *GameShrink* determines that state A is similar to state B, and then determines that state B is similar to state C, it will group A and C together, despite the fact that A and C may not be very similar. The quality of the abstraction can be even worse when a longer sequence of such comparisons leads to grouping together extremely different states. Stated differently, the greedy aspect of the algorithm leads to lopsided classes where large classes are likely to attract even more states into the class.
- The second drawback to *GameShrink* is that there is no way to directly specify how many classes the abstraction algorithm should yield (overall or at any specific betting round). Rather, there is a parameter (for each stage) that specifies a threshold of how different states can be and still be considered the same. If one knows how large an LP can be solved, one cannot create an LP of that size by specifying the number of classes in the abstraction directly; rather one must use trial-and-error (or some variant of binary search applied to the setting of multiple parameters) to pick the similarity thresholds (one for each betting round) in a way that yields an LP of the desired size.
- The third drawback to *GameShrink* is its scalability. In particular, the time needed to compute an abstraction for a three-round truncated version of Texas Hold'em was over a month. Furthermore, it would have to be executed in the inner loop of the parameter guessing algorithm of the previous paragraph (*i.e.*, once for each setting of the parameters).

In this subsection we describe a new abstraction algorithm that eliminates these problems.

We introduce a new structure for our algorithm, which we call the *abstraction tree*. For limit Texas Hold'em, the basic abstraction tree is initialized as follows. The root node contains $\binom{52}{2} = 1326$ children, one for each possible pair of hole cards that a player may be dealt. Each of these children has $\binom{50}{3}$ children, each corresponding to the possible flops that can appear after the two hole cards in the parent node have already been dealt. Similarly, the nodes at the next two levels have 47 and 46 children corresponding to the possible turn and river cards, respectively. Figure 6.1 provides an illustration.

This structure is by no means limited to poker. In terms of stage-based sequential games (Definition 12), we can derive the abstraction tree by considering the possible revelation of signals to one player as well as the possible stage games. (The abstraction tree for limit Texas Hold'em is much simpler since the stage graph is a line graph.)

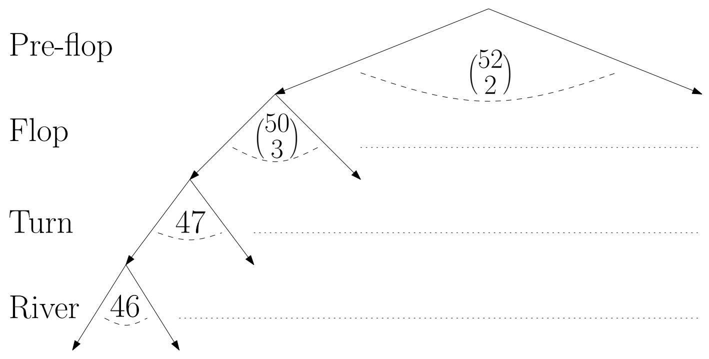

Figure 6.1: The initial abstraction tree of Texas Hold'em.

Suppose we wish to limit the number of filtered signals in the first stage to K_1. In Texas Hold'em, this corresponds to grouping each of the $\binom{52}{2} = 1326$ different hands into K_1 classes. We treat this as a clustering problem. To perform the clustering, we must first define a metric to determine the similarity of two states. Letting (w, l, d) be the number of possible wins, losses, and draws (based on the roll-out of the remaining signals), we compute the state's value as $w + d/2$, and we take the distance between two states to be the absolute difference between their values. This gives us the necessary ingredients to apply the k-means clustering algorithm [103]:

Algorithm 4 *k-means clustering*

1. *Create k centroid points in the interval between the minimum and maximum state values.*

2. *Assign each state to the nearest centroid.*

3. *Adjust each centroid to be the mean of their assigned state values.*

4. *Repeat steps 2 and 3 until convergence.*

This algorithm is guaranteed to converge, but it may find a local optimum. Therefore, in our implementation we run it several times with different starting points to try to find a global optimum. For a given clustering, we can compute the error (according to the value measure) that we would expect to have when using the abstraction.

For the later stages of the game, we again want to determine what the best abstraction classes are. Here we face the additional problem of determining how many children each parent in the abstraction tree can have. How should the right to have K_2 children (abstraction classes that have not yet been generated at this stage) be divided among the K_1 parents? We model and solve this problem as a 0-1 integer program [114] as follows. Our objective is to minimize the expected error in the abstraction. Thus, for each of the K_1 parent nodes, we run the k-means algorithm presented above for values of k between 1 and $K_2 - K_1$. We denote the expected error when node i has k children by $c_{i,k}$. We denote by p_i the probability of receiving a signal that is in abstraction class i (*i.e.*, in parent i). Based on these computations, the following 0-1 integer program finds the abstraction that minimizes the overall expected error for the second level:

$$\begin{array}{lrcl}
\min & \sum_{i=1}^{K_1} p_i \sum_{k=1}^{K_2-K_1} c_{i,k} x_{i,k} & & \\
\text{s.t.} & \sum_{i=1}^{K_1} \sum_{k=1}^{K_2-K_1} k x_{i,k} & \leq & K_2 \\
& \sum_{k=1}^{K_2-K_1} x_{i,k} & = & 1 \quad \forall i \\
& x_{i,k} & \in & \{0,1\}
\end{array}$$

The decision variable $x_{i,k}$ is set to 1 if and only if node i has k children. The first constraint ensures that the limit on the overall number of children is not exceeded. The second constraint ensures that a decision is made for each node. This problem is a generalized knapsack problem, and although NP-complete, can be solved efficiently using off-the-shelf integer programming solvers (*e.g.,* CPLEX solves this problem in less than one second at the root node of the branch-and-bound search tree).

We repeat this procedure in a top-down fashion for the other stages of the game. We refer to this algorithm as *expectation-based* abstraction.

As discussed, our technique optimizes the abstraction stage by stage, *i.e.*, level by level in the abstraction tree. A better abstraction (even for the same similarity metric) could conceivably be obtained by optimizing all rounds in one holistic optimization. However, that seems infeasible. First, the optimization problem would be nonlinear because the probabilities at a given level depend on the abstraction at previous levels of the tree. Second, the number of decision variables in the problem would be exponential in the size of the initial abstraction tree (which itself is large), even if the number of abstraction classes for each level is fixed.

6.4.1 *GS2*: Application of expectation-based abstraction to Texas Hold'em

We applied the expectation-based abstraction algorithm for constructing a Texas Hold'em player called *GS2*.

In the first phase, we solve an LP that models the strategic interaction in the first three rounds of the game, where the payoffs at the end of the third round are estimated based on an expectation of the actions in the fourth round. Thus, this model is based on actions in the entire game. (We describe the details of this payoff computation in Section 7.2.) We perform this first phase computation offline.

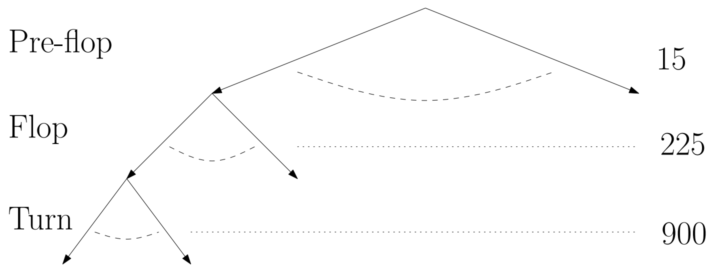

Figure 6.2: Abstraction tree for *GS2*'s Phase 1 equilibrium approximation.

Because the LP corresponding to this three-round version of Texas Hold'em is too large to solve, we must employ abstraction techniques to reduce the size of the LP. Based on the available computational resources for solving the LP, there is a limit to the number of strategically different hands that can be considered in each round. Based on the computational resources available to us, we determined we could handle up to 15 strategically different hands in the first round, 225 in the second round, and 900 in the third round. Solving the LP took 6 days and 70 gigabytes of RAM using the barrier method in CPLEX 10.0 (using one 1.65 GHz CPU on an IBM p570 computer). Of course, as computing speed increases over time and enables larger LPs to be solved, we can apply our algorithm to compute finer abstractions.

Although the first phase computation outputs strategies for the first three rounds, we only actually use the strategies it computes for the first two rounds. We don't use the third-round strategy for actual play because 1) it may be inaccurate because the fourth round was modeled based on expectations rather than game theory, as discussed above, and 2) we can use a finer abstraction if we compute the third-round strategy in real-time.

The strategies we use for the third and fourth rounds are computed in real-time while the game is being played. We do this so that the reasoning can be specific to the situation at hand (*i.e.*, cards on the table and betting history); the mere number of such situations precludes precomputing the answer to each one of them. Once the turn card appears at the beginning of the third betting round, we compute an equilibrium approximation for the remaining game. We do this as follows. First, we update the players' hand probabilities using Bayes' rule on the Phase 1 strategies and the observed betting history. Second, we use our automated abstraction algorithm to construct and solve a smaller LP. Here we are considering 10 strategically different hands in the third round, and 100 in the fourth round. This is based on computational experiments we performed to find the finest abstraction for which we could usually solve (at least near-optimally) the corresponding LP in under one minute.

For Phase 2, we compute a third and fourth-round abstraction using the same approach. We do this separately for each of the $\binom{52}{4}$ possible flop and turn combinations.[4]

6.4.2 Evaluation

We conducted a host of experiments against the Texas Hold'em programs *GS1*, *SparBot*, and *VexBot*. We used the *series competition* format of the first AAAI Computer Poker Competition (held in July 2006).[5] In our experiments, our player competed with each of

[4]Most of the computation time of the abstraction algorithm is spent running the k-means clustering algorithm. Our straightforward implementation of the latter could be improved by using sophisticated data structures such as a kd-tree [122] or performing bootstrap averaging to find the initial clusters [39]. This would also allow one to run the k-means clustering more times and thus have an even better chance of finding the global optimum of any individual k-means clustering problem.

[5]One difference between our experiments and the format of the AAAI poker competition is that the AAAI competition performed *duplicate matches*, in which the deck shuffles are stored for each match, and replayed with the players' roles reversed. Using this approach, if one player receives particularly lucky cards during a match, this will be offset by the duplicate match in which the other player receives the lucky cards. Unfortunately, we have not yet been able to run such experiments due to the fact that the other players are only available in the *Poker Academy* software package, which does not support duplicate matches.

the other players in 50 1,000-hand series, where the players' memories are reset before each series (this resetting only affects *Vexbot* since the other players do not perform any opponent modeling). The purpose of resetting the memories after each series is to give a learning bot a fair amount of time to develop an exploitative strategy, but not a completely unrealistic amount. 1,000 hands per series is more than reasonable since most games between humans last for at most a few hundred hands [75]. The AAAI competition also used 1,000 hands per series.

The results are presented in Table 6.1. Based on the winning records, our agent, generated using the techniques described above, outperforms the leading prior poker-playing programs. For *GS1* (the leading prior automatically generated agent for the game), this result is statistically significant according to the sign test, with p-value 3.06×10^{-4}. In terms of overall chips won, our program beats the prior game theory-based players, *GS1* and *Sparbot*, but loses to *Vexbot* by a small margin. The margin of victory over *GS1* is statistically significant as the estimated variance of small bets won or lost per hand is ± 0.0268 small bets (*i.e.*, two chips) per hand for 50,000 hands. However, the comparisons with *Sparbot* and *Vexbot* are in effect statistical ties.

Opponent	**Series won by *GS2***	**Sign test p-value**	**Win rate of *GS2*** (small bets per hand)
GS1	38 of 50	3.06×10^{-4}	$+0.0312$
Sparbot	28 of 50	4.80×10^{-1}	$+0.0043$
Vexbot	32 of 50	6.49×10^{-2}	-0.0062
GS2 w/o improved abstraction and w/o estimated payoffs	48 of 50	2.27×10^{-12}	$+0.0287$
GS2 w/o improved abstraction	35 of 50	6.60×10^{-3}	$+0.0273$
GS2 w/o estimated payoffs	44 of 50	3.24×10^{-8}	$+0.0072$

Table 6.1: Experimental results evaluating *GS2* against *GS1*, *Sparbot*, and *Vexbot*, as well as in self-play with various features removed.

Interestingly, our player has a better winning percentage against *Vexbot* than it does against *Sparbot*; yet based on the win rate of average chips won and lost per hand, it performs better against *Sparbot*. This is possibly due to the fact that *Vexbot* is able to exploit a weakness in our player that enables it to win a large amount of chips on a few hands, but those particular hands do not come up often enough to affect the match winning percentages. Another possibility is that our player is playing much more conservatively than *Vexbot*, which enables it to obtain a better winning percentage, but it is not playing aggressively enough on certain hands to capture a higher chip win rate. Exploring these possibilities could potentially lead to further improvements in our player.

In addition to evaluating the performance of our player against existing players, out of scientific interest we also wanted to measure the individual effect of our improved automated abstraction algorithm and our technique of estimating payoffs in a truncated game (described in Section 7.2). The fourth row in Table 6.1 reports results from the comparison between our player, and our player using the old version of the *GameShrink* algorithm (as used in *GS1*) and without estimating the payoffs of the truncated game (but instead using a uniform roll-out as in *Sparbot* and *GS1*). (Our technique for estimating payoffs of the truncated game is discussion in Section 7.2.) The introduction of these two new techniques is a clear winner, with a p-value for the winning percentage of 2.27×10^{-12} and even a statistically significant win rate in terms of the number of chips won. The last two rows of Table 6.1 report the performance boost that each of the two new techniques yields individually. The improved automated abstraction algorithm produces a greater performance gain than the technique of estimating payoffs in the truncated game, but each of the two techniques yields a dramatic statistically significant improvement in performance.

6.5 Potential-aware automated abstraction

The algorithm described in the previous subsection was based on a myopic expected-value computation, and used k-means clustering with integer programming to compute the abstraction. A state of the game was evaluated according to the probability of winning. The algorithm clustered together states with similar probabilities of winning, and it started computing the abstraction from the first round and then down through the card tree. This *top-down* algorithm generated the abstraction for *GS2*.

That approach does not take into account the *potential* of game states. For example, certain poker hands are considered *drawing hands* in which the hand is currently weak, but has a chance of becoming very strong. An important type of drawing hand is one in which the player has four cards of a certain suit (five are required to make a *flush*); at the present stage the hand is not very strong, but could become so if the required card showed up later in the game. Since the strength of such a hand could potentially turn out to be much different later in the game, it is generally accepted among poker experts that such a hand should be played differently than another hand with the same chance of winning, but without as much potential to improve.[6] However, if using the difference between probabilities of winning

[6]In the manual abstraction used in *Sparbot*, there are six buckets of hands where the hands are selected

as the metric for performing the clustering, the automated abstraction algorithm would consider these two very different situations to be quite similar.

One possible approach to handling the problem that certain game states with the same probability of winning may have different potential would be to consider not only the expected strength of a game state, but also its variance. In other words, the algorithm would be able to differentiate between two states that have the same probability of winning, but where one game state faces more uncertainty about what its final strength will be, while the other game state's strength is unlikely to change much. Although this would likely improve the basic abstraction algorithm, it does not take into account the different *paths of information revelation* that game states take in increasing or decreasing in strength. For example, two game states could have similar means and variances, but one state's strength may be determined after one more step, while the other state needs two more steps before its final strength is determined.

To address this, we introduce an approach where we associate with each state of the game a *histogram* over future possible states. This representation can encode all the pertinent information from the rest of the game, such as the probability of winning, the variance of winning, and the paths of information revelation. (In such a scheme, the k-means clustering step requires a distance function to measure the dissimilarity between different states. The metric we use is the usual L_2-distance metric. Given a finite set $\mathcal{S}$ of states where each hand i is associated with histogram h_i over the future possible states $\mathcal{S}$, the L_2-distance between states i and j is $dist(i, j) = \left[\sum_{s \in \mathcal{S}} (h_i(s) - h_j(s))^2\right]^{\frac{1}{2}}$.)

There are at least two prohibitive problems with the vanilla approach as stated. First, there are a huge number of possible reachable future states, so the dimensionality of the histograms is too large to do meaningful clustering with a reasonable number of clusters (*i.e.,* small enough to lead to an abstracted game that can be solved for equilibrium). Second, for any two states at the same level of the game, the descendant states are disjoint. Thus the histograms would have non-overlapping supports, so any two states would have maximum dissimilarity and thus no basis for clustering.

For both of these reasons (and for reducing memory usage and enhancing speed), we coarsen the domains of the histograms. First, instead of having histograms over individual states, we use histograms over abstracted states (clusters), which contain a number of states

based on likelihood of winning and one extra bucket for hands that an expert considered to have high potential [14]. In contrast, our approach is automated, and does its bucketing holistically based on a multi-dimensional notion of potential (so it does not separate buckets into ones based on winning probability and ones based on potential). Furthermore, its abstraction is drastically finer grained.

each. We will have, for each cluster, a histogram over clusters later in the game. Second, we restrict the histogram of each cluster to be over clusters at the next level of the game tree only (rather than over clusters at all future levels). However, we introduce a technique (a bottom-up pass of constructing abstractions up the tree) that allows the clusters at the next level to capture information from all later levels.

One way of constructing the histograms would be to perform a bottom-up pass of the stage graph: abstracting those stages that are leaves in the stage graph, creating histograms for parents of the leaf nodes nodes based on the leaf node clusters, then abstracting that second level, and so on. This is indeed what we do to find the abstraction for the first stage.

However, for later betting stages, we improve on this algorithm further by leveraging our knowledge of the fact that abstracted children of any cluster at levels above should only include states that can actually be children of the states in that cluster. We do this by *multiple* bottom-up passes, one for each cluster at the level above. For example, if a cluster in the first stage of Texas Hold'em contains only those states where the hand consists of two Aces, then when we are doing abstraction for the second stage, the bottom-up pass for that first stage cluster should only consider future states where the hand contains two Aces as the hole cards. This enables the abstraction algorithm to narrow the scope of analysis to information that is relevant given the abstraction that it made for earlier levels. The following subsections describe our abstraction algorithm in detail.

6.5.1 Computing the abstraction for the first stage

The first piece of the abstraction we compute is for the first stage. As discussed above, we will have, for each set of signals that could be observed in the first stage, a histogram over clusters of game states at stages that can follow the first stage. (These clusters are not necessarily the same that we will eventually use in the abstraction for those stages, discussed later.)

To obtain the clusters at the second (and lower) levels, we perform a bottom-up pass of the stage graph as follows. Beginning with the stages that correspond to leaf nodes in the stage graph, we cluster the possible signals into some small number of clusters[7] based on the probability of winning. Next, we proceed in a bottom-up order. For each stage and

[7]For our application to Texas Hold'em, the number of clusters at each stage was chosen to reflect the fact that when clustering data, the number of clusters needed to represent meaningful information should be at least the level of dimensionality of the data. So, the number of clusters at a certain stage should be at least as great as the number of clusters at stages that follow.

for the possible revealed signals, we compute its histogram over the clusters at the next stages, which we have already computed, and we cluster the possible signals into some small number of clusters. Once we get to the first stage we perform k-means clustering on these histograms to obtain the buckets that constitute our abstraction for the first stage. The value of k is specified as an input to the algorithm and corresponds to how many buckets we wish our abstraction to have in the first stage.

6.5.2 Computing the abstraction for intermediate stages

Just as we did in computing the abstraction for the first stage, we start by performing a bottom-up clustering, beginning at the stages that correspond to leaves in the stage graph. However, instead of doing this bottom-up pass once, we do it once for each bucket in the first stage. Thus, instead of considering all possible signals in each pass, we only consider those hands which contain as the first-stage signals those signals that are present in the particular first-stage bucket we are looking at.

At this point we have, for each first-stage bucket, a set of second-level clusters. For each first-stage bucket, we have to determine how many child buckets it should actually have. For each first-stage bucket, we run k-means clustering on its second-round clusters for several values of k. This yields, for each first-stage bucket and each value of k, an error measure for that bucket assuming it will have k children. (The error is the sum of each data point's L_2 distance from the centroid of its assigned cluster.)

As in the abstraction algorithm used by *GS2*, we formulate and solve an integer program (IP) to determine how many children each first-round bucket should have (*i.e.,* what k should be for that bucket). The IP simply minimizes the sum of the errors of the level-1 buckets under the constraint that their k-values do not sum to more than K, where K is specified as an input to the algorithm for each stage.

This process is repeated for all interior nodes of the stage graph.

6.5.3 Computing the abstraction for leaf stages

In game stages corresponding to leaf nodes in the stage graph, there is no need to use the sophisticated clustering techniques discussed above since the players will not receive any more information. Instead, we simply compute these final-stage abstractions based on each hand's probability of winning, exactly the way as was done for computing the abstraction for *GS2*.

6.5.4 *GS3*: Application of potential-aware abstraction and holistic game solving to Texas Hold'em

Before computing an abstraction, we need to determine the coarseness of the resulting abstraction. Ideally we would compute an abstraction as fine-grained as possible. However, we need to limit the fineness of the abstraction to ensure that we are able to compute an equilibrium approximation for the resulting abstracted game.

One important aspects of the abstraction is the *branching factor*. One intuitively desirable property is to have an abstraction where the relative amount of information revealed in each stage is similar to the relative amount revealed in the game under consideration. For example, it would likely not be effective to have an abstraction that only had one bucket for each of the first three rounds, but had 1000 buckets for the last round. Similarly, we don't want to have 100 buckets in the first round if we are going to only have 100 buckets in the second, third, and fourth rounds, since then no new information would be revealed after the first round.

One implication of this reasoning is that the branching factor going into the flop (where three cards are dealt) should be greater than the branching factor going into the turn or river (where only one card is dealt in each round). Furthermore, it seems reasonable to require that the branching factor of the flop be at least the branching factor of the turn and river *combined*, since more information is revealed on the flop than on the turn and river together.

Based on these considerations, and based on some preliminary experiments to determine the problem size we could expect our equilibrium-finding algorithm to handle, we settled on an abstraction that has 20 buckets in the first round, 800 buckets in the second round, 4,800 buckets in the third round, and 28,800 buckets in the fourth round. This implies a branching factor of 40 for the flop, 6 for the turn, and 6 for the river.

The prior game-theory based players (*GS1*, *GS2*, and *Sparbot*) computed strategies by first splitting the game into two phases, and then solving the phases separately and then gluing together the separate solutions. In particular, *GS1* considers rounds 1 and 2 in the first phase, and rounds 3 and 4 in the second phase. *GS2* considers round 1, 2, and 3 in the first phase, and rounds 3 and 4 in the second phase. *Sparbot* considers rounds 1, 2, and 3 in the first phase, and rounds 2, 3, and 4 in the second phase. These approaches allow for finer-grained abstractions than what would be possible if a single, monolithic four-round model were used. However, the equilibrium finding algorithm used in each of those players was based on standard algorithms for LP which do not scale to a four-round model (except possibly for a trivially coarse abstraction).

Solving the (two) different phases separately causes important strategic errors in the player (in addition to those caused by lossy abstraction). First, it will play the first phase of the game inaccurately because it does not properly consider the later stages (the second phase) when determining the strategy for the first phase of the game. Second, it does not accurately play the second phase of the game because strategies for the second phase are derived based on beliefs at the end of the first phase, which are inaccurate.[8]

Therefore, we want to solve for equilibrium while keeping the game in one holistic phase. Using a holistic four-round model makes the equilibrium computation a difficult problem, particularly since our abstraction is very fine grained. As noted earlier, standard LP solvers (like CPLEX's simplex method and CPLEX's interior-point method) are insufficient for solving such a large problem. Instead, we used an implementation of Nesterov's *excessive gap technique* algorithm [117], which was recently specialized for two-person zero-sum sequential games of imperfect information [74, 56]. This algorithm is a gradient-based algorithm which requires $O(1/\epsilon)$ iterations to compute an ϵ-equilibrium, that is, a strategy for each player such that his incentive to deviate to another strategy is at most ϵ. This algorithm is an anytime algorithm since at every iteration it has a pair of feasible solutions, and the ϵ does not have to be fixed in advance. After 24 days of computing on 4 CPUs running in parallel, the algorithm had produced a pair of strategies with $\epsilon = 0.027$ small bets.

Experiments with *GS3* against static opponents

We tested our player, *GS3*, against seven prior programs: *BluffBot*, *GS2*, *Hyperborean*,[9] *Monash-BPP*, *Sparbot*, *Teddy*, and *Vexbot*. To our knowledge, this collection of opponents represents the "best of breed" in heads-up limit Texas Hold'em computer poker players. It includes all competitors from the 2006 AAAI Computer Poker Competition.

We also tested *GS3* against two (self-explanatory) benchmark strategies: *Always Call* and *Always Raise*. Although these last two strategies are completely predictable, it has been pointed out that it is important to evaluate a player against a wide range of opponents [14].

BluffBot, *GS2*, *Hyperborean*, *Monash-BPP*, *Sparbot*, and *Teddy* are static players, that is, each of them uses a mixed strategy that does not change over time.[10] Of course, *Always*

[8]We address some of these shortcomings in Section 7.2, but the techniqes discussed there do not completely eliminate these problems.

[9]There are actually two versions of *Hyperborean*: *Hyperborean-Bankroll* and *Hyperborean-Series*. The differences between those two players are not publicly available. We tested against both versions.

[10]Since no information about *Bluffbot*, *Hyperborean*, and *Teddy* is publicly available, we are statistically

Opponent	**Number of hands played**	***GS3*'s win rate**	**Empirical std. dev.**	**95% confidence interval**
Always Call	50,000	0.532	4.843	[0.490, 0.575]
Always Raise	50,000	0.442	8.160	[0.371, 0.514]
BluffBot	20,000	0.148	1.823	[0.123, 0.173]
GS2	25,000	0.222	5.724	[0.151, 0.293]
Hyperborean-Bankroll	20,000	0.099	1.779	[0.074, 0.124]
Hyperborean-Series	20,000	0.071	1.812	[0.045, 0.096]
Monash-BPP	20,000	0.669	2.834	[0.630, 0.709]
Sparbot	200,000	0.033	5.150	[0.010, 0.056]
Teddy	20,000	0.419	3.854	[0.366, 0.473]

Table 6.2: Experiments evaluating *GS3* against static opponents. The win rate is the average number of small bets *GS3* won per hand. (The win rate against an opponent that always folds is 0.75.) *GS3* beats each opponent by a statistically significant margin.

Call and *Always Raise* are also static strategies.

Table 6.2 summarizes our experiments comparing *GS3* with the eight static opponents. One interpretation of the last column is that if zero is strictly below the interval, we can reject the null hypothesis "*GS3* is not better than the opponent" at the 95% certainty level. Thus, *GS3* beat each of the opponents with statistical significance.

The matches against *Always Call* and *Always Raise* were conducted within *Poker Academy Pro*, a commercially available software package that facilitates the design of and experimentation with poker-playing programs. We played these two strategies against *GS3* for 50,000 hands. Unsurprisingly, *GS3* beat these simple strategies very easily.

The matches against *GS2* and *Sparbot* were also conducted within *Poker Academy Pro*. *GS3* outplayed its predecessor, *GS2*, by a large margin. *Sparbot* provided *GS3* with the toughest competition, but *GS3* beat it, too, with statistical significance.

The matches against the other participants of the 2006 AAAI Computer Poker Competition beyond *GS2* (*BluffBot*, *Hyperborean*, *Monash-BPP*, and *Teddy*) were conducted on the benchmark server available for participants of that competition. One advantage of this testing environment is that it allows for *duplicate matches*, in which each hand is played twice with the same shuffle of the cards and the players' positions reversed. (Of course, the player's memories are reset so that they do not know that the same hand is being played

evaluating *GS3*'s performance against them as though they were static.

a second time.) This reduces the role of luck, so the empirical standard deviation is lower than it would be in a normal match. Each match against these four players consisted of 20,000 duplicate hands (40,000 total). An additional way of evaluating the players in the AAAI competition is to split the experiment for each pair of competitors into 20 equal-length series, and declare as the winner of the pair the player who wins a larger number of the 20 series. Under that measure, *GS3* beat each of the opponents 20-0, except for *Hyperborean-Bankroll*, which *GS3* beat 19-1, and *Hyperborean-Series*, which *GS3* best 16-4.

Experiments with *GS3* against a learning opponent, *Vexbot*

Vexbot does not employ a static strategy. It records observations about its opponents' actions, and develops a model of their style of play. It continually refines its model during play and uses this knowledge of the opponent to try to exploit his weaknesses [13].

Since *Vexbot* is remembering (and exploiting) information from each hand, the outcomes of hands in the same match are not statistically independent. Also, one known drawback of *Vexbot* is that it is possible for it to get stuck in a local minimum in its learned model [12, 16]. Hence, demonstrating that *GS3* beats *Vexbot* in a single match (regardless of the number of hands played) is not significant since it is possible that *Vexbot* happened to get stuck in such a local minimum. Therefore, instead of statistically evaluating the performance of *GS3* on a hand-by-hand basis as we did with the static players, we evaluate *GS3* against *Vexbot* on a match-by-match basis.

We conducted 20 matches of *GS3* against *Vexbot*. The design of each match was extremely conservative, that is, generous for *Vexbot*. Each match consisted of 100,000 hands, and in each match *Vexbot* started with its default model of the opponent. We allowed it to learn throughout the 100,000 hands in each match (rather than flushing its memory every so often as is customary in computer poker competitions). This number of hands is many more than would actually be played between two players in practice. For example, the number of hands played in each match in the 2006 AAAI Computer Poker Competition was only 1,000.

The match results are summarized in Table 6.3. In every match, *GS3* beat *Vexbot* by a large margin, with a mean win rate of 0.142 small bets per hand. The 95% confidence interval for the overall win rate is [0.133, 0.151].

One criticism that could possibly be made against the experimental methodology described above is that we did not allow *Vexbot* to learn for some period before we started recording the winnings. With this in mind, we also present (in the third column of Table 6.3) *GS3*'s win rate over the last 10,000 hands only, which illustrates how well *GS3* would perform if we allowed *Vexbot* to train for 90,000 hands before recording any win/loss information. As can be seen from the data, *GS3* still outperforms *Vexbot*, winning 0.147 small bets per hand on average, with a 95% confidence interval of [0.115, 0.179].

	Small bets *GS3* won per hand	
Match #	Over all 100k hands	Over final 10k hands
1	0.129	0.197
2	0.132	0.104
3	0.169	0.248
4	0.139	0.184
5	0.130	0.150
6	0.153	0.158
7	0.137	0.092
8	0.147	0.120
9	0.120	0.092
10	0.149	0.208
11	0.098	0.067
12	0.153	0.248
13	0.142	0.142
14	0.163	0.169
15	0.165	0.112
16	0.163	0.172
17	0.108	-0.064
18	0.180	0.255
19	0.147	0.143
20	0.118	0.138
Mean:	0.142	0.147
Std. dev:	0.021	0.073
95% CI:	[0.133, 0.151]	[0.115, 0.179]

Table 6.3: Experiments against *Vexbot*. The third column reports *GS3*'s win rate over 10,000 hands after *Vexbot* is allowed to train for 90,000 hands.

Results from the 2007 AAAI Computer Poker Competition

GS3 competed in the 2007 AAAI Computer Poker Competition. Table 6.4 shows how *GS3* fared against each of the competitors in the Equilibrium Competition.[11] *GS3* finished in third place out of 16 entries, beating all but two of the entries in head-to-head competition. *GS3*'s average winnings was second overall in the Equilibrium Competition.

Opponent	**Amount won by *GS3* per hand ± std. dev.**
Hyperborean07LimitEq1	-0.0320 ± 0.0030
IanBotLimit1	-0.0040 ± 0.0030
PokeMinnLimit1	0.1499 ± 0.0127
QuickLimit1	0.0729 ± 0.0069
GomelLimit2	0.1124 ± 0.0111
DumboLimitEq1	0.1597 ± 0.0084
DumboLimitEq2	0.1488 ± 0.0085
SequelLimit1	0.1396 ± 0.0072
SequelLimit2	0.1480 ± 0.0120
PokeMinnLimit2	0.1540 ± 0.0112
UNCCLimit1	0.4670 ± 0.0091
GomelLimit1	0.1065 ± 0.0104
LeRenardLimit1	0.1418 ± 0.0127
MonashBPPLimit1	0.4119 ± 0.0178
MilanoLimitEq1	0.4448 ± 0.0082
Average	0.1754 ± 0.0044

Table 6.4: Amount won by *GS3* against each of the competitors in the 2007 AAAI Computer Poker Competition, along with standard deviations.

[11]The full results of the Equilibrium Competition are available on-line at `http://www.cs.ualberta.ca/~pokert/2007/results/summarylimiteq.html`.

6.6 Strategy-based abstraction and results from the 2008 AAAI Computer Poker Competition

Our entry in the 2008 AAAI Computer Poker Competition, *GS4*, was similar to *GS3*. The main difference was the use of *strategy-based abstraction* in which information abstraction is performed based on an expectation of different hands being played similarly in equilibrium.

In the construction of *GS4*, we first solved a truncated three-round game that contained a lossless abstraction of 169 buckets for the first betting round, 6,760 buckets for the second betting round, and 54,080 for the third betting round. We used a simple fixed strategy for play on the fourth round to compute the payoffs. We applied our implementation of Nesterov's excessive gap technique to find an approximate solution to this problem over 90 days. Using this solution, we then performed a bucketing of the pre-flop hands using the L_1 distance between strategy vectors as the similarity metric. The rest of the rounds were abstracted using the potential-aware automated abstraction algorithm just as was done for *GS3*. After again applying our equilibrium-finding algorithm, we were able to compute near-optimal strategies for *GS4*.

The results for *GS4* from the 2008 AAAI Computer Poker Competition are in Table 6.5.[12] Overall, *GS4* had the highest win rate in terms of dollars won. However, it had only 4 wins and 4 losses when compared pairwise to the other programs.

6.7 Evaluation of information-based abstraction algorithms

As we have described so far in this section, automated information abstraction algorithms for sequential imperfect information games are a key component in developing competitive game theory-based agents. However, prior research and the experiments presented in this chapter so far did not investigate the relative performance of the different abstraction algorithms. Instead, agents have only been compared under confounding effects such as different granularities of abstraction and equilibrium-finding algorithms that yield different accuracies when solving the abstracted game.

In this subsection we provide a systematic evaluation of the information abstraction algorithms. Is one of them better than another? Does the answer depend on the granularity

[12]The full results of the Equilibrium Competition are available on-line at `http://www.cs.ualberta.ca/~pokert/2008/results/`.

Opponent	**Amount won by *GS3* per hand ± std. dev.**
Hyperborean08-Online	-0.030 ± 0.012
Hyperborean08-Equilibrium	-0.029 ± 0.009
Ian Fellows	-0.020 ± 0.015
GGValuta	-0.036 ± 0.009
PokeMinnLimit2	0.200 ± 0.018
PokeMinnLimit1	0.113 ± 0.016
GUS	0.695 ± 0.018
Dr. Sahbak	0.713 ± 0.007
Average	0.201 ± 0.007

Table 6.5: Amount won by *GS4* against each of the competitors in the 2008 AAAI Computer Poker Competition, along with standard deviations.

of the abstraction that is acceptable in the output (in practice this is constrained by the scalability of the equilibrium-finding algorithm that will take the abstracted game as input)? Furthermore, does the answer depend on whether the agent competes against another agent developed using abstraction, against equilibrium play, or against its nemesis?

Rhode Island Hold'em is a two-person zero-sum game. Thus, the equilibrium problem can be formulated as a linear program whose size is linear in the size of the game tree [130, 84, 160]. Although solving these linear programs becomes difficult in practice for large games, the scalability is adequate for the abstracted games (even losslessly abstracted ones) we discuss in this section. Hence, in our experimental evaluations, we test using optimal strategies for the abstracted games (rather than the approximately optimal strategies guaranteed by other equilibrium-finding approaches, *e.g.,* [56, 165, 166, 106, 57]). The tractability of finding optimal strategies under all granularities of abstraction is important because it allows us to isolate the performance of the abstraction from the performance of the equilibrium-finding algorithm (because the latter is exact here). This is the reason we use Rhode Island Hold'em—which is just under the threshold of what is solvable for equilibrium exactly—as the testbed, instead of, say, Texas Hold'em.

We thus conduct experiments on Rhode Island Hold'em poker where all the confounding effects can be controlled. We compare the algorithms against each other, against optimal play, and against each agent's nemesis. We also compare them based on the resulting game's value. Interestingly, for very coarse abstractions the expectation-based algorithms are better, but for moderately coarse and fine abstractions the potential-aware approach is superior. Furthermore, agents constructed using the expectation-based approaches are highly exploitable beyond what their performance against the game's optimal strategy would suggest.

In our experiments, we fix the number of first-round buckets to be $K_1 = 13$. This value allows for an optimal bucketing at that level; it can be obtained by simply applying *suit isomorphisms*. Since each agent only has one card and there are 13 different ranks, we can find the optimal 13 buckets by simply grouping the hands according to rank and ignoring the suit. Determining buckets for the second and third rounds is thus the main work of the abstraction algorithms.

One idea that has appeared in the literature is to consider the *square* of the probability of winning, rather than simply the probability [166]. The motivation is that the high-value hands (those with the higher probability of winning) should be more finely abstracted than the lower-value hands. Another motivating factor behind this approach is that it captures some of the variance in poker hands. Thus, we consider a simple variant of the expectation-based algorithm that instead uses the square of the probability of winning. To be precise, we take the value of hand i to be $v_i = (w_i + d/2)^2$. We refer to this algorithm as *expectation squared*.[13]

In the rest of this subsection we present our experimental results comparing the expectation-based (both the probability of winning and probability of winning squared variants) and potential-aware abstraction algorithms while varying the granularity of the abstractions. We denote an abstraction with K_1 first-round buckets, K_2 second-round buckets, and K_3 third-round buckets with the string K_1-K_2-K_3. For example, the abstraction granularity 13-25-125 has 13 first-round buckets, 25 second-round buckets, and 125 third-round buckets. The abstraction granularities we consider range from coarse (13-25-125) to fine (13-205-1774). At this fine granularity an equilibrium-preserving abstraction exists [61].

For two-person zero-sum sequential imperfect information games with perfect recall, the equilibrium problem is to find a solution to

$$\max_{x \in Q_1} \min_{y \in Q_2} x^{\mathrm{T}} A y = \min_{y \in Q_2} \max_{x \in Q_1} x^{\mathrm{T}} A y,$$

where Q_i is the set of *realization plans* for agent i and A is the payoff matrix. The sets of realization plans are derived from the sequence form representation of the game [84, 130,

[13]The resulting modified algorithm is slightly different than the one that has appeared previously [166] since we are still using an integer program to allocate the buckets, which enables non-uniform bucketing. The prior approach used fixed-size buckets.

160]. In our experiments, all payoff calculations are carried out exactly. This makes all of our results statistically significant since there is no variance in the expectation computation.

Compared to the computation time for finding an equilibrium, the time needed by the abstraction algorithms was insignificant, although the potential-aware algorithm takes slightly longer than the expectation-based algorithm since the former is doing clustering in the higher-dimensional space of histograms. We used CPLEX's interior-point linear programming solver for finding an exact equilibrium (up to numerical precision) of each of the abstracted games.

In Rhode Island Hold'em, agent 2 has an advantage. (Intuitively, she has more information when it is her turn to act.) Thus, to properly evaluate an agent's performance, it is necessary to consider that agent playing as both agent 1 and as agent 2 in turn. Where it is relevant, in the presentation of our results we average the performance of an agent playing each of the two roles.

We compared the agents constructed with the different information-based abstraction algorithms using four different evaluation criteria. The following four subsections discuss each of these.

6.7.1 Comparing agents head-to-head

The first criterion we use to compare the algorithms is to simply play the resulting strategies against each other in the full, unabstracted version of Rhode Island Hold'em. Formally, if (x_1, y_1) is the pair of strategies computed by the expectation-based algorithm and (x_2, y_2) is the pair of strategies computed by the potential-aware algorithm, the expected payoff to the expectation-based agent is

$$\frac{1}{2}x_1^{\mathrm{T}}Ay_2 - \frac{1}{2}x_2^{\mathrm{T}}Ay_1.$$

As can be seen in Table 6.6, the potential-aware algorithm beats both expectation-based algorithms for the abstraction granularities 13-50-250 and finer. However, interestingly, there is a cross-over: the expectation-based algorithms beat the potential-aware algorithm for the coarsest granularity 13-25-125. One hypothesis for why this is the case is that the dimensionality of the temporary states used in the bottom-up pass in the third-round (which must be smaller than the number of available second-round buckets in order for the clustering to discover meaningful centroids) is insufficient for capturing the strategically relevant aspects of the game. Another hypothesis is that since the potential-aware approach is trying to learn a more complex model (in a sense, clusters of paths of states) and the expectation-based model is trying to learn a less complex model (clusters of states, based on mere probability of winning), the former requires a larger dimension to capture this richness.

Granularity	**EB** vs. **EB**^{2}	**EB vs. PA**	**EB**^{2} **vs. PA**
13-25-125	0.1490	16.6223	17.0938
13-50-250	-0.1272	-1.0627	-0.5200
13-75-500	0.1787	-8.5784	-8.4891
13-100-750	0.2340	-6.9880	-7.1448
13-125-1000	0.1713	-6.4130	-6.3567
13-150-1250	0.1813	-5.5707	-5.6879
13-205-1774	0.0000	-0.0877	-0.0877

Table 6.6: Head-to-head comparison of the algorithms against one another.

The expectation-based algorithms eventually perform almost as well as the potential-aware algorithm when the abstraction granularity is 13-205-1774. The fact that the expectation-based algorithms still lose shows that those algorithms are unable to find the optimal abstraction even when enough buckets are allowed so that a lossless abstraction exists. (In fact, at that granularity, expectation-based abstraction is only able to distinguish between 204 second-round buckets and 1748 third-round buckets. This immediately implies that increasing the number of allowed buckets further does not improve its performance.) It is also interesting that after the cross-over the advantage of the potential-aware approach first increases and then decreases as both approaches get closer to optimal play as the allowed abstraction granularity gets finer and finer.

Comparing probability of winning versus probability of winning squared, we see that they perform similarly. Interesting, one does not dominate the other. For granularity 13-25-125, EB beats EB^2 head-to-head, but EB^2 does better against PA than EB does. For granularity 13-50-250, EB^2 beats EB head-to-head, and EB^2 does better against PA than EB. For the other granularities, EB outperforms EB^2 in both categories.

6.7.2 Comparing agents against equilibrium play

The second evaluation criterion is to play each of the abstractions against the optimal strategy (*i.e.*, equilibrium strategy) of the unabstracted game. If (x, y) is the pair of strategies computed by one of the abstraction algorithms and (x_*, y_*) is the optimal pair of strategies for the unabstracted game, the expected payoff to the abstracted agent is

$$\frac{1}{2}x^{\mathrm{T}}Ay_{*} - \frac{1}{2}x_{*}^{\mathrm{T}}Ay.$$

The results in Table 6.7 show that, as expected, the algorithms improve monotonically against the optimal strategy as finer-grained abstractions are allowed. Furthermore, the potential-aware algorithm has a better payoff than the expectation-based algorithm for granularity 13-50-250 and finer, and is *optimal* for the 13-205-1774 granularity, indicating that the potential-aware algorithm finds the *lossless* abstraction. (In fact, we verified that it finds the same abstraction as the *GameShrink* algorithm that finds lossless abstractions [61].) In contrast, we see that the expectation-based algorithm never finds a lossless abstraction, regardless of how fine an abstraction we allow it to make. This is due to the fact that sometimes two game states have exactly the same probability of winning, yet should be played differently.

Granularity	**EB Payoff**	**EB^2 Payoff**	**PA Payoff**
13-25-125	-25.0312	-25.0807	-41.7910
13-50-250	-19.6519	-19.2139	-18.2612
13-75-500	-15.4708	-15.3714	-10.5562
13-100-750	-11.9801	-12.0462	-5.4248
13-125-1000	-9.3718	-9.3465	-3.2392
13-150-1250	-6.8172	-6.9536	-1.5770
13-205-1774	-0.0877	-0.0877	0.0000

Table 6.7: Expected payoff to each agent when playing against the optimal agent.

6.7.3 Comparing agents against their nemeses: Worst-case performance

The third criterion examines the expected worst-case performance of the algorithms. This is done by computing a *best response strategy*—*i.e.*, a *nemesis*—for each of the two strategies, and then playing each strategy against its nemesis. If (x, y) is the strategy pair computed by one of the abstraction algorithms, the expected worst-case payoff is

$$\frac{1}{2}\min_{v \in Q_2} x^{\mathrm{T}}Av + \frac{1}{2}\max_{u \in Q_1} u^{\mathrm{T}}Ay.$$

Table 6.8 shows that the performance guarantees of each of the algorithms improve as finer abstraction is allowed. Again, the potential-aware algorithm outperforms the expectation-based algorithms for abstraction granularities 13-50-250 and finer, but the expectation-based algorithm provides a better bound for the coarsest granularity. The expectation-based algorithm using winning probability does better than the variant using winning probability squared.

Granularity	**EB Payoff**	**EB^2 Payoff**	**PA Payoff**
13-25-125	-160.527	-163.527	-204.022
13-50-250	-134.406	-134.946	-125.972
13-75-500	-97.970	-99.145	-68.238
13-100-750	-68.888	-72.218	-45.124
13-125-1000	-44.761	-47.334	-24.200
13-150-1250	-29.556	-31.212	-12.067
13-205-1774	-0.429	-0.429	$>$ -0.001

Table 6.8: Expected payoffs to various agents when playing againt their nemesis.

The fact that the payoff to PA is not exactly zero in the 13-205-1774 case is due to numerical rounding issues. Interior-point methods, such as the one used in the experiments, are not exact. An exact algorithm would require rational arithmetic (instead of floating-point arithmetic) which would be completely impractical.

6.7.4 Evaluating abstractions based on estimating the value of the game

Our fourth and final criterion evaluates the abstraction algorithms based on their ability to predict the value of the game. (Computing an accurate value may be useful, for example, to determine the fairness of the game.) If (x, y) is the strategy pair computed by one of the abstraction algorithms, the value estimate is simply given by $x^{\mathrm{T}}Ay$. Table 6.9 displays the results.

The expectation-based algorithms provide a better estimate of the value for the coarsest abstraction granularity, but the potential-aware algorithm provides a better estimate for all finer granularities. The error in the estimations of both algorithms decreases monotonically as the granularity increases, but, interestingly, the estimated values do not change monotonically.

	EB		**EB^2**		**PA**	
Granularity	**Value**	**Error**	**Value**	**Error**	**Value**	**Error**
13-25-125	-68.8710	4.6723	-69.1463	4.9476	-56.9853	7.2134
13-50-250	-59.6962	4.5024	-59.5952	4.6035	-62.2535	1.9452
13-75-500	-60.1006	4.0981	-59.9381	4.2606	-59.8614	4.3373
13-100-750	-60.3524	3.8462	-60.3941	3.8046	-63.8007	0.3980
13-125-1000	-61.6250	2.5737	-61.6343	2.5644	-63.9787	0.2200
13-150-1250	-61.8371	2.3615	-61.8717	2.3270	-64.0957	0.1030
13-205-1774	-64.2361	0.0374	-64.2361	0.0374	-64.1987	0.0000

Table 6.9: Comparison of the estimate of the value of the game given by the algorithms. The estimated value and distance from the actual value (error) of the payoff estimated by each agent are shown.

6.7.5 Summary of experiments comparing information abstraction algorithms

We performed a systematic comparison of automated abstraction algorithms for sequential imperfect information games. We examined three algorithms for information-based abstraction: expectation-based (EB), expectation squared (EB^2), and potential-aware (PA). Our experiments, conducted using the game of Rhode Island Hold'em poker—in order to isolate the abstraction issues from confounding effects—examined four criteria. The results were consistent across all four.

For extremely coarse abstractions, EB and EB^2 outperformed PA. As the abstraction becomes finer, PA becomes best, and is optimal if a fine enough abstraction is allowed.

Interestingly, agents generated using EB are substantially more exploitable (by a nemesis and by an optimal equilibrium-based player) than the head-to-head comparisons against potential-aware abstraction would suggest. EB^2 is even worse in this sense.

6.8 Conclusions and future research

In this chapter we investigated lossy information-based abstraction, which filters the information revealed to the players. We developed four increasingly sophisticated and effective algorithms that take into account strategically-relevant aspects of the game, and we performed a controlled experiment to pinpoint where the different algorithms demonstrate the best performance.

There are numerous directions for future research. The abstraction algorithms presented in this chapter are non-monotonic in the sense that increasing the fineness of the granularity of the abstraction does not necessarily lead to an increase in the quality of the solution. This was also observed in a recent paper [162]. One interesting line of research would be to investigate the capability of designing a monotonic abstraction algorithm, one that would guarantee that the quality of the resulting strategies improve as the abstraction becomes larger. Alternatively, it may be possible that the complexity of such an algorithm is as difficult as the equilibrium-finding problem itself.

Another related direction for future research concerns the development of abstraction algorithms with *ex ante* guarantees which would take the form of a user specifying a optimality tolerance, and the abstraction algorithm would output an abstraction that yielded strategies within that tolerance. Again, it may be the case that the complexity of such an algorithm is as difficult as the equilibrium-finding problem itself.

Chapter 7

Stage and Action Abstraction

7.1 Introduction

In this chapter we examine two more families of abstraction: *stage abstraction*, discussed in Section 7.2, and *action abstraction*, discussed in Sections 7.3 and 7.4.

7.2 Stage abstraction

Another type of abstraction that has been found to be useful is *stage abstraction*. This type of abstraction is potentially useful when the game is so large that performing the necessary amount of information abstraction to make equilibrium computation feasible would be so severe that the resulting strategies would perform very poorly.

A straightforward method of stage abstraction simply solves two halves of the game separately. The payoffs of the first half are computed assuming some "default" behavior for the rest of the game. For example, in both *Sparbot* [14] and *GS1* (Section 6.3), the payoffs that are computed for the leaf nodes at the end of the truncated game (Phase 1) are based on the betting history leading to that node and the expected value of winning that hand assuming that no more betting takes place in later rounds (*i.e.* the payoffs are averaged over the possible fourth-round cards drawn uniformly at random, and assuming that neither player bets in the final round). The second half of the game is solved using beliefs about imperfect information updated (according to Bayes' rule) based on the strategies that are computed for the first half. This simple method does not allow for the strategy analysis of the first half of the game to take into consideration the strategic situations that exist in the second half. This will clearly lead to sub-optimal strategies.

In this section, we present an improved technique for estimating the payoffs of the truncated game by simulating the actions in the remaining portion of the game. This allows the equilibrium-finding algorithm to take into account the entire game tree while having to explicitly solve only a truncated version. This section covers the idea in the context of Texas Hold'em.

Instead of ignoring the fourth-round betting, we in effect incorporate it into the truncated game tree by simulating the betting actions for the fourth round (using reasonable fixed randomized strategies for the fourth round), and then using these payoffs as the payoffs in the truncated game. This is intended to mitigate the negative effects of performing an equilibrium analysis on a truncated game.

At the beginning of the fourth round, each player has two hole cards and there are five community cards on the table. Letting (w, l, d) be the number of possible wins, losses, and draws for a player in that situation, we compute the hand's value using the formula $w - l + d/2$. For hands in the fourth round, this gives a value in the interval $[-990, 990]$. Using the history from 343,513 games of *Sparbot* in self-play (of which 187,850 went to the fourth round), we determined the probability of performing each possible action at each player's turn to act as a function of the hand value.[1] To illustrate this, Figures 7.1–7.3 show these smoothed probabilities for three particular points to act.

Of course, since these probabilities are only conditioned on the hand value (and ignore the betting history in the first three rounds), they do not exactly capture the strategy used by *Sparbot* in the fourth round. However, conditioning the probabilities on the betting history as well would have required a vast number of additional trials, as well as much more space to store the result. Conditioning the actions on hand value alone is a practical way of striking that trade-off.

We presented experimental results for this technique in Table 6.1. The addition of this technique led to a statistically significant improvement in our player.

[1]To mitigate the fact that 187,850 training examples do not densely cover all possible hand values, we bucketed hand values by rounding them to the nearest multiple of 25, and then smoothed by replacing each bucket's value by the average of it and both of its neighbors.

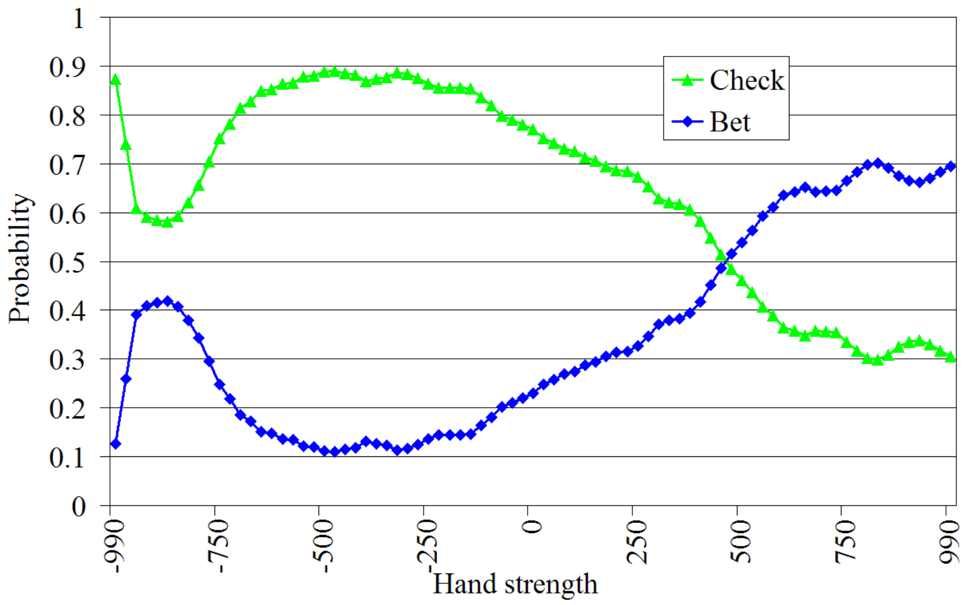

Figure 7.1: First player's empirical action probabilities (for checking and betting) as a function of hand strength at the beginning of the fourth betting round.

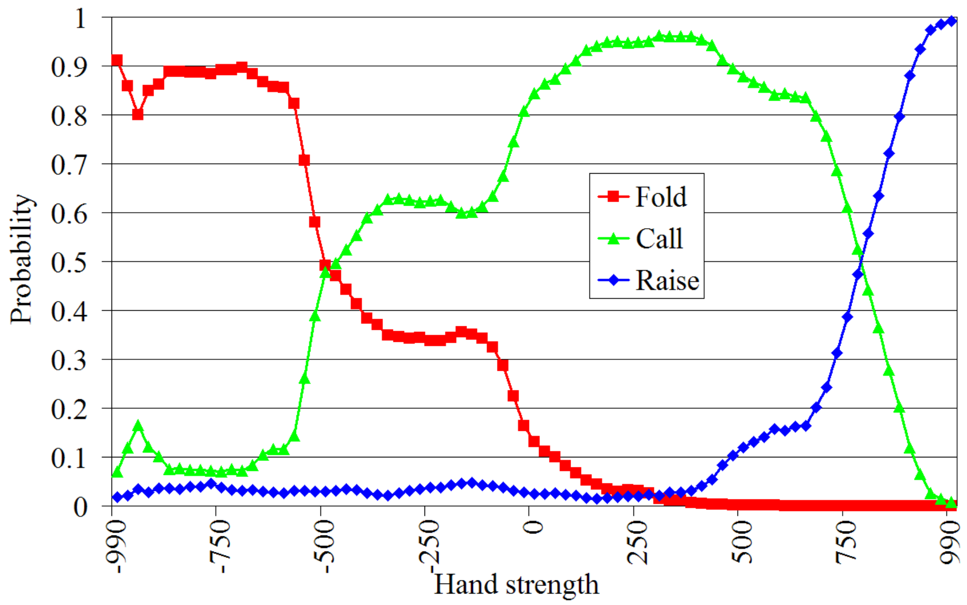

Figure 7.2: Second player's empirical action probabilities (for folding, calling, and raising) as a function of hand strength for the fourth betting round after the first player has bet.

7.3 Action abstraction in limit poker

In *action abstraction*, different actions in the original game are treated as a single action in the abstracted game. This type of abstraction is particularly useful in games where each

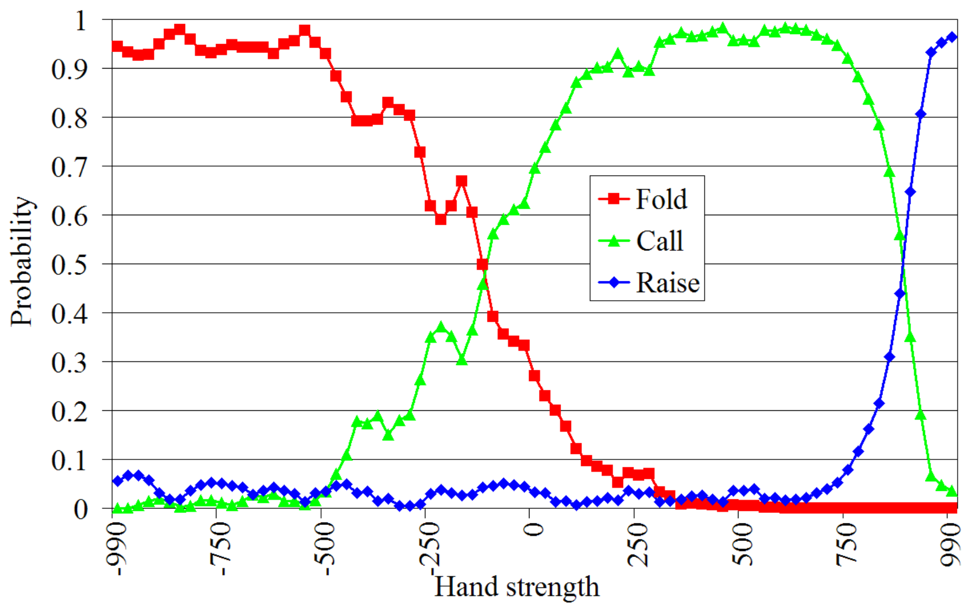

Figure 7.3: First player's empirical action probabilities (for folding, calling, and raising) as a function of hand strength for the fourth betting round after the first player has bet and the second player has raised.

player has a large—or even infinite—number of actions at each information set. In the case of limit poker, we evaluate an action abstraction technique that simply limits the number of bets a player may make in a betting round (Section 7.3). In no-limit poker, the space of betting actions is much richer, and a more sophisticated approach is needed. We describe a betting model and associated reverse mapping in Section 7.4. We also describe *Tartanian*, our game theory-based player for heads-up no-limit Texas Hold'em which uses this action abstraction technique, in Section 7.4.3.

In this section, we examine the effects of a particular form of action-based abstraction. In standard Rhode Island Hold'em, there is a limit of 3 bets in a single betting round. For a given information abstraction granularity, we can obtain a much smaller, and hence much easier to solve, equilibrium problem if we instead limit the agents to at most 1 or 2 bets in each round.

In limit Texas Hold'em, there is a limit of 4 bets in a single betting round. In the developement of prior game theory-based agents for Texas Hold'em, one common technique for decreasing the size of the game tree has been to consider only 3 bets instead of 4 [14, 59, 60]. The experiments in this section are designed to examine the effects of this type of abstraction.

In our experiment for measuring the effects of limiting the number of bets in the

	Payoff Against Equilibrium			**Payoff Against Nemesis**		
Granularity	**1 Bet**	**2 Bets**	**3 Bets**	**1 Bet**	**2 Bets**	**3 Bets**
13-25-125	-73.816	-44.154	-41.791	-359.566	-201.750	-204.022
13-50-250	-52.004	-22.162	-18.261	-365.047	-137.742	-125.972
13-75-500	-44.320	-15.324	-10.556	-310.879	-83.411	-68.238
13-100-750	-41.400	-9.396	-5.425	-312.997	-60.067	-45.124
13-125-1000	-39.532	-7.560	-3.239	-301.066	-44.699	-24.200
13-150-1250	-38.532	-5.806	-1.577	-298.511	-35.214	-12.067
13-205-1774	-37.311	-4.409	0.000	-293.850	-26.160	>-0.001

Table 7.1: Performance of potential-aware abstraction against an optimal strategy and against each agent's nemesis for varying numbers of bets. The first column gives the abstraction granularity used by the agent.

abstracted game, we computed equilibrium strategies for abstractions of Rhode Island Hold'em poker. Each abstracted game used potential-aware abstraction for information-based abstraction and the number of bets was limited to 1, 2, or 3. Table 7.1 displays the results.

Unsurprisingly, having the actual number of bets (3) in the action model is almost always better than having a fewer number of bets (1 or 2). (The one exception discovered in the experiments is the case where an agent has the extremely coarse-grained 13-25-125 abstraction and could have 2 versus 3 bets. This exception only holds under the method of comparison based on performance against the nemesis.) What is interesting is that the relative performance of the the agents with different numbers of bets becomes more important as the information abstraction becomes finer-grained. This is true both for measuring the payoff against an equilibrium strategy in the unabstracted game as well as for measuring the payoff against the agent's nemesis. This suggests that the action abstraction becomes increasingly important (in relative terms) as the information abstraction becomes finer grained.

Another point of interest is that there is a tradeoff between action abstraction and information abstraction. For example, given the choice between having an optimal (lossless) information abstraction and an action abstraction model that contains an additional bet but a worse information abstraction (13-25-125), it was almost always better to have the optimal information abstraction, at least with respect to measuring performance against an agent playing an equilibrium strategy. The one exception to this is the case in which an agent is using an optimal information abstraction (13-205-1774) and 1 bet in the action model versus the case of the coarse 13-25-125 information abstraction and 2 bets in the action model. When measuring performance using the payoff against a nemesis, it is better to have the coarser-grained information abstraction but with 2 bets in the action model.

7.4 Action abstraction in no-limit poker

The most immediate difficulty encountered when moving from limit to no-limit Texas Hold'em is in the development of a betting model. In limit Texas Hold'em, the players only ever have at most three possible actions available to them (fold, call, or raise). This small branching factor in the action sequences allows the model builder to include all possible actions in the model of the game used for the equilibrium analysis.[2]

In no-limit Texas Hold'em, on the other hand, the number of actions available to the players can be huge. For example, when the small blind makes his first action, he can fold, call, or raise to any (integral) amount between 4 and 1000, for a total of 999 possible actions. (If the bets were not limited to be integral amounts then the branching factor would actually be infinite.) Information sets (decision points) with high degree occur elsewhere in the game tree as well. Even if bets are limited to integers, the size of the unabstracted game tree of no-limit heads-up Texas Hold'em where each player has 1000 chips is approximately 10^{71} nodes, compared to "only" 10^{18} nodes in the limit variant.

In the remainder of this section, we discuss the design of our discretized betting model. This consists of two pieces: the choice of which bet amounts we will allow in our model (Section 7.4.1) and the mapping of actions in the real game back to actions in our abstracted game (Section 7.4.2).

7.4.1 Betting model

Although there are potentially a huge number of actions available to a player at most points of the game, in practice among human players, a few bets occur much more frequently than others. These include bets equal to half of the size of the current pot, bets equal to the size of the current pot, and *all-in* bets. We discuss each of these in turn.

[2]Of course an abstraction of the playing cards is still necessary in models of limit Texas Hold'em intended for equilibrium analysis.

- Bets equal to half of the size of the current pot are good *value bets*[3] as well as good *bluffs*. When a player has a strong hand, by placing a half-pot bet he is giving the opponent 3:1 *pot odds*.[4] For example, if a half-pot bet is placed on the river, then the opponent only needs to think that he has a 25% chance of winning in order for a call to be "correct". This makes it a good value bet for the opponent who has a good hand.

  Half-pot bets also make good bluffs: they only need to work one time in three in order for it to be a profitable play. This bet size is advocated by many poker experts as a good-size bet for bluffing [71].

- Bets equal to the size of the current pot are useful when a player believes that he is currently "in the lead", and does not wish to give the opponent a chance to draw out to a better hand (via the additional cards dealt later on in the hand). By placing a pot bet, the player is taking away the odds that the opponent would need to rationally call the bet—with almost any drawing hand, that is, a hand that is not good currently, but has the potential to improve with additional cards. (Half-pot bets are also good for this purpose in some situations.) It is usually not necessary to bet more than this amount.

  Pot bets are particularly useful pre-flop when the big blind, who will be *out of position* (i.e., acting first) in later betting rounds, wishes to make it more expensive for the small blind to play a particular hand.

- In most situations it is a bad idea to go all-in because if the opponent makes the call, he most likely has the better hand, and if he does not make the call, then nothing (or very little) is gained. However, this is a commonly used move (particularly by beginners). In some situations where the pot is large relative to the players' remaining chips, it makes more sense to employ the all-in move.

  Another good reason for including the all-in bet in the model is that it provides a level of robustness in the model. This aspect will be discussed further in Section 7.4.2.

There are also a few bets that are particularly poor or redundant actions, and therefore we do not include them in our betting model in order to keep it relatively small, thus gaining

[3]A bet is considered a *value bet* if the player placing the bet has a strong hand and aims to bet in a way that will entice the opponent into calling the bet. This increases the size of the pot, thus increasing the amount that the player placing the bet will likely win.

[4]*Pot odds* is the ratio of the current size of the pot to the current amount that a player needs to call. They are often used by human players as a guide for making decisions of whether to call or fold.

computational tractability.

- Making bets that are small relative to the pot are usually a bad idea. When facing such a bet, the opponent has terrific pot odds to make a call. Since the opponent can make the call with almost any hand, not much information about the opponent's hand is revealed. Also, since the bet is so small, it is not of much value to a player with a strong hand.

- Once a player's quantity of remaining chips is small relative to the pot, he is in a situation known as *pot-committed*. When facing a subsequent bet of any size, the player will be facing great pot odds and will almost surely be compelled to call (because he can call with whatever he has left, even if that amount is drastically smaller than the pot). In this sense, a rational player who is pot-committed is basically in the same situation as a player who went all-in already. Thus bets that lead to pot-committed situations are, in a sense, nearly redundant. Therefore, in order to reduce the action space for computational tractability, we advocate not allowing bets that put the player in a pot-committed situation. Similarly, we advocate not allowing bets that put the opponent in a pot-committed situation if he calls.

- In theory, the players could go back and forth several times within a betting round. However, such a sequence rarely occurs in practice. The most common sequences involve just one or two bets. In order to keep the betting model small, we advocate a cap of three bets within a betting round.[5]

Taking all of the above considerations into account, we designed our betting model to allow for the following actions:

1. The players always have the option of going *all-in*.

2. When no bets have been placed within a betting round, the actions available to the acting player are *check*, *bet* half the pot, *bet* the pot, or go *all-in*.

3. After a bet has been placed within a betting round, the actions available to the acting player are *fold*, *call*, *bet* the pot, or go *all-in*.

[5]After we developed our betting model, we observed that allowing an unlimited number of bets (in conjunction with a minimum bet size of half the pot) only increases the size of the betting model by 15%. Therefore, in future versions of our player, we plan to relax this constraint.

4. If at any point a bet of a certain size would commit more than half of a player's stack, that particular bet is removed from the betting model (except for all-in bets).

5. At most three bets (of any size) are allowed within any betting round.

The above model could most likely be improved further, particularly with the incorporation of a much larger body of domain knowledge. However, since our research agenda is that of designing game-independent solving techniques, we avoid that approach where possible. We propose as future research a more systematic automated approach to designing betting abstractions—and more generally, for discretizing action spaces in games.

7.4.2 Reverse mapping

Once the betting model has been specified and an equilibrium analysis has been performed on the game model, there still remains the question of how actions in the real game are mapped into actions in the abstracted game. For example, if the betting model contains half-pot bets and pot bets, how do we handle the situation when the opponent makes a bet of three-fourths of the pot? In this section we discuss several issues that arise in developing this *reverse mapping*, and discuss the different design decisions we made for *Tartanian*.

One idea is to map actions to the nearest possible action in terms of amount contributed to the pot. For example, if the betting model contains half-pot bets and pot bets, and the opponent bets four-fifths of the pot, we can treat this (in our model) as a pot-size bet. (Ties could be broken arbitrarily.) However, this mapping can be subject to exploitation. For example, consider the actions available to the small blind player after the initial blinds have been posted. At this point, the small blind has contributed one chip to the pot and the big blind has contributed two chips. According to our betting model, the options available to the small blind are to fold (adding zero chips), call (one chip), half-pot bet (three chips), pot bet (five chips), or all-in (999 chips). Clearly, there is a huge gap between contributing five chips and 999 chips. Suppose that the opponent in this situation actually contributes 500 chips. In absolute distance, this is closer to the pot bet than it is to the all-in bet. However, the bet is so large relative to the pot that for all practical purposes it would be more suitably treated as an all-in bet. If the opponent knows that we treat it as a five-chip bet, he can exploit us by using the 500-chip bet because we would call that with hands that are too weak.[6]

[6]The experimental results we present for *Tartanian* reflect the performance of a version of our player that used this simplistic mapping rule. There we discuss situations in which this mapping led to weak play.

Another possible way of addressing the interpolation problem would be to use randomization.[7] Suppose an action is played where a player contributes c chips to the pot. Suppose that the closest two actions in the betting model correspond to actions where the player contributes d_1 and d_2 chips, with $d_1 < c < d_2$. We could then randomly select the first action in the betting model with probability $p = 1 - \frac{c-d_1}{d_2-d_1}$ and select the second action with probability $1 - p$. This would help mitigate the above-mentioned example where a 500-chip bet is treated as a pot-size bet. However, this would still result in it being treated as a pot-size bet about half of the time.

Instead of using the absolute distances between bets for determining which actions are "closest", we instead advocate using a relative distance. Again considering the situation where the opponent contributes c chips and the two surrounding actions in the model contribute d_1 and d_2 chips, with $d_1 < c < d_2$, we would then compare the quantities $\frac{c}{d_1}$ and $\frac{d_2}{c}$ and choose the action corresponding to the smallest quantity. In the example where the small blind contributes 500 chips in his first action, the two quantities would be $\frac{500}{5} = 100$ versus $\frac{999}{500} = 1.998$. Hence, according to this metric, our reverse mapping would choose the all-in bet as desired.

7.4.3 *Tartanian*: Our first player for No-Limit Texas Hold'em

In *Tartanian*, we use the same automated abstraction algorithm as we used for *GS3*. The number of buckets we allow for each level are the inputs to the algorithm. We used 10 buckets for the first round, 150 for the second round, 750 for the third round, and 3750 for the fourth round. These numbers were chosen based on estimates of the size of problem that our equilibrium-finding algorithm, described below, could solve to high accuracy in a reasonable amount of time.

Once the discretized betting model and reverse mapping have been designed, and the card abstraction has been computed, we are ready to perform the final step, equilibrium computation. As we did with *GS3*, we use our implementation of Nesterov's *excessive gap technique* algorithm [117], which was recently specialized for two-person zero-sum sequential games of imperfect information [74, 56].

The most performance-critical operation in executing the excessive gap technique algorithm is the matrix-vector product operation. Thus, having custom software developed specifically for this purpose is important for the overall performance of the algorithm. We developed a tool for automatically generating the C++ source code for computing the re-

[7]A similar randomization technique has been proposed previously for mitigating this problem [3].

quired matrix-vector product based on an XML description of the game.

As mentioned above, the most intensive portion of the EGT algorithm is in computing matrix-vector products $\mathbf{x}^{\mathrm{T}}A$ and $A\mathbf{y}$. For small games, or games where the structure of the strategy space is quite simple, the source code for computing this product could be written by hand. For larger, more complicated games, the necessary algorithms for computing the matrix-vector product would in turn be more complicated. Developing this code by hand would be a tedious, difficult task—and it would have to be carried out anew for each game and for each betting discretization.

We can see two alternatives for handling this problem. The first, and most obvious, is to have a tree-like representation of the betting model built in memory. This tree could be built from a description of the game. Then, when the matrix-vector product operation is needed, a general algorithm could traverse this tree structure, performing the necessary computations. However, the performance of this algorithm would suffer some since there is the overhead of traversing the tree.

A second approach, which offers better performance, is to generate the C++ source code automatically for the game at hand. This eliminates the need for a tree-like representation of the betting model. Instead, for each node of the tree we simply have one line of source code which performs the necessary operation.

For this approach to work, we need some way of specifying a betting model. We accomplish this with our *Betting Model Language (BML)*, an XML-based description of all possible betting models for no-limit Texas Hold'em. Listing 7.1 contains a snippet of the BML file used by our player.

The BML file consists of a `<round>` section for each betting round (only parts of the first betting round are shown in Listing 7.1). Within each `<round>`, there are `<decision>` entries and `<leaf>` entries. The `<decision>` entries specify the actions available to each player at any stage of the game, as well as specifying certain indices (given via the `number` key) which are used by the equilibrium-finding algorithm for accessing appropriate entries in the strategy vectors.

The `<leaf>` entries encode the payoffs that occur at terminal sequences of the game. When a `<leaf>` has `type` equal to `'fold'`, then it contains a `payoff` value which specifies the payoff to player 1 in that case. Similarly, when a `<leaf>` has `type` equal to `'showdown'`, then it contains a `potshare` value which specifies the amount of chips that each player has contributed to the pot so far. (Of course, the actual payoffs in show-

```
<bml name='CustomBetting'>
    <round number='1'>
        <decisions>
            <decision player='2' sequence='' parent='-1'>
                <action name='F'  number='0' />
                <action name='C'  number='1' />
                <action name='R1' number='2' />
                <action name='A'  number='3' />
            </decision>
            <decision player='1' sequence='C' parent='-1'>
                <action name='k'  number='0' />
                <action name='r1' number='1' />
                <action name='a'  number='2' />
            </decision>
            <decision player='2' sequence='Ca' parent='1'>
                <action name='F'  number='19' />
                <action name='C'  number='20' />
            </decision>
            <decision player='1' sequence='A' parent='-1'>
                <action name='f'  number='32' />
                <action name='c'  number='33' />
            </decision>
            <!-- other decisions omitted... -->
        </decisions>
        <leaves>
            <leaf seq1='2'  seq2='19' type='fold'
                  sequence='CaF' payoff='2.0' />
            <leaf seq1='2'  seq2='20' type='showdown'
                  sequence='CaC' potshare='1000.0' />
            <leaf seq1='32' seq2='3'  type='fold'
                  sequence='Af'  payoff='-2.0' />
            <leaf seq1='33' seq2='3'  type='showdown'
                  sequence='Ac'  potshare='1000.0' />
            <!-- other leaves omitted... -->
        </leaves>
    </round>
    <!-- other rounds omitted... -->
</bml>
```

Listing 7.1: A snippet of the BML for our first-round betting model. The `r1` action indicates a pot-size bet.

down leaves also depend on the players' cards.)

Listing 7.2 contains a snippet of C++ code produced by our software for translating BML into C++. As can be seen, the code is very efficient as each leaf of the game tree is processed with only a few instructions in one line of code each.

```
void TexasMatrixNoLimit::multvec_helper_round1_fold
      (Vec& x, Vec& b, unsigned int i,
       unsigned int j, double prob) {
         b[i +  2] += x[j + 19] * prob *  2.0; // CaF
         b[i + 32] += x[j +  3] * prob * -2.0; // Af
         /* other payoffs omitted... */
}

void TexasMatrixNoLimit::multvec_helper_round1_showdown
      (Vec& x, Vec& b, unsigned int i, unsigned int j,
       double prob, double win) {
        b[i +  2] += x[j + 20] * win * prob * 1000.0; // CaC
        b[i + 33] += x[j +  3] * win * prob * 1000.0; // Ac
        /* other payoffs omitted... */
}
```

Listing 7.2: A snippet of the automatically-generated C++ code for computing the matrix-vector product.

Tartanian participated in the no-limit category of the 2007 AAAI Computer Poker Competition. Each of the 10 entries played head-to-head matches against the other 9 players in Doyle's no-limit Texas Hold'em poker. Each pair of competitors faced off in 20 *duplicate matches* of 1000 hands each. A duplicate match is one in which every hand is played twice with the same cards, but the players are switched. (Of course, the players' memories are reset so that they do not remember the hands the second time they are played.) This is to mitigate the element of luck inherent in poker since if one player gets a particularly lucky hand, then that will be offset by giving the other player that same good hand.

Table 7.2 summarizes the results.[8] *Tartanian* placed second out of the ten entries. The ranking system used in this competition was *instant runoff bankroll*. In that system, the total number of chips won or lost by each program is compared to all of the others. The entrant that loses the most is eliminated and finishes in last place; this ranking process iterates until there is a single winner.

Once the ranking process had only three remaining entries (*Tartanian*, *BluffBot*, and *Hyperborean*), 280 more duplicates matches were held in order to obtain statistical significance. Based on this total of 300 duplicate matches, *Tartanian* beat *Hyperborean* by 0.133 ± 0.039 small bets, but lost to *BluffBot* by 0.267 ± 0.032.

[8]The full competition results are available on the web at `http://www.cs.ualberta.ca/~pokert/`.

Opponent	**Amount won by *Tartanian* per hand ± std. dev.**
BluffBot20NoLimit1	-0.166 ± 0.074
Hyperborean07NoLimit1	-0.079 ± 0.148
SlideRuleNoLimit1	0.503 ± 0.148
GomelNoLimit1	3.161 ± 0.597
GomelNoLimit2	0.124 ± 0.467
MilanoNoLimit1	1.875 ± 0.377
ManitobaNoLimit1	4.204 ± 0.323
PokeMinnNoLimit1	-42.055 ± 0.606
ManitobaNoLimit2	5.016 ± 0.192
Average	-3.046 ± 0.170

Table 7.2: Amount won by *Tartanian* against each of the competitors in the 2007 AAAI No-Limit Computer Poker Competition, along with standard deviations.

An interesting phenomenon was that *Tartanian's* performance against *PokeMinn* was significantly worse than against any other opponent—despite the fact that *PokeMinn* fared poorly in the competition overall. We manually investigated the hand histories of this match-up and observed that *PokeMinn* had a tendency to place bets that were particularly ill-suited to our discretized betting model. For example, a common bet made by *PokeMinn* was putting in 144 chips pre-flop. As mentioned in Footnote 6, the version of our player in the competition was using the simplistic absolute rounding mapping and so it would treat this as a pot-size bet. However, it actually makes much more sense to treat this as an all-in bet since it is so large relative to the size of the pot. We expect that our improved rounding method based on relative distances, described in Section 7.4.2, will appropriately handle this.

7.4.4 *Tartanian2*: Better action abstraction

Our entry in the 2008 AAAI Computer Poker Competition, *Tartanian2*, differed from *Tartanian* in three ways. The first was that we used a finer-grained abstraction. The second was that we were able to run our equilibrium-finding algorithm longer and thus obtained an ϵ-equilibrium with a smaller ϵ. The third was that we used a slightly different action abstraction scheme. In particular, we modeled the following actions in the game.

1. The players always have the option of going *all-in*.
2. When no bets have been placed within a betting round, the actions available to the acting player are *check*, *bet* half the pot, *bet* the pot, or go *all-in*.
3. After a bet has been placed within a betting round, the actions available to the acting player are *fold*, *call*, *raise* two-thirds the value of the pot, or go *all-in*.
4. If at any point a bet of a certain size would commit more than half of a player's stack, that particular bet is removed from the betting model.
5. Any number of bets are allowed within any betting round.

The performance of *Tartanian2* in the competition is summarized in Table 7.3. Overall, *Tartanian2* won the largest number of chips, although in head-to-head competition it only defeated one of the other three opponents.

	Amount won by *Tartanian2*
Opponent	**per hand $\pm$ std. dev.**
BluffBot 3.0	-0.611 $\pm$ 0.091
Hyperborean08	-0.625 $\pm$ 0.091
Ballarat	5.371 $\pm$ 0.223
Overall	1.378 $\pm$ 0.091

Table 7.3: Amount won by *Tartanian2* against each of the competitors in the 2008 AAAI No-Limit Computer Poker Competition, along with standard deviations.

7.5 Conclusions and future research

We presented *Tartanian*, a game theory-based player for heads-up no-limit Texas Hold'em poker. To handle the huge strategy space of no-limit poker, we created a discretized betting model that attempts to retain the most important actions in the game. This also raised the need for a reverse model. Second, as in some prior approaches to game theory-based poker players, we employed automated abstraction for shrinking the size of the game tree based on identifying strategically similar card situations. Third, we presented a new technique for automatically generating the performance-critical portion of equilibrium-finding code based on data describing the abstracted game. The resulting player is competitive with the best existing computer opponents.

Throughout, we made many design decisions. In this research so far, we have made educated guesses about what good answers are to the many questions. In particular, the design of the discretized betting model (and reverse model) and the choice of the number of buckets for each level of the card abstraction were largely based on our own understanding of the problem. In the future, we would like to automate this decision-making process (and hopefully get better answers). Some concrete paths along these lines would be the development of an automated discretization algorithm for the betting model. This could attempt to incorporate a metric for the amount that is lost by eliminating certain strategies, and use this to guide its decisions as to what strategies are eliminated from the model. Another research direction involves developing a better understanding of the tradeoffs between abstraction size and solution quality. We would also like to understand in a more principled way how to set the number of buckets for the different levels of the abstracted card tree.

Part III

Equilibrium-Finding Algorithms

Chapter 8

Gradient-Based Algorithms for Solving Zero-Sum Games

8.1 Introduction

In this chapter, we present a new gradient-based algorithm for finding an ϵ-equilibrium in two-person zero-sum games. It applies to matrix games as well as sequential games of imperfect information. The Nash equilibrium problem for a two-person zero-sum game can be formulated as a saddle-point problem, which can in turn be cast as a linear program (LP). However, for many interesting instances of games, such as those that arise in real poker, these LPs are enormous and unsolvable via standard algorithms such as the simplex or interior-point methods.

To address this computional challenge, some alternative algorithms have been developed and have been shown to be effective in finding ϵ-equilibria, where neither player can benefit more than ϵ by deviating. These include an algorithm based on regret minimization [166] (which has iteration-complexity $\mathcal{O}\left(1/\epsilon^2\right)$) and iterative bundle-based methods [165, 106].

In this chapter we present algorithms based on Nesterov's [117, 118] first-order smoothing techniques. The main strengths are their simplicity and low computational cost of each iteration. Our first algorithm, based on Nesterov's excessive gap technique [117], finds an ϵ-equilibrium within $\mathcal{O}(1/\epsilon)$ iterations. In contrast, interior-point methods find an ϵ-equilibrium within $\mathcal{O}(\ln(1/\epsilon))$ iterations [163], but do not scale to large games due to memory requirements. With this observation in mind we also present an iterated version of Nesterov's smoothing technique for nonsmooth convex optimization [118] that runs in

$\mathcal{O}(\kappa(A)\ln(1/\epsilon))$ iterations. In terms of ϵ, the iteration complexity is thus the same as that of interior-point methods and exponentially better than that of prior first-order methods. The complexity also depends on a certain condition measure, $\kappa(A)$, of the payoff matrix A. Unlike interior-point methods, we inherit the manageable memory usage of prior first-order methods. So, our algorithm scales to large games and small ϵ.

8.2 First-order methods

Assume $Q \subseteq \mathbf{R}^n$ is a compact convex set and $f : Q \to \mathbf{R}$ is convex. Consider the convex optimization problem

$$\min\{f(x) : x \in Q\}. \tag{8.1}$$

This chapter is concerned with *first-order methods* for solving a particular form of problem (8.1). The defining feature of these methods is that the search direction at each main iteration is obtained using only first-order information, such as the gradient or subgradient of the function f. This feature makes their computational overhead per iteration extremely low, and hence makes them attractive for large-scale problems.

The complexity of first-order methods for finding an approximate solution to (8.1) depends on the properties of f and Q. For the setting where f is differentiable and ∇f, the gradient of f, is Lipschitz[1] and continuous, Nesterov [115] proposed a gradient-based algorithm with convergence rate $\mathcal{O}(1/\sqrt{\epsilon})$. In other words, within $\mathcal{O}(1/\sqrt{\epsilon})$ iterations, the algorithm outputs a value $x \in Q$ such that $f(x) \leq f(x') + \epsilon$ for all $x' \in Q$, including the optimal one. We refer to this algorithm as *Nesterov's optimal gradient algorithm* since it can be shown that for that smooth class of problems, no gradient-based algorithm has asymptotically faster convergence. A variant by Lan, Liu, and Monteiro [94] also features $\mathcal{O}(1/\sqrt{\epsilon})$ convergence and outperformed Nesterov's original algorithm in experiments.

For the setting where f is non-differentiable, *subgradient* algorithms are often used. They have complexity $\Theta(1/\epsilon^2)$ [68]. However, this pessimistic result is based on treating f as a black box, whose value and subgradient are available through an oracle. For a function f with a suitable structure, Nesterov [117, 118] devised a first-order method with convergence rate $\mathcal{O}(1/\epsilon)$.[2] The method is based on a *smoothing* technique. The idea is

[1]Recall that a function f is *Lipschitz with constant* L if $|f(x) - f(y)| \leq L\|x - y\|$ for all x and y in the domain of f.

[2]First-order algorithms have also proven to be effective for finding approximate solutions to large-scale LPs [11] and to large-scale nonlinear convex programs [150]. These approaches use $\mathcal{O}(1/\epsilon^2)$ iterations on non-smooth problems. (For a special class of continuously differentiable minimization problems (which is

that the structure of f can be used to construct a smooth function with Lipschitz gradient that resembles f. Then, a gradient algorithm (for example, Nesterov's optimal gradient algorithm) applied to the smooth function yields an approximate minimizer for f.

8.3 Excessive gap technique

One particular smoothing technique which we adapt in this section is Nesterov's *excessive gap technique (EGT)* [117]. It is a primal-dual gradient algorithm where the primal and dual functions, f and ϕ, are smoothed by adding a strongly convex and concave term to the minimization and maximization expressions that define them. Furthermore, across iterations of the algorithm, the amount of smoothing is reduced so that the smoothed primal and dual functions approach the original ones.

We now describe EGT in detail, specialized to extensive form games. For $i \in \{1, 2\}$, assume that S_i is the set of sequences of moves of player i and $Q_i \subseteq \mathbb{R}^{S_i}$ is the *set of realization plans* of player i [160]. Further, assume that d_i is a strongly convex function on Q_i, that is, there exists $\sigma_i > 0$ such that

$$d_i(\alpha\vec{z} + (1-\alpha)\vec{w}) \ \leq \ \alpha d_i(\vec{z}) + (1-\alpha)d_i(\vec{w}) - \frac{1}{2}\sigma_i\alpha(1-\alpha)\|\vec{z} - \vec{w}\|^2 \qquad (8.2)$$

for all $\alpha \in [0, 1]$ and $\vec{z}, \vec{w} \in Q_i$. The largest σ_i satisfying (8.2) is the *strong convexity parameter* of d_i. Without loss of generality, assume that $\min_{\vec{z}\in Q_i} d_i(\vec{z}) = 0$.

We use the *prox functions* d_1 and d_2 to *smooth* the non-smooth primal and dual functions, f and ϕ, as follows. For $\mu_1, \mu_2 > 0$ consider

$$f_{\mu_2}(\vec{x}) := \max_{\vec{y}\in Q_2} \{\langle A\vec{y}, \vec{x}\rangle + \mu_2 d_2(\vec{y})\}$$

and

$$\phi_{\mu_1}(\vec{y}) := \min_{\vec{x}\in Q_1} \{\langle A\vec{y}, \vec{x}\rangle - \mu_1 d_1(\vec{x})\}.$$

Because d_1 and d_2 are strongly convex, it follows [117] that f_{μ_2} and ϕ_{μ_1} are smooth. Weak duality implies $f(\vec{x}) \leq \phi(\vec{y})$ for all $\vec{x} \in Q_1, \vec{y} \in Q_2$. Consider the following related *excessive gap condition:*

$$f_{\mu_2}(\vec{x}) \geq \phi_{\mu_1}(\vec{y}). \qquad (8.3)$$

very different from our non-differentiable setting) the first-order algorithm presented by Smola *et al.* [150] runs in $\mathcal{O}(\ln(1/\epsilon))$ iterations.)

Let $D_i := \max_{\vec{z} \in Q_i} d_i(\vec{z})$. If $\mu_1, \mu_2 > 0$, $\vec{x} \in Q_1, \vec{y} \in Q_2$ and $(\mu_1, \mu_2, \vec{x}, \vec{y})$ satisfies (8.3), then Lemma 3.1 of [117] guarantees

$$0 \leq \phi_{\mu_1}(\vec{y}) - f_{\mu_2}(\vec{x}) \leq \mu_1 D_1 + \mu_2 D_2. \tag{8.4}$$

This suggests the following strategy to find an approximate solution to (10.1): generate a sequence $(\mu_1^k, \mu_2^k, \vec{x}^k, \vec{y}^k)$, $k = 0, 1, \ldots$, with μ_1^k and μ_2^k decreasing to zero as k increases, while maintaining the loop invariants that $\vec{x}^k \in Q_1$, $\vec{y}^k \in Q_2$, and $(\mu_1^k, \mu_2^k, \vec{x}^k, \vec{y}^k)$ satisfies (8.3). As μ_1^k and μ_2^k approach zero, the quantity $\mu_1 D_1 + \mu_2 D_2$ approaches zero, and thus the strategies $\vec{x}^k$ and $\vec{y}^k$ approach a solution to (10.1).

The building blocks of our algorithms are the mapping sargmax and the procedures `initial` and `shrink`, which we will now define. Let d be a strongly convex function with a convex and compact domain $Q \subseteq \mathbb{R}^n$. We define $\operatorname{sargmax}(d, \cdot) : \mathbb{R}^n \to Q$ as

$$\operatorname{sargmax}(d, \vec{g}) := \operatorname*{argmax}_{\vec{x} \in Q} \{\langle \vec{g}, \vec{x} \rangle - d(\vec{x})\}. \tag{8.5}$$

By Lemma 5.1 of [118], the following procedure `initial` finds a starting point $(\mu_1^0, \mu_2^0, \vec{x}^0, \vec{y}^0)$ that satisfies the excessive gap condition (8.3). The notation $\|A\|$ indicates an appropriate operator norm (see [118] and Examples 1 and 2 for details).

Algorithm 5 `initial`(A, d_1, d_2)

```
1. $\mu_1^0 := \mu_2^0 := \frac{\|A\|}{\sqrt{\sigma_1 \sigma_2}}$
2. $\hat{\vec{y}} := \operatorname{sargmax}\left(d_2, \vec{0}\right)$
3. $\vec{x}^0 := \operatorname{sargmax}\left(d_1, \frac{1}{\mu_1^0} A \hat{\vec{y}}\right)$
4. $\vec{y}^0 := \operatorname{sargmax}\left(d_2, \nabla d_2\left(\hat{\vec{x}}\right) + \frac{1}{\mu_2^0} A^{\mathrm{T}} \vec{x}^0\right)$
5. Return $(\mu_1^0, \mu_2^0, \vec{x}^0, \vec{y}^0)$
```

The following procedure `shrink` enables us to reduce μ_1 and μ_2 while maintaining (8.3).

Algorithm 6 `shrink`$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$

```
1. $\breve{\vec{y}} := \operatorname{sargmax}\left(d_2, -\frac{1}{\mu_2} A^{\mathrm{T}} \vec{x}\right)$
```

```
2. $\hat{\vec{y}} := (1-\tau)\vec{y} + \tau\check{\vec{y}}$
3. $\hat{\vec{x}} := \operatorname{sargmax}\left(d_1, \frac{1}{\mu_1}A\hat{\vec{y}}\right)$
4. $\tilde{\vec{y}} := \operatorname{sargmax}\left(d_2, \nabla d_2\left(\check{\vec{y}}\right) + \frac{\tau}{(1-\tau)\mu_2}A^{\mathrm{T}}\hat{\vec{x}}\right)$
5. $\vec{x}^+ := (1-\tau)\vec{x} + \tau\hat{\vec{x}}$
6. $\vec{y}^+ := (1-\tau)\vec{y} + \tau\tilde{\vec{y}}$
7. $\mu_2^+ := (1-\tau)\mu_2$
8. Return $(\mu_2^+, \vec{x}^+, \vec{y}^+)$
```

By Theorem 4.1 of [117], if the input $(\mu_1, \mu_2, \vec{x}, \vec{y})$ to `shrink` satisfies (8.3) then so does $(\mu_1, \mu_2^+, \vec{x}^+, \vec{y}^+)$ as long as τ satisfies $\tau^2/(1-\tau) \leq \mu_1\mu_2\sigma_1\sigma_2/\|A\|^2$. Consequently, the iterates generated by procedure `EGT` below satisfy (8.3). In particular, after N iterations, Algorithm `EGT` yields points $\vec{x}^N \in Q_1$ and $\vec{y}^N \in Q_2$ with

$$0 \leq \max_{\vec{x}\in Q_1}\langle A\vec{y}^N, \vec{x}\rangle - \min_{\vec{y}\in Q_2}\langle A\vec{y}, \vec{x}^N\rangle \leq \frac{4\,\|A\|}{N}\sqrt{\frac{D_1 D_2}{\sigma_1\sigma_2}}. \qquad (8.6)$$

Algorithm 7 `EGT`(A, d_1, d_2)

```
1. $(\mu_1^0, \mu_2^0, \vec{x}^0, \vec{y}^0) = \texttt{initial}(A, d_1, d_2)$
2. For $k = 0, 1, \ldots$:
    (a) $\tau := \frac{2}{k+3}$
    (b) If $k$ is even:    // shrink $\mu_2$
        i. $(\mu_2^{k+1}, \vec{x}^{k+1}, \vec{y}^{k+1}) := \texttt{shrink}(A, \mu_1^k, \mu_2^k, \tau, \vec{x}^k, \vec{y}^k, d_1, d_2)$
        ii. $\mu_1^{k+1} := \mu_1^k$
    (c) If $k$ is odd:    // shrink $\mu_1$
        i. $(\mu_1^{k+1}, \vec{y}^{k+1}, \vec{x}^{k+1}) := \texttt{shrink}(A^{\mathrm{T}}, -\mu_2^k, -\mu_1^k, \tau, \vec{y}^k, \vec{x}^k, d_2, d_1)$
        ii. $\mu_2^{k+1} := \mu_2^k$
```

Notice that Algorithm `EGT` is a *conceptual* algorithm that finds an ϵ-solution to (10.1). It is only conceptual because it requires computing the mappings $\operatorname{sargmax}(d_i, \cdot)$ (several times at each iteration), and there is no algorithm for solving them for arbitrary prox functions d_i. Consequently, the choice of d_1 and d_2 is critical for implementing Algorithm `EGT`. This will be discussed in the next subsection.

8.3.1 Nice prox functions

Assume Q is a convex and compact set. We say that a function $d : Q \to \mathbb{R}$ is a *nice prox function* for Q if it satisfies the following three conditions:

1. d is strongly convex and continuous everywhere in Q and is differentiable in the relative interior of Q;
2. $\min\{d(\vec{z}) : \vec{z} \in Q\} = 0$;
3. The mapping $\operatorname{sargmax}(d, \cdot)\colon \mathbb{R}^n \to Q$ is easily computable. For example, it has a closed-form expression.

As a building block for prox functions for the space we are interested in, we first review two examples of nice prox functions for the *simplex*

$$\Delta_n := \{\vec{x} \in \mathbb{R}^n : \vec{x} \geq 0, \sum_{i=1}^{n} x_i = 1\}.$$

Example 8.3.1 *Consider the* entropy *function $d(\vec{x}) = \ln n + \sum_{i=1}^{n} x_i \ln x_i$. The function d is strongly convex and continuous in Δ_n and $\min_{\vec{x}\in\Delta_n} d(\vec{x}) = 0$. It is also differentiable in the relative interior of Δ_n. It has strong convexity parameter $\sigma = 1$ for the L_1-norm in $\mathbb{R}^n$, namely, $\|\vec{x}\| = \sum_{i=1}^{n} |x_i|$. The corresponding operator norm, $\|A\|$, for this setting is simply the largest absolute value of any entry in A. Finally, the mapping* $\operatorname{sargmax}(d, \vec{g})$ *has the easily computable expression*

$$\operatorname{sargmax}(d, \vec{g})_j = \frac{e^{g_j}}{\sum\limits_{i=1}^{n} e^{g_i}}.$$

Example 8.3.2 *Consider the (squared)* Euclidean distance *to the center of Δ_n, that is, $d(\vec{x}) = \frac{1}{2}\sum_{i=1}^{n} \left(x_i - \frac{1}{n}\right)^2$. This function is strongly convex, continuous and differentiable in Δ_n, and $\min_{\vec{x}\in\Delta_n} d(\vec{x}) = 0$. It has strong convexity parameter $\sigma = 1$ for the Euclidean (L_2) norm, namely, $\|\vec{x}\| = (\sum_{i=1}^{n} |x_i|^2)^{1/2}$. The corresponding operator norm, $\|A\|$, for this setting is the spectral norm of A, that is, the largest singular value of A. Although the mapping* $\operatorname{sargmax}(d, \vec{g})$ *does not have a closed-form expression, it can easily be computed in $\mathcal{O}(n \log n)$ steps [31].*

In order to apply Algorithm `EGT` to problem (10.1) for sequential games we need nice prox functions for the realization sets Q_1 and Q_2 (which are more complex than the simplex discussed above in Examples 1 and 2). This problem was recently solved by Hoda, Gilpin, and Peña [74]:

Theorem 4 *Any nice prox function ψ for the simplex induces a nice prox function for a set of realization plans Q. The mapping* $\mathrm{sargmax}(d, \cdot)$ *can be computed by repeatedly applying* $\mathrm{sargmax}(\psi, \cdot)$.

8.3.2 Experimental setup

In the experiments of this paper we benchmark on five poker games ranging from relatively small to very large. We chose these problems because we wanted to evaluate the algorithms on real-world instances, rather than on randomly generated games (which may not reflect any realistic setting). Table 8.1 provides the sizes of the test instances. The first three instances, `10k`, `160k`, and `RI`, are abstractions of Rhode Island Hold'em poker [146] computed using the *GameShrink* automated abstraction algorithm [61]. The first two instances are lossy (non-equilibrium preserving) abstractions, while the `RI` instance is a lossless abstraction. The `Texas` and `GS4` instances are lossy abstractions of Texas Hold'em poker.

Name	**Rows**	**Columns**	**Non-Zero Entries**
`10k`	14,590	14,590	536,502
`160k`	226,074	226,074	9,238,993
`RI`	1,237,238	1,237,238	50,428,638
`Texas`	18,536,842	18,536,852	61,498,656,400
`GS4`	299,477,082	299,477,102	4,105,365,178,571

Table 8.1: Problem sizes (when formulated as a linear program) for the instances used in our experiments.

Due to the enormous size of the `GS4` instance, we do not include it in the experiments that compare better and worse techniques within our algorithm. Instead, we use the four smaller instances to find a good configuration of the algorithm, and we use that configuration to tackle the `GS4` instance. We then report on how well those resulting strategies did in the AAAI-08 Computer Poker Competition.

8.3.3 Experimental comparison of prox functions

Our first experiment compared the relative performance of the prox functions induced by the entropy and Euclidean prox functions described in Examples 1 and 2 above. Figure 8.1 shows the results. (Heuristics 1 and 2, described later, and the memory saving technique described later, were enabled in this experiment.) In all of the figures, the units of the vertical axis are the number of chips in the corresponding poker games.

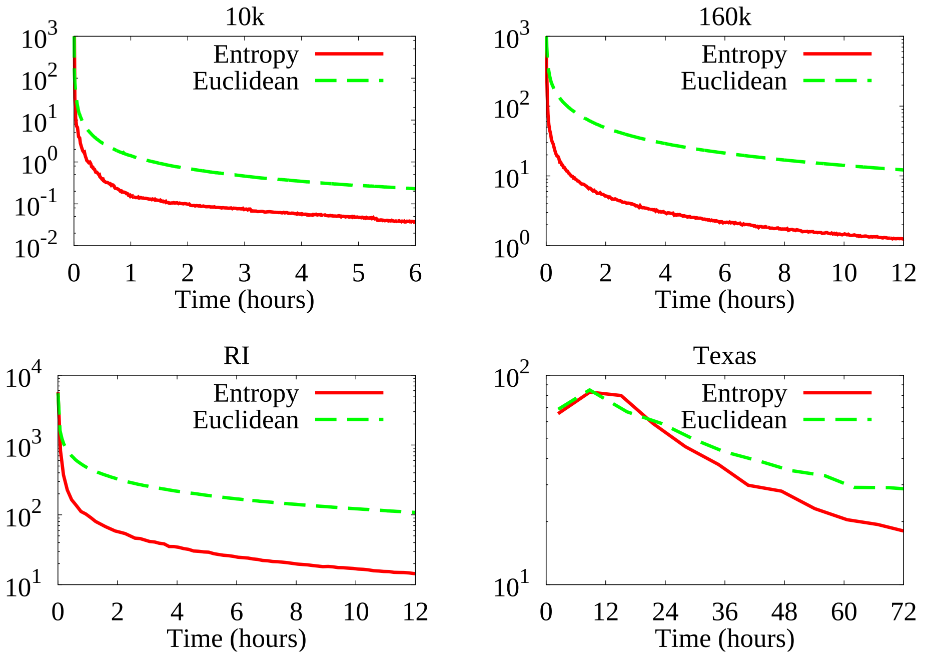

Figure 8.1: Comparison of the entropy and Euclidean prox functions. The value axis is the gap ϵ.

The entropy prox function outperformed the Euclidean prox function on all four instances. Therefore, in the remaining experiments we exclusively use the entropy prox function.

8.3.4 Heuristics for improving speed of convergence

Algorithm `EGT` has worst-case iteration-complexity $\mathcal{O}(1/\epsilon)$ and already scales to problems much larger than is possible to solve using state-of-the-art linear programming solvers (as we demonstrate in the experiments later in this paper). In this section we introduce two heuristics for further improving the speed of the algorithm, while retaining the guaranteed worst-case iteration-complexity $\mathcal{O}(1/\epsilon)$. The heuristics attempt to decrease μ_1 and μ_2 faster than prescribed by the basic algorithm, while maintaining the excessive gap condition (8.3). This leads to overall faster convergence in practice, as our experiments will show.

Heuristic 1: Aggressive μ reduction

The first heuristic is based on the following observation: although the value $\tau = 2/(k+3)$ computed in step 2(a) of Algorithm EGT guarantees the excessive gap condition (8.3), this is potentially an overly conservative value. Instead we can use an adaptive procedure to choose a larger value of τ. Since we now can no longer guarantee the excessive gap condition (8.3) *a priori*, we are required to do a *posterior* verification which occasionally necessitates an adjustment in the parameter τ. In order to check (8.3), we need to compute the values of f_{μ_2} and ϕ_{μ_1}. To that end, consider the following mapping smax, a variation of $\mathrm{sargmax}$. Assume d is a prox function with domain $Q \subseteq \mathbb{R}^n$. We define $\mathrm{smax}(d, \cdot) : \mathbb{R}^n \to \mathbb{R}$ as

$$\mathrm{smax}(d, \vec{g}) := \max_{\vec{x} \in Q}\{\langle \vec{g}, \vec{x}\rangle - d(\vec{x})\}. \tag{8.7}$$

It is immediate that $\mathrm{smax}(d, \cdot)$ is easily computable provided $\mathrm{sargmax}(d, \cdot)$ is. Notice that

$$\phi_{\mu_1}(\vec{y}) = \mu_1 \,\mathrm{smax}(d_1, \frac{1}{\mu_1} A\vec{y})$$

and

$$f_{\mu_2}(\vec{x}) = -\mu_2 \,\mathrm{smax}(d_2, -\frac{1}{\mu_2} A^{\mathrm{T}}\vec{x}).$$

To incorporate Heuristic 1 in Algorithm `EGT` we extend the procedure `shrink` as follows.

Algorithm 8 `decrease`$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$

```
1. $(\mu_2^+, \vec{x}^+, \vec{y}^+) :=$ shrink$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$
2. While $\mu_1 \,\mathrm{smax}(d_1, \frac{1}{\mu_1} A\vec{y}^+) > -\mu_2 \,\mathrm{smax}(d_2, \frac{-1}{\mu_2^+} A^{\mathrm{T}}\vec{x}^+)$    // $\tau$ is too big
    (a) $\tau := \tau/2$
    (b) $(\mu_2^+, \vec{x}^+, \vec{y}^+) :=$ shrink$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$
3. Return $(\mu_2^+, \vec{x}^+, \vec{y}^+, \tau)$
```

By Theorem 4.1 of [117], when the input $(\mu_1, \mu_2, \vec{x}, \vec{y})$ to `decrease` satisfies (8.3), the procedure `decrease` will halt.

Figure 8.2 demonstrates the impact of applying Heuristic 1. (For this experiment, Heuristic 2, described later, was not used. The memory saving technique, also described later, was used.) On all four instances, Heuristic 1 reduced the gap significantly. On the larger instances, this reduction was an order of magnitude.

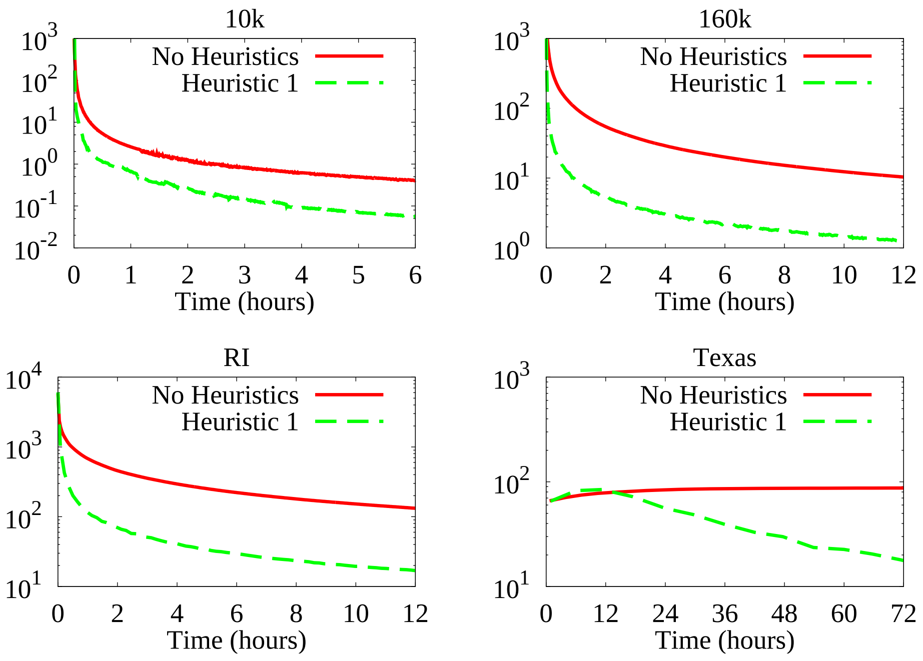

Figure 8.2: Experimental evaluation of Heuristic 1. The value axis is the gap ϵ.

Heuristic 2: Balancing and reduction of μ_1 and μ_2

Our second heuristic is motivated by the observation that after several calls to the `decrease` procedure, one of μ_1 and μ_2 may be much smaller than the other. This imbalance is undesirable because the larger one contributes the most to the worst-case bound given by (8.4). Hence after a certain number of iterations we perform a *balancing* step to bring these values closer together. The balancing consists of repeatedly shrinking the larger one of μ_1 and μ_2.

We also observed that after such balancing, the values of μ_1 and μ_2 can sometimes be further reduced without violating the excessive gap condition (8.3). We thus include a final reduction step in the balancing heuristic.

This balancing and reduction heuristic is incorporated via the following procedure. (We chose the parameter values (0.9 and 1.5) based on some initial experimentation.)

Algorithm 9 `balance`$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$

```
1. While $\mu_2 > 1.5\mu_1$     // shrink $\mu_2$
       $(\mu_2, \vec{x}, \vec{y}, \tau) :=$ decrease$(A, \mu_1, \mu_2, \tau, \vec{x}, \vec{y}, d_1, d_2)$
2. While $\mu_1 > 1.5\mu_2$     // shrink $\mu_1$
```

```
    $(\mu_1, \vec{y}, \vec{x}, \tau) :=$ decrease$(A^{\mathrm{T}}, -\mu_2, -\mu_1, \tau, \vec{y}, \vec{x}, d_2, d_1)$
3. While $0.9\mu_1 \operatorname{smax}(d_1, \frac{1}{0.9\mu_1} A\vec{y}) \le -0.9\mu_2 \operatorname{smax}(d_2, \frac{-1}{0.9\mu_2} A^{\mathrm{T}}\vec{x})$
   // decrease $\mu_1$ and $\mu_2$ if possible
    (a) $\mu_1 := 0.9\mu_1$
    (b) $\mu_2 := 0.9\mu_2$
4. Return $(\mu_1, \mu_2, \vec{x}, \vec{y}, \tau)$
```

We are now ready to describe the variant of EGT with Heuristics 1 and 2.

Algorithm 10 EGT-2

```
1. $(\mu_1^0, \mu_2^0, \vec{x}^0, \vec{y}^0) =$ initial$(A, d_1, d_2)$
2. $\tau := 0.5$
3. For $k = 0, 1, \ldots$:
    (a) If $k$ is even: // Shrink $\mu_2$
        i. $(\mu_2^{k+1}, \vec{x}^{k+1}, \vec{y}^{k+1}, \tau) :=$ decrease$(A, \mu_1^k, \mu_2^k, \tau, \vec{x}^k, \vec{y}^k, d_1, d_2)$
        ii. $\mu_1^{k+1} = \mu_1^k$
    (b) If $k$ is odd: // Shrink $\mu_1$
        i. $(\mu_1^{k+1}, \vec{y}^{k+1}, \vec{x}^{k+1}, \tau) :=$ decrease$(-A^{\mathrm{T}}, \mu_2^k, \mu_1^k, \tau, \vec{y}^k, \vec{x}^k, d_2, d_1)$
        ii. $\mu_2^{k+1} = \mu_2^k$
    (c) If $k \mod 100 = 0$ // balance and reduce
            $(\mu_1^k, \mu_2^k, \vec{x}^k, \vec{y}^k, \tau) :=$ balance$(A, \mu_1^k, \mu_2^k, \tau, \vec{x}^k, \vec{y}^k, d_1, d_2)$
```

Because Heuristic 2 is somewhat expensive to apply, we experimented with how often the algorithm should run it. (We did this by varying the constant in line 3(c) of Algorithm EGT-2. In this experiment, Heuristic 1 was turned off, but the memory-saving technique, described later, was used.) Figure 8.3 shows that it is always effective to use Heuristic 2 at least some, although the optimal frequency at which Heuristic 2 should be applied varies depending on the instance.

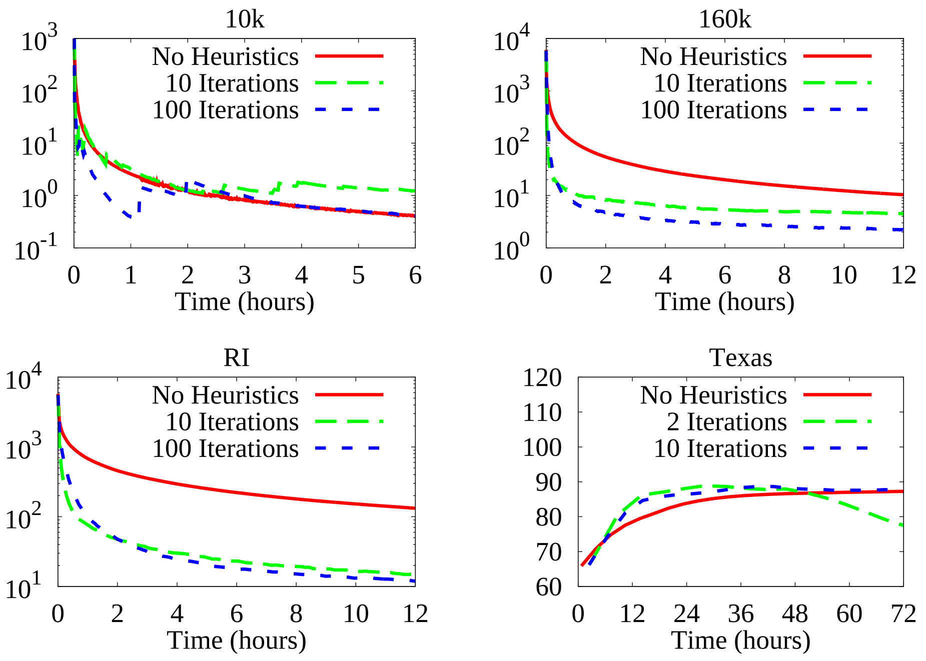

Figure 8.3: Heuristic 2 applied at different intervals. The value axis is the gap ϵ.

Theoretical complexity of excessive gap technique with heuristics (EGT-2)

We next show that Algorithm EGT-2 has the same overall speed of convergence $\mathcal{O}(1/\epsilon)$ of Algorithm EGT. The main ingredient in the proof is the following estimate of the reduction of μ_1 or μ_2 each time `decrease` is called.

Proposition 5 *Each call to* `decrease` *in Algorithm* EGT-2 *reduces μ_i to $\mu_i^+ = (1-\tau)\mu_i$, for $i \in \{1,2\}$ and $\tau \in (0,1)$, so that*

$$\frac{1}{\mu_i^+} \geq \frac{1}{\mu_i} + \frac{\sqrt{\sigma_1\sigma_2}}{4\sqrt{2}\|A\|}. \tag{8.8}$$

PROOF. By symmetry it suffices to prove (8.8) for $i = 1$. Since $\mu_i^+ = (1-\tau)\mu_i$, inequality (8.8) is equivalent to

$$\frac{\tau}{1-\tau} \geq \frac{\sqrt{\sigma_1\sigma_2}}{4\sqrt{2}\|A\|}\mu_1. \tag{8.9}$$

From the construction of `decrease` and Theorem 4.1 of [117] it follows that $\tau \geq \frac{\tau_{\max}}{2}$ where $\tau_{\max} \in (0,1)$ satisfies

$$\frac{\tau_{\max}^2}{1-\tau_{\max}} = \frac{\mu_1\mu_2\sigma_1\sigma_2}{\|A\|^2}. \tag{8.10}$$

Notice that $\frac{\mu_1\mu_2\sigma_1\sigma_2}{\|A\|^2} \leq 1$ because both μ_1 and μ_2 decrease monotonically and initially they are both equal to $\frac{\|A\|}{\sqrt{\sigma_1\sigma_2}}$. Hence from (8.10) it follows that $\tau_{\max} \in (0, 2/3)$. Therefore, since $\tau \geq \frac{\tau_{\max}}{2}$, to show (8.9) it suffices to show

$$\frac{\tau_{\max}}{1-\tau_{\max}} \geq \frac{\sqrt{\sigma_1\sigma_2}}{\sqrt{2}\|A\|}\mu_1. \tag{8.11}$$

By step 3(c) in Algorithm `EGT-2`, the values μ_1 and μ_2 satisfy

$$\frac{\mu_1}{2} \leq \mu_2 \leq 2\mu_1. \tag{8.12}$$

From (8.10) and (8.12) we obtain

$$\frac{\mu_1^2\sigma_1\sigma_2}{2\|A\|^2} \leq \frac{\tau_{\max}^2}{1-\tau_{\max}} \leq \left(\frac{\tau_{\max}}{1-\tau_{\max}}\right)^2, \tag{8.13}$$

and (8.11) follows. □

We now obtain a complexity bound for Algorithm `EGT-2` analogous to the bound (8.6) for Algorithm `EGT`.

Theorem 6 *After N calls to* `decrease`*, Algorithm* `EGT-2` *returns a pair $\vec{x}^N \in Q_1, \vec{y}^N \in Q_2$ satisfying*

$$0 \leq \max_{\vec{x}\in Q_1}\langle A\vec{y}^N, \vec{x}\rangle - \min_{\vec{y}\in Q_2}\langle A\vec{y}, \vec{x}^N\rangle \leq \frac{12\sqrt{2}\|A\|}{N}\frac{(D_1+D_2)}{\sqrt{\sigma_1\sigma_2}}. \tag{8.14}$$

PROOF. From Proposition 5, it follows that after N calls to `decrease`, the iterates $\vec{x}^N \in Q_1, \vec{y}^N \in Q_2$ satisfy the excessive gap condition for μ_1^N, μ_2^N with

$$\frac{1}{\mu_1^N} + \frac{1}{\mu_2^N} \geq \frac{1}{\mu_1^0} + \frac{1}{\mu_2^0} + \frac{\sqrt{\sigma_1\sigma_2}N}{4\sqrt{2}\|A\|} > \frac{\sqrt{\sigma_1\sigma_2}N}{4\sqrt{2}\|A\|}. \tag{8.15}$$

From (8.12) and (8.15) we get

$$\frac{2+1}{\mu_i^N} \geq \frac{\sqrt{\sigma_1\sigma_2}N}{4\sqrt{2}\|A\|}, \quad \text{for } i \in \{1,2\}. \tag{8.16}$$

Since $\vec{x}^N \in Q_1, \vec{y}^N \in Q_2$ satisfy the excessive gap condition for μ_1^N, μ_2^N, inequalities (8.4) and (8.16) yield

$$0 \leq \max_{\vec{x}\in Q_1}\langle A\vec{y}^N, \vec{x}\rangle - \min_{\vec{y}\in Q_2}\langle A\vec{y}, \vec{x}^N\rangle \leq \mu_1^N D_1 + \mu_2^N D_2 \leq \frac{12\sqrt{2}\|A\|(D_1+D_2)}{\sqrt{\sigma_1\sigma_2}N}.$$

□

From Theorem 6 it follows that after $N = \mathcal{O}(1/\epsilon)$ calls to `decrease` Algorithm `EGT-2` returns a pair $\vec{x}^N \in Q_1, \vec{y}^N \in Q_2$ satisfying

$$0 \leq \max_{\vec{x} \in Q_1} \langle A\vec{y}^N, \vec{x} \rangle - \min_{\vec{y} \in Q_2} \langle A\vec{y}, \vec{x}^N \rangle \leq \epsilon.$$

However, the appropriate measure of the overall computational work in Algorithm `EGT-2` is the number of calls to the subroutine `shrink`. Each call to `decrease` requires one call to `shrink` plus an extra call to `shrink` whenever τ needs to be reduced. Hence the total number of calls to `shrink` is N plus the total number of reductions in τ throughout the algorithm. Since τ is reduced by 1/2 each time, the total number of reductions in τ is exactly equal to $\log_2(0.5/\tau^N)$. From inequality (8.13), we get $\tau^N = \Omega(\mu_1^N) = \Omega(\mu_2^N) = \Omega(\epsilon)$, so the number of extra calls to `shrink` is $\mathcal{O}(\log(1/\epsilon))$. Therefore the total number of calls to `shrink` in Algorithm `EGT-2` is $\mathcal{O}(1/\epsilon) + \mathcal{O}(\log(1/\epsilon)) = \mathcal{O}(1/\epsilon)$.

8.4 Solving matrix games with $\mathcal{O}(\log 1/\epsilon)$ convergence

In this section we describe a smoothing method for the min-max matrix game problem

$$\min_{x \in \Delta_m} \max_{y \in \Delta_n} x^{\mathrm{T}} A y = \max_{y \in \Delta_n} \min_{x \in \Delta_m} x^{\mathrm{T}} A y \tag{8.17}$$

where $\Delta_m := \{x \in \mathbf{R}^m : \sum_{i=1}^m x_i = 1, x \geq 0\}$ is the set of mixed strategies for a player with m pure strategies. The game interpretation is that if player 1 plays $x \in \Delta_m$ and player 2 plays $y \in \Delta_n$, then 1 receives payoff $-x^{\mathrm{T}} A y$ and 2 receives payoff $x^{\mathrm{T}} A y$.

Nesterov [118] formulated a first-order smoothing technique for solving for each agent's strategy in a matrix game separately. We present that idea here, but applied to a formulation where we solve for both players' strategies at once.

Problem (8.17) can be rewritten as the primal-dual pair of nonsmooth optimization problems

$$\min\{f(x) : x \in \Delta_m\} = \max\{\phi(y) : y \in \Delta_n\}$$

where

$$\begin{aligned} f(x) &:= \max\left\{x^{\mathrm{T}} A v : v \in \Delta_n\right\}, \\ \phi(y) &:= \min\left\{u^{\mathrm{T}} A y : u \in \Delta_m\right\}. \end{aligned}$$

For our purposes it will be convenient to cast this as the primal-dual nonsmooth convex minimization problem

$$\min\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\}, \tag{8.18}$$

where

$$F(x,y) = \max\left\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay : (u,v) \in \Delta_m \times \Delta_n\right\}. \tag{8.19}$$

Observe that $F(x,y) = f(x) - \phi(y)$ is convex and $\min\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\} = 0$. Also, a point $(x,y) \in \Delta_m \times \Delta_n$ is an ϵ-solution to (8.17) if and only if $F(x,y) \leq \epsilon$.

Since the objective function $F(x,y)$ in (8.18) is nonsmooth, a subgradient algorithm would be appropriate. Thus, without making any attempt to exploit the structure of our problem, we would be faced with a worst-case bound on a subgradient-based algorithm of $\mathcal{O}(1/\epsilon^2)$. However, we can get a much better bound by exploiting the structure of our problem as we now show.

The following objects associated to Equation (8.18) will be useful later. Let

$$\mathit{Opt} := \operatorname{Argmin}\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\}$$

be the set of all optimal solutions and let $\mathit{dist} : \Delta_m \times \Delta_n \to \mathbf{R}$ be the distance function to the set Opt, *i.e.*,

$$\mathit{dist}(x,y) := \min\{\|(x,y) - (u,v)\| : (u,v) \in \mathit{Opt}\}.$$

Let $(\bar{u},\bar{v}) \in \Delta_m \times \Delta_n$ and $\mu > 0$. Consider the following smoothed version of F:

$$F_\mu(x,y) = \max\left\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \frac{\mu}{2}\|(u,v) - (\bar{u},\bar{v})\|^2 : (u,v) \in \Delta_m \times \Delta_n\right\}. \tag{8.20}$$

Let $(u(x,y), v(x,y)) \in \Delta_m \times \Delta_n$ denote the maximizer in (8.20). This maximizer is unique since the function

$$x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \frac{\mu}{2}\|(u,v) - (\bar{u},\bar{v})\|^2$$

is strictly concave in u and v [118]. It follows from [118, Theorem 1] that F_μ is smooth with gradient

$$\nabla F_\mu(x,y) = \begin{bmatrix} 0 & A \\ -A^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix},$$

and ∇F_μ is Lipschitz with constant $\frac{\|A\|^2}{\mu}$.[3] Let

$$D := \max\left\{\frac{\|(u,v) - (\bar{u},\bar{v})\|^2}{2} : (u,v) \in \Delta_m \times \Delta_n\right\}.$$

Nesterov's optimal gradient algorithm applied to the problem

$$\min\{F_\mu(x,y) : (x,y) \in \Delta_m \times \Delta_n\} \tag{8.21}$$

[3] $\|A\|$ denotes the *matrix norm* of matrix A which is associated with some vector norm. We will use the Euclidean norm (L_2-norm) for which it can be shown that $\|A\| = \sqrt{\lambda(A^{\mathrm{T}}A)}$ where $\lambda(M)$ denotes the largest eigenvalue of matrix M.

yields the following algorithm. Assume $(x_0, y_0) \in \Delta_m \times \Delta_n$ and $\epsilon > 0$ are given.

```
smoothing$(A, x_0, y_0, \epsilon)$

  1. Let $\mu = \frac{\epsilon}{2D}$ and $(w_0, z_0) = (x_0, y_0)$
  2. For $k = 0, 1, \dots$
      (a) $(u_k, v_k) = \frac{2}{k+2}(w_k, z_k) + \frac{k}{k+2}(x_k, y_k)$
      (b) $(x_{k+1}, y_{k+1}) =$
          $\text{argmin}\left\{\nabla F_\mu(u_k, v_k)^{\mathrm{T}}((x, y) - (u_k, v_k)) + \frac{\|A\|^2}{2\mu}\|(x, y) - (u_k, v_k)\|^2 : (x, y) \in \Delta_m \times \Delta_n\right\}$
      (c) If $F(x_{k+1}, y_{k+1}) < \epsilon$ Return
      (d) $(w_{k+1}, z_{k+1}) =$
          $\text{argmin}\left\{\sum_{i=0}^{k} \frac{i+1}{2}\nabla F_\mu(u_i, v_i)^{\mathrm{T}}((w, z) - (u_i, v_i)) + \frac{\|A\|^2}{2\mu}\|(w, z) - (x_0, y_0)\|^2 : (w, z) \in \Delta_m \times \Delta_n\right\}$
```

Proposition 7 *Algorithm* smoothing *finishes in at most*

$$k = \frac{2\sqrt{2} \cdot \|A\| \cdot \sqrt{D} \cdot dist(x_0, y_0)}{\epsilon}$$

first-order iterations.

PROOF. This readily follows from [94, Theorem 9] applied to the prox-function

$$d(u, v) = \frac{1}{2}\|(u, v) - (\bar{u}, \bar{v})\|^2,$$

which we used for smoothing in Equation (8.20). □

Note that the vectors $\bar{u}$, $\bar{v}$ can be *any* vectors in Δ_m and Δ_n. In our implementation, we take these vectors to be those corresponding to a uniformly random strategy.

8.4.1 Iterated smoothing scheme for matrix games

We are now ready to present the main contribution of this chapter. The new algorithm is a simple extension of Algorithm smoothing. At each iteration we call the basic smoothing subroutine with a target accuracy. Between the iterations, we reduce the target accuracy by $\gamma > 1$. Consider the following iterated first-order method for minimizing $F(x, y)$.

```
$\mathbf{iterated}(A, x_0, y_0, \gamma, \epsilon)$

  1. Let $\epsilon_0 = F(x_0, y_0)$

  2. For $i = 0, 1, \ldots$

       • $\epsilon_{i+1} = \frac{\epsilon_i}{\gamma}$
       • $(x_{i+1}, y_{i+1}) = \mathbf{smoothing}(A, x_i, y_i, \epsilon_{i+1})$
       • If $F(x_{i+1}, y_{i+1}) < \epsilon$ halt
```

While the modification to the algorithm is simple, it yields an exponential speedup with respect to reaching the target accuracy ϵ:

Theorem 8 *Each call to* **smoothing** *in Algorithm* **iterated** *halts in at most*

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \sqrt{D}}{\delta(A)} \tag{8.22}$$

first-order iterations, where $\delta(A)$ is a finite condition measure of the matrix A.

Algorithm **iterated** *halts in at most*

$$\frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$$

outer iterations, that is, in at most

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \ln(2\|A\|/\epsilon) \cdot \sqrt{D}}{\ln(\gamma) \cdot \delta(A)} \tag{8.23}$$

first-order iterations.

By setting $\gamma = e \approx 2.718...$ above gives the bound

$$\frac{2\sqrt{2} \cdot e \cdot \|A\| \cdot \ln(2\|A\|/\epsilon)\sqrt{D}}{\delta(A)}.$$

It can be shown that this is the optimal setting of γ for the overall complexity bound in Theorem 8.

For the proof of Theorem 8, we need to introduce the condition measure $\delta(A)$.

8.4.2 The condition measure $\delta(A)$

In this subsection we develop the technical results necessary for showing our main complexity result. We define the condition measure of a matrix A as

$$\delta(A) = \sup_{\delta} \left\{ \delta : dist(x,y) \leq \frac{F(x,y)}{\delta} \quad \forall (x,y) \in \Delta_m \times \Delta_n \right\}.$$

Notice that $\delta(A)$ can be geometrically visualized as a measure of "steepness" of the function $F(x,y)$. We can relate this to $\kappa(A)$ by defining $\kappa(A) := \|A\|/\delta(A)$. The following technical lemma shows that $\delta(A) > 0$ for all A.

Lemma 9 *Assume $A \in \mathbf{R}^{m\times n}$ and F is as in* (8.19). *There exists $\delta > 0$ such that*

$$dist(x,y) \leq \frac{F(x,y)}{\delta} \text{ for all } (x,y) \in \Delta_m \times \Delta_n. \tag{8.24}$$

PROOF. Since the function $F : \Delta_m \times \Delta_n \to \mathbf{R}$ is polyhedral, its epigraph $epi(F) = \{(x,y,t) : t \geq F(x,y),\ (x,y) \in \Delta_m \times \Delta_n\}$ is polyhedral. It thus follows that

$$epi(F) = conv\{(x_i,y_i,t_i) : i = 1,\ldots,M\} + \{0\} \times \{0\} \times [0,\infty)$$

for a finite set of points $(x_i,y_i,t_i) \in \Delta_m \times \Delta_n \times \mathbf{R}_+,\ i = 1,\ldots,M$. Therefore F can be expressed as

$$F(x,y) = \min \left\{ \sum_{i=1}^{M} t_i\lambda_i : \sum_{i=1}^{M} (x_i,y_i)\lambda_i = (x,y),\ \lambda \in \Delta_M \right\}. \tag{8.25}$$

Since $\min\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\} = 0$, we have $\min\{t_i, i = 1,\ldots,M\} = 0$. Without loss of generality assume $t_1 \geq t_2 \geq \cdots \geq t_N > 0 = t_{N+1} = \cdots = t_M$. We assume $N \geq 1$ as otherwise $Opt = \Delta_m \times \Delta_n$ and (8.24) readily holds for any $\delta > 0$. Thus $Opt = conv\{(x_i,y_i) : i = N+1,\ldots,M\}$. Let

$$\begin{aligned} \delta &:= \frac{t_N}{\max\{\|(x_i,y_i) - (x,y)\| : i = 1,\ldots,N, (x,y) \in Opt\}} \\ &= \frac{t_N}{\max\{\|(x_i,y_i) - (x_j,y_j)\| : i = 1,\ldots,N, j = N+1,\ldots,M\}} \end{aligned}$$

We claim that δ satisfies (8.24). To prove this claim, let $(x,y) \in \Delta_m \times \Delta_n$ be any arbitrary point. We need to show that $dist(x,y) \leq F(x,y)/\delta$. Assume $F(x,y) > 0$ as otherwise there is nothing to show. From (8.25) it follows that

$$(x,y) = \sum_{i=1}^{M} (x_i,y_i)\lambda_i,\ F(x,y) = \sum_{i=1}^{M} t_i\lambda_i = \sum_{i=1}^{N} t_i\lambda_i$$

for some $\lambda \in \Delta_M$. Let $\mu := \sum_{i=1}^N \lambda_i > 0$, and let $\tilde{\lambda} \in \Delta_N$ be the vector defined by putting $\tilde{\lambda}_i := \lambda_i/\mu,\ i = 1, \ldots, N$. In addition, let $(\hat{x}, \hat{y}) = \sum_{i=1}^N (x_i, y_i)\tilde{\lambda}_i = \sum_{i=1}^N (x_i, y_i)\lambda_i/\mu \in \Delta_m \times \Delta_n$, and $(\tilde{x}, \tilde{y}) \in \mathit{Opt}$ be as follows

$$(\tilde{x}, \tilde{y}) := \begin{cases} \displaystyle\sum_{i=N+1}^{M} (x_i, y_i)\lambda_i/(1-\mu) & \text{if} \quad \mu < 1 \\ (x_M, y_M) & \text{if} \quad \mu = 1 \end{cases}$$

Then $(x, y) = \mu(\hat{x}, \hat{y}) + (1-\mu)(\tilde{x}, \tilde{y})$ and consequently

$$\begin{aligned} \|(x,y) - (\tilde{x}, \tilde{y})\| &= \mu\|(\hat{x}, \hat{y}) - (\tilde{x}, \tilde{y})\| \\ &= \mu \left\| \sum_{i=1}^N \tilde{\lambda}_i((x_i, y_i) - (\tilde{x}, \tilde{y})) \right\| \\ &\le \mu \sum_{i=1}^N \tilde{\lambda}_i \|(x_i, y_i) - (\tilde{x}, \tilde{y})\| \\ &\le \mu \max\{\|(x_i, y_i) - (x, y)\| : i = 1, \ldots, N,\ (x, y) \in \mathit{Opt}\} \\ &= \tfrac{\mu t_N}{\delta}. \end{aligned}$$

To finish, observe that

$$F(x, y) = \sum_{i=1}^N t_i \lambda_i = \mu \sum_{i=1}^N t_i \tilde{\lambda}_i \ge \mu t_N.$$

Therefore,

$$\mathit{dist}(x, y) \le \|(x, y) - (\tilde{x}, \tilde{y})\| \le \mu t_N/\delta \le F(x, y)/\delta.$$

□

8.4.3 Proof of Theorem 8

By construction, for each $i = 0, 1, \ldots$ we have

$$\mathit{dist}(x_i, y_i) \le \frac{\epsilon_i}{\delta(A)} = \frac{\gamma \cdot \epsilon_{i+1}}{\delta(A)}.$$

The iteration bound (8.22) then follows from Proposition 7.

After N outer iterations Algorithm **iterated** yields $(x_N, y_N) \in \Delta_m \times \Delta_n$ with

$$F(x_N, y_N) < \epsilon_N = \frac{F(x_0, y_0)}{\gamma^N} \le \frac{2\|A\|}{\gamma^N}.$$

Thus, $F(x_N, y_N) < \epsilon$ for $N = \frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$ and (8.23) follows from (8.22).

8.4.4 The subroutine *smoothing* for matrix games

Algorithm **smoothing** involves fairly straightforward operations except for the solution of a subproblem of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|(u,v)\|^2 - (g,h)^{\mathrm{T}}(u,v) : (u,v) \in \Delta_m \times \Delta_n\right\}.$$

This problem in turn separates into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in \Delta_m\right\}. \tag{8.26}$$

Problem (8.26) can easily be solved via its Karush-Kuhn-Tucker optimality conditions:

$$u - g = \lambda\mathbf{1} + \mu,\ \lambda \in \mathbf{R},\ \mu \in \mathbf{R}^m_+,\ u \in \Delta_m,\ u^{\mathrm{T}}\mu = 0.$$

From these conditions it follows that the solution to (8.26) is given by

$$u_i = \max\{0, g_i - \lambda\},\ i = 1, \ldots, m,$$

where $\lambda \in \mathbf{R}$ is such that $\sum_{i=1}^m \max\{0, (g_i - \lambda)\} = 1$. This value of λ can be computed in $\mathcal{O}(m\ln(m))$ steps via a binary search in the sorted components of the vector g.

8.5 Solving sequential games with $\mathcal{O}(\log 1/\epsilon)$ convergence

Algorithm **iterated** and its complexity bound can be extended to sequential games. The Nash equilibrium problem of a two-player zero-sum sequential game with imperfect information can be formulated using the sequence form representation as the following saddle-point problem [84, 130, 160]:

$$\min_{x\in Q_1}\max_{y\in Q_2} x^{\mathrm{T}}Ay = \max_{y\in Q_2}\min_{x\in Q_1} x^{\mathrm{T}}Ay. \tag{8.27}$$

In this formulation, the vectors x and y represent the strategies of players 1 and 2 respectively. The strategy spaces $Q_i \subseteq \mathbf{R}^{S_i},\ i = 1,2$ are the sets of realization plans of players 1 and 2 respectively, where S_i is the set of sequences of moves of player i.

The approach we presented for equilibrium finding in matrix games extends to sequential games in the natural way: recast (8.27) as a nonsmooth convex minimization problem

$$\min\{F(x,y) : (x,y) \in Q_1 \times Q_2\}, \tag{8.28}$$

for

$$F(x,y) = \max\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay : (u,v) \in Q_1 \times Q_2\}. \tag{8.29}$$

Algorithms **smoothing** and **iterated** extend to this context by replacing Δ_m and Δ_n with Q_1 and Q_2, respectively. Proposition 7 and Theorem 8 also extend in the same fashion. However, the critical subproblem in the subroutine **smoothing** becomes more challenging, as described next.

8.5.1 The subroutine ***smoothing*** for sequential games

Here we describe how to solve each of the two argmin subproblems of **smoothing** in the sequential game case. Each of those two subproblems decomposes into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in Q\right\}, \tag{8.30}$$

where Q is a set of realization plans.

Our algorithm for this is a generalization of the solution approach described above for the case $Q = \Delta_k$. In order to describe it, we use some features of the sets of realization plans in the sequence form representation of sequential games. A detailed discussion of the sequence form can be found in [160]. Recall that an extensive form sequential game is given by a tree, payoffs at the leaves, chance moves, and information sets [120]. Each node in the tree determines a unique *sequence* of choices from the root to that node for each one of the players. Under the assumption of perfect recall, all nodes in an information set u of a player define the same sequence σ_u of choices.

Assume U is the set of information sets of a particular player. For each $u \in U$ let C_u denote the set of choices for that player. Then the set of sequences S of the player can be written as

$$S = \{\emptyset\} \cup \{\sigma_u c : u \in U,\, c \in C_u\}$$

where the notation $\sigma_u c$ denotes the sequence of moves σ_u followed by the move c. A *realization plan* for this player is a non-negative vector $x : S \to \mathbf{R}$ that satisfies $x(\emptyset) = 1$, and

$$-x(\sigma_u) + \sum_{c \in C_u} x(\sigma_u c) = 0$$

for all $u \in U$.

It is immediate that the set of realization plans of the player as above can be written in the form

$$\{x \geq 0 : Ex = e\}$$

for some $(1 + |U|) \times |S|$ matrix E with entries $\{0, 1, -1\}$ and the $(1 + |U|)$-dimensional vector $e = (1, 0, \ldots, 0)^{\mathrm{T}}$. It also follows that sets of realization plans are *complexes*. A complex is a generalization of a simplex, and can be recursively defined as follows:

(C1) The empty set $\emptyset$ is a complex.

(C2) Assume $Q_j \subseteq \mathbf{R}^{d_j}$ for $j = 1, \ldots, k$ are complexes. Then the following set is a complex

$$\left\{\left(u^0, u^1, \ldots, u^k\right) \in \mathbf{R}^{k+d_1+\cdots+d_k} : u^0 \in \Delta_k,\ u^j \in u^0_j \cdot Q_j,\ j = 1, \ldots, k\right\}.$$

(The operation $u^0_j \cdot Q_j$ multiplies all elements of Q_j by u^0_j.)

(C3) Assume $Q_j \subseteq \mathbf{R}^{d_j}$ for $j = 1, \ldots, k$ are complexes. Then the following set (their Cartesian product) is a complex

$$\left\{\left(u^1, \ldots, u^k\right) \in \mathbf{R}^{d_1+\cdots+d_k} : u^j \in Q_j,\ j = 1, \ldots, k\right\}.$$

Note that any simplex is a complex: Δ_k is obtained by applying **(C2)** with $Q_j = \emptyset,\ j = 1, \ldots, k$.

Given a complex $Q \subseteq \mathbf{R}^d$ and a vector $g \in \mathbf{R}^d$, define the *value function* $v_{Q,g} : \mathbf{R}_+ \to \mathbf{R}$ as

$$v_{Q,g}(t) := \min\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q\right\}.$$

It is easy to see that $v_{Q,g}$ is differentiable in $\mathbf{R}_+$. Let $\lambda_{Q,g} = v'_{Q,g}$ and let $\theta_{Q,g}$ be the inverse function of $\lambda_{Q,g}$. It is easy to see that $\lambda_{Q,g}$ is strictly increasing in $\mathbf{R}_+$. In particular, its minimum value is $\lambda_{Q,g}(0)$. The function $\theta_{Q,g}$ can be defined in all of $\mathbf{R}$ by putting $\theta_{Q,g}(\lambda) := 0$ for all $\lambda \leq \lambda_{Q,g}(0)$. Finally, define the minimizer function $u_{Q,g} : \mathbf{R}_+ \to Q$ as

$$u_{Q,g}(t) := \operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q\right\}.$$

The recursive algorithm **ComplexSubproblem** below computes the functions $v_{Q,g}$, $\lambda_{Q,g}$, $\theta_{Q,g}$, and $u_{Q,g}$ for any given complex Q. In particular, it computes the solution $u_{Q,g}(1)$ to the subproblem (8.30). The algorithm assumes that either Q is as in **(C2)** and $g = \left(g^0, g^1, \ldots, g^k\right) \in \mathbf{R}^{k+d_1+\cdots+d_k}$, or Q is as in **(C3)** and $g = \left(g^1, \ldots, g^k\right) \in \mathbf{R}^{d_1+\cdots+d_k}$.

ComplexSubproblem(Q, g)

1. If Q is as in **(C2)** then

   (a) For $i = 1, \dots, k$ let $\tilde{\lambda}_i : \mathbf{R}_+ \to \mathbf{R}$ and $\tilde{\theta}_i : \mathbf{R} \to \mathbf{R}_+$ be

   $$\tilde{\lambda}_i(t) := t - g_i^0 + \lambda_{Q_i,g^i}(t), \quad \tilde{\theta}_i := \tilde{\lambda}_i^{-1}.$$

   (b) Let $\theta_{Q,g} := \sum_{i=1}^k \tilde{\theta}_i$ and $\lambda_{Q,g} := \theta_{Q,g}^{-1}$

   (c) Let $u_{Q,g} : \mathbf{R}_+ \to Q$ be

   $$u_{Q,g}(t)_i^0 := \tilde{\theta}_i(\lambda_{Q,g}(t))$$

   and

   $$u_{Q,g}(t)^i := u_{Q_i,g^i}\left(u_{Q,g}(t)_i^0\right)$$

   for $i = 1, \dots, k$.

2. If Q is as in **(C3)** then

   (a) Let $\lambda_{Q,g} := \sum_{i=1}^k \lambda_{Q_i,g^i}$ and $\theta_{Q,g} = \lambda_{Q,g}^{-1}$

   (b) Let $u_{Q,g} : \mathbf{R}_+ \to Q$ be

   $$u_{Q,g}(t)^i := u_{Q_i,g^i}(t)$$

   for $i = 1, \dots, k$.

While we presented Algorithm **ComplexSubproblem** in recursive form for pedagogical reasons, for efficiency purposes we implemented it as a dynamic program. The implementation first performs a bottom-up pass that computes and stores the functions $\lambda_{Q,g}$. Subsequently a top-down pass computes the components of the minimizer $u_{Q,g}(t)$.

Theorem 10 *Algorithm* **ComplexSubproblem** *is correct. In addition, the function $\lambda_{Q,g}$ is piecewise linear. Furthermore, if Q is as in* **(C2)** *or is as in* **(C3)***, then the total number of breakpoints $B(Q, g)$ of $\lambda_{Q,g}$ is at most*

$$\sum_{i=1}^k \max\{B(Q_i, g_i), 1\}.$$

If the breakpoints of λ_{Q_i,g^i} are available, then the breakpoints of $\lambda_{Q,g}$ can be constructed in

$$\mathcal{O}(B(Q, g) \ln(B(Q, g)))$$

steps, i.e.*, this is the run time of Algorithm* **ComplexSubproblem**.

PROOF. First, assume that Q is as in **(C2)**. Then the value function $v_{Q,g}(t)$ can be written as

$$v_{Q,g}(t) = \min\left\{\frac{1}{2}\left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}} u^0 + \sum_{j=1}^{k} v_{Q_j,g^j}\left(u_j^0\right) : u^0 \in t\cdot\Delta_k\right\}. \tag{8.31}$$

This is a constrained optimization problem in the variables u^0. Its Karush-Kuhn-Tucker optimality conditions are

$$\begin{aligned} u_j^0 - g_j^0 + \lambda_{Q_j,g^j}\left(u_j^0\right) &= \lambda + \mu_j, \\ \lambda \in \mathbf{R},\ \mu &\in \mathbf{R}_+^k, \\ u^0 \in t\cdot\Delta_k,\ \ \mu^{\mathrm{T}} u^0 &= 0. \end{aligned} \tag{8.32}$$

By basic differentiability properties from convex analysis (see, *e.g.* [73, Chapter D]), it follows that $\lambda_{Q,g}(t) = v'_{Q,g}(t)$ is precisely the value of λ that solves the optimality conditions (8.32). From these optimality conditions, we get $u_j^0 = \tilde\theta_j(\lambda),\ j = 1,\ldots,k$ for the functions $\tilde\theta_j$ constructed in step 1 of Algorithm **ComplexSubproblem**. Hence

$$t = \sum_{j=1}^{k} u_j^0 = \sum_{j=1}^{k} \tilde\theta_j(\lambda).$$

Therefore, $\theta_{Q,g} = \sum_{j=1}^{k} \tilde\theta_j$. This shows the correctness of Steps 1.a and 1.b of Algorithm **ComplexSubproblem**. Finally, the correctness of Step 1.c follows from (8.31) and (8.32).

On the other hand, if Q is as in **(C3)** then the value function $v_{Q,g}(t)$ can be decoupled as follows

$$v_{Q,g}(t) = \sum_{i=1}^{k} \min\left\{\frac{1}{2}\left\|u^i\right\|^2 - \left(g^i\right)^{\mathrm{T}} u^i : u^i \in t\cdot Q_i\right\} = \sum_{i=1}^{k} v_{Q_i,g^i}(t). \tag{8.33}$$

This yields the correctness of Steps 2.a and 2.b.

The piecewise linearity of $\lambda_{Q,g}$ readily follows from the correctness of Algorithm **ComplexSubproblem**. As for the number of breakpoints, consider first the case when Q is as in **(C2)**. Observe that the number of breakpoints of $\tilde\theta_i$ is the same as that of $\tilde\lambda_i$, which is either the same as that of λ_{Q_i,g^i} (if $Q_i \neq \emptyset$) or 1 (if $Q_i = \emptyset$). To get the bound on $B(Q,g)$, note that the total number of breakpoints of $\lambda_{Q,g}$ is the same as that of $\theta_{Q,g}$, which is at most the sum of the number of breakpoints of all $\tilde\theta_i,\ i = 1,\ldots,k$. The breakpoints of $\theta_{Q,g}$ can be obtained by sorting the breakpoints of all of the θ_i together. This can be done in $\mathcal{O}(B(Q,g)\ln(B(Q,g)))$ steps. In the case when Q is as in **(C3)** the number of breakpoints of $\lambda_{Q,g}$ is at most the sum of the number of breakpoints of all $\lambda_i,\ i = 1, \dots, k$. The breakpoints of $\lambda_{Q,g}$ can be obtained by sorting the breakpoints of all of the λ_i together. This can be done in $\mathcal{O}(B(Q,g)\ln(B(Q,g)))$ steps.

$\square$

8.5.2 *ComplexSubproblem* example

We include a simple example to illustrate Algorithm **ComplexSubproblem**, as well as the use of our recursive definition of complexes. For simplicity of the example, let $Q_1 = \Delta_2$ and $Q_2 = \emptyset$. Then applying the recursive definition of complexes, **(C2)**, we get that Q is the set

$$\left\{ \left(u^0, u^1\right) : u^0 \in \Delta_2, u^1 \in u_1^0 \cdot Q_1 \right\}.$$

In a sequential game corresponding to this set of realization plans, the player first chooses among actions a_1^0 and a_2^0, with probabilities u_1^0 and u_2^0, respectively, and conditioned on choosing action a_1^0, the player may be asked to choose among actions a_1^1 and a_2^1, which are played with probabilities u_1^1/u_1^0 and u_2^1/u_1^0, respectively. (Note that there is no u^2 in the above equation since $Q_2 = \emptyset$, *i.e.*, if the agent plays a_2^0, he will have no further actions.) The complex Q can also be written as $\{x \geq 0 : Ex = e\}$ for

$$E = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 1 \end{bmatrix},\ e = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Now, given input vector $g^1 \in \mathbf{R}^2$, we define the value function for Q_1 as

$$v_{Q_1,g^1}\left(u_1^0\right) := \min \left\{ \frac{1}{2} \left\|u^1\right\|^2 - \left(g^1\right)^{\mathrm{T}} u^1 : u^1 \in u_1^0 \cdot Q_1 \right\}.$$

Then, as was done in the proof of Theorem 10, we can write the value function for Q as

$$v_{Q,g}(t) := \min \left\{ \frac{1}{2} \left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}} u^0 + v_{Q_1,g^1}\left(u_1^0\right) : u^0 \in t \cdot \Delta_k \right\}$$

for $g = (g^0, g^1) \in \mathbf{R}^4$. This is the problem that **ComplexSubproblem** is trying to solve in our example.

We first demonstrate the algorithm as it executes **ComplexSubproblem**(Q_1, g^1), *i.e.*, the bottom of the recursion. Since Q_1 has no "sub-complexes", we have

$$\begin{aligned} \tilde{\lambda}_1(t) &:= t - g_1^1, \\ \tilde{\lambda}_2(t) &:= t - g_2^1. \end{aligned}$$

The equations are graphed on the left in Figure 8.4. Step 1 of the algorithm constructs the $\tilde{\theta}_i$ functions to be the inverse of the $\tilde{\lambda}_i(t)$ functions. Once these inverses are computed, Step 2 of the algorithm adds the $\tilde{\theta}_i$ functions to obtain the θ_{Q_1,g^1} function, which is in turn inverted to construct the λ_{Q_1,g^1} function. This process of inverting, adding, and inverting again has a more intuitive description in the form of a "horizontal addition" operation on the $\tilde{\lambda}_i$ functions. In such an operation, two functions are added as normal, except we flip the axis of the graph so that the x-axis and y-axis are switched. This operation is illustrated in Figure 8.4. The graph on the left in Figure 8.4 contains the $\tilde{\lambda}_i(t)$ functions. These functions are "horizontally added" to obtain λ_{Q_1,g^1} on the right in Figure 8.4.

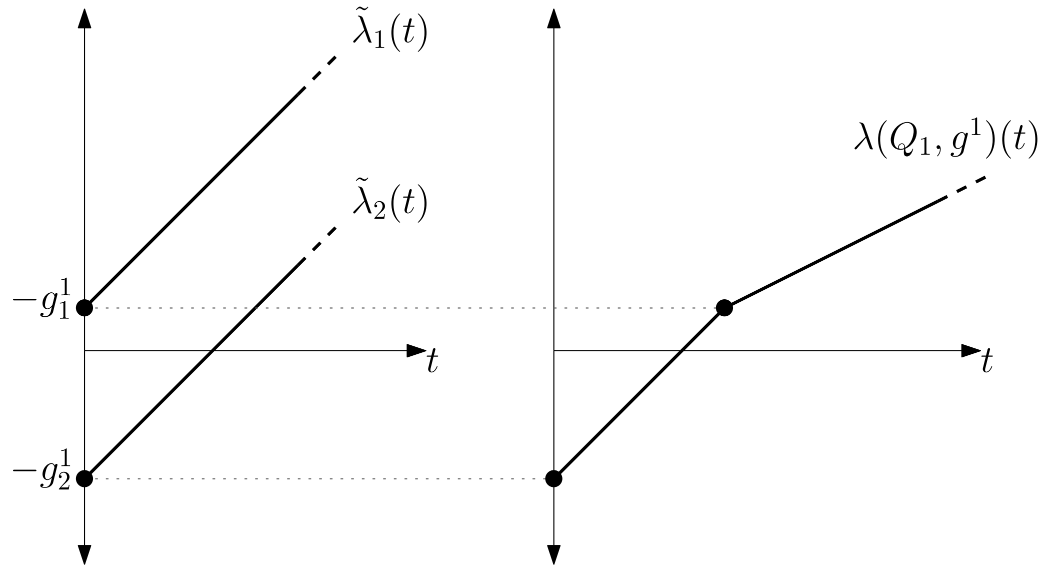

Figure 8.4: An illustration of Steps 1 and 2 of Algorithm **ComplexSubproblem** applied to Q_1 and g^1.

At non-bottom parts of the recursion ($\lambda_{Q,g}$ in our example) we construct the piecewise linear functions similarly, except that we have to take into account subsequent actions using the piecewise linear functions (function $\lambda_{Q_1,g^1}(t)$ in our example) already computed for the nodes below the current node in the recursion tree:

$$\begin{aligned}\tilde{\lambda}_1(t) &:= t - g_1^0 + \lambda_{Q_1,g^1}(t),\\ \tilde{\lambda}_2(t) &:= t - g_2^0\end{aligned}$$

The "horizontal addition" operation for this case is depicted in Figure 8.5.

Since $\lambda_{Q_1,g^1}(t)$ and $\lambda_{Q,g}$ are piecewise linear, our implementation simply represents them as a set of breakpoints, which are represented by solid circles in Figures 8.4 and 8.5.

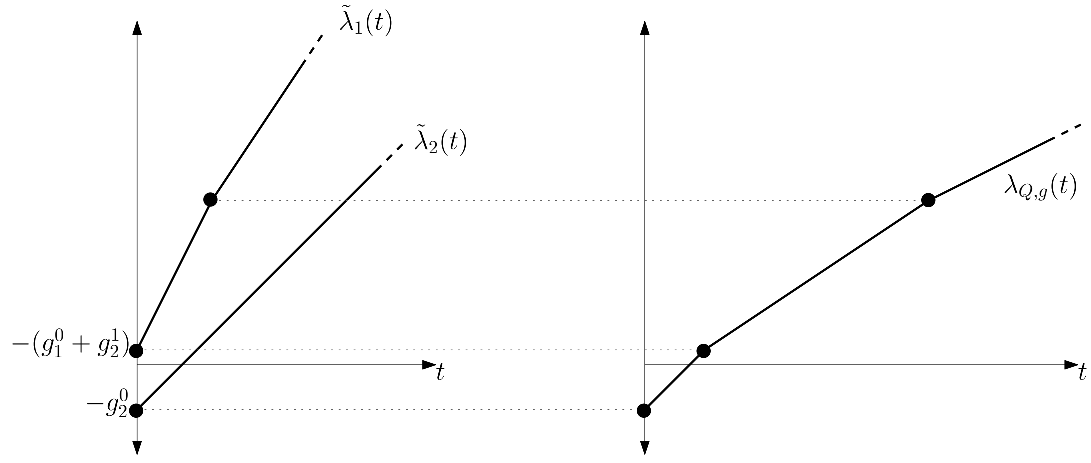

Figure 8.5: An illustration of Steps 1 and 2 of Algorithm **ComplexSubproblem** applied to Q and g.

Given that we have finally constructed the piecewise linear function at the root, we can determine the values of u^0 and u^1 that solve the optimization problem in (8.30) as described in Step 3 of Algorithm **ComplexSubproblem**. Specifically, we first take $t = 1$ and solve for u^0. To do this, we evaluate $\lambda_{Q,g}(1)$. Then we find the values of u_1^0 and u_2^0 such that

$$\begin{aligned}\tilde{\lambda}_1(u_1^0) &= \lambda_{Q,g}(1),\\ \tilde{\lambda}_2(u_2^0) &= \lambda_{Q,g}(1).\end{aligned}$$

This last operation is straightforward since the functions in question are monotonically increasing and piecewise linear.

Once we have computed u_1^0, we can evaluate $\lambda_{Q_1,g^1}(u_1^0)$ and find u_1^1 and u_2^1 that satisfy

$$\begin{aligned}\tilde{\lambda}_1(u_1^1) &= \lambda_{Q_1,g^1}(u_1^0),\\ \tilde{\lambda}_2(u_2^1) &= \lambda_{Q_1,g^1}(u_1^0).\end{aligned}$$

Again, this operation is easy due to the functions being monotonically increasing and piecewise linear. This completes the execution of Algorithm **ComplexSubproblem** on our example.

8.6 Computational experiments on $\mathcal{O}(\log 1/\epsilon)$ algorithm

In this section we report on our computational experience with our new method. We compared our **iterated** algorithm against the basic **smoothing** algorithm. We tested the algorithms on matrix games as well as sequential games.

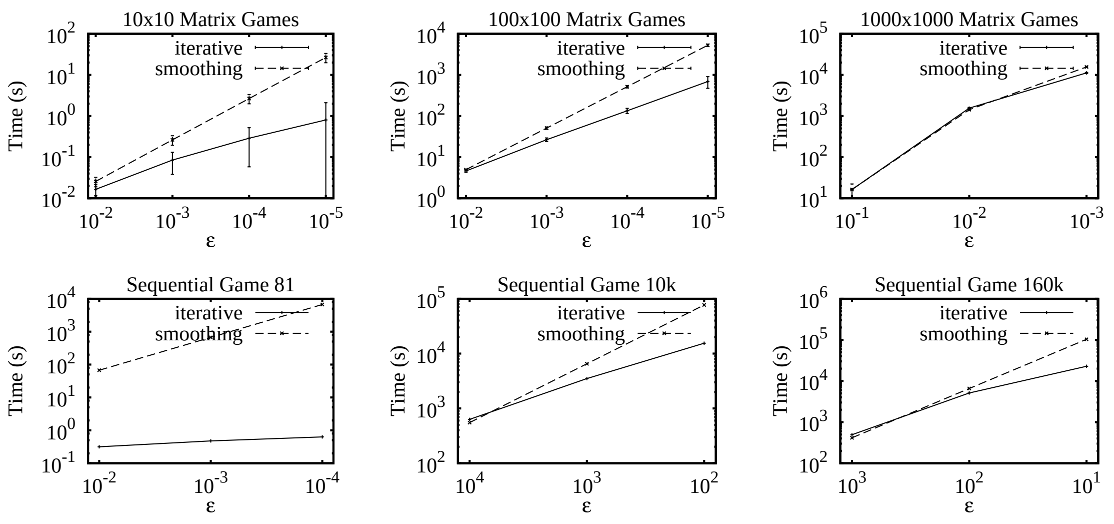

Figure 8.6: Time taken (in seconds) for each algorithm to find an ϵ-equilibrium for various values of ϵ.

For matrix games, we generated 100 games of three different sizes where the payoffs are drawn uniformly at random from the interval $[-1, 1]$. This is the same instance generator as in Nesterov's [118] experiments.

For sequential games, we used the benchmark instances `81`, `10k`, and `160k` which have been used in the past for benchmarking equilibrium-finding algorithms for sequential imperfect-information games [56]. These instances are all abstracted versions of Rhode Island Hold'em poker [146], and they are named to indicate the number of variables in each player's strategy vector.

Figure 8.6 displays the results. Each graph is plotted with ϵ on the x-axis (using an inverse logarithmic scale). The y-axis is the number of seconds (using a logarithmic scale) needed to find ϵ-equilibrium for the given ϵ. The matrix game graphs also display the standard deviation.

In all settings we see that our **iterated** algorithm indeed outperforms the **smoothing** algorithm (as the worst-case complexity results would suggest). In fact, as the desired accuracy increases, the relative speed difference also increases.

We also tested a version of our iterated smoothing algorithm that used the Lan *et al.* [94] variant of Nesterov's algorithm for smooth optimization (subroutine $\mathrm{smoothing}$). The only difference in that subroutine is in Step 2(d) of $\mathrm{smoothing}$. Although the guarantee of Theorem 8 does not hold, that version performed almost identically.

8.7 Summary

We presented new algorithms for finding ϵ-equilibria in two-person zero-sum games. They apply to both matrix and sequential games. In one algorithm, we adapt Nesterov's excessive gap technique to the problem of finding equilibria in two-person zero-sum games. We present heuristics which speed up the basic version of the algorithm, and we prove that the worst-case theoretical guarantees still hold when we use our heuristics.

We described another algorithm that has convergence rate $\mathcal{O}(\kappa(A)\ln(1/\epsilon))$, where $\kappa(A)$ is a condition measure of the matrix A. In terms of the dependence on ϵ, this matches the complexity of interior-point methods and is exponentially faster than prior first-order methods. Furthermore, our algorithms, like other first-order methods, uses dramatically less memory than interior-point methods, indicating that it can scale to games much larger than previously possible.

Our $\mathcal{O}(\log 1/\epsilon)$ scheme supplements Nesterov's first-order smoothing method with an outer loop that lowers the target ϵ between iterations (this target affects the amount of smoothing in the inner loop). We find it surprising that such a simple modification yields an exponential speed improvement, and wonder whether a similar phenomenon might occur in other optimization settings as well. Finally, computational experiments both in matrix games and sequential games show that a significant speed improvement is obtained in practice as well, and the relative speed improvement increases with the desired accuracy (as suggested by the complexity bounds).

Chapter 9

Sampling for Speeding Up Gradient-Based Algorithms

9.1 Introduction

First-order (*i.e.*, gradient-based) methods for solving two-person zero-sum sequential games of imperfect information are important tools in the construction of game theory-based agents. The computation time per iteration is typically dominated by matrix-vector product operations involving the payoff matrix A. In this chapter we describe a randomized sampling technique that approximates A with a sparser matrix $\tilde{A}$. Then an approximate equilibrium for the original game is found by finding an approximate equilibrium of the sampled game. We experimentally evaluate both static and dynamic sampling.

9.2 Main idea

The basic idea of gradient-based algorithms as applied to convex optimization problems is to estimate a good search direction based on information that is local to the current solution. Then, the algorithms take a step in a direction related to the gradient (different variants choose the exact direction and step size differently), and repeat the process from the new solution. As discussed above, the matrix-vector product is a crucial component of these algorithms. In this chapter we develop a faster algorithm for estimating the matrix-vector product, and hence estimating the gradient.

The specific technique we develop is based on *sampling*. In a game, there are random moves by nature that occur throughout the game. (In poker, these correspond to deals of cards.) Instead of exhaustively evaluating all possible sequences of moves by nature, we can instead estimate this expectation by sampling a much smaller number of sequences.

For a given game, let Θ denote the possible sequences of chance moves. In the sequence form representation of zero-sum games, each leaf in the game tree corresponds to a non-zero entry in the payoff matrix A. Each leaf also corresponds to a particular sequence of chance moves. Thus, we can partition the non-zero entries of A into a collection of non-overlapping matrices $\{A_\theta\}_{\theta\in\Theta}$ and we can calculate the matrix-vector product as a sum of products of these matrices:

$$Ay = \sum_{\theta\in\Theta} A_\theta y.$$

Instead of evaluating $A_\theta y$ for all $\theta \in \Theta$, we can estimate Ay by evaluating the products only for a small, randomly-sampled subset $\tilde{\Theta} \subset \Theta$:

$$Ay \approx \sum_{\theta\in\tilde{\Theta}} z_\theta A_\theta y.$$

The z_θ in the above equation are normalization constants that depend on the specific sampling performed, and are computed by simply making sure that the sampled probabilities sum to one.

Having described the sampling technique, there remains the question of how to leverage it in gradient-based equilibrium-finding algorithms. We address this question in the following two subsections.

9.3 Static sampling

In this section we describe a simple approach to incorporating sampling in gradient-based equilibrium-finding algorithms. The purpose of this subsection is primarily to illuminate the behavior of sampling algorithms rather than develop a strong algorithm. The main idea is to simply perform a single sampling with a fixed proportion of sampling up front, and then apply the equilibrium-finding algorithm using this sampling for the matrix. Specifically:

Step 1 Randomly generate the sample $\tilde{\Theta} \subset \Theta$ so that $|\tilde{\Theta}| = \lceil p|\Theta|\rceil$ where $p \in (0, 1)$ is the proportion of sampling that is specified as input.

Step 2 Run the gradient-based algorithm, but replace all matrix-vector products with the sampled version.

For this algorithm, we performed experiments on Rhode Island Hold'em poker [146], a benchmark problem for evaluating game-theoretic algorithms. Although it has been solved optimally [61], it remains useful as a test problem. The gradient-based algorithm used was the excessive gap technique for sequential games [117, 74], including both of the published heuristics for speeding it up and the decomposed matrix representation [56].

Figure 9.1 displays the results. Each of the plots shown is a log-log plot displaying data from running the algorithm with the amount of sampling specified as input ranging between 0.5% and 64%. Each plot contains two lines. The top line displays the real gap for a given point in time. The bottom line displays the gap in the sampled game—*i.e.,* the gap under the (incorrect) assumption that the game being solved actually is the sampled matrix.

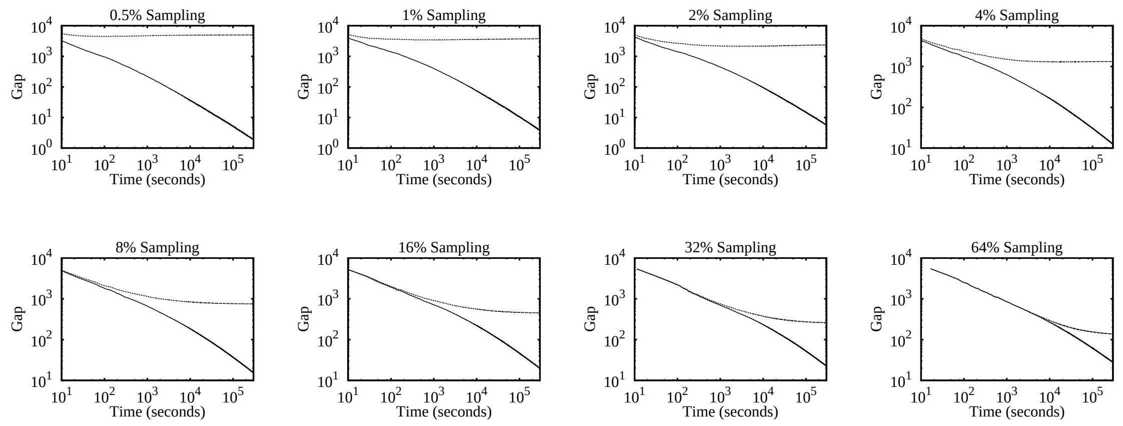

Figure 9.1: Duality gap versus time for varying levels of sampling. The solid, bottom line in each graph indicates the duality gap for the game corresponding to the sampled game. The dotted, top line in each graph indicates the duality gap for the original game. The effects of overfitting are indicated by a separation of the curves. As the level of sampling increases, the effect of overfitting takes longer to manifest.

Divergence of the two lines provides evidence of overfitting. In fact, in the plots with the smallest amount of sampling, we see that the gap actually *increases* after some time. This is very strong evidence of overfitting. The figures provide a clear trend showing that for larger amounts of sampling, the effects of overfitting take longer to appear.

Figure 9.2 provides another view of a subset of the data from Figure 9.1. This plot illustrates the selective superiority of using various levels of sampling. Using a relatively small amount of sampling (4%) initially provides the best performance. However, as the effects of overfitting begin to manifest, it becomes better to use an increased amount of sampling (16%) even though that initially provides worse performance. Finally, it becomes best to use an even larger amount of sampling (64%). (There are of course intermediate levels of sampling that could be performed. We omitted these from the graph to enhance the readability.)

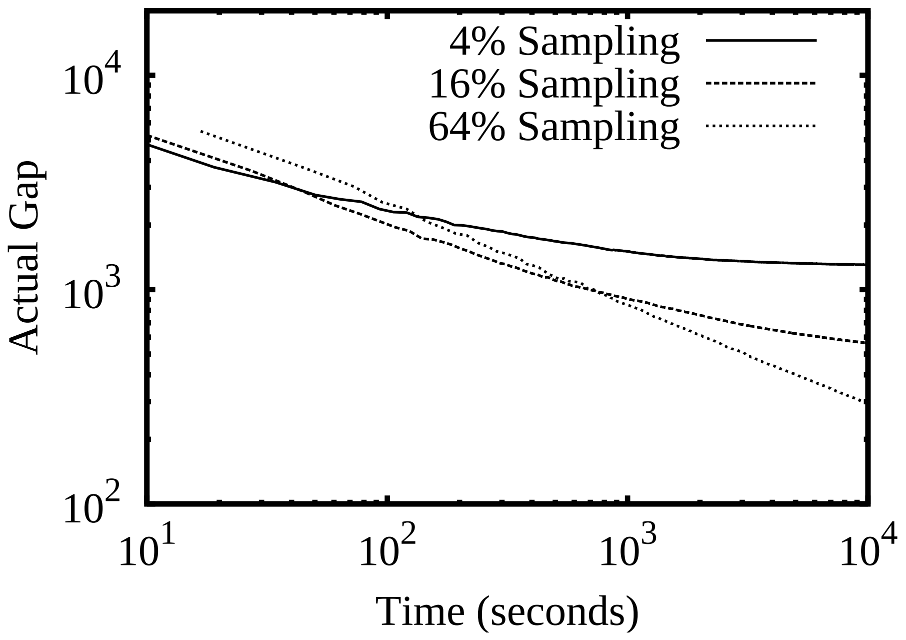

Figure 9.2: Duality gap for the original game as a function of time, for varying levels of sampling. Initially, the best performance is achieved using the smallest level of sampling. As time increases, and the effects of overfitting appear, the best performance is achieved using an increased amount of sampling.

This observation suggests an algorithm where we dynamically change the level of sampling based on properties observed at run-time. We discuss such an algorithm next.

9.4 Dynamic sampling

To avoid the problem of overfitting, we develop a dynamic sampling procedure that attempts to detect the occurrence of overfitting and then increases the level of sampling.

The detection of overfitting could conceivably be done in a number of ways. In this initial investigation, we adopt a fairly simple rule: if the actual gap, as measured every 50 iterations, ever increases, we consider this to be overfitting. We only do this every 50 iterations, rather than every iteration, for two reasons. First, computing the actual gap is expensive since it requires performing two matrix-vector products in the full (*i.e.*, unsampled) game—precisely the problem we are trying to avoid. Second, for the gradient-based algorithms we are using, the actual gap does not decrease monotonically. It is quite common for the value to fluctuate a bit from iteration to iteration. However, over a longer period, such as 50 iterations, this non-monotonicity is unlikely to be observed (unless, of course, overfitting is occurring).

Once we detect overfitting, we increase the amount of sampling that is performed and resume running the gradient-based algorithm as follows.

Step 1 Initialize $p = 0.01$ and initialize (x, y) arbitrarily.

Step 2 Randomly generate the sample $\tilde{\Theta} \subset \Theta$ so that $|\tilde{\Theta}| = \lceil p|\Theta| \rceil$.

Step 3 Run the gradient-based algorithm starting at (x, y) until overfitting is detected (if $p = 1$ then run the gradient-based algorithm indefinitely).

Step 4 Let $p \leftarrow \min\{2p, 1\}$.

Step 5 Go to Step 2.

So, in each phase, we double the level of sampling. It would certainly be reasonable to implement other strategies for increasing the amount of sampling as well. One could also use other criteria for triggering the next level of sampling, for example, if the sampled gap diverges too much from the actual gap. We leave the investigation of such other techniques for future research.

Figure 9.3 shows the results from running our dynamic sampling algorithm on a large instance of an abstraction of Heads-Up Limit Texas Hold'em poker. This problem instance is much larger than the Rhode Island Hold'em, and the relative amount of time spent performing the matrix-vector product is even larger, which should make our technique particularly effective. As can be seen in the plot, this is the case. For example, to reach a gap of 20 using the non-sampled version of the algorithm takes 32 hours. The sampled version only takes 3.7 hours to achieve the same actual gap.

The curve for the dynamic sampling algorithm has a noticeable "jump". This corresponds to the time when the algorithm detects overfitting and increases the amount of sampling. The underlying gradient-based algorithm must perform some modifications to the current strategies in order to continue. In our experiments we are using Nesterov's excessive gap technique for the gradient-based algorithm and we have to modify the starting solution to ensure that the *excessive gap condition* is satisfied (see [117] for details).

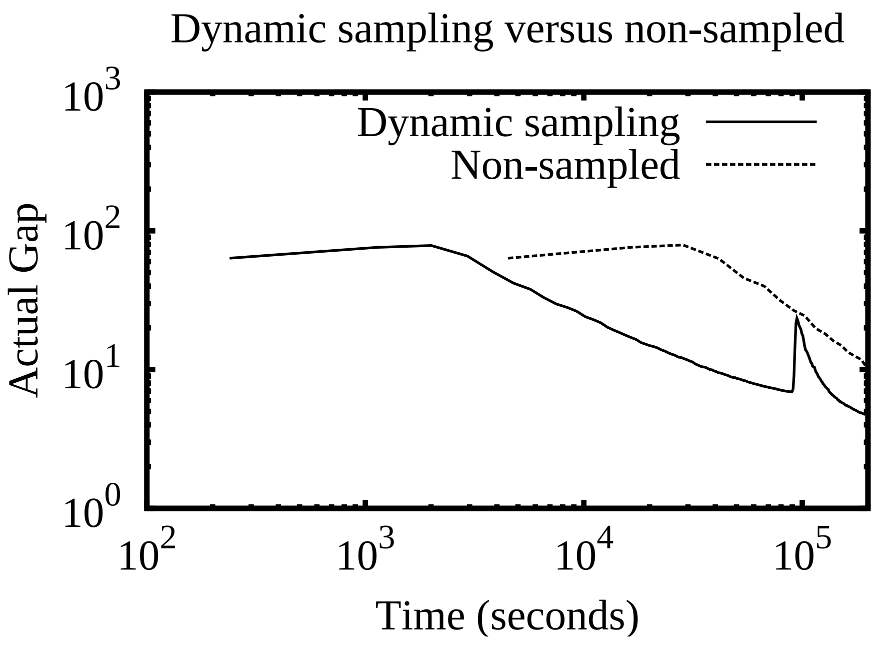

Figure 9.3: Duality gap versus time for the original non-sampling approach and the dynamic sampling approach.

This change in (x, y) causes the jump. Fortunately, the algorithm recovers from the jump quickly.

9.5 Summary

Sampling has been applied in the context of a regret-minimization approach to equilibrium-finding algorithms [166]. As is the case in our approach, that prior approach also sampled from the moves of nature. However, the use of that sampling was to estimate regret. In contrast, our sampling is used for approximating the result of a matrix-vector product for the purpose of estimating a gradient.

Another difference is that the prior approach performed a new sampling at each iteration. In contrast, we perform a single sampling up front, and only perform a new sampling when we increase the amount of sampling performed in the dynamic sampling variant of our approach.

We have also implemented a version in which we perform a new sampling at each iteration. However, that version performed quite poorly in our initial experiments, indicating that conducting a new sampling at each iteration is not appropriate for use in conjunction with gradient-based algorithms.

We also conducted exploratory experiments where our algorithm re-sampled (with the same level of sampling) every few iterations. That approach was ineffective at reducing the actual gap.

Chapter 10

Implementation of Gradient-Based Algorithms

10.1 Introduction

As discussed previously, for two-player zero-sum sequential games of imperfect information the Nash equilibrium problem can be formulated using the sequence form representation [130, 84, 160] as the following saddle-point problem:

$$\max_{\mathbf{x}\in Q_1} \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}\rangle = \min_{\mathbf{y}\in Q_2} \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}, \mathbf{x}\rangle. \tag{10.1}$$

When applying gradient-based algorithms to the above formulation (for example, using the algorithm of Chapter 8), the main computational bottleneck is in computing the matrix-vector products $A\mathbf{y}$ and $A^{\mathrm{T}}\mathbf{x}$. In this chapter, we take advantage of the structure of the problem to improve the performance of this operation both in terms of time and memory requirements. As a result, we are able to handle games that are several orders of magnitude larger than games that can be solved using conventional linear programming solvers. For example, we compute approximate solutions to an abstracted version of Texas Hold'em poker whose LP formulation has 18,536,842 rows and 18,536,852 columns, and has 61,450,990,224 non-zeros in the payoff matrix. This is more than 1,200 times the number of non-zeros in the Rhode Island Hold'em problem mentioned above. Since conventional LP solvers require an explicit representation of the problem (in addition to their internal data structures), this would require such a solver to use more than 458 GB of memory *simply to represent the problem.* On the other hand, our algorithm only requires 2.49 GB of memory.

10.2 Customizing the algorithm for stage-based sequential games

The bulk of the computational work at each iteration of a gradient-based algorithm consists of some matrix-vector multiplications $\mathbf{x} \mapsto A^{\mathrm{T}}\mathbf{x}$ and $\mathbf{y} \mapsto A\mathbf{y}$. These operations are by far the most expensive, both in terms of memory (for storing A) and time (for computing the product).

10.2.1 Addressing the space requirements

To address the memory requirements, we exploit the problem structure to obtain a concise representation for the payoff matrix A. This representation relies on a uniform structure that is present in poker games and many other games. For example, the betting sequences that can occur in most poker games are independent of the cards that are dealt. This conceptual separation of betting sequences and card deals is used by our automated abstraction algorithms. Analogously, we can decompose the payoff matrix based on these two aspects.

The basic operation we use in this decomposition is the *Kronecker product*, denoted by $\otimes$. Given two matrices $B \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{p\times q}$, the Kronecker product is

$$B \otimes C = \begin{bmatrix} b_{11}C & \cdots & b_{1n}C \\ \vdots & \ddots & \vdots \\ b_{m1}C & \cdots & b_{mn}C \end{bmatrix} \in \mathbb{R}^{mp\times nq}.$$

For ease of exposition, we explain the concise representation in the context of Rhode Island Hold'em poker [146], although the general technique applies much more broadly. The payoff matrix A can be written as

$$A = \begin{bmatrix} A_1 & & \\ & A_2 & \\ & & A_3 \end{bmatrix}$$

where $A_1 = F_1 \otimes B_1$, $A_2 = F_2 \otimes B_2$, and $A_3 = F_3 \otimes B_3 + S \otimes W$ for much smaller matrices F_i, B_i, S, and W. The matrices F_i correspond to sequences of moves in round i that end with a fold, and S corresponds to the sequences in round 3 that end in a showdown. The matrices B_i encode the betting structures in round i, while W encodes the win/lose/draw information determined by poker hand ranks.

Given this concise representation of A, computing $\mathbf{x} \mapsto A^{\mathrm{T}}\mathbf{x}$ and $\mathbf{y} \mapsto A\mathbf{y}$ is straightforward, and the space required is sublinear in the size of the game tree. For example, in Rhode Island Hold'em, the dimensions of the F_i and S matrices are 10×10, and the dimensions of B_1, B_2, and B_3 are 13×13, 205×205, and 1,774 $\times$ 1,774, respectively—in contrast to the A-matrix, which is 883,741 $\times$ 883,741. Furthermore, the matrices F_i, B_i, S, and W are themselves sparse which allows us to use the Compressed Row Storage (CRS) data structure (which stores only non-zero entries).

The data instances we performed the tests on are described in Table 8.1.

Table 10.1 clearly demonstrates the extremely low memory requirements of the EGT algorithms. Most notably, on the `Texas` instance, both of the CPLEX algorithms require more than 458 GB simply to *represent* the problem. In contrast, using the decomposed payoff matrix representation, the EGT algorithms require only 2.49 GB. Furthermore, in order to solve the problem, both the simplex and interior-point algorithms would require additional memory for their internal data structures.[1] Therefore, the EGT family of algorithms is already an improvement over the state-of-the-art (even without the heuristics).

Name	**CPLEX IPM**	**CPLEX Simplex**	**EGT**
`10k`	0.082 GB	$>$ 0.051 GB	0.012 GB
`160k`	2.25 GB	$>$ 0.664 GB	0.035 GB
`RI`	25.2 GB	$>$ 3.45 GB	0.15 GB
`Texas`	$>$ 458 GB	$>$ 458 GB	2.49 GB

Table 10.1: Memory footprint in gigabytes of CPLEX interior-point method (IPM), CPLEX Simplex, and EGT algorithms. CPLEX requires more than 458 GB for the `Texas` instance.

10.2.2 Speedup from parallelizing the matrix-vector product

To address the time requirements of the matrix-vector product, we can effectively parallelize the operation by simply partitioning the work into n pieces when n CPUs are available. The speedup we can achieve on parallel CPUs is demonstrated in Table 10.2. The instance used for this test is the `Texas` instance described above. The matrix-vector product operation scales linearly in the number of CPUs, and the time to perform one iteration of the algorithm (using the entropy prox function and including the time for applying Heuristic 1) scales nearly linearly, decreasing by a factor of 3.72 when using 4 CPUs.

[1]The memory usage for the CPLEX simplex algorithm reported in Table 10.1 is the memory used after 10 minutes of execution (except for the `Texas` instance which did not run at all as described above). This algorithm's memory requirements grow and shrink during the execution depending on its internal data structures. Therefore, the number reported is a lower bound on the maximum memory usage during execution.

CPUs	matrix-vector product		EGT iteration	
	time (s)	speedup	time (s)	speedup
1	278.958	1.00x	1425.786	1.00x
2	140.579	1.98x	734.366	1.94x
3	92.851	3.00x	489.947	2.91x
4	68.831	4.05x	383.793	3.72x

Table 10.2: Effect of parallelization for the `Texas` instance.

10.3 Exploiting ccNUMA architecture

There has lately been a rapidly accelerating trend in computer architecture towards systems with many numbers of processors and cores. At the highest end of the computing spectrum, this is illustrated by the development of the *cache-coherent Non-Uniform Memory Access (ccNUMA)* architecture. A NUMA architecture is one in which different processors access different physical memory locations at different speeds. Each processor has fast access to a certain amount of memory (near it in practice), but if it accesses memory from another physical location the memory access will be slower. However, all of the memory is addressable from every processor.

A NUMA architecture is *cache-coherent* if the hardware makes sure that writes to memory in a physical location are immediately visible to all other processors in the system. This greatly simplifies the complexity of software running on the ccNUMA platform. In fact, code written using standard parallelization libraries on other platforms will usually work without modification on a ccNUMA system. However, as we will discuss in Section 10.3.1, to fully take advantage of the performance capabilities of the hardware, our matrix-vector multiplication algorithm for games needs to be redesigned somewhat. Since the matrix-vector product accounts for such a significant portion of the time (this is true even when using sampling as discussed in the previous section), developing an algorithm that scales well in the number of available cores could have a significant impact in practice.

In Section 10.3.1 we describe our implementation of a parallel matrix-vector product algorithm specialized for the ccNUMA architecture. We present experimental results in Section 10.3.2.

10.3.1 Details of our approach

Processors in a ccNUMA system have fast access to local memory, and slower access to memory that is not directly connected. Thus, it would be ideal for every processor to primarily access data from its local memory bank. The ccNUMA system maps virtual memory to physical memory based on which processor first writes to a particular memory location. Thus, a common technique when developing software for a ccNUMA platform is to include an initialization step in which all of the threads are created, and they allocate their own memory and write initialization values to every memory location.

We use this approach in our algorithm. Given that we have access to N cores, we partition the matrix into N equal-size pieces and communicate to each thread which segment of the matrix it should access. Then each thread allocates its own memory and loads the pertinent information describing its submatrix. Thus, each thread will have the most pertinent data loaded in the physical memory closest to the core on which it is running.

This approach may not be as effective if the memory requirements are severe. The machine we use in our experiments, for example, has 8 GB of fast (local) memory for every 4 cores. Thus, if there is a requirement for more than 2 GB per core, then the memory will—by necessity—have to be stored on non-adjacent memory banks, hampering performance. For the problem sizes in which we are presently interested, these memory limits are large enough—given that we use the decomposed matrix representation developed in [56]. (To be specific, the size of the matrix representation of the abstracted Texas Hold'em instance used in these experiments is 16GB overall, and it gets divided equally among the cores. Therefore, if we have at least 8 cores at our disposal, the memory image of each entire 4-core block fits in the block's fast 8GB memory.)

10.3.2 Experiments

We developed and tested our algorithm on a ccNUMA machine with 768 cores and 1.5 TB RAM. Due to the high demand for computing time on this valuable supercomputer, it is currently difficult to gain access to large segments of time, or a large number of cores at any given time. Thus, in our experiments we only present results where our software is using up to 64 cores. In the future, as ccNUMA machines become more prevalent, we plan to evaluate our approach on even greater numbers of cores.

We tested the time needed for computing a matrix-vector product for an abstracted instance of Texas Hold'em poker. We compared our algorithm against the standard parallelization approach that does not take into account the unique physical characteristics of the ccNUMA architecture. Figure 10.1 displays the results. Our new approach is always faster, and at 64 cores it is more than twice as fast.

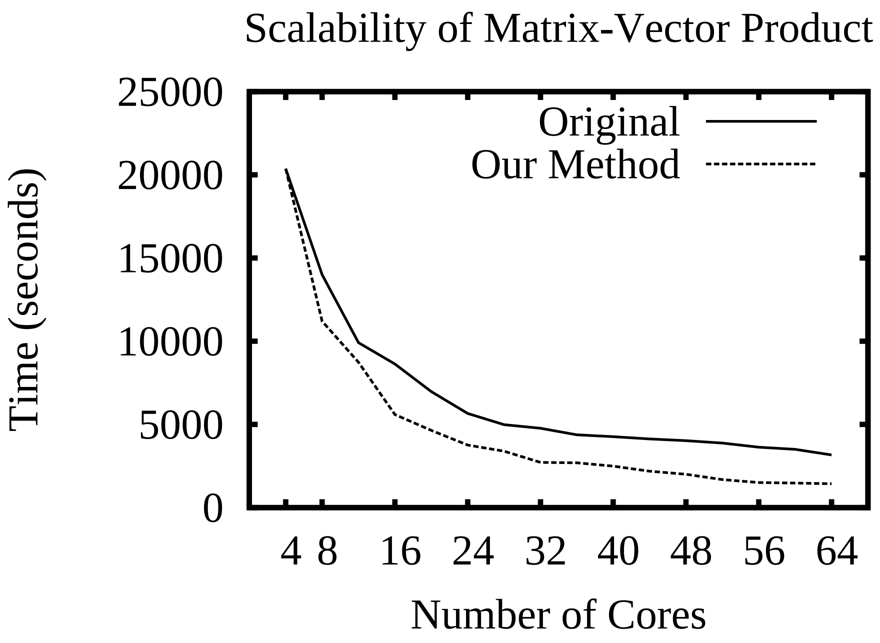

Figure 10.1: Computation time needed for one matrix-vector product computation as the number of available cores increases.

10.4 Summary

For poker games and similar games, we introduced a decomposed matrix representation that reduces storage requirements drastically. We also showed near-perfect efficacy of parallelization on a uniform-memory access architecture with four processors. For a non-uniform memory access architecture, we showed promising (but not linear) scalability up to 64 processors. Overall, our techniques enable one to solve orders of magnitude larger games than the prior state of the art.

Although current general-purpose simplex and interior-point solvers cannot handle problems of more than around 10^6 nodes [61], it is conceivable that specialized versions of these algorithms could be effective. However, taking advantage of the problem structure in these linear programming methods appears to be quite challenging. For example, a single interior-point iteration requires the solution of a symmetric non-definite system of equations whose matrix has the payoff matrix A and its transpose A^{T} in some blocks. Such a step is inherently far more complex than the simple matrix-vector multiplications required in gradient-based algorithms. On the upside, overcoming this obstacle would enable us to capitalize on the superb speed of convergence of interior-point methods. We leave the study of these alternative algorithms for Nash equilibrium finding as future work.

Chapter 11

Solving Repeated Games of Imperfect Information

11.1 Introduction

As discussed throughout this thesis, an important topic in computational game theory is the study of algorithms for computing equilibrium strategies for games. Without such algorithms, the elegant game theory solution concepts would have little to offer in the way of guidance to designers and implementers of game-theoretic agents. On the contrary, equipping agents with these algorithms would enable them to use strategies that are determined by a game-theoretic analysis. In many games—including all finite two-person zero-sum games—such strategies are optimal for the agent regardless of the opponents' actions.

Most work on algorithms for equilibrium-finding (including the work discussed until now in this thesis) has focused on the non-repeated setting in which a game is played only once. However, most agent interactions happen many times. For example, participants in a market will likely encounter the same buyers and sellers repeatedly (hence the importance of reputation). In this chapter, we explicitly study a setting that models repeated interactions.

In addition to encountering one another multiple times, agents are typically endowed with some private information about the state of the world, that is, which game is being played. As does most prior work, we model this private information by treating the game as one of incomplete information. What makes the work in this chapter unique is that we address computational considerations in repeated games of incomplete information, in which the agents' private information may be revealed over time—possibly inadvertently—

by their actions.

One stream of related research on algorithms for repeated games falls under the category of *multiagent learning* (*e.g.*, [52, 99, 24, 76]) which is usually applied either when the rules of the game are unknown (*e.g.*, when a player is initially completely uninformed about the payoffs of the game) or when directly solving for good strategies is too difficult in a computational sense [134]. In the setting we study in this chapter, both players know the rules of the game, and we demonstrate that it is not computationally infeasible to compute optimal strategies.

A closely related piece of work is due to Littman and Stone [100] who designed an algorithm for finding Nash equilibria in repeated games. That work studied the setting where both players know which stage game is being repeated. In our setting, the key difference is that only one of the players knows that.

In this chapter we study a model of repeated interaction known as *two-person zero-sum repeated games of incomplete information*. We first review the necessary theory (Section 11.2) and present some illustrative examples (Section 11.3) of this class of games. Following that, for the case where one player is *informed* about the state of the world and the other player is *uninformed*, we derive a non-convex mathematical programming formulation for computing the value of the game (Section 11.4.1). This is a complicated optimization problems for which standard optimization algorithms do not apply. We describe and analyze a novel efficient algorithm for solving this problem to within arbitrary accuracy in Section 11.4.2. In Section 11.5.1 we demonstrate how the solution to our optimization problem yields an optimal strategy for the informed player. We also give an algorithm for the uninformed player to play optimally, in Section 11.5.2. Finally, in Section 11.6 we conclude and present some open problems.

11.2 Preliminaries

In this section we review the definitions and concepts on which we will build. We first review in Section 11.2.1 some basic game theory for two-person zero-sum games with complete information (in which both players are fully informed about which game is being played). In Section 11.2.2 we review single-shot two-person zero-sum games with incomplete information (in which the players are only partially informed about the game being played). We conclude this section with the relevant theory of infinitely-repeated two-person zero-sum games with incomplete information (Section 11.2.3).

The material in this section is largely based on Aumann and Maschler's early work on the subject [6]. Myerson provides a textbook introduction [111] and Sorin provides a thorough treatment of two-person zero-sum repeated games [151].

11.2.1 Complete-information zero-sum games

A two-person zero-sum game with complete information is given by $A \in \mathbb{Q}^{m\times n}$ with entries A_{ij}.[1] In this game, player 1 plays an action in $\{1,\dots,m\}$ and player 2 *simultaneously* plays an action in $\{1,\dots,n\}$. If player 1 chooses i and player 2 chooses j, player 2 pays player 1 the amount A_{ij}. In general, the players may employ *mixed strategies* which are probability distributions over the available actions, denoted Δ_m and Δ_n, respectively, where

$$\Delta_m = \left\{ \mathbf{p} \in \mathbb{R}^m : \sum_{i=1}^{m} p_i = 1, \mathbf{p} \geq 0 \right\}$$

and similarly for Δ_n. If $\mathbf{x} \in \Delta_m$ and $\mathbf{y} \in \Delta_n$, player 2 pays player 1 the quantity $\mathbf{x}A\mathbf{y}$ in expectation, where we take $\mathbf{x}$ to be a row vector and $\mathbf{y}$ to be a column vector.

Player 1, wishing to maximize the quantity $\mathbf{x}A\mathbf{y}$, while knowing that player 2 is a minimizer, faces the following optimization problem:

$$\max_{\mathbf{x}\in\Delta_m} \min_{\mathbf{y}\in\Delta_n} \mathbf{x}A\mathbf{y}. \tag{11.1}$$

Similarly, player 2 faces the problem

$$\min_{\mathbf{y}\in\Delta_n} \max_{\mathbf{x}\in\Delta_m} \mathbf{x}A\mathbf{y}. \tag{11.2}$$

The celebrated *minimax theorem* states that the values of these two problems are equal and can be simultaneously solved [158]. Hence, we may consider the problem of solving the following equation:

$$\max_{\mathbf{x}\in\Delta_m} \min_{\mathbf{y}\in\Delta_n} \mathbf{x}A\mathbf{y} = \min_{\mathbf{y}\in\Delta_n} \max_{\mathbf{x}\in\Delta_m} \mathbf{x}A\mathbf{y}.$$

If $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ are solutions to the above problem then we say that $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ are *minimax solutions* and we define the value of the game $v(A) = \bar{\mathbf{x}}A\bar{\mathbf{y}}$. (It is easy to see that $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ also satisfy the weaker solution concept of Nash equilibrium, although in this chapter we focus on the stronger minimax solution concept.)

[1] We have already discussed complete information zero-sum games previously in this thesis. We repeat the presentation here to emphasize the relationship between the complete information theory and the incomplete information theory.

We can reformulate player 1's problem (11.1) as the following linear program:

$$\max\{z : z\mathbf{1} - \mathbf{x}A \leq 0, \mathbf{x}\mathbf{1} = 1, \mathbf{x} \geq 0\} \tag{11.3}$$

where $\mathbf{1}$ is the all-ones column vector of appropriate dimension. Similarly, player 2's problem can be formulated as:

$$\min\{w : w\mathbf{1} - A\mathbf{y} \geq 0, \mathbf{1}^T\mathbf{y} = 1, \mathbf{y} \geq 0\}. \tag{11.4}$$

Thus, we can find minimax solutions for both players in polynomial time using linear programming.[2]

Later in this chapter we also consider the *average* game which is given by a set of K possible games $\hat{A} = \{A^1, \ldots, A^K\}$ with $A^i \in \mathbb{Q}^{m\times n}$, and a probability distribution $\mathbf{p} \in \Delta_K$. The possible games $\hat{A}$ are common knowledge, but neither player knows which game is actually being played. The actual game is chosen according to $\mathbf{p}$. Without knowing the actual game, but knowing the distribution, both players play strategies $\mathbf{x} \in \Delta_m$ and $\mathbf{y} \in \Delta_n$. When the game A^i is drawn, player 2 pays player 1 $\mathbf{x}A^i\mathbf{y}$. Hence, the expected payment made in this game is

$$\sum_{i=1}^K p_i \mathbf{x}A^i\mathbf{y} = \sum_{i=1}^K \mathbf{x}(p_iA^i)\mathbf{y} = \mathbf{x}\left(\sum_{i=1}^K p_iA^i\right)\mathbf{y}.$$

Thus the average game is equivalent to playing the matrix game given by $A = \sum_{i=1}^K p_iA^i$ and we can compute the value of the average game using linear programming as applied to the matrix game A. We define the value of the matrix game as

$$v(\mathbf{p}, \hat{A}) = v\left(\sum_{i=1}^K p_iA^i\right).$$

11.2.2 Incomplete-information zero-sum games

A two-person zero-sum game with incomplete information is given by matrices $A^{kl} \in \mathbb{Q}^{m\times n}$ for each $k \in \{1, \ldots, K\}$ and $l \in \{1, \ldots, L\}$. In this game, k is drawn according to some common-knowledge distribution $\mathbf{p} \in \Delta_K$ and the value k is communicated to player 1 (but not player 2). Similarly, l is drawn according to some common-knowledge distribution $\mathbf{q} \in \Delta_L$ and is communicated to player 2 only. Having learned their respective

[2]We note that (11.3) and (11.4) are duals of each other and hence the strong duality theorem of linear programming can be used to prove the minimax theorem. See [32, pp. 230–233] for more details on the relationship between linear programming and zero-sum games.

private values of k and l, both players play mixed strategies $\mathbf{x}^k$ and $\mathbf{y}^l$ yielding the expected payment $\mathbf{x}^k A^{kl} \mathbf{y}^l$ from player 2 to player 1. Given the probability distributions $\mathbf{p}$ and $\mathbf{q}$ and strategies $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^K\}$ and $\mathbf{Y} = \{\mathbf{y}^1, \ldots, \mathbf{y}^L\}$, the expected payment made in the game is

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{l=1}^{L} p_k q_l A_{ij}^{kl} x_i^k y_j^l.$$

There is a linear programming formulation for finding minimax solutions for two-person zero-sum games with incomplete information [125], but we do not describe it here as we do not need it in this chapter.

11.2.3 Repeated incomplete-information zero-sum games

A repeated two-person zero-sum game with incomplete information is defined by the same data as the single-shot setting described above, but the game is played differently. As before, k is drawn according to $\mathbf{p} \in \Delta_K$ and communicated to player 1 only, and l is drawn according to $\mathbf{q} \in \Delta_L$ and communicated to player 2 only. Now, however, the game A^{kl} is repeated. After each stage i, the players only observe each other's actions, s_1^i and s_2^i. In particular, the payment $A^{kl}_{s_1^i s_2^i}$ in each round i is not observed. (If it were, the players would quickly be able to "reverse engineer" the actual state of the world.)

The class of games described in the previous paragraph are two-person zero-sum repeated games *with lack of information on both sides*, since both players 1 and 2 are uninformed about some aspect of the true state of the world. In this chapter, we limit ourselves to two-person zero-sum repeated games *with lack of information on one side*. In this setting, player 1 (the *informed player*) is told the true state of the world. Player 2, the *uninformed player*, is not told anything, *i.e.,* we assume $|L| = 1$. We must consider this more specific class of games for several reasons.

- The first reason is that the notion of *value* in games with lack of information on both sides is unclear. There are two approaches to studying the repeated games in this model. The first is to consider the n-stage game, denoted Γ_n, and its value $v(\Gamma_n)$ (which clearly exists since Γ_n is finite), and then determine $\lim_{n\to\infty} v(\Gamma_n)$, if it exists. (Observe that Γ_1 is the same as the game described in Section 11.2.2.) The second approach is to consider the infinitely-repeated game, denoted Γ_∞, and determine its value $v(\Gamma_\infty)$ directly, if it exists. In two-person zero-sum repeated games with lack of information on one side, both $\lim_{n\to\infty} v(\Gamma_n)$ and $v(\Gamma_\infty)$ exist and are equal, so either approach is suitable [6]. However, there are games with lack of information on both sides for which $v(\Gamma_\infty)$ does not exist (although it is known that $\lim_{n\to\infty} v(\Gamma_n)$ exists [107]).

- The second reason involves the choice of modeling *payoffs* in repeated games with incomplete information. As discussed above, the players do not observe the payoffs in each stage. How then does one even evaluate a player's strategy or consider the notion of value? It is customary to consider the mean-payoff, so we say that after the N-th stage, player 1 receives a payoff

  $$\frac{1}{N}\sum_{i=1}^{N} A^{kl}_{s_1^i s_2^i}$$

  from player 2.

  As an alternative to the mean-payoff model, one could also study the seemingly more natural λ-discounted game Γ_λ as $\lambda \to 0$. This game is repeated infinitely many times and player 1 receives the payoff

  $$\sum_{i=1}^{\infty} \lambda(1-\lambda)^{i-1} A^{kl}_{s_1^i s_2^i}$$

  from player 2. It can be shown that

  $$v(\Gamma_\infty) = \lim_{n\to\infty} v(\Gamma_n) = \lim_{\lambda\to 0} v(\Gamma_\lambda)$$

  where $v(\Gamma_\infty)$ exists [151, Lemma 3.1]. Since, as discussed in the previous point, this condition holds for games with lack of information on one side, we have that the value of the game is the same regardless of which payoff metric we choose.

  Given this flexibility in choice of payoff measure, we simply restrict ourselves to studying the mean-payoff game for computational reasons.

- Finally, as we will elaborate in Section 11.4, our optimization approach for computing optimal strategies in the game depends heavily on a characterization of equilibrium strategies that unfortunately only holds for games with lack of information on one side.

In summary, we limit ourselves to the one-sided lack of information setting for various conceptual and computational reasons. Developing concepts and algorithms for the more general two-sided setting is an important area of future research which we discuss further in Section 11.6.

11.3 Examples

Before describing our algorithms for computing both the value and optimal strategies for games in our model, we discuss some classical examples that illustrate the richness and complexity of our model.

In any repeated game with incomplete information, the players must take into account to what extent their actions reveal their private information, and to what extent this revelation will affect their future payoffs. In the following three subsections, we present three examples where the amount of information revelation dictated by the optimal strategy differs. The first two examples are due to Aumann and Maschler [6] and the third is due to Zamir [164].

11.3.1 Completely unrevealing strategy

Our first example is given in Figure 11.1. There are two possible states of the world, A and B, and each is chosen by nature with probability 0.5. The true state of the world is communicated to player 1.

State A

	L	R
U	1	0
D	0	0

State B

	L	R
U	0	0
D	0	1

Figure 11.1: The stage games for the unrevealing example. If the state of the world is State A, then the game on the left is played. Otherwise, the game on the right is played.

Consider what happens if player 1, after being informed that the true state is A, plays U every time. (This is a weakly dominant strategy for player 1 in the state A game.) Eventually it will occur to player 2, who is observing player 1's actions, that player 1 is playing U because the players are in state A and player 1 is hoping to get the payoff of 1. Observing this, player 2 will switch to playing R, guaranteeing a payoff of 0.

A similar line of reasoning in the case of state B appears to demonstrate that player 1 can only achieve a long-term payoff of 0 as player 2 will eventually figure out what actual game is being played. However, somewhat unintuitively, consider what happens if player 1 *ignores* her private signal. Then no matter what strategy player 1 uses, player 2 will not be able to infer the game being played. In fact, it as if the players are playing the *average* game in Figure 11.2. In this game, both players' (unique) optimal strategy is to play both of their actions with probability 0.5, for an expected payoff of 0.25. Thus player 1 achieves an average expected payoff of 0.25, compared to the payoff of 0 that she would get if player 2 were able to infer the actual game from player 1's action history. Note that player 1's strategy is *completely non-revealing* since player 2 will never be able to determine the true state of the world based on player 1's actions.

	L	R
U	1/2	0
D	0	1/2

Figure 11.2: The *average* game corresponding to the case where player 1 ignores her private information in the game in Figure 11.1.

Although we do not prove it here, the above strategy for player 1 is optimal. Intuitively, if player 1 were to slightly alter her strategy to take advantage of her private information, player 2 would observe this and would then deviate to the unique best response to player 1's altered strategy. Thus, any advantage player 1 could possibly get would be short-term, and would not be nearly enough to compensate for the long-term losses that player 2 would be able to inflict.

11.3.2 Completely revealing strategy

Our second example is given in Figure 11.3. Again, there are two possible states of the world, A and B, and each is chosen by nature with probability 0.5. The true state of the world is communicated to player 1.

State A

	L	R
U	-1	0
D	0	0

State B

	L	R
U	0	0
D	0	-1

Figure 11.3: The stage games for the revealing example. If the state of the world is State A, then the game on the left is played. Otherwise, the game on the right is played.

Here, a payoff of 0 is clearly the best player 1 can hope for, and this outcome can be achieved using the following strategy: "Always play D if the state of the world is A;

otherwise always play U". No matter what strategy player 2 uses, player 1 obtains a payoff of 0. Since this is the highest possible payoff, this is clearly an optimal strategy. Note that this strategy is *completely revealing* since player 2 will be able to determine the true state of the world.

11.3.3 Partially revealing strategy

Our third example is given in Figure 11.4. Again, there are two possible states of the world, A and B, and each is chosen by nature with probability 0.5. The true state of the world is communicated to player 1.

State A

	L	M	R
U	4	0	2
D	4	0	-2

State B

	L	M	R
U	0	4	-2
D	0	4	2

Figure 11.4: The stage games for the partially-revealing example. If the state of the world is State A, then the game on the left is played. Otherwise, the game on the right is played.

If, as in the first example, player 1 completely ignores her private information, the game reduces to the one in Figure 11.5. In this case, player 2 can always play R and thus guarantee a payout of at most 0.

State A

	L	M	R
U	2	2	0
D	2	2	0

Figure 11.5: The *average* game corresponding to the case where player 1 ignores her private information in the game in Figure 11.4.

If, on the other hand, player 1 completely reveals her private information, then player 2 will have the following optimal strategy: "Always play M if the inferred state is A; otherwise always play L". Again, player 2 is able to guarantee a maximum payout of 0.

Suppose player 1 now employs the following strategy: "If the state of the world is A, then with probability 0.75 always play U, otherwise always play D; if the state of the world is B, then with probability 0.25 always play U, otherwise always play D". Suppose further that player 2 knows this strategy. If player 2 observes that player 1 is always playing U, then player 2 can infer that

$$\begin{aligned} Pr[A|U] &= \frac{Pr[U|A]Pr[A]}{Pr[U]} \\ &= \frac{Pr[U|A]Pr[A]}{Pr[U,A]+Pr[U,B]} \\ &= \frac{0.75 \cdot 0.5}{0.5 \cdot 0.75 + 0.5 \cdot 0.25} \\ &= \frac{3}{4}. \end{aligned}$$

Hence, player 2 is faced with the decision problem

	L	M	R
U	3	1	1

and so can do by best by achieving a payout of 1. A similar computation shows the same is true if player 2 observes player 1 to always be playing D. Therefore, player 1 achieves a payoff of 1, which is better than she would have done had she either completely revealed her private information or completely ignored her private information. By *partially revealing* this information, she has boosted her payoff.

11.4 Optimization formulation

In this section, we review Aumann and Maschler's theorem for characterizing the value of two-person zero-sum repeated games with lack of information on one side. Unfortunately, this theorem does not include an algorithm for actually computing the value. Using this characterization we derive a non-convex optimization problem for computing the value of the game (Section 11.4.1). Non-convex optimization problems are in general $\mathcal{NP}$-complete [54] so there is little hope of employing a general-purpose algorithm for solving non-convex optimization problems. Instead, we give a specialized algorithm that computes the value of the game to within additive error ϵ for any given target accuracy $\epsilon > 0$ (Section 11.4.2). For games with a constant number of world states (but with a non-constant number of actions available to the players) we can compute such a solution in time polynomial in the number of actions and $\frac{1}{\epsilon}$.

11.4.1 Derivation of the optimization formulation

Consider the two-person zero-sum game with incomplete information given by matrices $A^k \in \mathbb{Q}^{m\times n}$ for $k \in \{1,\ldots,K\}$, and let $\mathbf{p}^* \in \Delta_K$ be the probability with which k is chosen and communicated to player 1. (This is a game with lack of information on one side only, so player 2 does not receive any information about the true state of the world.) Denote the infinitely-repeated version of this game as Γ_∞. We are interested in computing $v(\Gamma_\infty)$.

Recall the *average game* which for some $\mathbf{p} \in \Delta_K$ has payoff matrix

$$A(\mathbf{p}) = \sum_{i=1}^{K} p_i A^i.$$

As discussed in Section 11.2.1, the value of this game is $v(\mathbf{p}, A) = v(A(\mathbf{p}))$ and can be computed using LP. In what follows we omit A when it can be inferred from context, and instead simply discuss the value function $v(\mathbf{p})$.

Consider now the *concavification* of $v(\mathbf{p})$ with respect to $\mathbf{p}$, that is, the point-wise smallest (with respect to $v(\cdot)$) concave function that is greater than v for all $\mathbf{p} \in \Delta_K$. Letting $v' \geq v$ denote that $v'(\mathbf{p}) \geq v(\mathbf{p})$ for all $\mathbf{p} \in \Delta_K$, we can formally write

$$\mathbf{cav}\ v(\mathbf{p}) = \inf_{v'} \left\{ v'(\mathbf{p}) : v' \text{ concave}, v' \geq v \right\}.$$

Aumann and Maschler's surprising and elegant result [6] states that $v(\Gamma_\infty) = \mathbf{cav}\ v(\mathbf{p})$. Our goal of computing $v(\Gamma_\infty)$ can thus be achieved by computing $\mathbf{cav}\ v(\mathbf{p})$.

A basic result from convex analysis [28] shows that the convex hull of an n-dimensional set S can be formed by taking convex combinations of $n+1$ points from S. Hence, the K-dimensional point $(\mathbf{p}, \mathbf{cav}\ v(\mathbf{p}))$ can be represented as the convex combination of $K+1$ points $(\mathbf{p}^i, v(\mathbf{p}^i))$, $i \in \{1,\ldots,K+1\}$. (Note that $\mathbf{p}$ is $(K-1)$-dimensional, not K-dimensional.) In particular, for any $(\mathbf{p}, \mathbf{cav}\ v(\mathbf{p}))$, there exists $\alpha \in \Delta_{K+1}$ and points

$$\left\{ \left(\mathbf{p}^1, v\left(\mathbf{p}^1\right)\right), \ldots, \left(\mathbf{p}^{K+1}, v\left(\mathbf{p}^{K+1}\right)\right) \right\}$$

such that

$$\mathbf{p} = \sum_{i=1}^{K+1} \alpha_i \mathbf{p}^i$$

and

$$\mathbf{cav}\ v(\mathbf{p}) = \sum_{i=1}^{K+1} \alpha_i v(\mathbf{p}^i).$$

Hence, we can rewrite the problem of computing $\mathbf{cav}\ v(\mathbf{p})$ as the following optimization problem:

$$
\text{(P1)} \qquad
\begin{array}{ll}
\max & \displaystyle\sum_{i=1}^{K+1} \alpha_i v(\mathbf{p}^i) \\
\text{such that} & \displaystyle\sum_{i=1}^{K+1} \alpha_i \mathbf{p}^i = \mathbf{p} \\
& \mathbf{p}^i \in \Delta_K \text{ for } i \in \{1, \ldots, K+1\} \\
& \alpha \in \Delta_{K+1}
\end{array}
$$

A solution to Problem P1 therefore yields the value of Γ_∞. Unfortunately, this does not immediately suggest a good algorithm for solving this optimization problem. First, the optimization problem depends on the quantities $v(\mathbf{p}^i)$ for variables $\mathbf{p}^i$. As discussed in Section 11.2.1, the value $v(\mathbf{p}^i)$ is itself the solution to an optimization problem (namely a linear program), and hence a closed-form expression is not readily available. Second, the first constraint is non-linear and non-convex. Continuous optimization technology is much better suited for convex problems [116], and in fact non-convex problems are $\mathcal{NP}$-complete in general [54]. In the following subsection, we present a numerical algorithm for solving Problem P1 with arbitrary accuracy ϵ.[3]

11.4.2 Solving the formulation

Our algorithm is closely related to *uniform grid methods* which are often used for solving extremely difficult optimization problems when no other direct algorithms are available [116]. Roughly speaking, these methods discretize the feasible space of the problem and evaluate the objective function at each point. Our problem differs from the problems normally solved via this approach in two ways. The first is that the feasible space of Problem P1 has additional structure not normally encountered. Most uniform grid methods have

[3]Unfortunately, our algorithm cannot solve Problem P1 exactly, but rather only to within additive error ϵ. However, this appears unavoidable since there are games for which the (unique) optimal play of player 1 involves probabilities that are irrational numbers. An example of such a game is due to Aumann and Maschler [6, p. 79–81]. This immediately implies that there does not exist a linear program whose coefficients are arithmetically computed from the problem data whose solution yields an optimal strategy to Problem P1 since every linear program with rational coefficients has a rational solution. Furthermore, any numerical algorithm will not be able to compute an exact solution for similar reasons. Note that this is very different from the case of two-person zero-sum games in which optimal strategies consisting of rational numbers as probabilities always exist.

a hyper-rectangle as the feasible space. In contrast, our feasible space is the product of several simplices of different dimension, which are related to each other via a non-convex equality constraint (the first constraint in Problem P1). Second, evaluating the objective function of Problem P1 is not straightforward as it depends on several values of $v(\mathbf{p}^i)$ which are themselves the result of an optimization problem.

As above we consider a game given by K matrices A^k, but now for simplicity we require the rational entries A^k_{ij} to be in the unit interval $[0,1]$. This is without loss of generality since player utilities are invariant with respect to positive affine transformations and so the necessary rescaling does not affect their strategies.

Our algorithm for solving Problem P1 within additive error ϵ proceeds as follows.

Procedure SolveP1

1. Let $C = \lceil \frac{1}{\epsilon} \rceil$.
2. Let $\mathbf{Z}(C) = \left\{ \left(\frac{d_1}{C}, \ldots, \frac{d_K}{C} \right) : d_i \in \mathbb{N}, \sum_{i=1}^{K} d_i = C \right\}$. Denote the points of $\mathbf{Z}(C)$ as $\mathbf{p}^1, \ldots, \mathbf{p}^{|\mathbf{Z}(C)|}$.
3. For each point $\mathbf{p}^i \in \mathbf{Z}(C)$, compute and store $v(\mathbf{p}^i)$ using linear program (11.3).
4. Solve linear program P2:

$$\begin{array}{lll} & \max & \displaystyle\sum_{i=1}^{|\mathbf{Z}(C)|} \alpha_i v(\mathbf{p}^i) \\ (\mathrm{P2}) & \text{such that} & \displaystyle\sum_{i=1}^{|\mathbf{Z}(C)|} \alpha_i \mathbf{p}^i \leq \mathbf{p} \\ & & \alpha \in \Delta_{|\mathbf{Z}(C)|} \end{array}$$

5. Output the value of linear program P2 as an approximation of the value of the game.

We now analyze the above algorithm. We first recall the following classic fact about the value function $v(\mathbf{P})$.

Lemma 11 *$v(\mathbf{p})$ is Lipschitz with constant 1, that is,*

$$|v(\mathbf{p}) - v(\mathbf{p}')| \leq \|\mathbf{p} - \mathbf{p}'\|_\infty.$$

PROOF. This is immediate in our case since we assume that all entries of A are in the range $[0, 1]$. □

Before presenting our main theorem, we present a technical lemma which shows that our discretization (*i.e.,* our design of $\mathbf{Z}(C)$, is sufficiently refined. The proof is immediate from the definition of $\mathbf{Z}(C)$ and is omitted.

Lemma 12 *Let $\mathbf{p} \in \Delta_K$, $\epsilon > 0$, and $C = \lceil \frac{K}{\epsilon} \rceil$. There exists $\mathbf{q} \in \mathbf{Z}(C)$ such that*

$$\|\mathbf{p} - \mathbf{q}\|_\infty \leq \frac{1}{C}.$$

Theorem 13 *Let $v(\Gamma_\infty)$ be the value of the infinitely-repeated game and let v^* be the value output by the above algorithm with input $\epsilon > 0$. Then*

$$v(\Gamma_\infty) - v^* \leq \epsilon.$$

PROOF. We only need to show that there exists a feasible solution to the linear program P2 whose objective value satisfies the inequality (the optimal answer to the linear program could be even better). Let $\bar{\alpha}, \bar{\mathbf{p}}^1, \ldots, \bar{\mathbf{p}}^{K+1}$ be optimal solutions to Problem P1. We construct a feasible solution α to linear program P2 as follows. For each $i \in \{1, \ldots, K+1\}$, choose $\mathbf{p}^j \in \mathbf{Z}(C)$ such that $\|\bar{\mathbf{p}}^i - \mathbf{p}^j\|_\infty$ is minimized (breaking ties arbitrarily) and set $\alpha_j = \bar{\alpha}_i$. Assign $\eta(i) = j$. Leave all other entries of α zero. Let $N = \{i : \alpha_i > 0\}$ be the index set for the positive entries of α and let

$$v = \sum_{i=1}^{|\mathbf{Z}(C)|} \alpha_i v(\mathbf{p}^i).$$

Clearly, this is a lower bound on the objective value of linear program P2. Now we can write:

$$\begin{aligned}
v(\Gamma_\infty) - v &= \sum_{i=1}^{K+1} \bar{\alpha}_i v\left(\bar{\mathbf{p}}^i\right) - \sum_{i=1}^{|\mathbf{Z}(C)|} \alpha_i v\left(\mathbf{p}^i\right) \\
&= \sum_{i=1}^{K+1} \bar{\alpha}_i v\left(\bar{\mathbf{p}}^i\right) - \sum_{i \in N} \alpha_i v\left(\mathbf{p}^i\right) \\
&= \sum_{i=1}^{K+1} \bar{\alpha}_i v\left(\bar{\mathbf{p}}^i\right) - \sum_{i=1}^{K+1} \alpha_{\eta(i)} v\left(\mathbf{p}^{\eta(i)}\right) \\
&= \sum_{i=1}^{K+1} \bar{\alpha}_i v\left(\bar{\mathbf{p}}^i\right) - \sum_{i=1}^{K+1} \bar{\alpha}_{\eta(i)} v\left(\mathbf{p}^{\eta(i)}\right) \\
&= \sum_{i=1}^{K+1} \bar{\alpha}_i \left[v\left(\bar{\mathbf{p}}^i\right) - v\left(\mathbf{p}^{\eta(i)}\right)\right] \\
&\leq \sum_{i=1}^{K+1} \bar{\alpha}_i \left\|\bar{\mathbf{p}}^i - \mathbf{p}^{\eta(i)}\right\|_\infty \\
&\leq \sum_{i=1}^{K+1} \bar{\alpha}_i \frac{1}{C} = \frac{1}{C} \leq \epsilon
\end{aligned}$$

The first inequality is by Lemma 11, the second inequality is by Lemma 12, and the third inequality is by the definition of C. □

We now analyze the time complexity of our algorithm. We first state a simple lemma.

Lemma 14 *For $C = \lceil \frac{1}{\epsilon} \rceil$ the set*

$$\mathbf{Z}(C) = \left\{ \left(\frac{d_1}{C}, \ldots, \frac{d_K}{C} \right) : d_i \in \mathbb{N}, \sum_{i=1}^{K} d_i = C \right\}$$

defined in step 2 of the above algorithm satisfies

$$|\mathbf{Z}(C)| = \binom{C+K-1}{K-1} \leq (C+1)^K.$$

We analyze our algorithm in terms of the number of linear programs it solves. Note that each linear program is solvable in polynomial-time, *e.g.*, by the ellipsoid method [81] or by interior-point methods [163]. Step 3 of the algorithm clearly makes $|\mathbf{Z}(C)|$ calls to a linear program solver. By Lemma 14, we have $|\mathbf{Z}(C)| \leq (C+1)^K$. Each of these linear program has $m+1$ variables and $n+1$ constraints, where m and n are the numbers of actions each player has in each stage game.

Similarly, the linear program solved in step 4 of the algorithm also has at most $|\mathbf{Z}(C)| \leq (C+1)^K$ variables. Hence we have:

Theorem 15 *The above algorithm solves $(C+1)^K$ linear program that are of size polynomial in the size of the input data, and solves one linear program with $(C+1)^K$ variables.*

Therefore, for a fixed number of possible states K, our algorithm runs in time polynomial in the number of actions available to each player and in $\frac{1}{\epsilon}$.

11.5 Finding the players' strategies

In this section, we demonstrate how we can use the output of our algorithm to construct the player's strategies. The strategy of player 1 (the informed player) can be constructed explicitly from the values of the variables in the linear program P2 solved in our algorithm. Player 2 (the uninformed player) does not have such an explicitly represented strategy. However, we can use existing approaches (in conjunction with the output from our algorithm) to describe a simple algorithmic procedure for player 2's strategy.

11.5.1 The informed player's strategy

As alluded to in the examples, player 1's optimal strategy is of the following form. Based on the revealed choice of nature, player 1 performs a type-dependent lottery to select some distribution $\mathbf{q} \in \Delta_K$. Then she always plays as if the stage game were the average game induced by a distribution that does not reveal the true state of the world to player 2.

Let $\alpha, \mathbf{p}^1, \ldots, \mathbf{p}^{K+1}$ be solutions to Problem P1. The following strategy is optimal for player 1 [6].

> Let $k \in \{1, \ldots, K\}$ be the state revealed (by nature) to player 1. Choose $i \in \{1, \ldots, K+1\}$ with probability
>
> $$\frac{\alpha_i p_k^i}{p_k}$$
>
> where $\mathbf{p} = \{p_1, \ldots, p_k\}$ is the probability that natures chooses state k. Play the mixed equilibrium strategy corresponding to the average game given by distribution $\mathbf{p}^i$ in every stage.

Thus the informed player is using her private information once and for all at the very beginning of the infinitely-repeated game, and then playing always as if the game were actually the average game induced by the distribution $\mathbf{p}^i$. The strength of this strategy lies in the fact that player 2, after observing player 1's strategy, is unable to determine the actual state of nature *even after learning which number i player 1 observed in her type-dependent lottery*.

This surprisingly and conceptually simple strategy immediately suggests a similar strategy for player 1 based on the output of our algorithm. Let α be a solution to linear program P2 solved during the execution of our algorithm, let $\{\mathbf{p}^1, \dots, \mathbf{p}^{|\mathbf{Z}(C)|}\} = \mathbf{Z}(C)$, and let k be the actual state of nature. The strategy is as follows: "Choose $i \in \{1, \dots, |\mathbf{Z}(C)|\}$ with probability

$$\frac{\alpha_i p_k^i}{p_k}.$$

Play a mixed equilibrium strategy to the average game corresponding to the distribution $\mathbf{p}^i$ in every stage thereafter". Using reasoning completely analogous to the reasoning of Aumann and Maschler [6], this strategy guarantees player 1 a payoff of at least $v(\Gamma_\infty) - \epsilon$. (If we were able to solve Problem P1 optimally then we would have $\epsilon = 0$.)

11.5.2 The uninformed player's strategy

We now describe how player 2's strategy can be constructed from the solution to our algorithm. Unlike in the case of player 1, there is no concise, explicit representation of the strategy. Rather the prescription is in the form of an algorithm.

The driving force behind this technique is Blackwell's approachability theory [20] which applies to games with vector payoffs.[4] The basic idea is that player 2, instead of attempting to evaluate her expected payoff, instead considers her *vector payoff*, and then attempts to force this vector payoff to *approach* some set.

Because $\mathbf{cav}\ v(\mathbf{p})$ is concave, there exists $\mathbf{z} \in \mathbb{R}^K$ such that

$$\sum_{i=1}^{K} p_i z_i = \mathbf{cav}\ v(\mathbf{p})$$

and

$$\sum_{i=1}^{K} q_i z_i \geq \mathbf{cav}\ v(\mathbf{q}), \quad \forall \mathbf{q} \in \Delta_K.$$

Let $S = \left\{\mathbf{s} \in \mathbb{R}^K | \mathbf{s} \leq \mathbf{z}\right\}$. Blackwell's approachability theorem states that there exists a strategy for player 2 such that for *any* strategy of player 1, player 2 can receive a vector payoff arbitrarily close (in a precise sense) to the set S. Since S can be interpreted as the

[4]We do not include a full description of this theory here. Complete descriptions are provided by Myerson [111, pp. 357–360], Sorin [151, Appendix B], or Blackwell's original paper [20].

set of affine functions majorizing $v(\mathbf{p})$ (and, hence, majorizing $\mathbf{cav}\ v(\mathbf{p})$), then player 2 can force a payout arbitrarily close to the payoff that player 1 guarantees.

Following from the above discussion, we can state the following optimal strategy for player 2. For each stage n and each $i \in K$, let u_n^i be player 2's payout to player 1 (given that the state of the world is actually i). Now define

$$w_n^i = \frac{\sum_{j=1}^n u_j^i}{n}$$

to be player 2's average payoff vector to player 1. In stage 1 and in any stage n where $\mathbf{w}_n = (w_n^1, \ldots, w_n^K) \in S$, let player 2 play an arbitrary action (this is acceptable since so far player 2 is doing at least as well as possible). At stage n where $\mathbf{w}_n \notin S$, let player 2 choose her move according to a distribution $\mathbf{y}$ satisfying

$$\min_{\mathbf{y} \in \Delta_n} \max_{\mathbf{x} \in \Delta_m} \sum_{i=1}^{K} \left(w_{n-1}^i - \varepsilon^i(\mathbf{w}_{n-1})\right) \mathbf{x} A^i \mathbf{y} \tag{11.5}$$

where $\varepsilon(\mathbf{w}_{n-1})$ is the (unique) point in S that is closest to $\mathbf{w}_{n-1}$. Blackwell's approachability theorem [20] states that player 2's vector payoff converges to S regardless of player 1's strategy.

The above discussion thus shows how player 2, using the information output by our algorithm, can be used to generate a strategy achieving the optimal payoff. In each stage, at most all that is required is solving an instance of Equation 11.5, which can be solved in polynomial time using linear programming.

11.6 Conclusions and future directions

In this chapter we studied computational approaches for finding optimal strategies in repeated games with incomplete information. In such games, an agent must carefully weigh the tradeoff between exploiting its information to achieve a short-term gain versus carefully concealing its information so as not to give up a long-term informed advantage. Although the theoretical aspects of these games have been studied, this is the first work to develop algorithms for *solving* for optimal strategies. For the case where one player is *informed* about the true state of the world and the other player is *uninformed*, we derived a non-convex mathematical programming formulation for computing the value of the game, as well as optimal strategies for the informed player. We then described an algorithm for solving this difficult optimization problem to within arbitrary accuracy. We also described a method for finding the optimal strategy for the uninformed player based on the output of the algorithm.

Directions for future research are plentiful. This chapter has only analyzed the case of one-sided information. Developing algorithms for the case of lack of information on both sides would be an interesting topic. However, this appears difficult. For one, the notion of the value of the game is less well understood in these games. Furthermore, there is no obvious optimization formulation that models the equilibrium problem (analogous to Problem P1). Other possible directions include extending this to non-zero-sum games [72] as well as to games with many players. Again, these tasks appear difficult as the characterizations of equilibrium strategies become increasingly complex.

Yet another possible algorithmic approach would be to tackle the problem via Fenchel duality (Rockafellar [129] is a standard reference for this topic). Fenchel duality has been employed as an alternative method of proving various properties about repeated games [41, 42]. In particular, the Fenchel biconjugate of $v(\mathbf{p})$ yields $\mathbf{cav}\ v(\mathbf{p})$. Given the close relationship between Fenchel duality and optimization theory, an intriguing possibility would be to use Fenchel duality to derive an improved optimization algorithm for the problem studied in this chapter.

The class of games we study is a special case of the more general class of *stochastic games* [145]. That class of games allows for a much richer signaling structure (rather than the limited signaling structure we consider in which only the players' actions are observable), as well as transitioning to different stage games based on the choices of the players and possible chance moves. Developing a solid algorithmic understanding of the issues in those richer games is another important area of future research.

Part IV

Summary

Chapter 12

Summary

In this thesis we presented several algorithms for abstracting and solving sequential imperfect information games.

On the abstraction side, we investigated three broad families of abstraction: *information abstraction*, *action abstraction*, and *stage abstraction*. Our abstraction techniques apply to n-person general-sum games. Within information abstraction, we developed both lossless and lossy abstraction algorithms. In the context of lossless abstraction, we introduced the ordered game isomorphic abstraction transformation. Our lossless abstraction algorithm, *GameShrink*, exhaustively applies the transformations and we used it to solve Rhode Island Hold'em poker, a game tree with more than 50 million nodes. At the time of our solving this game, it was the largest sequential imperfect information game solved by over four orders of magnitude. In the context of lossy information-based abstraction, we developed four increasingly sophisticated and effective algorithms that take into account strategically-relevant aspects of the game. The first algorithm is a relaxation of our *GameShrink* lossless abstraction algorithm. The second algorithm applies k-means clustering and integer programming to find optimized, balanced abstractions. The third algorithm extends the second algorithm to incorporate potential into the metric for identifying strategically similar states. The fourth algorithm makes use of strategy-based abstraction. We performed controlled experiments to identify where each of the algorithms are best applicable.

In the context of no-limit Texas Hold'em poker, we developed and applied action abstraction for discretizing the huge action space. In addition to developing this action abstraction, we also developed a method for mapping real-world actions into our model's representation of actions.

We evaluated all of our abstraction algorithms in the context of poker, and our algorithms have led to the creation of poker-playing programs that are among the very best in the world. In the 2008 AAAI Computer Poker Competition, the latest versions of our limit and no-limit Texas Hold'em programs won the most chips overall.

On the equilibrium-finding side, we developed new algorithms for finding ϵ-equilibria in two-person zero-sum games. In one algorithm, we adapted Nesterov's excessive gap technique to the equilibrium-finding problem, and we introduced heuristics which speed up the algorithm in practice, while not sacrificing the theoretical worst-case guarantees of the basic algorithm. Our experiments indicate an order of magnitude in speed improvement through the use of our heuristics.

We also presented a gradient-based algorithm for finding ϵ-equilibria that is exponentially faster than the prior gradient-based algorithms in terms of its dependence on ϵ. The worst-case number of iterations needed to find an ϵ-equilibria for that algorithm is $\mathcal{O}(\log 1/\epsilon)$ which matches the worst-case guarantees of memory-intensive interior-point methods.

On the implementation side, we developed a novel matrix representation technique that dramatically reduces the memory requirements of our gradient-based algorithms. The key insight in this technique was the separation of player actions from informational signals. We developed a sampling technique that allows us to find ϵ-equilibria even more rapidly, and as an orthogonal technique we also developed parallelization schemes that allow our algorithm to take advantage of the modern supercomputing architecture ccNUMA. All of these techniques apply to all of the gradient-based algorithms we discussed in this thesis.

In a slightly tangential research direction, we also developed an equilibrium-finding algorithm for the particular case of repeated games of imperfect information with asymmetric information.

Bibliography

[1] Tim Abbott, Daniel Kane, and Paul Valiant. On the complexity of two-player win-lose games. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2005.

[2] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Math. Annalen*, 99:118–133, 1928.

[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] David Applegate, Guy Jacobson, and Daniel Sleator. Computer analysis of sprouts. Technical Report CMU-CS-91-144, Carnegie Mellon University, 1991.

[6] Robert J. Aumann and Michael Maschler. *Repeated Games with Incomplete Information*. MIT Press, 1995. With the collaboration of R. Stearns. This book contains updated versions of four papers originally appearing in *Report of the U.S. Arms Control and Disarmament Agency*, 1966–68.

[7] Richard Bellman. On games involving bluffing. *Rendiconti del Circolo Matematico di Palermo*, 1(2):139–156, 1952.

[8] Richard Bellman and David Blackwell. Some two-person games involving bluffing. *Proceedings of the National Academy of Sciences*, 35:600–605, 1949.

[9] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*. Academic Press, New York, 1983.

[10] Nivan A. R. Bhat and Kevin Leyton-Brown. Computing Nash equilibria of action-graph games. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Banff, Canada, 2004. AUAI Press.

[11] Daniel Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems*. Kluwer International Series, Dordrecht, 2002.

[12] Darse Billings. *Algorithms and Assessment in Computer Poker*. PhD thesis, University of Alberta, 2006.

[13] Darse Billings, Michael Bowling, Neil Burch, Aaron Davidson, Rob Holte, Jonathan Schaeffer, Terrance Schauenberg, and Duane Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Proceedings of the 4th International Conference on Computers and Games (CG)*, pages 21–34, Ramat-Gan, Israel, July 2004. Springer-Verlag.

[14] 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.

[15] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[16] Darse Billings and Morgan Kan. A tool for the direct assessment of poker decisions. *ICGA Journal*, 29(3):119–142, 2006.

[17] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 493–499, Madison, WI, 1998.

[18] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Poker as an experimental testbed for artificial intelligence research. In *Proceedings of Canadian Society for Computational Studies in Intelligence*, 1998.

[19] Darse Billings, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron. Using probabilistic knowledge and simulation to play poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Orlando, FL, 1999.

[20] David Blackwell. An analog of the minmax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6:1–8, 1956.

[21] Ben Blum, Christian R. Shelton, and Daphne Koller. A continuation method for Nash equilibria in structured games. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. Morgan Kaufmann.

[22] Béla Bollobás. *Combinatorics*. Cambridge University Press, Cambridge, 1986.

[23] Émile Borel. *Traité du calcul des probabilités et ses applications*, volume IV of *Applications aux jeux des hazard*. Gauthier-Villars, Paris, 1938.

[24] Ronen Brafman and Moshe Tennenholtz. A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence*, 121:31–47, 2000.

[25] George W. Brown. Iterative solutions of games by fictitious play. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. John Wiley & Sons, 1951.

[26] Kevin Burns. Heads-up face-off: On style and skill in the game of poker. In *Style and Meaning in Language, Art, Music, and Design: Papers from the 2004 Fall Symposium*, pages 15–22, Menlo Park, California, 2004. AAAI Press.

[27] Kevin Burns. Pared-down poker: Cutting to the core of command and control. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 234–241, Colchester, UK, 2005.

[28] C. Carathéodory. Uber den Variabiletätsbereich der Fourier'schen Konstanten von positiven harmonischen Funktionen. *Rendiconti del Circolo Matematico de Palermo*, 32:193–217, 1911.

[29] André Casajus. Weak isomorphism of extensive games. *Mathematical Social Sciences*, 46:267–290, 2003.

[30] 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.

[31] Fabian A. Chudak and Vania Eleutério. Improved approximation schemes for linear programming relaxations of combinatorial optimization problems. In *IPCO*, pages 81–96, Berlin, Germany, 2005.

[32] Vasek Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, NY, 1983.

[33] Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

[34] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[35] William H. Cutler. An optimal strategy for pot-limit poker. *American Mathematical Monthly*, 82:368–376, 1975.

[36] 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.

[37] Aaron Davidson. Opponent modeling in poker: Learning and acting in a hostile environment. Master's thesis, University of Alberta, 2002.

[38] 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.

[39] Ian Davidson and Ashwin Satyanarayana. Speeding up k-means clustering by bootstrap averaging. In *IEEE Data Mining Workshop on Clustering Large Data Sets*, 2003.

[40] Boudewijn P. de Bruin. Game transformations and game equivalence. Technical note x-1999-01, University of Amsterdam, Institute for Logic, Language, and Computation, 1999.

[41] B. De Meyer. Repeated games, duality and the central limit theorem. *Mathematics of Operations Research*, 21:237–251, 1996.

[42] B. De Meyer and D. Rosenberg. "Cav u" and the dual game. *Mathematics of Operations Research*, 24:619–626, 1999.

[43] Xiaotie Deng, Christos Papadimitriou, and Shmuel Safra. On the complexity of equilibria. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 67–71, 2002.

[44] Nikhil R. Devanar, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, pages 389–395, 2002.

[45] Susan Elmes and Philip J. Reny. On the strategic equivalence of extensive form games. *Journal of Economic Theory*, 62:1–23, 1994.

[46] Chris Ferguson and Thomas S. Ferguson. On the Borel and von Neumann poker models. *Game Theory and Applications*, 9:17–32, 2003.

[47] Chris Ferguson, Tom Ferguson, and Céphas Gawargy. Uniform(0,1) two-person poker models, 2004. Available at `http://www.math.ucla.edu/~tom/papers/poker2.pdf`.

[48] Nicholas V. Findler. Studies in machine cognition using the game of poker. *Communications of the ACM*, 20(4):230–245, 1977.

[49] Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[50] Yoav Freund and Robert Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[51] Lawrence Friedman. Optimal bluffing strategies in poker. *Management Science*, 17(12):764–771, 1971.

[52] Drew Fudenberg and David Levine. *The Theory of Learning in Games*. MIT Press, 1998.

[53] Sam Ganzfried and Tuomas Sandholm. Computing an approximate jam/fold equilibrium for 3-agent no-limit Texas hold'em tournaments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2008.

[54] Michael Garey and David Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[55] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.

[56] 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.

[57] Andrew Gilpin, Javier Peña, and Tuomas Sandholm. First-order algorithm with $\mathcal{O}(\log(1/\epsilon))$ convergence for ϵ-equilibrium in games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2008.

[58] Andrew Gilpin and Tuomas Sandholm. Optimal Rhode Island Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1684–1685, Pittsburgh, PA, 2005. AAAI Press / The MIT Press. Intelligent Systems Demonstration.

[59] Andrew Gilpin and Tuomas Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1007–1013, 2006.

[60] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1168–1175, 2007.

[61] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM*, 54(5), 2007.

[62] 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 National Conference on Artificial Intelligence (AAAI)*, 2008. Short paper.

[63] Andrew Gilpin and Tuomas Sandholm. Solving two-person zero-sum repeated games of incomplete information. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2008.

[64] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2007.

[65] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2008.

[66] Matthew L. Ginsberg. Partition search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 228–233, Portland, OR, 1996. AAAI Press.

[67] Matthew L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, Stockholm, Sweden, 1999. Morgan Kaufmann.

[68] J.-L. Goffin. On the convergence rate of subgradient optimization methods. *Mathematical Programming*, 13:329–347, 1977.

[69] A. J. Goldman and J. J. Stone. A symmetric continuous poker model. *Journal of Research of the National Institute of Standards and Technology*, 64(B):35–40, 1960.

[70] Srihari Govindan and Robert Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110:65–86, 2003.

[71] Dan Harrington and Bill Robertie. *Harrington on Hold'em Expert Strategy for No-Limit Tournaments, Vol. 1: Strategic Play*. Two Plus Two, 2004.

[72] Sergiu Hart. Nonzero-sum two-person repeated games with incomplete information. *Mathematics of Operations Research*, 10:117–153, 1985.

[73] Jean-Baptiste Hirriart-Urruty and Claude Lemaréchal. *Fundamentals of Convex Analysis*. Springer-Verlag, Berlin, 2001.

[74] Samid Hoda, Andrew Gilpin, and Javier Peña. Smoothing techniques for computing Nash equilibria of sequential games. Available at Optimization Online, 2008.

[75] 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, July 2005.

[76] Junling Hu and Michael P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[77] R. Isaacs. A card game with bluffing. *American Mathematical Monthly*, 62:99–108, 1955.

[78] Kamal Jain, Mohammad Mahdian, and Amin Saberi. Approximating market equilibria. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2003.

[79] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

[80] S. Karlin and R. Restrepo. Multi-stage poker models. In *Contributions to the Theory of Games*, volume 3 of *Annals of Mathematics Studies, Number 39*, pages 337–363. Princeton University Press, Princeton, New Jersey, 1957.

[81] Leonid Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20:191–194, 1979.

[82] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[83] Elon Kohlberg and Jean-Francois Mertens. On the strategic stability of equilibria. *Econometrica*, 54:1003–1037, 1986.

[84] Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, October 1992.

[85] Daphne Koller and Nimrod Megiddo. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory*, 25:73–92, 1996.

[86] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[87] Daphne Koller and Brian Milch. Multi-agent influence diagrams for representing and solving games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1027–1034, Seattle, WA, 2001.

[88] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, July 1997.

[89] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 343–350, Stockholm, Sweden, 1999.

[90] David M. Kreps and Robert Wilson. Sequential equilibria. *Econometrica*, 50(4):863–894, 1982.

[91] H. W. Kuhn. Extensive games. *Proc. of the National Academy of Sciences*, 36:570–576, 1950.

[92] 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.

[93] H. W. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 2 of *Annals of Mathematics Studies, 28*, pages 193–216. Princeton University Press, Princeton, NJ, 1953.

[94] Guanghui Lan, Zhaosong Lu, and Renato D. C. Monteiro. Primal-dual first-order methods with $O(1/\epsilon)$ iteration-complexity for cone programming, 2006. Available at Optimization Online.

[95] Carlton Lemke and J. Howson. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics*, 12:413–423, 1964.

[96] Kevin Leyton-Brown and Moshe Tennenholtz. Local-effect games. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. Morgan Kaufmann.

[97] Richard Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 36–41, San Diego, CA, 2003. ACM.

[98] Richard J. Lipton and Neal E. Young. Simple strategies for large zero-sum games with applications to complexity theory. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 734–740, Montreal, Quebec, Canada, 1994.

[99] Michael Littman. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 157–163, 1994.

[100] Michael Littman and Peter Stone. A polynomial-time Nash equilibrium algorithm for repeated games. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 48–54, San Diego, CA, 2003.

[101] Chao-Lin Liu and Michael Wellman. On state-space abstraction for anytime evaluation of Bayesian networks. *SIGART Bulletin*, 7(2):50–57, 1996. Special issue on Anytime Algorithms and Deliberation Scheduling.

[102] R. Duncan Luce and Howard Raiffa. *Games and Decisions*. John Wiley and Sons, New York, 1957. Dover republication 1989.

[103] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, California, 1967. University of California Press.

[104] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, NY, 1995.

[105] Richard D. McKelvey and Andrew McLennan. Computation of equilibria in finite games. In H. Amann, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, pages 87–142. Elsevier, 1996.

[106] H. Brendan McMahan and Geoffrey J. Gordon. A fast bundle-based anytime algorithm for poker and other convex games. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Juan, Puerto Rico, 2007.

[107] Jean-Franois Mertens and Shmuel Zamir. The value of two-person zero-sum repeated games with lack of information on both sides. *International Journal of Game Theory*, 1:39–64, 1971.

[108] Donald Michie. Game-playing and game-learning automata. In L. Fox, editor, *Advances in Programming and Non-Numerical Computation*, pages 183–200. Pergamon, New York, NY, 1966.

[109] Peter Bro Miltersen and Troels Bjerre Sørensen. Computing sequential equilibria for two-player games. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 107–116, Miami, FL, 2006. SIAM.

[110] Peter Bro Miltersen and Troels Bjerre Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.

[111] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, 1991.

[112] John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[113] John F. Nash and Lloyd S. Shapley. A simple three-person poker game. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 105–116. Princeton University Press, Princeton, NJ, 1950.

[114] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[115] Yurii Nesterov. A method for unconstrained convex minimization problem with rate of convergence $O(1/k^2)$. *Doklady AN SSSR*, 269:543–547, 1983. Translated to English as *Soviet Math. Docl.*

[116] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[117] Yurii Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005.

[118] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[119] Donald J. Newman. A model for "real" poker. *Operations Research*, 7:557–560, 1959.

[120] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[121] Christos Papadimitriou and Tim Roughgarden. Computing equilibria in multi-player games. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–91, Vancouver, BC, Canada, 2005. SIAM.

[122] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining*, pages 277–281, 1999.

[123] Andrés Perea. *Rationality in extensive form games*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[124] Avi Pfeffer, Daphne Koller, and Ken Takusagawa. State-space approximations for extensive form games, July 2000. Talk given at the First International Congress of the Game Theory Society, Bilbao, Spain.

[125] J.-P. Ponssard and Sylvain Sorin. The L-P formulation of finite zero-sum games with incomplete information. *International Journal of Game Theory*, 9:99–105, 1980.

[126] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

[127] David H. Reiley, Michael B. Urbancic, and Mark Walker. Stripped-down poker: A classroom game with signaling and bluffing, February 2005. Working paper. Available at `http://economics.eller.arizona.edu/downloads/working_papers/Econ-WP-05-11.pdf`.

[128] Julia Robinson. An iterative method of solving a game. *Annals of Mathematics*, 54:296–301, 1951.

[129] R. Tyrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[130] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[131] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[132] M. Sakaguchi. A note on the disadvantage for the sente in poker. *Mathematica Japonica*, 29:483–489, 1984.

[133] M. Sakaguchi and S. Sakai. Partial information in a simplified two person poker. *Mathematica Japonica*, 26:695–705, 1981.

[134] Tuomas Sandholm. Perspectives on multiagent learning. *Artificial Intelligence*, 171:382–391, 2007.

[135] Tuomas Sandholm and Andrew Gilpin. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006.

[136] 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, Pittsburgh, PA, 2005. AAAI Press / The MIT Press.

[137] Rahul Savani and Bernhard von Stengel. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 258–267, Rome, Italy, 2004. IEEE Computer Society Press.

[138] Herbert E. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal of Applied Mathematics*, 15:1328–1343, 1967.

[139] Jonathan Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, New York, 1997.

[140] Jonathan Schaeffer. The games computers (and people) play. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 50, pages 189–266. Academic Press, 2000.

[141] Terence Conrad Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[142] Alex Selby. Optimal heads-up preflop poker, 1999. http://www.archduke.demon.co.uk/simplex/.

[143] Reinhard Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 12:301–324, 1965.

[144] Reinhard Selten. Evolutionary stability in extensive two-person games – correction and further development. *Mathematical Social Sciences*, 16:223–266, 1988.

[145] Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39:1095–1100, 1953.

[146] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 333–345, London, UK, 2002. Springer-Verlag.

[147] 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.

[148] David Sklansky. *The Theory of Poker*. Two Plus Two Publishing, fourth edition, 1999.

[149] Stephen J. J. Smith, Dana S. Nau, and Thomas Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[150] Alexander J. Smola, S. V. N. Vishwanathan, and Quoc Le. Bundle methods for machine learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2007.

[151] Sylvain Sorin. *A First Course on Zero-Sum Repeated Games*. Springer, 2002.

[152] 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.

[153] Nathan Sturtevant, Martin Zinkevich, and Michael Bowling. Prob-max^n: Opponent modeling in n-player games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1057–1063, Boston, MA, 2006.

[154] Ken Takusagawa. Nash equilibrium of Texas Hold'em poker, 2000. Undergraduate thesis, Stanford University.

[155] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[156] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995.

[157] F. Thompson. Equivalence of games in extensive form. RAND Memo RM-759, The RAND Corporation, January 1952.

[158] John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

[159] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[160] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[161] Bernhard von Stengel. Computing equilibria for two-person games. In Robert Aumann and Sergiu Hart, editors, *Handbook of game theory*, volume 3. North Holland, Amsterdam, The Netherlands, 2002.

[162] Kevin Waugh, Dave Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathology in extensive games. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2009.

[163] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, PA, 1997.

[164] S. Zamir. Repeated games of incomplete information: Zero-sum. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory, Vol. I*, pages 109–154. North Holland, 1992.

[165] Martin Zinkevich, Michael Bowling, and Neil Burch. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2007.

[166] 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.