University of Alberta

Library Release Form

Name of Author: Michael Bradley Johanson

Title of Thesis: Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player

Degree: Master of Science

Year this Degree Granted: 2007

_____

Michael Bradley Johanson

Date: _____

Too much chaos, nothing gets finished. Too much order, nothing gets started.

— Hexar's Corollary

University of Alberta

ROBUST STRATEGIES AND COUNTER-STRATEGIES: BUILDING A CHAMPION LEVEL COMPUTER POKER PLAYER

by

Michael Bradley Johanson

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2007

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player** submitted by Michael Bradley Johanson in partial fulfillment of the requirements for the degree of **Master of Science**.

_____

Michael Bowling
Supervisor

_____

Duane Szafron

_____

Michael Carbonaro
External Examiner

Date: _____

To my family:
my parents Brad and Sue Johanson,
and my brother, Jeff Johanson.

Abstract

Poker is a challenging game with strong human and computer players. In this thesis, we will explore four approaches towards creating a computer program capable of challenging these poker experts. The first approach is to approximate a Nash equilibrium strategy which is robust against any opponent. The second approach is to find an exploitive counter-strategy to an opponent. We will show that these counter-strategies are brittle: they can lose to arbitrary other opponents. The third approach is a compromise of the first two, to find robust counter-strategies. The four approach is to combine several of these agents into a team, and learn during a game which to use. As proof of the value of these techniques, we have used the resulting poker programs to win an event in the 2007 AAAI Computer Poker Competition and play competitively against two human poker professionals in the First Man-Machine Poker Championship.

Acknowledgements

This work would not have been possible without the valuable guidance and support (or alternatively, positive order and chaos) provided by many people.

- First among these is my supervisor, **Michael Bowling**, for giving me the freedom and support to work on several projects before settling into my work on computer poker. Thanks to his guidance, the last two years of graduate school have been rewarding beyond my expectations.

- Next is **Martin Zinkevich**. During a discouraging point in my research, Marty asked if I would like to help on a new direction he was exploring. The following four months of exciting progress and fun collaboration resulted in the majority of the work I will present in this thesis. Because of his influence, I am left in the unusual position of being *more* excited about my topic as I finish my thesis than when I started my research.

- My first exposure to research in computing science began when **Ryan Hayward** hired me as a summer student for his research project on Hex. Without his influence and that of **Yngvi Bjørnsson**, I would not be half the programmer I am today, and would not have started my graduate studies.

- From the Computer Poker Research Group, **Neil Burch** and **Darse Billings** deserve special thanks. Neil's assistance and coding prowess has helped me solve several problems, and he has shown me by example what a stable, clear and extensible code base should look like. Darse has used his vast knowledge of games and his friendly demeanor to reveal far more depth to the games domain than I had previously noticed.

- The other members of the CPRG — **Jonathan Schaeffer, Rob Holte, Duane Szafron, Morgan Kan, Nolan Bard, Josh Davidson, Carmelo Piccione, Andrew Albert and John Hawkins** — also deserve my thanks. Never before have I worked with a group that got *closer* and *stronger* as a deadline approached. Under pressure during the Man-Machine poker match in Vancouver, we were always a team.

- Finally, I would like to thank **Jessica Enright, Paul Berube, Curtis Onuczko, Jeff Siegel, and Brad Joyce** and many others, for keeping me sane along the way. Thanks - it's been fun.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Playing Games

The use of games as a testbed for artificial intelligence predates the existence of the first modern computers. In 1952, Alan Turing had written an algorithm for playing chess, but did not have access to a computer on which to run it. Instead, he performed the necessary computations by hand on paper, acting as an aptly named Turing machine. Although his algorithm lost its one recorded match [12, p. 440], the experiment was a precursor to what has become a successful line of research. The artificial intelligence techniques that researchers have developed to play games such as chess have found many applications in the study of artificial intelligence, and in computing science in general.

Games have several compelling features that make them well-suited to be a benchmark for progress in artificial intelligence:

- **Finite game state and action space**. Tasks like the games of chess and checkers have a large but finite number of possible states, defined by the permutations of pieces on the board, and the players must choose between a limited number of actions. Tasks with a limited number of states and possible actions are conceptually simpler, allowing researchers and programmers to focus more on the artificial intelligence task and less on the intricacies of the domain.

- **Clear measure of success**. A game of chess can only end in three ways: a win, a loss, or a draw. The players may additionally consider degrees of success, such as winning the game as fast as possible, or with most of one's pieces intact. Even these alternate goals are quantifiable, and it is still clear that the program either works as intended (by winning) or not (by losing).

- **Existence of experts to compare against**. For tasks where success can be measured, two players can attempt the same task or compete against each other. Over repeated trials, we can determine if one player is better than the other. Through comparisons and competitions between different techniques for solving an artificial intelligence task, we can determine the circumstances in which one technique is more effective than the other. Furthermore, for games

such as chess, checkers, bridge or poker, there is a set of human enthusiasts that possess a wide range of ability. By comparing our artificially intelligent programs against humans, and human experts in particular, we can measure the progress of artificial intelligence.

In this thesis, we will continue this line of research into games by considering the challenging and popular game of Texas Hold'em poker. The main contributions of this work are three new methods for creating artificially intelligent programs that play games, and the demonstration of an established technique for combining those programs into a team. While these techniques can be applied towards a wide variety of games, we will focus on using Texas Hold'em as a benchmark. The agents created using these methods have competed in and won an international competition for computer poker players, and have been shown to be competitive with two of the world's best human players during the First Man-Machine Poker Championship. By playing competitively against the world's best poker players — both computer and human — we have demonstrated the effectiveness of our techniques.

In Section 1.2, we will describe the history of game playing programs and identify the features that are used to partition games into different categories. In Section 1.3, we will explain the mechanics of Texas Hold'em, and describe the qualities of this game (as compared to other games) that make it an interesting domain for artificial intelligence research. In Section 1.4 we will outline the contributions of this thesis in detail.

1.2 Beating humans at their own games

Since Turing's chess game in 1952, computer programmers have produced several examples of game playing programs that have approached and surpassed the best human players. A few of the prominent successes are:

- **Checkers**. Chinook is a checkers playing program developed by a team at the University of Alberta, led by Jonathan Schaeffer. In 1994, Chinook earned the right to challenge Marion Tinsley for the World Championship title. Chinook won the title after Dr. Tinsley forfeited the match due to health concerns. Chinook has since defended the title against other human masters [26].

- **Chess**. Deep Blue is a chess playing program developed at IBM, by a team led by Feng-hsiung Hsu and Murray Campbell. In 1997, Deep Blue played against chess world champion Garry Kasparov in an exhibition match, and won the match 3.5-2.5 [13].

- **Othello**. Logistello is an othello program developed by Michael Buro. In 1997, Logistello played against othello world champion Takeshi Murakami in an exhibition match, and won the match 6-0 [9].

Several other games also have strong computer agents that are competitive with or surpass the best human players. Scrabble, backgammon, and awari are examples of such games. Some of these games have common attributes, such as checkers, chess and othello. The players alternate turns taking actions with deterministic consequences, and the entire state of the game is visible to both players at all times. In games such as Scrabble and poker, some information is hidden from one or more of the players. In games such as backgammon, poker and blackjack, there is an element of chance that makes it impossible to determine precisely what will happen in the game's future. We can classify games based on these features:

By the term **perfect information game**, we refer to games where all players can determine the exact state of the game. In games like chess and checkers, this is done by looking at the board, where the pieces determine the game state. In contrast, games like poker and Scrabble are called **imperfect information games**, as there is some information known to some players but not others. In poker, each player has cards that only they can see. In Scrabble, each player can view their own tiles, but not that of their opponent.

By the term **deterministic game**, we refer to games where each action has a fixed, consistent outcome. For example, in chess, choosing to move a pawn forward one square always results in the same result — the pawn advances one square. In these games, it is possible to explore all possible lines of play, and choose actions that have guaranteed outcomes. Games like Monopoly or Risk are examples of **stochastic games**, where either the player's actions or the "chance" player affects the game in unpredictable ways. In Risk, the action to attack another player has several possible outcomes determined by random dice rolls. In Monopoly, the player is forced to roll the dice, determining the distance that their piece moves. Their actions (to buy or not buy a property) have deterministic outcomes, but the outcome of the game is affected by random chance. In such games, we consider a third player — the "chance" player — to take actions according to some distribution. In Monopoly, the chance player's actions determine the distance that the player's piece will move; in Risk, the chance player's actions determine if an attack is successful or not. These stochastic elements mean that the agents cannot be sure of the outcome of choosing certain actions.

Many of the computer programs strong enough to challenge human experts play games that are deterministic and have perfect information. Chess, checkers, and othello are all examples of such games. In these games, the well-known technique of alpha-beta search can be used to explore deep into the game tree, in order to choose actions that a worst-case opponent cannot do well against. Although games with these attributes may have differences — some games may be longer, or have a larger branching factor — there is at least an intuition that the techniques that have worked well in these games (alpha-beta search, opening books, endgame databases, and so on) should also apply to other deterministic, perfect information games. Therefore, to pursue new avenues for research, computing scientists have examined other types of games. We present two well known examples:

- **Maven** is a world-champion level Scrabble player, written by Brian Sheppard [28]. Scrabble

is an imperfect information game, in that the players cannot see the tiles that the other player holds. It also has stochastic elements, in that the players draw random tiles to refill their rack after placing a word. Maven uses selective sampling roll-outs to choose its actions. To choose its actions, the program samples many likely opponent racks of tiles, and simulates the value of its actions given those racks.

- **TD-Gammon** is a Backgammon player of comparable strength to the best human players, written by Gerry Tesauro [30]. Backgammon is a stochastic game, where dice rolls determine the available options for moving pieces around the board. Reinforcement learning is used to train the program.

In the next section, we will present the game of Texas Hold'em poker, which has a combination of features that are not well represented by the games we have discussed.

1.3 Texas Hold'em Poker

In 1997, the Games Research Group at the University of Alberta formed the Computer Poker Research Group (CPRG) to focus on the game of poker as a domain for new research. Poker is a card game for two or more players that has several interesting features that are not well addressed by the traditional approaches used in perfect information, deterministic games:

- **Imperfect information**. Each player holds cards that only they can observe. There are a large number of possible opponent hands, which can subtly change in strength depending on cards revealed throughout the game. Part of the challenge of poker is the importance of inferring what cards the opponent might be holding, given their actions. It is equally important to choose deceptive actions to avoid revealing the nature of one's own cards.

- **Stochastic outcomes**. The cards dealt to the players are selected at random, and the strength of each player's hand may vary greatly on each round. A successful player must be able to choose actions while considering the risks involved. With a strong hand, actions may be chosen so as to scare other players into exiting the game, to avoid the possibility of an unlucky card making other player's hand stronger. Alternatively, a strong hand can be played deceptively to encourage other players to stay in the hand, and thus increase the reward at the end of the game. A hand might be weak in the current round, but have some probability of becoming the strongest hand if a particular card is dealt. A successful player must be able to recognize these situations and play accordingly.

- **Exploitation is important**. In many of the games mentioned previously, the players are trying to win, and not necessarily to win by a large margin. In poker, the players are trying to win by as large a margin as possible. While there is still merit in designing a program that cannot

be beaten by its worst-case opponent, an excellent poker player will adapt to their opponents' strategy to exploit them.

- **Partially observable information**. The hidden information is not always revealed at the end of the game. This means that the player must not only manage their risk during the game, but that they cannot always confirm that their actions were correct after the game. This makes the opponent modeling task difficult, as players may sometimes choose to play suboptimally to reveal the opponents' hidden cards, in order to improve their opponent model.

We will now provide an introduction to the game of poker and the rules of Texas Hold'em, the particular poker variant that we are interested in. Afterwards, with a basic understanding of the mechanics of the game, we will revisit these features in more detail.

1.3.1 Poker and Heads-Up Texas Hold'em

Poker is a class of games; there are over 100 variants of poker, most of which have similar rules and themes. Over the last few years, poker and its variants have enjoyed a rush of popularity. This is partially due to the new availability of online casinos that allow players to play poker online, instead of in casinos or informal cash games. Poker tournaments and poker-oriented shows have also become popular on TV, introducing novices to a game that they may not have encountered before.

Poker involves two or more players who play a series of short **games** against each other. Each player is dealt cards from a standard playing card deck, with which they form a five-card **hand**. Each possible hand is associated with a category that determines its strength as compared to other hands. Examples of these categories include "One Pair" (having two cards of the same rank), "Flush" (five cards of the same suit), or "Full House" (two cards with the same rank, and three cards of a different rank). The players place wagers that their hand will be the strongest at the end of the game. Each wager is called a bet, and the sum of the wagers is called the pot. Instead of betting, players can leave the game, surrendering any chance of winning the pot. At the end of each game, the remaining player with the strongest hand wins the pot, and another game begins. The goal is to win as much money as possible over the series of games. This emphasizes one of the features of poker that we discussed in Section 1.2: it is important to win as much as possible from each opponent. A player that wins a little against every opponent can lose to a player that loses a little to half of the players, and wins a lot from the remaining players. In fact, this was the result of one of the events in the 2007 AAAI Computer Poker Competition, which will be discussed in Chapter 7.

In particular, we are interested in the variant of poker known as **Texas Hold'em**. Texas Hold'em is considered to be the most strategic variant of poker, in that it requires more skill and is less influenced by luck than other poker variants. Texas Hold'em is a game for two to ten players that advances through four rounds, which we will briefly describe here. A detailed introduction to the

rules is available online [31].

During each round, the players will participate in a **round of betting**, in which they will alternate taking one of the following actions:

- **Fold**. The player exits the game, relinquishing any chance of winning the pot.
- **Call**. The player increases the size of their wager to match the highest wager of the other players, and places this amount into the pot. If no wagers have been placed in the current round, this action is called a "check".
- **Bet**. The player places a new wager in the pot, which other players must call if they wish to continue playing. If another player has placed a wager that has not yet been called, then the player matches that wager before placing their own. In this case, the action is called a "raise".

When all of the players have acted and have either folded or called the wagers, the game progresses to the next round. If at any time only one player remains in the game (that is, if the other players have folded), then that player wins the pot without revealing their cards, and the game is over.

Each game progresses through four rounds, called the Preflop, Flop, Turn and River, and ends with a Showdown:

- **Preflop**. One player is designated the dealer. The two players to the left of the dealer are called the Small Blind and the Big Blind, and are forced to make a bet (also known as an ante). The dealer then deals two **private cards** (also known as **hole cards**) to each player, which only they can see or use. Then, starting with the player to the left of the Big Blind, the players begin a round of betting.
- **Flop**. The dealer deals three **public cards** to the table. These cards are also called **board cards** or **community cards**, and all of the players may see and use these cards to form their poker hand. After the cards are dealt, the player to the left of the dealer begins another round of betting.
- **Turn**. The dealer deals one additional public card to the table, and the player to the left of the dealer begins another round of betting.
- **River**. The dealer deals one final public card to the table, and the player to the left of the dealer begins the final round of betting.
- **Showdown**. All players still in the game reveal their cards. The player with the strongest hand wins the pot, and a new game begins. In the case of a tie, the pot is divided between the players with the strongest hands.

1.3.2 Variants of Texas Hold'em Poker

Texas Hold'em has several variants that determine the size of the wagers that the players are allowed to make; of these variants, we will describe two. In the **Limit** variant, a fixed bet size is chosen before the start of the game, such as \$10/\$20. This means that during the Preflop and Flop rounds, all bets and raises are \$10; in the Turn and River rounds, all bets and raises are \$20. These values are called the Small Bet and the Big Bet. At the start of the game, the Small Blind and Big Blind are forced to place bets equal to one half of a Small Bet and a Small Bet, respectively.

In the **No-Limit** variant, the size of the Small Blind and Big Blind are set before the start of the first game. When given a chance to act, each player may bet any amount of money equal or greater than the size of the previous bet, up to the total amount of money they have available. When a player bets all of their money, it is called **going all-in**.

If more than two players are playing, it is called a **Ring** game. If the game is being played by only two players, it is called a **Heads-Up** game. A common convention in Heads-Up games is to reverse the betting order in the Preflop round, such that the dealer places the small blind and acts first. This is done to reduce the advantage of being second to act.

In this thesis, we will only consider Heads-Up Limit Texas Hold'em. The techniques described here have also been used to produce Heads-Up No-Limit poker agents, which were competitive in the 2007 AAAI Computer Poker Competition. However, there are currently many more programs available that play Limit than No-Limit, and so we will focus on the Limit variant where we have a wide variety of established players for comparison.

1.3.3 Poker Terminology

Several other terms in the poker lexicon should be defined before continuing. A more comprehensive vocabulary can be found in [4, Appendix A].

- **Bluff**. A bluff is a bet that is made with a weak hand, to convince the opponent that the player holds strong cards. This can result in an immediate win (if the opponent folds as as result), and also serves to obfuscate future bets with strong hands, as the opponent may believe the player is attempting to bluff again.

- **Semi-bluff**. A semi-bluff is a bluff that is made with an inferior hand that has the potential to improve to a game-winning hand, if certain board cards are revealed.

- **Trapping**. A trapping action is when a player with a strong hand passes up an opportunity to bet, to convince the opponent that the player's hand is weak. Two examples of trapping are the **check-raise**, where a player checks with the intent of raising if the opponent responds by betting, and the **slow-play**, where a player checks or calls with the intent of raising in a future round.

- **Value bet**. A bet made to increase the value of a pot that the player expects to win with their current hand.

1.3.4 Poker Features

In Section 1.3, we mentioned four features of Texas Hold'em poker that made it an excellent domain for artificial intelligence research. Once again, these features were:

- **Imperfect information**. Players must be able to reason about the strength of the hidden cards their opponents hold, and choose actions that are profitable without revealing information about their own hand.
- **Stochastic outcomes**. Players must be able to choose profitable actions in an uncertain environment, where they can quickly change from a winning position to a losing one and vice versa.
- **Exploitation is important**. Players must be able to model their opponent and adapt their play accordingly, in order to maximize their long-term winnings.
- **Partially observable information**. Players must be able to construct opponent models even though there is some information that they will never have access to.

For many poker experts, poker has become a lucrative career. Poker professionals, unlike chess or checkers professionals, can win large cash prizes by playing in tournaments. For example, in the 2005 World Series of Poker, $52,818,610 in prize money was won by players, with $7.5 million going to the first place finisher of the main event [32]. With such a large monetary incentive, one can expect the best human experts to take the game very seriously and to display a high level of skill.

Skilled human opposition at a range of skill levels is readily available: from within the CPRG, online through "play money" webpages, and recently, from human experts. In July of 2007, the University of Alberta Computer Poker Research Group hosted the First Man-Machine Poker Championship, held at the AAAI conference in Vancouver. Over two days, ten poker agents, nine of which were created by the techniques described in this thesis, were used to compete in four duplicate matches against two world class human professional poker players — Phil Laak and Ali Eslami. The results of this match will be discussed in Chapter 7.

Therefore, to the above list of four important features of poker, we add one more:

- **Availability of many players of varying skill levels** — poker's current popularity means that there are players at all skill levels from beginners to world champions willing to compete against artificially intelligent agents.

Some of these features occur in other games. However, games that include even a few of these features tend to not be as well studied as deterministic, perfect information games such as chess,

checkers, hex, awari, go, othello, amazons, Chinese chess, shogi, sokoban, lines of action, domineering, and many others. Games with stochastic elements and imperfect information represent a promising research area that has not received as much attention as it should.

When we use games as a research domain, we discover techniques that have applications beyond the games domain. To name one example, the research towards heuristic search in games at the University of Alberta has been applied to civil engineering tasks such as optimally placing storm drains and sewers in cities. Dr. Jonathan Schaeffer, however, claims that the advances that come from research towards stochastic, imperfect information games such as poker will have much broader applicability to real-life problems than the advances that have come from deterministic, perfect information games [29, 19]. There is a strong intuition behind this: the real world *is* unpredictable and partially observable, and real-world tasks often involve working with or against other agents whose actions affect your own.

1.4 Contributions of This Thesis

Having established the necessary background information and motivated the topic, we can now present in detail our contributions. In this thesis, we will discuss three new techniques that can be used to produce strategies for playing any stochastic, hidden information game. We will also demonstrate the use of an established experts algorithm technique for combining these strategies into a team.

We will begin in Chapter 2 by describing the theoretical foundation on which this work relies. We will also describe the methods by which we evaluate our poker programs, and give examples of several recent successful poker programs.

Each of the following techniques will then be discussed in its own chapter:

- **Counterfactual Regret Minimization** (CFR). There are well-known techniques for finding Nash Equilibria in small abstracted versions of Texas Hold'em. In Chapter 3, we will show a new technique for quickly finding Nash Equilibria in much larger abstractions than were previously possible. This is possible because this new approach has much smaller memory requirements than established approaches: the new technique's memory requirements scale with the number of information sets, instead of the number of game states. As they play close to a Nash equilibria, the CFR agents have theoretical bounds on their maximum exploitability. We will show that the agents produced by this technique are stronger than all of our benchmark agents.

- **Frequentist Best Response** (FBR). Given an arbitrary opponent, how can we develop an effective counter-strategy that can defeat it? In Chapter 4, we will define a new technique for producing these counter-strategies, use it to model a wide range of opponents, and show that it performs better than previously known techniques.

- **Restricted Nash Response**. The agents produced by the Frequentist Best Response technique are brittle — they perform well against their intended opponents, but are very exploitable and can perform very poorly against arbitrary opponents. The agents produced by the Counterfactual Regret Minimization technique are robust — their worst-case opponent cannot exploit them, but they are not able to fully exploit weak opponents. The Restricted Nash Response technique is a compromise — it produces agents that are robust against arbitrary opponents, yet are also capable of exploiting a subset of possible opponents. Like the CFR agents, the technique provides a theoretical bound on their maximum exploitability. In Chapter 5, we will explain how this technique works, and show that the programs produced by this technique perform well against a wide variety of benchmark programs, losing only slightly to the new CFR agents while defeating other opponents by higher margins.

- **Teams of Agents**. The three new techniques described previously all produce independent poker strategies with different merits and weaknesses. Against an arbitrary opponent, it may not initially be clear which type of agent to use against it. Instead of just using one agent, we will consider a set of agents to be a team, and use a "coach" that dynamically chooses which agent to use. In Chapter 6, we show that by using established techniques from the experts paradigm that we can use several Poker agents and learn online which one is most effective against an opponent. This produces one poker program that is stronger than any of its individual components.

The poker programs produced as a result of these new techniques have recently competed in two significant competitions. In Chapter 7, we will present the results of the 2007 AAAI Computer Poker Competition and the First Man-Machine Poker Championship.

Finally, we will summarize the contributions of this thesis in Chapter 8, the conclusion, and will describe the promising new directions for this research that have been revealed in the wake of the two 2007 competitions.

1.5 Author's Contributions

The techniques to be presented in Chapter 3, Chapter 5 and Chapter 6 were developed in collaboration with Martin Zinkevich and Michael Bowling. In particular, the original idea, theoretical foundation and a prototype implementation of the Counterfactual Regret Minimization approach are the contributions of Martin Zinkevich. The author's contribution was practical implementation and optimization of a program that uses this technique. The author then used the program to collect the results presented in this thesis and to produce the competitive poker strategies which were entered into the competitions described in Chapter 7. In these chapters, we will take care to state the portions of the work that are the author's contribution, and the portions that were contributed by others.

Chapter 2

Background and Related Work

There is a long history of research into creating agents for playing zero-sum, imperfect information games such as poker. In this section, we will review some of the recent work upon which this thesis depends.

First, we will present additional background related to creating and evaluating computer poker agents. In Section 2.1, we will describe several different types of strategies that could be used in poker. In Section 2.2, we will explain the methods by which we will evaluate the poker programs that we create. In Section 2.3, we will describe a variety of benchmark programs against which we will compare our new poker programs. In Section 2.5, we will explain how Texas Hold'em poker (a game with 10^{18} states), can be abstracted to a manageable size without affecting the strategic elements of the game.

Next, we will begin laying a foundation for our descriptions of past approaches and our own new contributions. In Section 2.4, we will define extensive form games, sequence form, and the variables and terminology that will be used throughout this thesis.

Finally, in Section 2.6, we will review a selection of past approaches to creating poker agents. In Section 2.6.1, we will discuss the simulation-based approaches that the CPRG used for its first poker agents. In Section 2.6.2, we will review the most successful approaches to date: strategies that approximate a Nash equilibrium, resulting in very robust players. In Section 2.6.3, we will explain best response strategies. In Section 2.6.4, we will consider adaptive players that change their play to defeat their opponent. Finally, in Section 2.7, we will review one known approach for combining poker strategies into a team, and a "coach" agent that chooses which strategy to use from game to game.

2.1 Types of poker strategies

Before describing some of the poker programs that have already been developed, it is useful to consider the different types of strategies that a player could use when playing the game. In Section 1.3.4, we mentioned that one of the features of poker is that exploitation is important: the goal is to win as much money as possible from each opponent. This means that there is not a "correct way" to play poker, like there is in games that have recently been solved such as awari [23] or checkers [25]. Instead, the correct strategy to use should ideally depend on the opponent that is being faced.

Against a weak or known opponent, this may mean using a strategy designed to exploit their faults. Through examining histories of past games or through online learning, one can build a model of the opponent, and act in such a way as to maximally exploit the model. If the model is very accurate, then this may have a high win rate. If the model is inaccurate, however, it can lose badly.

Against an unknown or stronger opponent, we may want to adopt a strategy that is very difficult to exploit. The standard way of thinking about such a strategy, in any game, is the concept of a Nash equilibrium. A Nash equilibrium is a strategy for each player of the game, with the property that no single player can do better by changing to a different strategy. There can be several different (and possibly infinitely many) equilibria for any given game, but if the game is two-player and zero-sum, every Nash equilibrium provides the same payoffs to the players. In a repeated game where the players change positions, such as heads-up poker, this is a very useful property — if both players are playing an equilibrium strategy, the expected score for both players will be zero. If one player plays the equilibrium strategy, since their opponent cannot do better by playing a strategy other than the equilibrium, they can expect to do no worse than tie the game. In poker, using such a strategy allows us to defend against any opponent, or allows us to learn an opponent's tendencies safely for several hands before attempting to exploit them.

When trying to find a Nash equilibrium in a complex game, we can rarely arrive at the precise equilibrium. Instead, we approximate the Nash equilibrium with an **ϵ-Nash equilibrium** strategy, where ϵ is a measure of how far from the equilibrium the strategy is. Since a Nash equilibrium strategy should expect to get a value of no less than 0 against any opponent, ϵ is the value of the best response to the strategy. Other ways to say this are that that the strategy is ϵ **suboptimal** or **exploitable**.

A common theme we will explore when considering poker strategies is the tradeoff between exploiting an opponent and one's own capacity to be exploited. If we use a strategy that is specifically designed to beat one opponent, we are exploiting them but are also opening ourselves up to be exploited by a different strategy. If we choose to minimize our own exploitability by playing very close to an equilibrium, then we have to sacrifice our ability to exploit an opponent. It would be very valuable to have strategies along this line, and not just at these two extremes. Furthermore, we would like to obtain more than a linear tradeoff when we do this: we want to get more than we give up.

Instead of just having one well-designed strategy, we would also like to have a variety of strategies to choose from. For example, we may want to consider a set of strategies to be a team, from which we will choose one strategy at a time to play the game. One approach could be to randomly select strategies from a pool, and set a higher probability of choosing strategies that have historically been successful. A more complicated approach may be to start with an equilibrium strategy until we discover an opponent's weakness, and then use the appropriate response to the weakness.

These types of strategies are presented as examples we are interested in for the purposes of this thesis. In this thesis, we will describe methods for producing poker agents that play according to each of these strategies — specific responses to opponents, careful equilibria, exploitative-but-robust compromises, and teams of strategies with varying abilities.

2.2 Evaluating a poker program

When humans play Limit Hold'em, they often use the value of the Small Bet (or, equivalently, the Big Blind) as their base unit of money. Since players can play at different speeds, or (if online) play on several tables at the same time, they usually measure their success by the number of small bets they win per game. Darse Billings, a poker expert and a researcher in the CPRG, claims that a good player playing against weaker opponents can expect to make 0.05 small bets per game. This number may seem surprisingly low to people new to the game, but at a \$10/\$20 table playing at 40 games per hour, this translates to \$20 per hour [4, p. 65].

Our poker agents play significantly faster than 40 games per hour. In fact, most of our poker programs can play thousands of games per second, which allows us to play millions of games of poker to compare the relative strength of our programs. Over millions of games, the variance is reduced such that measuring our performance in small bets/game (sb/g) becomes unwieldy due to the number of decimal points. Therefore, for computer competitions, we choose to measure our performance in millibets/game (mb/g), where a millibet is 0.001 small bets.

Variance is a challenge in poker. On each game, the typical standard deviation of the score is ± 6 sb/g (6000 mb/g) [7, p. 13]. If two closely matched poker players are playing a match and one is 10 mb/g better than the other, it can take over one million hands to determine with 95% confidence that the better player has won [15, p. 1]. One simple way to get an accurate result, then, is simply to play several million hands of poker. This is possible if we are playing two programs against each other, as 10 million games can be played in parallel in a matter of minutes. As human players play comparatively slowly (40 games/hour) and their play degrades over time due to fatigue, hunger, washroom needs and other typical human concerns, playing one million hands over 25,000 continuous hours is not an option. Instead, we use two other techniques to reduce the variance: duplicate games and DIVAT analysis.

2.2.1 Duplicate games

In bridge, a standard convention is to play duplicate games. At one table, teams A and B receive fixed cards when they play against each other. At another table, teams C and D receive the same cards. If teams A and C received the same cards, then they had the same opportunities. By comparing their scores against each other, they can determine which team did better with the resources they had.

We have adopted this convention for poker. When we run a match between two programs (A and B), we first play a series of games, with the cards being dealt according to a random number generator given a certain seed. Then, we reset the programs so that they do not remember the previous match, switch their starting positions, and replay the same number of games with the same cards. We add each player's performance in each position together, and compare the total scores. Since each player has now received the same opportunities — the same lucky breaks and the same unfortunate losses — the variance on each hand is much lower. In his PhD thesis, Billings experimentally measured the standard deviation of a duplicate match at ± 1.6 sb/g (1600 mb/g) [7, p. 17]. When we run a match between two poker programs, we typically play 5 million hands of duplicate poker. This means 5 million hands on either side of the cards, resulting in 10 million hands total. This provides us with a 95% confidence interval for the mean of 2 mb/g, which is usually enough for us to determine if one player is stronger than another.

Although duplicate poker does considerably reduce variance, it is still subject to luck. For example, consider two players A and B that are playing the same hand against opponents. With a weak Preflop hand, A might fold on the Preflop and take a small penalty. B might call, receive a tremendously lucky set of cards on the Flop, and win a large pot as a result. In this example, a lucky outcome has had a large effect on the duplicate score, and created variance. To combat this effect, we simply play a large number of games when two computer players are competing.

Once again, playing against human opponents is more complicated than playing against computer opponents. During the second half of a duplicate match, the competitors will have already seen the opposite side of their cards, breaking the assumption that the two sets of games are independent. Computer programs do not object to having their memories reset, but humans are not so agreeable. Instead, we can perform the duplicate experiment by playing against two humans that are working as a team. In one room, we play a match between program A and human B, and in a separate room, human D will play against program C. The same cards will be dealt in each room, with A and D receiving the same cards and playing in the same seat. Afterwards, the two humans and two programs combine their scores, and we can determine which team performed better with the same opportunities. This approach is most effective when both teammates are playing according to a similar style. If they are radically different, such as if one player is very aggressive and the other is very conservative, then more situations will arise where one team will win or lose both sides of the same hand, resulting in less of a variance reduction.

This duplicate poker convention was used at the 2007 Man-Machine Poker Championship, where

two human experts played 4 sets of 500 duplicate hands (4000 hands total) against the CPRG's poker agents. Over each 500 hand match, however, the variance was still quite high. While the duplicate money total was used to declare a winner in each match, for our own purposes we used another tool, called DIVAT analysis, to reduce more variance from the score.

2.2.2 DIVAT Analysis

DIVAT is a technique proposed by Billings and Kan [4, 15, 7], and analyzed further by Zinkevich et al [33]. If poker is a game where skill plus luck equals money, then DIVAT is a technique for subtracting the luck out of the equation. It is a tool that can be run after a match is complete, and requires full information.

When used, the DIVAT program examines each hand and considers how a baseline strategy would play both sides of the cards. This baseline can be any strategy, but in the CPRG's implementation it is a bet-for-value strategy: it bets according to the strength of its cards, without trying to bluff, slowplay, or do any other "tricky" actions. By comparing the player's actions against the baseline, we can identify situations where the the player took actions that resulted in more or less money than the baseline would have made. If the player wins a large pot because of a lucky card revealed on the river, then the baseline also wins this large pot, and the DIVAT program does not reward the player for this win. However, if the player takes an action that the baseline would not have, then the DIVAT program rewards or punishes the player.

These rewards and penalties are combined to form a score that gives a value to the player's skill, in small bets/game. The DIVAT program has been proven to be unbiased by Zinkevich et al [33], meaning that the expected value of the player's DIVAT score is equal to the expected value of the money earned by the player. The standard deviation of the DIVAT score is dependent on the players involved; Billings and Kan show examples of the standard deviation being reduced to as little as ± 1.93 sb/g [7, p. 17]. The duplicate and DIVAT approaches can be combined to produce a duplicate DIVAT metric, capable of reducing the standard deviation to ± 1.18 sb/g [7, p. 18].

These techniques allow us to evaluate a poker agent in far fewer hands than would otherwise be possible. Throughout this thesis, the resulting score of a match between two agents will be shown in millibets per game (mb/g) and will be accompanied by the 95% confidence interval of the result. All matches will be run in duplicate, to reduce variance. In the case of matches against human opponents, DIVAT analysis will also be performed, to reduce the variance as much as possible.

2.3 Benchmark programs

To evaluate the poker agents produced by the techniques in this thesis, we will have them compete against the following opponents:

2.3.1 Best Responses

A best response to a program is an optimal strategy for playing against that program. Techniques for approximating such an optimal strategy will be discussed later in Chapter 4; we will call these approximations of optimal counter-strategies **abstract game best responses**. A match between a program and its abstract game best response gives one indication of how much the program *can* be beaten by. This is a worst-case analysis: an opponent without a perfect opponent model is unlikely to win at the same rate as the abstract game best response strategy.

2.3.2 Poker Academy

Poker Academy is a poker training program produced by BioTools, Inc. Poker Academy includes two strong poker programs, Sparbot and Vexbot, that can compete against humans and new poker programs. Sparbot and Vexbot were developed by the CPRG, and have been licensed to BioTools for use in Poker Academy. As this is a standard program that can be bought and used by anyone, Sparbot and Vexbot are common benchmarks for researchers.

2.3.3 CPRG Programs

The CPRG has several poker agents that have become internal benchmarks that we compare our new programs to. Throughout this thesis, the majority of the results presented will come from matches between our new poker agents and these benchmark agents. These benchmark programs are:

- **PsOpti4, PsOpti6 and PsOpti7** are ϵ-Nash equilibrium strategies produced by the techniques described in [3]. PsOpti4 and PsOpti6 were combined to form Hyperborean06, the winner of the 2006 AAAI Computer Poker Competition. PsOpti4 is less exploitable than PsOpti6 and PsOpti7, but PsOpti6 and PsOpti7 play a strategically different style that is useful against some opponents. Poker Academy's Sparbot is PsOpti4 marketed under a different name.

- **Smallbot 1239, 1399 and 2298** are ϵ-Nash equilibria strategies produced using a recently published technique [34]. 1239 and 1399 are weaker than PsOpti4, and 2298 was the CPRG's strongest program until the arrival of the programs described in this thesis. Recently, Zinkevich, Bowling and Burch verified that if Smallbot2298 had competed in the 2006 AAAI Computer Poker Competition, it would have won [34, p. 792].

- **Attack60 and Attack80** are "attack" strategies, similar to best responses in that they are intended to defeat particular opponent strategies. They were generated as byproducts of Smallbot2298. They are theoretically very exploitable, but form interesting opponents when we are considering counter-strategies.

While these poker agents are strong opponents, they tend to come from two insular "families": the PsOptis and the Smallbots. Although the CPRG has produced many strong poker agents, it is important to carefully consider the results of matches against externally produced poker agents.

2.3.4 2006 AAAI Computer Poker Competition Programs

After the 2006 AAAI Computer Poker Competition [18], a benchmark server was established so that the competitors could test new programs against any of the 2006 entries. Although the CPRG's entry (Hyperborean06) won the competition, several other strong and interesting programs were entered. They include, in the order of their placement in the competition:

- **BluffBot**, produced by Salonen [24].
- **GS2**, produced by Gilpin and Sandholm of Carnegie Mellon University. It plays according to an epsilon Nash equilibria strategy [10].
- **Monash-BPP**, produced by Korb et al from Monash University. It uses Bayesian reasoning to adjust its play to suit its opponents [17].
- **Teddy**, produced by Lynge from Denmark. Teddy is a simple agent that always attempts to raise at every opportunity.

2.3.5 2007 Computer Poker Competition

As a result of the techniques described in this thesis, two new poker agents were created that were then entered into the 2007 AAAI Computer Poker Competition. Fifteen competitors from seven countries submitted a total of 43 new poker agents in three different competitions, giving us the opportunity to compare our poker agents against the world's best new computer poker agents. The results of the match will be explored in Chapter 7.

2.3.6 First Man-Machine Poker Championship

At the 2007 AAAI conference, the University of Alberta hosted the First Man-Machine Poker Championship. In this event, two strong poker professionals, Phil Laak and Ali Eslami, competed as a team in duplicate matches against several of the poker programs produced using the techniques described in this thesis. This comparison to human professional players gave us valuable insights into the strengths and weaknesses of our agents, and an estimate of how well our program's performance compares to that of strong humans. The results of this match will also be explored in Chapter 7.

2.4 Extensive Form Games and Definitions

Games such as chess or checkers can be straightforwardly represented by game trees. A game tree is a directed tree that has one root, corresponding to the initial state of the game. Each game state where it is one player's turn to act is represented as a choice node in the tree. The edges from this choice node to other nodes represent the legal actions the player can choose from, and the states that those actions will lead to. The terminal nodes of the tree represent the end of the game. Each

terminal node holds the utility for each players to have reached that outcome. When a game is represented in this manner, it is called an **extensive form** game.

When we are using a game tree to represent stochastic games such as backgammon or poker, we need a way to represent the chance outcomes that occur during the game — the roll of the dice or the dealing of the cards. We do this by introducing the chance player. The chance events in the game are represented by a choice node for the chance player, where each action is a possible chance outcome. At each choice node for the chance player, they choose their actions according to a set distribution.

In the case of imperfect information games, the players may not be able to differentiate between different game states. For example, at the start of a poker game, each player has received their own cards (the chance player has acted), but they do not know what cards their opponent is holding. Thus, if they hold K♠K♡, they cannot tell if they are in the game state where their opponent holds 2♡7♣ or A♠A♢. We use the term **information set** to refer to a set of game states between which one player cannot differentiate. Since a player cannot tell the difference between states in the same information set, they must choose their actions according to the same distribution for all game states in the information set. Note that, in games like poker, the number of game states is far larger than the number of information sets. An extensive form game tree for an imperfect information game is a game tree where each choice node is a member of one information set.

2.4.1 Definitions

The following formalism of extensive games is due to Osborne and Rubinstein [22]:

Definition 1 *[22, p. 200] a finite extensive game with imperfect information has the following components:*

- *A finite set N of **players**.*
- *A finite set H of sequences, the possible **histories** of actions, such that the empty sequence is in H and every prefix of a sequence in H is also in H. $Z \subseteq H$ are the terminal histories (those which are not a prefix of any other sequences). $A(h) = \{a : (h, a) \in H\}$ are the actions available after a nonterminal history $h \in H$,*
- *A function P that assigns to each nonterminal history (each member of $H \backslash Z$) a member of $N \cup \{c\}$. P is the **player function**. $P(h)$ is the player who takes an action after the history h. If $P(h) = c$ then chance determines the action taken after history h.*
- *A function f_c that associates with every history h for which $P(h) = c$ a probability measure $f_c(\cdot|h)$ on $A(h)$ ($f_c(a|h)$ is the probability that a occurs given h), where each such probability measure is independent of every other such measure.*
- *For each player $i \in N$ a partition $\mathcal{I}_i$ of $\{h \in H : P(h) = i\}$ with the property that $A(h) = A(h')$ whenever h and h' are in the same member of the partition. For $I_i \in \mathcal{I}_i$*

*we denote by $A(I_i)$ the set $A(h)$ and by $P(I_i)$ the player $P(h)$ for any $h \in I_i$. $\mathcal{I}_i$ is the **information partition** of player i; a set $I_i \in \mathcal{I}_i$ is an **information set** of player i.*

- *For each player $i \in N$ a utility function u_i from the terminal states Z to the reals $\mathbf{R}$. If $N = \{1, 2\}$ and $u_1 = -u_2$, it is a **zero-sum extensive game**. Define $\Delta_{u,i} = \max_z u_i(z) - \min_z u_i(z)$ to be the range of utilities to player i.*

In the above description, we have defined the concept of the information partition without stating how the information partition is chosen. The standard information partition used for studying imperfect information games is to have each information set for player i contain the game states (or equivalently, histories) that vary only by the hidden information which player i cannot see. Figure 2.1 shows an example: the two choice nodes vary vary only because of our opponent's cards, and an information set contains these game states. From our perspective, we cannot tell if the opponent has the pair of twos or the pair of kings. During a game, we only know the information set we are in, and not the particular game state within that information set. Since we cannot tell the difference between the game states within the information set, any plan we have of how to act from that information set must be used for all game states within the set. In Figure 2.1, we cannot decide to raise when the opponent has the pair of twos and call when they have the pair of kings. Since we cannot tell which state we are in, we must choose an action (or probability distribution over actions) to use when we encounter the information set.

Now that we have defined the notion of a game, we will describe what we mean by a strategy. A strategy is a static plan for playing the game. A strategy does not change over time or adapt to any opponent; it is simply a formula for how to act at each possible information set. A **pure strategy** is a strategy where, for every information set, one action is always selected from that information set. For example, a strategy that always bets when it holds a pair of aces during the Preflop could be part of a pure strategy. A **behavioral strategy** is a strategy that selects actions with different probability distributions for each information set. For example, a strategy that sometimes calls and sometimes raises when holding a pair of aces during the Preflop would be a behavioral strategy. Note that the space of all possible behavioral strategies is infinite, as the range of probabilities that can be assigned is continuous.

We will now formally define the idea of strategies and strategy profiles [36]:

Definition 2 *A **strategy of player i** σ_i in an extensive game is a function that assigns a distribution over $A(I_i)$ to each $I_i \in \mathcal{I}_i$, and Σ_i is the set of strategies for player i. A **strategy profile** σ consists of a strategy for each player, $\sigma_1, \sigma_2, \ldots$, with σ_{-i} referring to all the strategies in σ except σ_i.*

Let $\pi^\sigma(h)$ be the probability of history h occurring if players choose actions according to σ. We can decompose $\pi^\sigma = \Pi_{i \in N \cup \{c\}} \pi_i^\sigma(h)$ into each player's contribution to this probability. Hence, $\pi_i^\sigma(h)$ is the probability that if player i plays according to σ then for all histories h' that are a proper prefix of h with $P(h') = i$, player i takes the corresponding action in h. Let $\pi_{-i}^\sigma(h)$ be

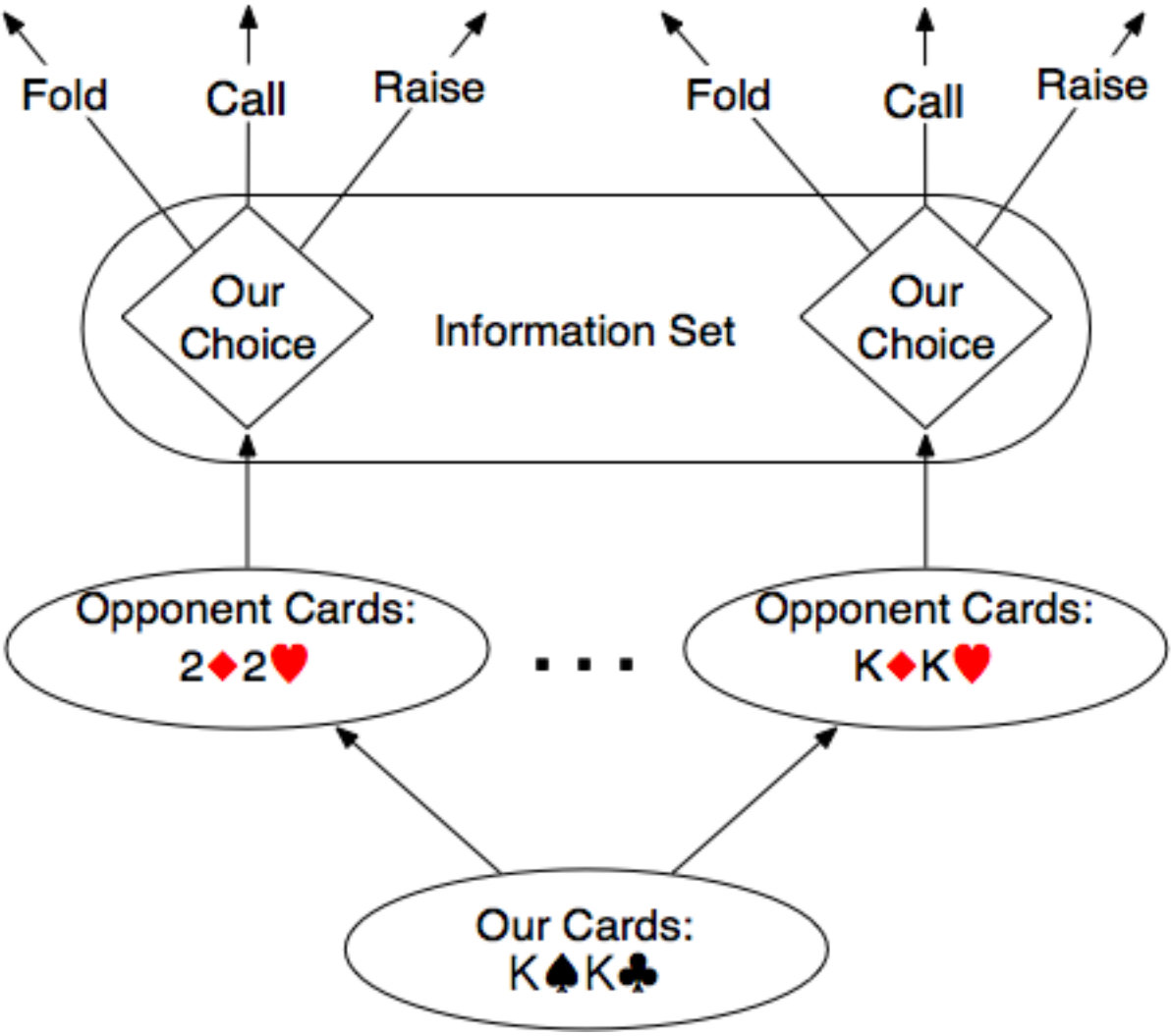


Figure 2.1: A poker example of information partition of the state space into information sets. We cannot distinguish between the choice nodes descendent from the two chance nodes that assign cards to our opponent. An information set contains these game states that we cannot distinguish between.

the product of all players' contribution (including chance) except player i. For $I \subseteq H$, define $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$, as the the probability of reaching a particular information set given σ, with $\pi_i^\sigma(I)$ and $\pi_{-i}^\sigma(I)$ defined similarly.

The overall value to player i of a strategy profile is then the expected payoff of the resulting terminal node, $u_i(\sigma) = \sum_{h \in Z} u_i(h)\pi^\sigma(h)$.

This formal description of strategies, strategy profiles, and histories will be used when describing other work in this area and in the contributions of this thesis.

2.4.2 Nash Equilibria

Now that we have defined strategies and strategy profiles, we will revisit the concept of the Nash equilibria and define it formally. A **Nash equilibrium** is a strategy profile σ where no player can increase their utility by unilaterally changing their strategy:

$$u_1(\sigma) \geq \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) \qquad u_2(\sigma) \geq \max_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2'). \tag{2.1}$$

This means that for player 1, there is no other strategy in Σ_1 that would produce more utility against σ_2 than its strategy in σ. The same is true of player 2.

In the huge poker abstractions that we are interested in solving, it is not feasible to find the precise Nash equilibrium. Instead, we try to find approximations of Nash equilibria. An **ϵ-Nash equilibrium** is a strategy profile σ where no player can increase their utility by more than ϵ by unilaterally changing their strategy:

$$u_1(\sigma) + \epsilon \geq \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) \qquad u_2(\sigma) + \epsilon \geq \max_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2'). \tag{2.2}$$

This means that for player 1, there is no strategy in Σ_1 that produces more than ϵ more utility against σ_2 than its strategy in σ.

2.4.3 Sequence Form

Strategies for extensive form games can be represented in several ways. One straightforward way would be to enumerate all possible information sets and record the probabilities of taking each action from that information set.

An alternate method of representing a strategy is to store the probability of playing along each sequence of actions. Consider a sequence of actions by a player and its opponents that have reached a terminal state. Assuming that the chance player and the opponent play to reach this outcome, we can find the probability of the player selecting their actions in this sequence. This is simply $\pi_i^\sigma(h)$ as defined above, and is known as a realization weight.

A set of realization weights defines a strategy. To find the strategy's action probabilities at an information set during a game, for each action we can find the sum of the realization weights

associated with terminal nodes reachable after taking that action. Storing our strategies in this way requires memory proportional to the number of terminal sequences.

In 1994, Koller, Megiddo and von Stengel proposed using the sequence form as a way to use linear programming to find a Nash equilibrium strategy [16]. A linear program can be created to find optimal realization weights subject to constraints (action probabilities are non-negative and sum to 1). The result is a pair of strategies that are best responses to each other: a Nash equilibrium. This approach was a large improvement over previous techniques, and in addition to other applications, was used to produce several strong poker programs [3, 10]. Examples of poker programs created by this technique will be discussed in Section 2.6.2.

2.5 Abstraction

Heads-up Limit Texas Hold'em is a game with approximately $3.16 * 10^{17}$ nonterminal game states and $3.19 * 10^{14}$ information sets. Since a behavioral strategy consists of a probability distribution over actions for each information set, using two 8 byte floating point numbers to store each 3 action probability distribution would require more than 4.5 petabytes of memory to store one strategy[1]. Our best known approach to calculating an approximate Nash equilibrium requires time proportional to the number of information sets; when the number of information sets is this large, the problem appears intractable.

We need a method to create an abstract version of poker that has fewer game states and information sets, with the property that the abstract game shares the same strategic properties as the full game. By doing so, we can create winning strategies in the abstract game that will also perform well in the real game.

2.5.1 Card Isomorphisms

The simplest abstraction is to abstract out the suits of the cards. For example, we can merge the game states for A♣2♣ and A♡2♡ or K♣7♠ and K♡7♢ into the same states. This abstraction does not lose any information, as there is no strategic value to choosing different actions on identical hands that vary only by a suit rotation. This offers a reduction in the number of game states of at most $4!$ [4, p. 83], but does not reduce the state space by the amount we require.

2.5.2 Action Abstraction

Another way to reduce the size of the game is to limit the number of actions that are available. In Limit Hold'em, for example, we can restrict the maximum number of bets to three bets per round instead of four. In theory, this limits the maximum amount it is possible to win from an opponent, but cases where the betting reaches its limit are not common in practice. In one experiment, Zinkevich et

[1]The sequence form is more compact, as it stores one probability of reaching each terminal node. In sequence form, there are $5.42 * 10^{14}$ histories, and a strategy would require approximately 3.8 petabytes

al found that an approximate Nash equilibrium in a game with a betting abstraction which considered at most 2 bets in the Preflop and 3 in the remaining rounds was 11 mb/g exploitable in its own abstraction, and 27 mb/g exploitable in the same card abstraction with full betting [34]. While this is not a trivial difference, the state space reduction is considerable, reducing the number of nonterminal betting sequences from 6378 to 2286. However, this abstraction on its own is still not sufficient to reduce the game to a tractable size.

2.5.3 Bucketing

A common and successful technique for reducing the size of the game to a tractable size is **bucketing**. On each round, we will partition the possible cards held by a player and on the board into a fixed number of **buckets**, with the intent that hands with similar strategic properties share the same bucket. One approach for doing this is to divide hands into buckets based on their strength, such that weak hands are grouped into low numbered buckets, and strong hands are grouped into high numbered buckets.

The **bucket sequence** is the sequence of buckets that the player's cards were placed into on each round. For example, if a player had a weak hand on the Preflop and the Flop cards made it a strong hand, then their hand may have been in bucket 1 on the Preflop and bucket 5 on the Flop. In the bucket abstraction, a strategy is defined over bucket sequences, and not over cards. This means that a strategy has to act with the same action probabilities for all hands with the same bucket sequence. A hand that progresses from bucket 1 to 5 is strategically distinct from one that progressed from bucket 4 to 5, but any two hands that progress through the same bucket sequence are treated as if they were identical.

This approach allows us to greatly reduce the number of game states. If we select a small number of buckets (5 or 10, for example), then our 1326 possible combinations of Preflop cards get reduced to a tractable number. As a consequence, strategies in the abstract game may no longer be capable of optimal play in the real game, as there may be subtle differences between hands in the same bucket that require different action probabilities.

Assuming an even distribution of hands into buckets, if we increase the number of buckets we use, each bucket will contain fewer hands. Since all hands in a bucket must be played according to the same action probabilities, this leads to fewer cases where a strategy is forced to consider suboptimal action probabilities for a particular hand. In other words, as we use more buckets, we get closer to playing the real game, and the performance of our strategies should increase.

To completely define the abstract game, we need two more pieces of information. First, we need to know the probability of every bucket sequence winning a showdown against every opponent bucket sequence. This tells us about the strength of the bucket sequence we are in. Second, we need to know the probability, given a bucket sequence, of reaching every possible bucket on the following round. This tells us about the properties of our current bucket sequence; for example, does it have

the potential to progress to a bucket sequence with a high probability of defeating the opponent? Is it likely to win against some of the possible opponent buckets, but is unlikely to improve during the next round?

To make the most of the buckets we choose to use, however, we would like to sort our card hands into buckets such that all of the cards in each bucket are strategically similar. In other words, we want to sort hands into buckets such that we would choose similar action probabilities for all hands in the bucket. To do this, we will choose a function that defines a many-to-one mapping of hands into bucket sequences. The card isomorphism abstraction described in Section 2.5.1 is one simple example of this. In the remainder of this section, we will present three alternate methods for partitioning the possible hands into bucket sequences.

Before continuing, we will note that the number associated with each bucket does not necessarily reflect the strength of the hands associated with it; each bucket is an abstract collection of hands that can be assigned by an arbitrary ranking. However, in practice, we find it convenient to number our buckets such that hands in a higher numbered bucket are "stronger" than those in a lower numbered bucket. We will continue to use this convention for examples, although it does not hold in theory.

2.5.4 PsOpti Bucketing

As we mentioned earlier, one straightforward way to partition hands into buckets sequences is to consider the "strength" of the hands. If a hand is at a showdown (there are no additional cards to be dealt), we will use the term **hand strength** to refer to the probability of that hand being stronger than a uniformly random opponent hand, where a tie counts as half a win. To measure the strength of a hand when there are more cards to be dealt, we will use a metric called **7 card hand rank** [3].

To calculate the 7 card hand rank metric for a given a set of cards (for example, 2 private cards on the Preflop, or 2 private cards and 3 public cards on the Flop), we roll out the remaining public cards and opponent private cards, and find the overall probability of this hand winning a showdown against a random two-card hand. Since the metric values the original hand as the expected value of the roll-out hand strength, we will also call it $E[HS]$.

The metric for any hand is then a number in the range [0,1]. We can then use this $E[HS]$ metric to order all possible hands at each round; a hand with a greater value is "more likely" to win at a showdown [2].

We could use this method alone to sort hands into buckets. If we wanted to use N buckets, then we could assign the hands where their $E[HS]$ value is in [0, $1/N$] to bucket 1, ($1/N$, $2/N$] to bucket 2, and so on. We call this uniform bucketing, as each bucket contains an equal sized range of possible $E[HS]$ values.

However, this approach assumes that all possible hands can be accurately evaluated along one

[2]Note that the $E[HS]$ metric assumes that the opponent's private cards are selected randomly. Hands with a higher $E[HS]$ rank are only "more likely" to win under the assumption that our opponent will possess any pair of private cards with equal probability. This is unrealistic; an opponent that has bet aggressively might be more likely to hold strong cards.

dimension, and this is not the case. Some hands can improve in strength dramatically as more cards are added, whereas others cannot. For example, the Preflop hand 8♣9♣ has the potential to turn into a very strong hand if the Flop contains three clubs (making a Flush) or a 5, 6, and 7 (making a Straight). A Preflop hand such as 2♠7♡, is less likely to be used in a Flush or a Straight. We are interested in the variance in the strength of a hand as more cards are dealt, as hands with high variance (they are likely to increase or decrease in strength dramatically) are strategically distinct from other hands. For now, we will concentrate on hands that are likely to improve, and we will say that these hands have **potential**.

For the program PsOpti4 [3], the CPRG decided to handle potential in the following way. One bucket was reserved for weak hands with potential: a hand with a $E[HS]$ value below a threshold, but which will improve above another threshold with some probability. The remaining $(N-1)$ buckets were used for bucketing based on the $E[HS]$ score as described above, with the higher buckets comprising a narrower range of $E[HS]$ values. This was done to give more information about the top-ranked hands that will be involved in higher betting (and thus more important) games. A similar approach was used to create the abstraction for the PsOpti6 and PsOpti7 programs.

2.5.5 More Advanced Bucketing

Currently, the CPRG employs a different method of abstracting the game, which involves four changes from the simple $E[HS]$ bucketing described above. This method was used to generate the abstractions used for the agents described in this thesis, and so we will explore it in more detail. The four changes are called **expected hand strength squared**, **nested bucketing**, **percentile bucketing**, and **history bucketing**.

Expected Hand Strength Squared

Earlier, we explained how potential was not well measured by the $E[HS]$ metric. In our new bucketing approach, we roll out the cards for each hand and measure the expected value of the square of the hand strength. This approach (abbreviated as $E[HS^2]$) also assigns values in the range [0, 1] to hands, but assigns a higher value to hands that have potential to improve in strength.

For example, consider two hands, a and b. During the roll-out, we find that a can reach two hand strength values, with strength 0.4 and 0.6. The $E[HS]$ value for a is then $E[HS](a) = (0.4 + 0.6)/2 = 0.5$, and the $E[HS^2]$ value is $E[HS^2](a) = (0.4^2 + 0.6^2)/2 = 0.25$. For hand b, we find that it can reach two hand strength values, with strengths 0.2 and 0.8. The $E[HS]$ value for b is then $E[HS](b) = (0.2 + 0.8)/2 = 0.5$, and the $E[HS^2]$ value is $E[HS^2](b) = (0.2^2 + 0.8^2)/2 = 0.68$. While both of these hands have the same $E[HS]$ value, the $E[HS^2]$ metric considers the hand with high potential to be more similar to hands with high hand strength. In practice, the betting strategy associated with high potential hands is more similar to that of hands with low potential but high hand strength than it is to hands with moderate hand strength. The $E[HS^2]$ metric places high potential

and high hand strength hands in the same buckets; the $E[HS]$ metric does not. In practice, we have found that agents produced with the $E[HS^2]$ abstraction perform better.

Nested Bucketing

The nested bucketing feature is closely tied to the rationale behind the $E[HS^2]$ metric. In the $E[HS^2]$ metric, we have taken two features (potential and hand strength) and placed them onto a one-dimensional ranking used to divide cards into buckets, while giving potential more value than it had with the $E[HS]$ metric. We have also considered other bucketing systems that use two dimensions, and nested bucketing is a step in that direction.

First, we use the $E[HS^2]$ metric to collect our possible hands into N sets. Then, within each set, we use the $E[HS]$ metric to split the set into M buckets. This allows us to separate the high potential from the high value hands, producing $N * M$ buckets.

Percentile Bucketing

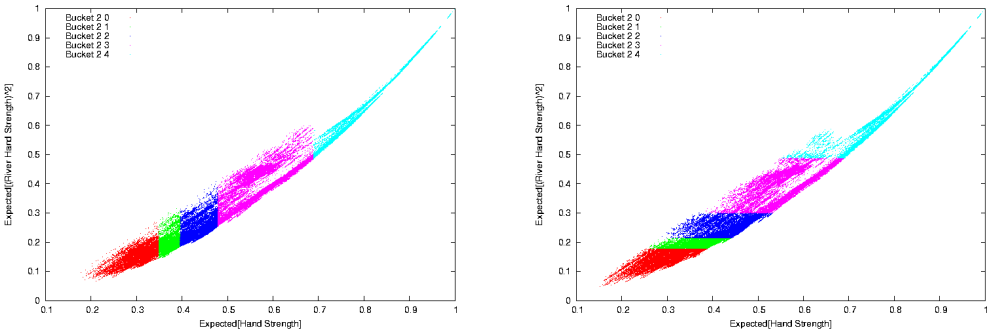
In the abstraction used to produce PsOpti4, each bucket specified a range of $E[HS]$ values; hands with an $E[HS]$ value within that range were assigned to that bucket. This led to buckets containing an asymmetrical number of hands. In part, this was intended, as the ranges were chosen to permit fewer hands into the highest valued buckets, to provide better discriminatory power for these important hands. However, some buckets were either empty (there were no hands on that round whose $E[HS]$ value fell within the bounds) or contained so few hands as to be inconsequential.

To address this problem, we use percentile bucketing. If N buckets are being used, the bottom $100/N$ percent of hands are assigned to the lowest bucket. The next bucket is assigned the next $100/N$ percent, and so on. This ensures that all buckets contain nearly a nearly equal of hands. This loses the discriminatory power of having fewer hands in some important buckets, but ensures that all buckets are used.

History Bucketing

In the abstraction used to produce PsOpti4, the bucket boundaries on each round were chosen independently of the buckets observed on previous rounds. For example, a Flop bucket always contained hands within the $E[HS]$ range [x, y], regardless of which Preflop bucket was observed. A hand may have had a much lower probability of progressing from Preflop bucket 1 (a weak hand) to Flop bucket 5 (a strong hand) than progressing from Preflop bucket 5 to Flop bucket 5, but the bucket sequences 1:5 and 5:5 indicate hands with $E[HS]$ values in the same range.

In the CPRG's current bucketing method, **history buckets** are used. We choose ranges for buckets on the Flop, Turn and River that are dependent on the bucket sequence that came before it. This means that instead of one set of buckets on the Flop, we have a different set of Flop bucket boundaries for every Preflop bucket. For example, the hands in Preflop bucket 1 can progress to N



(a) Expected Hand Strength Bucketing on the Flop

(b) Expected Hand Strength Squared Bucketing on the Flop

Figure 2.2: (a) $E[HS]$ and (b) $E[HS^2]$ Bucketing. Each point represents one hand, graphed by its score on the $E[HS]$ and $E[HS^2]$ metrics. Both graphs show the hands on the Flop that were in bucket 2 of 5 on the Preflop. The colors indicate the bucket groupings on the Flop. These graphs were produced by Carmelo Piccione.

Flop buckets with ranges chosen to suit the hands that came from Preflop bucket 1, while a different set of N Flop buckets is created for hands that were in Bucket 2 on the Preflop. We are no longer choosing boundaries for buckets, but for bucket sequences.

For example, assume we are using 5 buckets on each round. The bucket sequence 1:1 (Preflop bucket 1 and Flop bucket 1) might contain cards with an $E[HS^2]$ range of [0, 0.1], while the bucket sequence 1:5 might contain cards with the $E[HS^2]$ range of [0.4, 1]. Since very few low-ranked cards might progress to such a high hand strength, the size of the 1:5 bucket sequence is large to accommodate the top 20% of hands. Alternatively, the bucket 5:1 (Preflop bucket 5 and Flop bucket 1) might contain cards with an $E[HS^2]$ range of [0, 0.3] (since high hand strength hands are unlikely to fall too much), while the bucket 5:5 might contain cards with an $E[HS^2]$ range of [0.9, 1] (since high strength hands are more likely to remain high).

By the time the River is reached, three previous buckets have been observed and a very narrow range of values can be associated with the fourth bucket. This precision gives strategies in these abstract games an advantage over strategies in abstractions without history bucketing. Since our abstractions retain the property of perfect recall (a strategy always remembers its past bucket sequence), using the history bucketing technique does not increase the number of game states or impose extra overhead over not using it.

Summary

Using the techniques described above, the CPRG has created abstract games that use 5, 6, 8 and 10 buckets on each round. We have also constructed abstract games that have 10 and 12 nested buckets: 5 and 6 split by $E[HS^2]$, and then split into two by $E[HS]$. As we increase the number of buckets, we are able to generate poker agents that use strategies that are increasingly effective in the real game

of poker. In fact, we have yet to see an example where increasing the number of buckets has led to an inferior program. However, while the number of bucket sequences grows as the quartic of the number of buckets, the improvement to an ϵ-Nash equilibrium strategy empirically appears to grow only linearly; see Table 3.1 in Chapter 3 and note the performance of CFR5, 6, 8, and 10 against PsOpti4 and Smallbot2298. Thus, compared to the number of bucket sequences, we are obtaining diminishing returns from increasing the size of our abstraction. This is not unexpected, and is a useful property: since a small number of buckets is sufficient for strong play, we can concentrate on small abstractions in which we can quickly produce strategies.

2.6 Related approaches to creating poker agents

There is a long history of techniques for creating poker agents, which have met with varying degrees of success. In the remainder of this chapter, we will examine several of the recent techniques as case studies. Each of these techniques represents either a foundation upon which the techniques in this thesis were based, or a source of agents against which we can compare our new poker agents.

2.6.1 Simulation Based Systems

The Computer Poker Research Group's first poker agents were Loki and its successor, Poki. In addition to heads-up Hold'em, Poki also plays ring Hold'em (more than 2 players). It is not only the CPRG's only poker agent capable of doing so, but is also the best known artificial agent for ring games [4, p. 201].

Poki is described as a **Simulation Based System** [4]. During the Preflop round, Poki uses an expert system to guide the betting strategy. In the post-flop rounds, Poki uses the hand strength metric and a measure of potential as inputs into a formula that determine possible actions. It then uses Monte Carlo roll-outs biased by an opponent model to pick the action with the highest expected value.

In ring games, Poki was found to be competitive with human players of low-limit games, but was defeated by more skilled opposition. In heads-up games, however, Poki was not competitive with even moderate humans. The opponent modeling program Vexbot, to be discussed in Section 2.6.4, was able to defeat it by 601 millibets/hand — a very large margin [27]. Against PsOpti4, an approximate Nash equilibria program that plays a cautious game without trying to exploit its opponent, Poki lost 93 millibets/hand. In other games, Poki has been consistently beaten at 800 millibets/game, which is more than it would lose by folding every hand at the start of each game [4, p. 93].

Poki was designed to play ring games, and heads-up games are significantly different. In heads-up play, aspects of poker such as the need for deception and opponent modeling are much more pronounced. Since neither player is likely to have a very strong hand, players are forced to play with weak hands more often, and as a result must be better at obfuscating their strength and inferring that of their opponents. In ring games, it is more likely that at least one player at the table will have a

strong hand, and so bluffing actions are less likely to succeed. Poki was not designed to excel at tricky play, and so it was too predictable and trusting of its opponents' actions.

2.6.2 ϵ-Nash Equilibria Strategies

In Section 2.1, we described the idea of ϵ-Nash equilibrium strategies and motivated their use. An equilibrium strategy plays close to the Nash equilibrium in its abstraction, making it near-unbeatable by any other strategy in that abstraction. If the abstraction is strategically similar to the real game, then these equilibrium strategies can also be very difficult to defeat in the full game. There have been several approaches towards creating ϵ-Nash equilibria agents for poker. In this section, we will present three examples. In Chapter 3 we will describe a new approach to finding ϵ-Nash equilibrium strategies, and it will be useful to compare it against these existing techniques.

PsOpti4, PsOpti6, and PsOpti7

PsOpti4, PsOpti6, and PsOpti7 play in abstractions similar to the one described above in Section 2.5.4. The PsOpti family of strategies are created by converting the abstract extensive game into sequence form. The sequence form can then be treated as a series of constraints in a linear program, using the approach developed by Koller, Megiddo and von Stengel [16]. An LP solver can then be used to find an ϵ-Nash equilibrium strategy.

In the case of the PsOpti family of programs, however, the LP needed to solve the entire abstract game was too large to fit into a computer's memory. To simplify the problem, two additional simplifications were used. First, action abstraction was used so that one less bet was allowed on each round (2 on the Preflop, 3 on the other rounds). Second, the game was broken up into several smaller parts of the game tree. First, a 3-round Preflop strategy that assumes the players call on the River was solved. This strategy was used only for Preflop play. Then, seven 3-round Flop strategies were created; each one assumes a different Preflop betting sequence (check-call, check-bet-call, etc) was followed. This combination of strategies was then used to compose the overall strategy. The Preflop strategy was only used on the Preflop, after which the post-flop strategy that matched the Preflop betting was followed.

This disconnect between the Preflop strategy and the strategy used in the rest of the game was a flaw in the PsOpti agents that could be exploited. When the CPRG moved to strategies that formed one cohesive strategy for the whole game, such as Smallbot2298 (to be discussed in Section 2.6.2) or the new ϵ-Nash equilibria strategies discussed in Chapter 3, we saw an immediate increase in performance.

GS1 and GS2

GS1 [10] and GS2 [11], produced by Gilpin and Sandholm, are members of the Gameshrink family of ϵ-Nash equilibria agents. In the 2006 and 2007 AAAI Computer Poker Competitions, GS2 and its

successor GS3 have shown to be competitive opponents, taking 3rd place in the 2006 Series event, 3rd place in the 2007 Equilibria and Online Learnings events, and 2nd place in the 2007 No-Limit event.

While their approach shares a common foundation with that of the CPRG's PsOpti agents, they use different terminology. In their approach, a hand strength metric such as our $E[HS]$ is called an **ordered signal**; instead of cards, the agent receives a signal whose value corresponds to the probability of winning the hand. Instead of the term "bucket", they use the term **information filter**, which coarsens the ordered signal into a discrete value.

Both GS1 and GS2 compute a truncated ϵ-Nash equilibrium strategy to use in the Preflop and Flop rounds. For GS1, this truncated strategy considers the betting and cards on the Preflop and Flop rounds; GS2 considers the betting and cards on the Preflop, Flop and Turn rounds. Both programs use terminal nodes after this point to represent the estimated outcome of the remaining rounds. This estimate of the outcome depends on the remaining cards (the probability of winning) and the remaining betting (the magnitude of the win or loss). Both GS1 and GS2 find the probability of winning by rolling out the remaining filtered signals. GS1 assumes that both players call during the Turn and River rounds. GS2 uses an estimate of the betting strategy for Sparbot (the commercial name for PsOpti4), conditioned only on the cards and not on the earlier betting, as an estimate for the betting actions by both players on the River.

After the Turn card is dealt, in real time, the program constructs a linear program and solves it to find a new equilibrium strategy to use on the Turn and River rounds. At this point, the Preflop, Flop, and Turn cards are known, so the LP uses a precomputed abstraction that corresponds to this particular combination of Flop and Turn cards. For speed, 135,408 of these precomputed abstractions are calculated offline. This provides a much more precise idea of the hand strength at this point in the game than the past filtered signals (buckets) could provide. Bayes rule is used to find estimates of the opponent's hand strength given their Preflop and Flop betting, with the assumption that the opponent is also playing according to the precomputed strategy. The LP to be solved is thus designed for the current game state: it considers only the betting history that actually occurred, and the abstraction is tailored for the public cards that were dealt.

This LP is then solved in a separate process while the game is being played. Whenever an action is required, the LP can be interrupted and an intermediate solution can be used to choose an action. While this allows the agent to produce actions on demand, there is a danger that, as the LP solution is further refined, this intermediate action is no longer part of the solution to the LP. This is called "falling off the tree" — since the strategy would never have selected that action (it is no longer part of the game tree), it is unable to suggest future actions after that action. In these cases, the agent simply calls for the remainder of the game.

The main advantage of this approach is that, in theory, the equilibria strategy for the Turn and River rounds will be more accurate than an equilibria strategy in a normal bucket abstraction. This

is because of the precise card information from the Flop and Turn cards.

However, there are several disadvantages to this approach. One significant disadvantage is that solving the LP in real-time is not a task that can be computed quickly. In the 2006 AAAI Series event, the players were allocated up to 60 seconds per hand to make their decisions. This was an unusually large allotment of time for poker games. In the 2006 AAAI Bankroll event, there was a 7 second time limit per game. In practice, even generating the LP cannot be done quickly enough to be used during play.

Just as the Preflop / Postflop split was a disadvantage for the PsOpti agents, the split between early game and late game play was a large disadvantage for GS2. By assuming that both players will only call during the Turn and the River, the Preflop and Flop strategy does not have an accurate estimate of the utility of certain lines of play. In poker, there is a concept called **implied odds** that is relevant here; if you have a strong hand that is likely to win, you need to consider not only the money in the pot but also your bets that your opponent must call in future rounds. Since the Turn and River rounds have larger bet increments than the Preflop and Flop, assuming that both players will simply call (as GS1 does) gives too little utility to strong hands, and too much utility to weak hands. Another disadvantage is the assumption that the opponent's hand distribution is dependent on GS2's preflop strategy. If the opponent does not follow the same Preflop betting strategy as GS2 (and playing by a different strategy is very likely), then the LP used later in the game assumes an inaccurate strength for the opponent's hand.

Smallbot2298

Smallbot2298 [34] is a recent addition to the CPRG's collection of ϵ-Nash equilibrium agents, and before the agents produced by the techniques described in this thesis, it was our strongest agent. It was produced by a method that does not directly involve solving a linear program, as was used for the PsOpti family and for GS2. It was also one of the first ϵ-Nash equilibrium strategy to use one consistent, whole-game strategy, as opposed to the overlapping strategies used for PsOpti and GS2.

Smallbot2298 was produced by the novel "Range of Skill" algorithm developed by Zinkevich et al [34]. The guiding idea is to consider creating a sequence of agents, where each one can defeat the agents earlier in the sequence by at least ϵ. For any game and a value for ϵ, this sequence has a maximum length — eventually, the sequence approaches within ϵ of a Nash equilibrium, and no further agents are able to defeat it by more than ϵ. An illustrative example is to consider games such as Go, Chess, and Tic Tac Toe. The sequence of human Tic Tac Toe players would be very short; one claim that Go is more complex than chess is that the sequence of human experts for go is longer than that of chess [21].

To construct this sequence of agents, we apply two algorithms. The first algorithm, generalized best response, considers a bimatrix game where player 1 is restricted to a set of allowed strategies S'_1, and player 2's set S'_2 contains one arbitrary strategy. Next, we compute a Nash equilibrium in

this bimatrix game, where players 1 and 2 produce mixed strategies from the sets S'_1 and S'_2. We then calculate the best response to player 1's equilibrium strategy, and add it to S'_2, giving player 2 another option. After many repetitions of this algorithm, we return player 2's half of the equilibrium strategy, σ_2, as a best response to player 1's entire set of strategies S'_1.

The second algorithm, range of skill, repeatedly calls the generalized best response algorithm. We start by initializing a set of "allowed strategies" to contain an arbitrary strategy, and use generalized best response to calculate an equilibrium strategy for one player in the restricted game where the other player may only mix between the set of allowed strategies. This resulting strategy is capable of defeating any of the strategies in the set. We then add that strategy to the set of allowed strategies, and call generalized best response again.

Each strategy returned by the generalized best response algorithm in this way is a member of the "range of skill" described earlier, as each one is capable of defeating the strategies generated before it. As our sequence of strategies grows, the most recent strategies at the end of the sequence approach a Nash equilibrium.

Memory requirements are both the main advantage *and disadvantage* of the Range of Skill algorithm. Since the generalized best response algorithm only considers finding an ϵ-Nash equilibrium in the restricted bimatrix game where players must pick from a fixed set of strategies, it avoids the problem of solving an LP for the 4-round abstract game that hinders the PsOpti and Gameshrink approaches. This memory efficiency is what made it possible to produce one of the first full-game strategies for Texas Hold'em, avoiding the strategy fragments that the PsOpti and Gameshrink programs used.

The main disadvantage of the Range of Skill algorithm is the amount of intermediate data that must be stored. Each intermediate strategy generated by the algorithm must be stored for use in future steps, such as when constructing the utility matrix. This data can be stored on a hard disk as opposed to in RAM, but must still be loaded and unloaded intermittently. The storage required for the strategies and the time cost of loading these strategies from disk can make this approach difficult to scale to larger abstractions.

2.6.3 Best Response

Given a poker strategy σ, it is natural to wonder what the strongest possible opponent to that strategy is. This is a common question in game theory, and the counter-strategy is called the **best response**. Calculating a best response to the opponent's strategy is very computationally expensive, and so we compromise by calculating a best response strategy to σ in the same abstract game that σ plays. The resulting strategy is an approximation of the real best response, and we call it an abstract game best response. At every information set, the abstract game best response strategy chooses the action that maximizes its utility, given the probability of σ reaching every terminal node descendant from every game state in the information set and the utility associated with that terminal node. A formal

description of this algorithm will be presented in Chapter 4.

The abstract game best response strategies allow us to answer important questions about other strategies. For example, if σ is an ϵ-Nash equilibrium, playing the abstract game best response strategy against σ tells us how far from the equilibrium σ is in its own abstraction — in other words, it tells us the ϵ for this ϵ-Nash equilibrium strategy. If we intend to use an exploitative program to win as much money as possible from σ, the abstract game best response gives us a lower bound on the exploitability of σ. Since the abstract game has less information than the full game, a best response in the full game (or in a more informative abstract game) can achieve an even larger result than an abstract game best response.

The abstract game best response approach has requirements that limit its use. First, the algorithm needs to know how the strategy acts at every information set. This means that it is difficult to calculate an abstract game best response to an opponent's strategy, unless they choose to provide you with its details. Second, the abstract game best response has to be calculated in the same abstraction as the original strategy. These factors limit its usefulness against arbitrary opponents, but it remains a useful tool for evaluating other strategies that you create.

2.6.4 Adaptive Programs

Vexbot [5] and BRPlayer [27] are different types of poker agent than those that we have discussed so far. Vexbot and its successor, BRPlayer, are adaptive programs that build an opponent model online, and use strategies calculated to exploit that model.

Vexbot and BRPlayer use miximix search. This is similar to the expectimax search technique used for stochastic games, extended to accomodate imperfect information games. In a miximix search, we search a game tree where an opponent model predicts the opponent's action probabilities at each of their choice nodes, and predicts the utility for reaching each terminal node. A special case of miximix search, called miximax, chooses the one action at each information set that maximizes the expected utility. In miximix search, the goal is to obtain a high utility value, but to use a mixed strategy to avoid being predictable to the opponent.

The opponent model needed by miximix search must predict two types of information: opponent action probabilities at their choice nodes, and the utility for reaching each terminal node. To predict the opponent action probabilities, Vexbot and BRPlayer measure the frequency of the opponent's actions for each betting history. Note that this approach does not consider the public card information. During the miximix search, at opponent choice nodes, these frequency counts are used to predict their actions.

The utility for reaching a terminal node is the product of the size of the pot (determined by the betting sequence) and the probability of winning (determined by the players' private cards and the public cards). Since the betting sequence and our own cards are known, we need to predict the strength of the cards the opponent is holding. For each terminal node, Vexbot and BRPlayer maintain

a histogram that stores the observed frequency with which the opponent reached that terminal node with given ranges of hand strength. For example, at a terminal node with a large pot, the opponent might have displayed cards with a hand strength in the range 0.0-0.2 twice (when they bluffed) and in the range 0.8-1.0 four times. This histogram can then be used to predict our probability of winning with a particular hand. Using the earlier example, if Vexbot has a hand strength of 0.5, it can expect to win $2/6$ times (the frequency with which the opponent bluffs) and lose $4/6$ times (the frequency with which the opponent has legitimate hands).

On each hand of play, Vexbot and BRPlayer use miximix search and their opponent model to choose actions. As each action and showdown is observed, the opponent model is updated, to provide a more accurate model for future hands. Early in the game, observations are still sparse and the terminal node histograms described above may not yet have enough information to make accurate predictions. To solve this problem, Vexbot considers the histograms of similar terminal nodes and weights them according to their similarity and number of observations. BRPlayer considers more types of similar situations than Vexbot does, and this is the difference between the two implementations [27, p. 73].

The main advantage of Vexbot and BRPlayer is their ability to learn online and adapt their play to exploit their opponent. In experiments performed by Billings and colleagues [5], Vexbot was shown to defeat PsOpti4, and to defeat a variety of opponents by a much greater margin than PsOpti4 did. This shows the value of an adaptive strategy over a Nash equilibrium strategy. As we mentioned in our description of poker, exploitation is important: the goal is not necessarily to *not lose* , but to win as much as possible from each opponent.

Vexbot and BRPlayer have several disadvantages. One of these is that at the start of a match, Vexbot and BRPlayer do not know the rules of poker, and they can develop impossible beliefs about the game. The miximax search models the effect of chance nodes on their own hands, but there is nothing in the program or the opponent model that defines how the opponent's hand is drawn from the deck. Instead, the knowledge about the possible hands the opponent can have is learned through observation, and is represented in the histograms at the terminal nodes. If the game starts with a lucky set of cards for the opponent, for example, Vexbot and BRPlayer can learn that their opponent *always* wins at showdowns and *never* folds. These observations can affect the miximix search such that it avoids lines of play where it (incorrectly) believes it will always lose; it may take many more hands of observations to correct this mistake.

An additional disadvantage is that Vexbot and BRPlayer can require a large number of hands before developing a useful model of their opponent. Schauenberg ran three experiments where BRPlayer tried to model and defeat PsOpti4 [27, p. 70]. In one of these experiments, BRPlayer required approximately 15,000 games before it did better than break even against PsOpti4. In the other two experiments, BRPlayer required approximately 170,000 games before it did better than break even. Competitions such as the AAAI Computer Poker Competition obtain their results by

running several matches of at most 3,000 games. In this short of a timespan, Vexbot and BRPlayer may not have enough time to develop a useful opponent model.

2.7 Teams of programs

If you have several poker agents to choose from, one approach is to combine the agents into a team and use a meta-agent to select which agent to use for any given hand, based on each agent's past performance. This is a difficult task, as poker has high variance from hand to hand. If the variance is not taken into account, an unlucky series of hands may give one agent such a poor past performance that it will not be chosen again. In this section, we will present one past example of a team of agents that the CPRG has entered into a competition, and discuss an established framework for using teams of agents that will be used later in this thesis.

2.7.1 Hyperborean06 and Darse's Rule

In the 2006 AAAI Computer Poker Competition, the CPRG entry, called Hyperborean, was actually two programs: PsOpti4 and PsOpti6, as described in Section 2.3.3. Although both are ϵ-Nash equilibria programs and PsOpti4 is less exploitable (and thus by one metric, superior to) PsOpti6, we found that the two agents played with slightly different styles. Against some opponents, PsOpti6 performed better than PsOpti4, and vice versa. Another motivation for using PsOpti6 was that PsOpti4 was publicly available under the name Sparbot in the commercial program Poker Academy. Although PsOpti4 was believed to be stronger, it was important to have another strategy to switch to in case an opponent entered a counter-strategy to PsOpti4. To try to increase our winnings against exploitable opponents, and to avoid being exploited by a counter-strategy, the two programs were entered as a team, with a "coach" agent that would choose which agent to use on each hand.

The coach was guided by a simple rule called "Darse's Rule". After each hand played by either agent, we knew the score for that hand and, if the hand went to a showdown, the opponent's private information. Since the score on each hand has high variance (+- 6 sb/g, as mentioned in Section 2.2), it used a variant of DIVAT analysis called Showdown DIVAT to reduce the variance when choosing which strategy to use.

For each agent, we tracked the sum of the Showdown DIVAT scores from the hands they had played. On each hand, we divided the accumulated Showdown DIVAT score by the square root of the number of hands that agent had played; the agent with the higher resulting value was chosen to play the next hand. This algorithm was slightly adjusted by variables to give an advantage to programs that had not yet been used, to ensure that each agent received a chance to display their value.

In a post-tournament analysis, we found that PsOpti6 was more effective than PsOpti4 when playing against some opponents, and that the coach using Darse's rule selected PsOpti6 more often in such circumstances. By using the coach and selecting between the two programs, Hyperborean06

performed better than either strategy on its own would have. We will revisit this idea in Chapter 6, where we will show that a team of robust agents can be more effective than an equilibrium strategy against unknown opponents.

2.7.2 UCB1

UCB1 is a regret minimizing algorithm for acting in the multi-arm bandit task [1]. In the multi-arm bandit task, there are several slot machines that each have a different unknown reward distribution. The goal is to maximize the reward gained from the machines. Since the reward distribution for each machine is not known, a policy must be established that trades off exploration (trying a variety of machines to learn the expected value of each one) and exploitation (using the machine with the highest experimental expected value to maximize earnings).

Define an allocation strategy A to be a method for choosing the next slot machine to play, given the history of choices and rewards. Then, we can define our **regret** to be the difference between the reward we *could* have received by always picking the best slot machine and the reward we *did* receive by following allocation strategy A. Regret is thus similar to the economics notion of an opportunity cost: it is the penalty you suffer by not picking a different choice.

The UCB1 algorithm defines an allocation strategy where the regret from following the strategy grows logarithmically in the number of trials; this means that the *average* regret approaches zero. This makes it useful for selecting which slot machine to use. Our task in managing a poker team is very similar. If we are playing against a stationary opponent, then there is an unknown utility for playing each of our agents against the opponent. The utilities are subject to variance and we may not have prior knowledge of their distribution. During the match, we need to try different agents to find out which has the highest expected utility, and also maximize our winnings by using our current best agent.

The UCB1 allocation strategy is [1]:

Initialize by using each agent once. Then, use the agent j that maximizes $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$, where $\bar{x}_j$ is the average utility for agent j, n is the total number of trials, and n_j is the number of times agent j has been selected.

(2.3)

In Section 6, we will describe how to use the UCB1 algorithm to choose agents from a team of strategies when competing against opponents.

2.8 Summary

In this chapter, we have presented examples of several techniques for poker agents. In the remainder of this thesis, we will present a series of techniques that each surpass the approaches we have just described. We will show how to calculate approximations to a Nash equilibrium in larger abstractions than previously possible, how to calculate "good responses" with fewer restrictions than the traditional best response algorithm, and robust counter-strategies that exploit an opponent while minimizing their own exploitability. We will begin presenting the contributions of this work with Counterfactual Regret Minimization, a new technique for finding ϵ-Nash equilibrium strategies.

Chapter 3

Playing to Not Lose: Counterfactual Regret Minimization

3.1 Introduction

In this chapter, we will present a new technique for finding ϵ-Nash equilibrium strategies, which were first mentioned in Section 2.1. Examples of past approaches to finding ϵ-Nash equilibria strategies were discussed in Section 2.6.2. These programs approximate an unbeatable strategy, with the intent that they will do no worse than tie against any opponent. If the opponents (humans, for example) make mistakes, then the equilibrium strategy can win over time. This property makes equilibrium strategies valuable in any game, and they have been particularly successful in poker.

In Section 2.6.2, we described three past approaches for finding equilibria strategies (such as PsOpti, Smallbot and GS2) in Heads-Up Limit Texas Hold'em. One common drawback to these approaches is that they all require memory linear in the number of game states. Past results have shown that the quality of our strategies directly improves as we increase the size of our abstraction; in general, an equilibrium in a 5-bucket game will lose to an equilibrium in a 6-bucket game. While there is an orthogonal advantage to improving the *quality* of the abstraction, we are interested in solving the largest abstractions possible. In short, we want to use the largest abstraction available that fits within the limits of the memory of our computers and the time we have available.

In this chapter we will propose a new approach, called Counterfactual Regret Minimization (CFR), which requires memory linear in the number of *information sets*, not game states. This allows us to solve much larger abstractions than were possible with the previous methods. This produces ϵ-Nash equilibrium strategies that are closer to the real game's Nash equilibrium. Using this approach, we found ϵ-Nash equilibria strategies in abstractions two orders of magnitude larger than had been achieved by previous methods. [1]

[1] The theoretical background of the Counterfactual Regret Minimization technique was developed by Zinkevich, and is presented in a paper coauthored by Zinkevich, the author, Bowling and Piccione [36]. The theoretical background of CFR is not a contribution of this thesis. The author's contribution in this chapter is the practical implementation, optimization and parallelization of a program for generating CFR strategies. In this chapter, we will present an overview of the technique, and will highlight the author's contributions and the results.

In Section 3.2, we will provide a high-level description of the technique. In Section 3.3, we will explain the key theoretical foundations of the technique; the full details and theoretical proofs can be found in a separate technical report [35]. In Section 3.4.3, we will explain the author's main contribution to the technique: the optimizations and parallelization that allow the technique to efficiently solve one of the largest known poker abstractions. In Section 3.5, we will compare the poker strategies produced by this technique to several benchmark programs. Finally, in Section 3.6 we will summarize the technique and reconsider the advantages and disadvantages of ϵ-Nash equilibrium strategies.

3.2 Overview

We first encountered the concept of **regret** in Section 2.7.2, in our description of the UCB1 algorithm for choosing the best agent from a team of poker players. Informally, regret is similar to the economics concept of an **opportunity cost** — if you take some action a and receive utility $u(a)$ when you could have taken a utility maximizing action a^* to receive utility $u(a^*)$, then your regret is the difference $u(a^*) - u(a)$.

In the CFR approach, we will construct two half-strategies, σ_{dealer} and $\sigma_{opponent}$, who will play repeated games of poker against each other. We refer to them as "half-strategies" because each only plays half of the game: one plays from the dealer's perspective, and the other plays from the opponent's perspective. They will start the match with an arbitrary set of action probabilities, such as folding, calling and raising from all information sets with equal probability. After each hand, the half-strategies will adjust their play to minimize their regret. Specifically, at each information set, we will establish a regret minimizing agent that acts so as to minimize the regret of its own subtree, given the opponent's current strategy.

As the number of training games played by the pair of half-strategies increases, their regret minimizing behavior will cause them to approach a Nash equilibrium in the abstract game. At any point, we can combine the half-strategies into a full strategy and measure their progress towards this goal by considering the best response. As the number of training games increases, the utility of the best response to the strategy falls. In Section 3.5, we will present experimental results that show the rate at which the strategy's exploitability decreases. The improvement in play is dramatic at first, and eventually succumbs to diminishing returns, where each additional hand results in only a small improvement. By graphing the improvement in play in several different sizes of abstractions, we also show that the number of iterations needed to reach a certain level of exploitability is depends linearly on the number of information sets.

The benefits of this approach are as follows:

- **Lower memory requirements**. Since CFR requires memory linear in the number of information sets, it requires fewer resources than traditional linear programming methods.

- **Parallel computation**. In Section 3.4.3, we will present a method for parallelizing the algorithm so that more computers to be used. This improves the speed of the program and also uses more memory distributed across multiple machines, allowing us to solve larger abstract games.

3.3 Formal Description

3.3.1 ϵ-Nash Equilibria, Overall Regret, and Average Strategies

In this section, we will explain how we can approach a Nash equilibrium through regret minimization. We will start by considering what it means to measure our regret when we pick a strategy to play a game like poker. Player i plays T games of poker and their strategy $\sigma_i^t \in \Sigma_i$ is the strategy they use on game t. Then, the **average overall regret** of player i over this series of T games is:

$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} \left(u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t) \right) \tag{3.1}$$

The player's average overall regret is thus the average of the differences in utility between the strategies chosen for each game, and the single strategy that would have maximized utility over all T games.

We can also define the **average strategy** used by player i used during this series of games. For each information set $I \in \mathcal{I}_i$, for each action $a \in A(I)$, we define the average strategy as:

$$\bar{\sigma}_i^T(I)(a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}. \tag{3.2}$$

Thus, the average strategy defines its action probabilities for each information set to be the action probabilities of each strategy, weighted by the probability of the strategy reaching that information set.

In [36, Theorem 2], Zinkevich states the following well known theorem that links the concepts of the average strategy, overall regret, and ϵ-Nash equilibria:

Theorem 3 *In a zero-sum game at time T, if both players' average overall regret is less than ϵ, then $\bar{\sigma}^T$ is a 2ϵ equilibrium.*

Let us now consider the sequence of strategies that each player selects over the T games. If both players are learning over the course of these games such that their average overall regret approaches 0 as t goes to infinity, then their average strategies approach a Nash equilibrium. What is needed, then, is a **regret minimizing** algorithm for selecting σ_i^t. Such an algorithm can then be used in self-play over a series of games to approach an ϵ-Nash equilibrium. After any number of games, we can find the value of ϵ by measuring the average overall regret for all players, or by calculating a best response to the strategy profile σ. Additionally, if our regret minimizing algorithm provides bounds on the average overall regret, then it also bounds the rate of convergence to the Nash equilibrium.

3.3.2 Counterfactual Regret

We will propose a new regret minimization algorithm for selecting the sequence of strategies. Instead of trying to minimize one regret value for an entire game, we will instead decompose our regret minimization task into several small scenarios where we can minimize regret independently. We call these independent regret values **counterfactual regret**, and we will minimize the counterfactual regret values encountered at every information set. The sum of our counterfactual regret forms a bound on the overall regret; by minimizing our counterfactual regret, we thus minimize our overall regret, and approach a Nash equilibria.

We will start by considering the utility for a player to reach a certain information set, and the utility resulting from each action they can take from that information set. Let us define $u_i(\sigma, h)$ as the expected utility for player i if the players reach history h and then play according to strategy profile σ. We define **counterfactual utility** $u_i(\sigma, I)$ to be the expected utility given that information set I is reached and all players play using strategy profile σ except that player i plays to reach I. Recall that our notation for the probability of reaching a particular information set I, given that player i is trying to do so, is $\pi^{\sigma}_{-i}(I)$. We call this "counterfactual" utility because it is the value to player i of reaching information set I *if the player had tried to do so* .

We need to define one final term in order to consider strategies that diverge only at the actions the player is considering at one information set. Define $\sigma|_{I \to a}$ to be a strategy profile identical to σ, except that player i will choose to take action a whenever I is reached. We can now define **immediate counterfactual regret** to be a player's average regret for their actions at I, if they had tried to reach it:

$$R^T_{i,\text{imm}}(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(I) \left(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I) \right) \tag{3.3}$$

In this equation, a is selected as the maximum utility action to take in I over the T samples; $u_i(\sigma^t|_{I \to a})$ is the utility for player i of all players playing according to σ^t, except for i selecting action a at I. We subtract the utility $u_i(\sigma^t, I)$ to find our regret. To find our **immediate counterfactual regret**, we weight this utility with the probability that the other players play according to σ^t. We need only to minimize the positive portion of the regret, and so we define $R^{T,+}_{i,\text{imm}}(I) = \max(R^T_{i,\text{imm}}(I), 0)$.

With this background established, we can now present our first key result:

Theorem 4 $R^T_i \leq \sum_{I \in \mathcal{I}_i} R^{T,+}_{i,\text{imm}}(I)$

Theorem 4 states that average overall regret is bounded by the sum of the independent immediate counterfactual regret values. Through Theorem 3, we then find that by minimizing immediate counterfactual regret, our strategy profile σ approaches a Nash equilibrium. All that remains is to provide a strategy for minimizing immediate counterfactual regret at each information set.

3.3.3 Minimizing Immediate Counterfactual Regret

In this section, we will show how counterfactual regret can be used to set the action probabilities of a strategy, in order to minimize counterfactual regret on future iterations. With this accomplished, we will have completed our task — by finding a player's counterfactual regret and updating its action probabilities according to two equations, we will have specified a system for using self-play to find ϵ-Nash equilibria.

In Equation 3.3, we stated that our immediate counterfactual regret was a consequence of our choice of actions at that information set. In that equation, we considered our regret to be the difference in utility between the actions taken by our strategy σ_i, and the single action that would have maximized our utility for all examples of that information set, weighted by the probability of reaching that information set on iteration t, if we had tried to reach that information set. Now, we will instead measure our regret for not taking each possible action. Associated with each action in each information set, we will maintain the following regret value:

$$R_{i,imm}^{T}(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)\left(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I)\right) \tag{3.4}$$

These $R_i^T(I,a)$ values tell us, over T samples, how much we regret not folding, calling, or raising instead of taking the actions specified by σ_i. As we are concerned with positive regret, we define $R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0)$. Then, to determine the new strategy to use at time $T+1$, we set the action probabilities as follows:

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & \text{if } \sum_{a\in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases} \tag{3.5}$$

The relationship between positive counterfactual regret and the new strategy's action probabilities is simple. Each action is selected in proportion to the accumulated positive counterfactual regret we have had for not selecting that action in the past. In information states where there is no positive regret, we set the action probabilities to a default setting — in this case, equal probability to each action.

3.3.4 Counterfactual Regret Minimization Example

The above formal description of minimizing counterfactual regret may appear complex, but the computation is actually straightforward. We will now present a short example of how counterfactual regret is minimized at one choice node.

First, it is important to recognize that when the algorithm is calculating the counterfactual regret and adjusting the action probabilities for both strategies, a game is not actually being played. Instead, the algorithm has one strategy for each player, and it knows the details of those strategies. The algorithm can calculate the expected value for each of a player's actions by simulating the remainder

of the game with those strategies and the possible chance outcomes that can occur. If one takes the perspective of the strategy and not the algorithm, this appears impossible — how can the strategy calculate the expected value, since it does not know its opponent's action probabilities? This misconception is common, and it is important to remember that it is the *algorithm* that uses its perfect information to calculate regret and adjust the action probabilities.

For our first example, we will refer to Figure 3.1. The algorithm starts by initializing both strategies to a uniform distribution: at each choice node, the strategies for player 1 and 2 will fold, call, and raise with equal probability. In this example, we will consider the actions taken by a player at one of the many information sets. This information set occurs near the start of the game. Each player has received cards, and the player's cards are in bucket 3 of 5 (a moderate hand). The opponent has just made their first action: a bet. With their cards, according to their strategy, they would have placed this bet half of the time.

By examining the game tree following each action and using the two players' strategies, we can find the expected value of each action for the player. In this example, we find that the expected value for folding, calling and raising is -3, 6, and 9, respectively. The player's action probabilities (1/3, 1/3, 1/3) are given beside the name of each action. We can now find the current expected value for this player at this choice node, given the players' strategies and the chance outcomes: $-3 * 1/3 + 6 * 1/3 + 9 * 1/3 = 4$.

Next, we calculate the regret for each action. Recall that regret is the difference in expected value between always taking an action and the expected value of the strategy. For example, the regret for folding is $(-3 - 4 = -7)$. The regret for calling and raising is 2 and 5, respectively. If we use the usual English meaning of regret, we could say that we "regret" not calling or raising more often, and that we "regret" folding as much as we did.

As mentioned previously, counterfactual regret is regret weighted by the opponent's probability of reaching this information set. In this example, the opponent had a 0.5 probability of betting, and so we weight the regret for each action by 0.5, giving the player counterfactual regrets of (-3.5, 1, 2.5). We will keep track of the accumulated counterfactual regret from every time we visit this information set.

Finally, we assign new action probabilities at this information set proportional to the positive accumulated counterfactual regret. This means that there is no probability of folding (because the fold counterfactual regret is negative), a $(1/2.5 \approx 0.3)$ probability of calling, and a $(2.5/3.5 \approx 0.7)$ probability of raising. On future visits to this information set, such as in the next example, this probability will be used.

To show the importance of weighting the regret by the probability of the opponent's actions, we will present a second example in Figure 3.2. In the first example, the player likely had a stronger hand than the opponent: they won if the called or raised. In this example, the opponent has a stronger hand. This results in a negative expected value for each action, and they also bet with probability 1.

As in the first example, we calculate the expected value for each action (-3, -6, -9), and using the probabilities chosen earlier, we find the expected value of the new strategy to be -8.1. The regret for each action is (5.1, 2.1, -0.9). This means that we "regret" not folding or calling more often in this game state.

To find counterfactual regret, we weight the regret by the opponent's probability of reaching this information set. In this example, this probability is 1. The effect of this weighting is that it places more weight on regrets that are likely to occur. The regrets incurred in this second example are given twice the importance as the regrets incurred in the first example, as this situation is more likely to occur.

By adding the counterfactual regret from this example to that of the first example, we arrive at an accumulated counterfactual regret of (1.6, 3.1, 1.6). We then assign new action probabilities proportional to this accumulated counterfactual regret: ($1.6/6.3 \approx 0.25$, $3.1/6.3 \approx 0.5$, $1.6/6.3 \approx 0.25$). This set of action probabilities will be used on future visits to this information set.

These two examples demonstrate how, over time, the action probabilities at each information set are changed to minimize the counterfactual regret caused by any of the game states in the information set.

3.3.5 Bounds on Regret

Theorem 5 *If player i selects actions according to Equation 3.5 then $R^T_{i,\text{imm}}(I) \leq \Delta_{u,i}\sqrt{|A_i|}/\sqrt{T}$ and consequently $R^T_i \leq \Delta_{u,i}|\mathcal{I}_i|\sqrt{|A_i|}/\sqrt{T}$ where $|A_i| = \max_{h:P(h)=i} |A(h)|$.*

The range of utilities ($\Delta_{u,i}$), the maximum number of actions ($\sqrt{|A_i|}$), and the number of information sets ($|\mathcal{I}_i|$) are constants; as the number of iterations ($\sqrt{T}$) is in the denominator, Theorem 5 shows that average overall regret decreases as we increase the number of iterations. With this result, we conclude that by updating our probabilities according to Equation 3.5, we can use self-play to approach a Nash equilibrium. We also note that the theorem tells us that the average overall regret grows linearly with the number of information sets, and the number of iterations required grows quadratically with the number of information sets.

In the next section, we will explain how we have applied this technique to Texas Hold'em.

3.4 Applying Counterfactual Regret Minimization to Poker

In this section, we will describe the details of applying this technique to the game of Limit Heads-Up Texas Hold'em. We will start in Section 3.4.1 by describing a general implementation of the algorithm described above in Section 3.3. In Section 3.4.2, we will describe the poker specific program we use, which takes advantage of features of the game to more quickly approach a Nash equilibrium. Then, in Section 3.4.3, we will discuss optimizations made by the author to enable

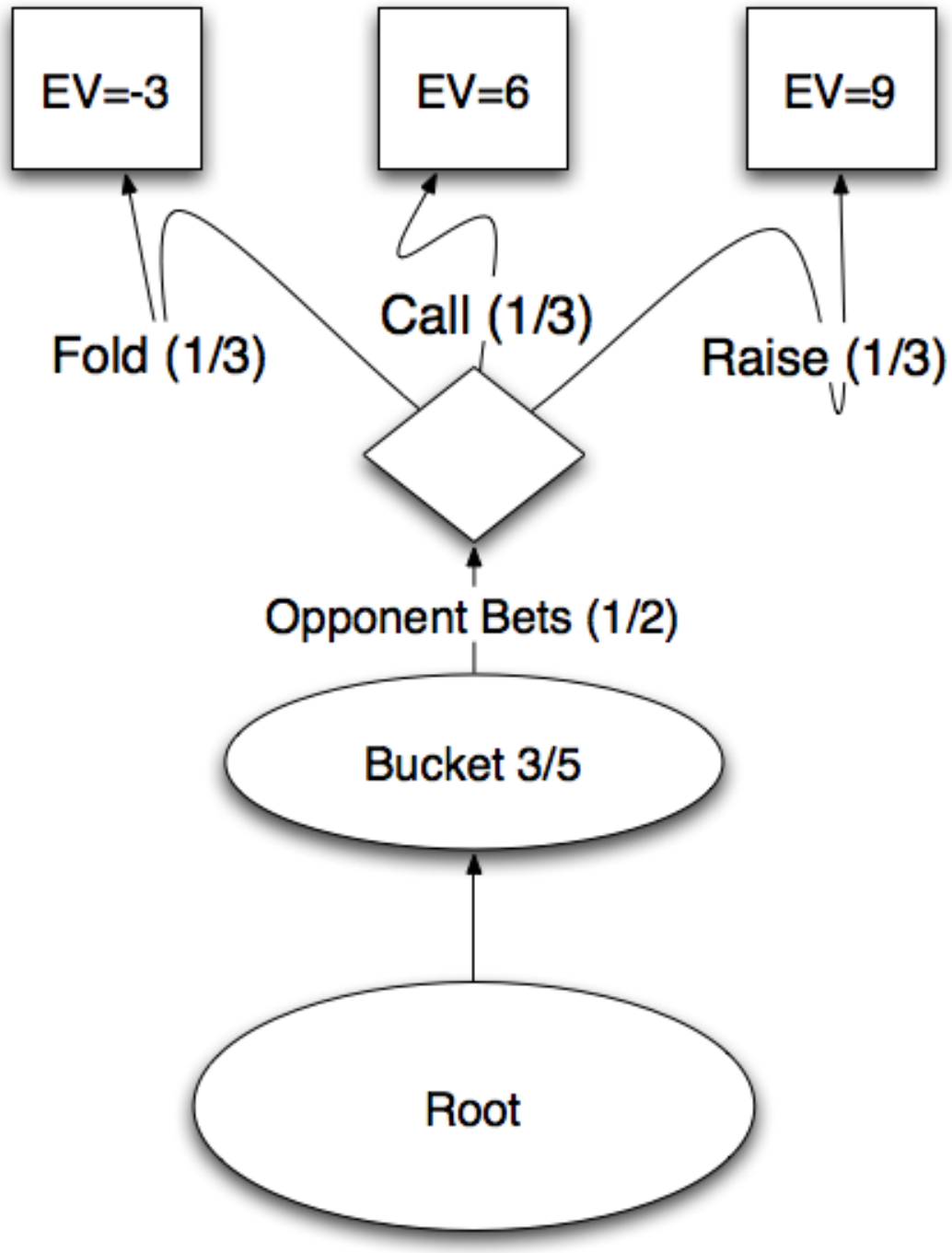


Figure 3.1: The first example of counterfactual regret minimization at a choice node. See Section 3.3.4 for an explanation.

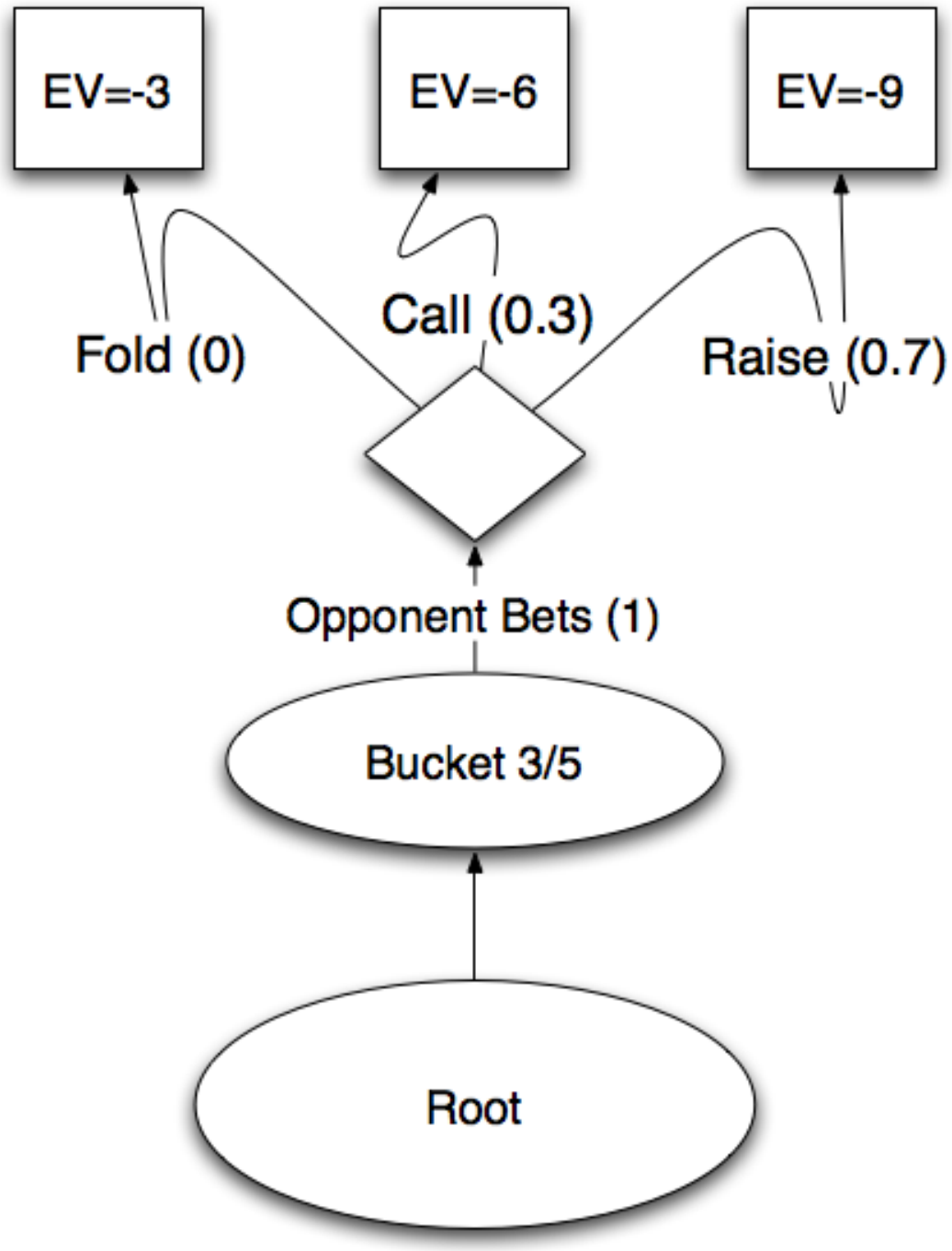


Figure 3.2: The second example of counterfactual regret minimization at a choice node. See Section 3.3.4 for an explanation.

this poker specific program to solve poker abstractions two orders of magnitude larger than had previously been achieved.

3.4.1 General Implementation

To implement this program to solve general extensive form games, we first create an information set tree for each player. This tree represents the game as viewed by one player; each information set from the extensive form game tree is represented by a choice node in the information set tree. At each choice node in the information set tree, we will track the accumulated counterfactual regret $R_i^T(I, a)$ for each action, and the action probabilities for that player's strategy. We initialize the strategy to a default setting, and then begin playing a series of games between the two players. We will refer to each game as one iteration.

Equation 3.4 and Equation 3.5 are the two key functions of the program for implementing this algorithm. On each iteration, for each player, we walk over the extensive form game tree. Note that the extensive form game tree does not have to be stored in memory, as we only have to visit each of its states. At each choice node for a player, we use Equation 3.4 to calculate the regret for each action, and we add this to the accumulated counterfactual regret associated in the corresponding node of the player's information set tree. Then, we walk each player's information set tree, and use Equation 3.5 to update the action probabilities at each choice node. Since each iteration walks the entire extensive form game tree, every iteration is considering all of the possible chance outcomes for the players.

Since this approach requires only that we walk the extensive form game tree but not store it, our memory requirements are only the two information set trees. Using this general approach, as we increase the number of iterations, the average strategies stored in the two information set trees will approach a Nash equilibrium.

In Section 3.3.5, we stated the number of iterations needed grows quadratically with the number of information sets. In poker, this bound can be made even tighter, as described in the technical report [35, Equation 25]. When using the general implementation for poker, the number of iterations relies on the number of betting sequences, but does not depend on the size of the card abstraction: if we were to double the size of our card abstraction, we only have to perform the same number of iterations to reach the same distance from a Nash equilibrium. However, since each iteration visits every game state, the *time cost per iteration* does increase.

3.4.2 Poker Specific Implementation

The game of poker has far more game states than information sets. This is because of the private information that our opponent holds; in the real game, there are 2,450 card combinations (169 after removing isomorphisms) that they could be holding. In an abstraction that has 5 buckets on each round, there are 5 game states for every Preflop information set; on the Flop there are 25, 125 on

the Turn and 625 on the River. Since a strategy must use the same actions in each game state in an information set, it may not be necessary to consider all of the game states to choose one action distribution for the information set, if instead we consider a representative sample of the game states.

In the general implementation described earlier, each iteration involved considering all possible private card holdings for both players. In our poker specific implementation, we will randomly sample a hand for each player and the board cards, and then walk only the portion of the extensive form game tree where those chance outcomes occur. In our abstractions of poker, we refer to this sequence of private and public cards as a **joint bucket sequence**. Another perspective on this difference between the general and poker specific implementation is that in the general implementation, the chance player plays according to a mixed strategy, where all chance outcomes are possible. In the poker-specific implementation, the chance player plays a pure strategy, where only one outcome is possible at each chance node, and this pure strategy changes on each iteration. For each iteration, a large part of the extensive form game tree is now unreachable, because the chance player's actions only allow for 18,496 states and 6,378 information sets to be reached. On each iteration, only one game state in each reachable information state is considered; over time, by visiting a sample of the states in each information set, the strategy's actions become increasingly suited for all of the game states in the set. In Section 3.5, we will present experimental results that show that this poker specific implementation causes a quadratic decrease in the number of states visited per iteration, and only causes a linear increase in the required number of iterations.

3.4.3 Optimizations

Part of the author's contribution to this work was to develop a fast, memory efficient implementation of the algorithm. In this section, we will briefly touch on the speed and memory optimizations performed, and describe how the algorithm can be parallelized.

Speed and Memory improvements

Since the strategy profile converges to a Nash equilibrium based on the number of iterations, it is important to traverse the game trees as efficiently as possible. It is possible to perform cutoffs at some points during the tree traversal, to avoid updating the regret values if the probability of an opponent reaching a particular history becomes 0. In practice, this is a very significant savings: if the opponent never raises with a weak hand in the Preflop, then we can avoid traversing large parts of the tree.

A careful implementation can also yield a large decrease in the memory requirements of the program. By designing our data structure to use as little memory as possible per node in the information set tree, we can make more memory available to solve larger abstractions. One "trick" of particular use in poker was to eliminate the terminal nodes from the tree. In our original implementation, every terminal node in the tree stored the amount of money in the pot (the utility for the game's winner).

Due to the branching nature of trees, there are more terminal nodes than any other type of node. However, since Limit Heads-Up Texas Hold'em can only end in very few ways (between 1.5 and 24 small bets in the pot, for a limited number of showdowns or folds), we created only 138 unique terminal nodes and structured our tree to avoid duplication.

Parallel Implementation

To compute our CFR poker agents, we used a cluster of computers where each node has 4 CPUs and 8 gigabytes of RAM, connected by a fast network. However, any one node did not have enough main memory to store both the details of a 10-bucket abstraction and the two information set trees required to compute a strategy profile. Our original goal in designing a parallel version of the program was to store parts of the game tree on different computers to make use of distributed memory. While meeting that requirement, we found that the algorithm is easily modified to allow parallel traversals of the game trees when updating the counterfactual regret and action probabilities.

We start with the observation that Limit Hold'em has seven Preflop betting sequences that continue to the Flop (i.e. betting sequences without folds): check-call, check-bet-call, check-bet-raise-call, check-bet-raise-raise-call, bet-call, bet-raise-call, and bet-raise-raise-call. After one of these Preflop betting sequences, to perform the recursive regret and probability updates, we only require the portion of both players' game trees that started with that Preflop betting sequence.

We use a client/server layout for our parallel approach. The server holds the details of the card abstraction and only the Preflop portion of both players' information set trees. On each of seven additional computers, we store the portion of both players' game trees that start with one of the Preflop betting sequences. At the start of each iteration, the server generates N joint bucket sequences, performs the update functions on its part of the game tree, and then contacts each of the clients. Each client is given the bucket sequences and the probability that each player would play that Preflop betting sequence, given the Preflop bucket. With this information, for each bucket sequence, the clients can run their update functions in parallel and contact the server when they are finished.

In theory, this algorithm is almost embarrassingly parallelizable. The server does a very fast iteration over its small Preflop game tree, and then contacts the clients with a short message over a fast network. Each of the clients then does the time consuming part of the task in parallel, with no need for communication between peers, or even with the server until the task is complete.

In practice, we get a sublinear speedup: 3.5x when using 8 CPUs, instead of the 7x speedup that we might expect[2]. This is because not all of the Preflop betting sequences are equally likely. For example, the check-bet-raise-raise-call sequence indicates that both players have a strong hand, and are willing to repeatedly bet before seeing the Flop cards. The probability of both players following this betting sequence with a moderate hand may be zero, in which case the computation is pruned.

[2]The highest speedup we would expect is 7x instead of 8x, since the server is idle while the clients are working

This means that the computer reserved for handling this betting sequence is not used to capacity. The computers responsible for the check-call, check-bet-call, and bet-call betting sequences are the bottlenecks as these betting sequences are likely to occur with most of the possible deals.

Even with this load balancing issue, however, the parallel version of the program is of considerable use. It satisfies the main goal of increasing the available memory by giving us access to 64 gigabytes of RAM across 8 computers. Using this parallel version, we have solved games with as many as 12 buckets on each round to within 2 millibets/game of a Nash equilibrium.

3.5 Experimental Results

In this section, we will present results that show the effectiveness of this technique as applied to Texas Hold'em. We will start by examining how quickly the algorithm approaches a Nash equilibrium in a variety of abstractions. Then, we will use the strategies generated by this method to play Texas Hold'em against the two best poker agents that the CPRG has previously produced, and will show that increasing the size of our abstraction consistently leads to better play. Finally, we will consider the 2006 AAAI Computer Poker Competition, and show that if one of our new ϵ-Nash equilibria agents had been submitted to that competition, it would have won by a large margin.

3.5.1 Convergence to a Nash Equilibrium

Using the Counterfactual Regret Minimization technique, we found ϵ-Nash equilibria for abstractions with 5, 6, 8, and 10 buckets per round. In Figure 3.3 (a), we show the size of each abstraction in game states, the number of iterations of the algorithm we performed, the time taken to run the program, and the exploitability of the resulting strategy. We also note here that since we are using the poker specific implementation where bucket sequences are sampled, the time taken per iteration is affected by the size of the abstraction only insofar as technical details such as memory locality are concerned.

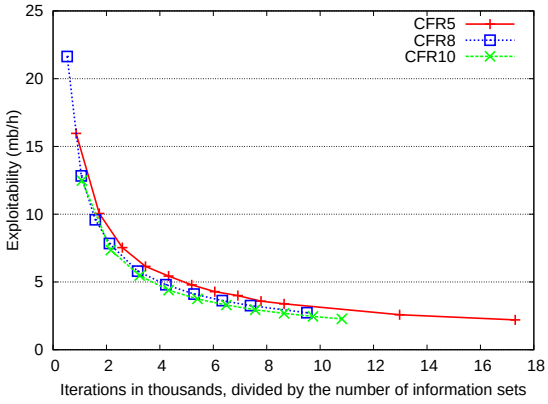
In Figure 3.3 (b), we show a graph of the convergence rates of strategies in the 5, 8, and 10 bucket abstractions. The y-axis shows the exploitability of the strategy, while the x-axis shows the number of iterations performed, normalized by the number of information sets in the abstraction. By dividing by the number of information sets, we show that the convergence rates are nearly identical, thus showing that in practice, the number of iterations needed to find an ϵ-Nash equilibrium grows linearly in the number of information sets. To find ϵ-Nash equilibrium strategies in a new abstraction, we can use this result to estimate how many iterations (and thus how much time) will be required.

The 10-bucket abstraction presented in this figure was one of the poker agents the CPRG submitted to the 2007 AAAI Computer Poker Competition. To produce this agent, the algorithm was run for 2 billion iterations over 14 days, using the parallel implementation and 4 CPUs. The resulting strategy was 2.2 mb/g exploitable in its own abstraction. The algorithm converges very quickly, though. After only 200 million iterations (10% of the total computation time, which required 2

Abs	Size (game states) ($\times 10^9$)	Iterations ($\times 10^6$)	Time (h)	Exp (mb/h)
5	6.45	100	33	3.4
6	27.7	200	75	3.1
8	276	750	261	2.7
10	1646	2000	326†	2.2

†: parallel implementation with 4 CPUs

(a)



(b)

Figure 3.3: (a) Number of game states, number of iterations, computation time, and exploitability (in its own abstract game) of the resulting strategy for different sized abstractions. (b) Convergence rates for three different sized abstractions. The x-axis shows the number of iterations divided by the number of information sets in the abstraction.

	PsOpti4	Smallbot2298	CFR5	CFR6	CFR8	CFR10	Average
PsOpti4	0	-28	-36	-40	-52	-55	-35
Smallbot2298	28	0	-17	-24	-30	-36	-13
CFR5	36	17	0	-5	-13	-20	2
CFR6	40	24	5	0	-9	-14	7
CFR8	52	30	13	9	0	-6	16
CFR10	55	36	20	14	6	0	22
Max	55	36	20	14	6	0	

Table 3.1: A crosstable showing the performance of several ϵ-Nash equilibrium strategies. Winnings are in millibets/game for the row player in full Texas Hold'em. Matches with PsOpti4 used 10 duplicate matches of 10,000 hands each and are significant to 20 mb/g. Other matches used 10 duplicate matches of 500,000 hands each are are significant to 2 mb/g.

days), it was already less than 13 mb/g exploitable. Recall that, due to our poker specific implementation, only 18,496 game states are visited per iteration. After 200 million iterations, each game state had been visited less than 2.5 times on average. The fact that the algorithm had already found a robust strategy with many observations of each information set but so few observations of each game state lends support to our claim that we can find appropriate actions in each information set by observing repeated samples of game states in that information set.

3.5.2 Comparison to existing programs

In this section, we will use the poker agents produced by the Counterfactual Regret Minimization technique to play games of Texas Hold'em against the strongest available poker programs. We will start by competing against the two strongest poker agents the CPRG has produced, and will evaluate

	Hyperborean	Bluffbot	Monash	Teddy	Average
Smallbot2298	61	113	695	474	336
CFR8	106	170	746	517	385

Table 3.2: Crosstable showing the performance of a Counterfactual Regret Minimization ϵ-Nash equilibrium agent and a recently published equilibrium strategy against the competitors of the 2006 AAAI Computer Poker Competition Online Learning event. Winnings are in mb/g for the row player in full Texas Hold'em.

our claim that an increase in the abstraction size generally leads to an increase in performance. Table 3.1 shows the results of this experiment. PsOpti4 and Smallbot2298 are the two strongest ϵ-Nash equilibria strategies that the CPRG has produced by older techniques. In these results, we show that a CFR strategy from our smallest (5 bucket) abstraction is able to defeat both of them. As we increase the size of the abstraction to 6, 8, and 10 buckets, we find that our performance against PsOpti4 and Smallbot2298 and smaller CFR strategies increases with each step.

Next, we will use one of our poker agents to recreate the 2006 AAAI Computer Poker Competition, and show that if it had been entered, it would have won. Table 3.2 shows a comparison between the 8-bucket CFR strategy from Figure 3.1, competing against the four competitors from the 2006 AAAI Computer Poker Competition's Online Learning event. In the paper that describes the technique that generated Smallbot2298, the analogous experiment was performed [34, p. 792]; for comparison, we have reproduced those results in this table. CFR8 defeats Smallbot2298 and all of the competitors from the competition, and also wins by a larger margin against each than Smallbot2298. From this result, we conclude that if CFR8 had been entered into the competition, with or without Smallbot2298 also being entered, CFR8 would have won by a substantial margin.

3.6 Conclusion

In this chapter, we presented a new technique for finding ϵ-Nash equilibria strategies in large abstract games, such as our abstractions of Texas Hold'em poker. We showed that, by solving larger equilibria than were previously possible, we have been able to produce new equilibrium strategies that have defeated all other known poker agents. In Chapter 7, we will show additional results of these agents being used to compete in the 2007 AAAI Computer Poker Competition and the First Man-Machine Poker Championship.

In poker, however, it is not sufficient merely to *not lose* to each of your opponents. As we mentioned in Section 1, one of the important features of poker is that exploitation is important. A strategy that never loses but does not win by as much as possible can lose a tournament to a player that occasionally loses, but wins by a large margin when it wins. The game RoShamBo (also known as Rock-Paper-Scissors) has been used to demonstrate this effect [6, p. 25]. In RoShamBo, the equilibrium strategy is trivially found and used by humans: simply randomly select each action $1/3$

of the time. However, the equilibrium strategy has an expected value of 0 against any opponent. In the International RoShamBo Programming Competition, several exploitable strategies are entered by the tournament organizer; exploitative strategies win against them, while the equilibrium cannot, and thus has no chance of winning the tournament.

In poker, we can learn this lesson from RoShamBo. The CFR ϵ-Nash equilibria strategies can win a competition which ranks players based on the number of matches they won, but may not win when the competition is ranked by the players' total winnings. We need to consider how much it is possible to defeat an opponent by, and measure how much of that exploitability our CFR strategies are achieving. Equilibrium strategies are useful tools, but to *win* the game instead of *not lose* it, we may need to explore other strategies that are capable of stepping away from the equilibrium point in order to try to exploit an opponent.

Chapter 4

Playing to Win: Frequentist Best Response

4.1 Introduction

A best response to a strategy σ_{opp} is the strategy that maximally exploits σ_{opp}. Knowing this strategy and the utility of using it against σ_{opp} confers several advantages. For example, if σ_{opp} is an ϵ-Nash equilibrium strategy, the utility tells us how far σ_{opp} is from a Nash equilibrium. If we are trying to use another strategy to defeat σ_{opp}, comparing that strategy's utility to the utility of the best response tells us how much of the possible exploitation we are achieving.

In Texas Hold'em, calculating a best response in the "real", unabstracted game is very computationally expensive, so we are forced to compromise by calculating a best response in the same abstract game that the strategy uses. The resulting counter-strategy may not be able to exploit σ_{opp} by as much as a best response in the real game can. For this reason, a best response in an abstract game can be called a "good response" to σ_{opp}, as opposed to the "best" response in the real game. However, we will use the more precise term **abstract game best response**, to indicate the strategy within a certain abstraction that is the best response to σ_{opp}. These abstract game best responses are still valuable, as they give us a lower bound on the exploitability of σ_{opp}; we know that the best response can achieve at least the same utility against σ_{opp} as the abstract game best response.

However, the abstract game best response algorithm has three drawbacks. The first is that it requires knowledge of the abstraction in which σ_{opp} plays. The second is that it requires the strategy σ_{opp} itself: to calculate a best response, we need to know how σ_{opp} will act in every information set. The third is that the resulting abstract game best response must be constructed in the same abstraction that σ_{opp} plays in.

In this chapter, we will present a technique called Frequentist Best Response (FBR) that addresses these three drawbacks of the abstract game best response algorithm. In Section 4.2, we will formally define the abstract game best response algorithm. In Section 4.3, we will present and formally define the FBR algorithm. In Section 4.5, we will show results of FBR's effectiveness against

the CPRG's benchmark programs, and compare it to the opponent modeling agent BRPlayer. Finally, in Section 4.6, we will describe the strengths and weaknesses of the resulting FBR strategies, and motivate our next technique, which allows us to quickly find ϵ-Nash equilibria in large abstract games.

4.2 Best Response

Computing a best response to a strategy is a well-known procedure using the tools of game theory. However, as mentioned above, computing a best response to a strategy in the unabstracted game of Texas Hold'em is not feasible, due to memory limitations and the computation time required. Therefore, as described in Section 2.6.3, we are forced to compute the best response to a σ_{opp} inside of the same abstract game that σ_{opp} uses. This abstract game best response is the strategy in the abstract game that maximally exploits σ_{opp}. Through experiments, we have found that these abstract game best responses are still "good" responses in the real game of poker when used against the strategies they are designed to defeat.

Formally, consider an abstract game A, and let Σ^A be the set of possible strategies in A. Given one such strategy $\sigma \in \Sigma^A$, we define the best response to σ_{opp} as a strategy $\sigma_{BR(opp)} \in \Sigma^A$ that has the maximum expected value of any strategy in Σ^A when played against σ_{opp}. We can compute the best response as described in Algorithm 1. Briefly, we do this by recursing over all of $\sigma_{BR(opp)}$'s information sets. For each game state in the information set, we find the probability of σ_{opp} reaching every terminal node descendant from that game state, and the associated utility of both players reaching that terminal node. We then find the expected utility for each action, and select the single action for $\sigma_{BR(opp)}$ that maximizes this utility.

Abstract game best response strategies have three drawbacks:

- **Requires the abstraction.** To compute a best response to σ_{opp}, we need to know everything about the abstraction that σ_{opp} plays in: the buckets, the transition probabilities, and the utilities.
- **Requires the strategy.** We need to know the action probability distribution of σ_{opp} for every information set.
- **Counter-strategies share the abstraction.** The best response strategy we generate plays in the same abstract game as σ_{opp}.

In the following section, we will describe Frequentist Best Response, a variant on the best response algorithm that avoids or lessens each of these drawbacks.

Algorithm 1 BESTRESPONSE(A, I, S)

Require: An abstraction of the game A
Require: A node I of the information set tree for game A
Require: A strategy σ_{opp}, being a mapping from information sets to action probabilities
Ensure: A strategy $BR(\sigma)$, being a mapping from information sets to action probabilities
Ensure: An expected utility $u(BR(\sigma), \sigma)$, being the expected utility of using strategy $BR(\sigma)$ against strategy σ_{opp}.

if I is a terminal node **then**
  Let u_I be the utility of reaching information set I.
  Let p be $\Pi^{\sigma}(g)$, the probability of σ_{opp} reaching game state $g \in I$
  Let u_g be the utility of reaching game state $g \in I$.
  $u_I = \frac{1}{\sum_{g \in I} p} \left(\sum_{g \in I} p u_g \right)$
  Return u_I
else if I is a choice node for $BR(\sigma)$ **then**
  Let $I(a)$ be the information set resulting from taking action a in information set I
  Find the action a that maximizes $u(a) =$ BESTRESPONSE$(A, I(a), \sigma)$
  Set $BR(\sigma)(I)$ to select action a
  Return u_a
else if I is a choice node for σ_{opp} **then**
  for Each action a available in I **do**
    Let $I(a)$ be the information set resulting from taking action a in information set I
    BESTRESPONSE$(A, I(a), \sigma)$
  end for
end if

4.3 Frequentist Best Response

There are several situations where the abstract game best response algorithm is not suitable. Two examples are:

- **Calculating the abstract game best response to a competitor's black box strategy**. By a "black box strategy", we refer to a program which you can play poker against, but not examine to directly find its action probability distribution at each information set. If you are playing in a competition such as the annual AAAI Computer Poker Competition, you may be curious to know how exploitable an opponent's previous year's entry was. However, since you cannot examine the strategy to find the action probabilities, the abstract game best response algorithm cannot be used.

- **Calculating exploitability in a more informed game**. If you have created a strategy σ_{opp} in some abstraction A, you can use the abstract game best response algorithm to find its exploitability. However, if you create a new abstraction A', you may wonder how exploitable σ_{opp} is by strategies that play in A'. Since the abstract game best response algorithm can only create strategies that play in the same abstraction, it is not suitable for this task. This is particularly useful if σ_{opp} is an ϵ-Nash equilibrium strategy, since σ_{opp} is minimally exploitable in its own abstraction, but may still be vulnerable in other abstractions.

Our approach for avoiding the drawbacks of best response is called Frequentist Best Response (FBR), and it was first described in [14]. It is an offline approach that requires the ability to play against an opponent strategy, without requiring direct access to the action probabilities that the strategy contains. In the remainder of this section, we will describe the approach, and highlight four parameters that affect the quality of the counter-strategies that it produces. In Section 4.4, we will present experimental results of the effects of these parameters, and justify the settings we use in FBR.

We will now begin our description of the FBR algorithm. For an opponent strategy σ_{opp}, we will observe many full information examples of σ_{opp} playing the unabstracted game of poker. Using this information, we will use frequency counts of its actions to form an opponent model in our choice of abstract game. Then, we will use the best response algorithm in Algorithm 1 to calculate a good response counter-strategy to the opponent model. We will then evaluate the technique by using the counter-strategy to play against σ_{opp} in the real game. We will now describe each step of this process in more detail, and point out the choices that affect FBR's performance.

4.3.1 Obtaining the training data

The first step of the process is obtaining a large set of example games of σ_{opp} playing the unabstracted game of poker with full information. By "full information", we mean that we need to see all private and public cards for every game, even in cases where one player folds. This technique will require a large number of example games, and more example games are needed as we increase the size of the abstraction we wish to use. For our smallest abstraction (5 buckets per round), we require between 100,000 and 1 million training games. For larger abstractions (such as 8 buckets per round), more games are required. For the technique to work well, we need to observe σ_{opp} playing in a wide variety of situations. This means that the type of opponent that σ_{opp} is playing against in the training data is also important. The "training opponent" for σ_{opp} should evenly explore the different lines of play that σ_{opp} can reach.

Parameter 1: Collecting enough training data

As we use more training data to create our opponent model, the accuracy of the model and the performance of the FBR counter-strategy against σ_{opp} improves. There are diminishing returns, however, since there is a bound on how exploitable σ_{opp} is by any strategy in the abstraction. Our first decision to make, then, is to choose how much training data to generate and use.

Parameter 2: Choosing an opponent for σ_{opp}

Earlier, we mentioned that the choice of opponent that we observe σ_{opp} playing against for these training games is significant. Let us call this opponent σ_{train}. For FBR to work well, we need to observe σ_{opp} acting several times in every reachable information set of the abstraction we will

choose later. If we accomplish this, then our opponent model will be accurate and the counter-strategy will perform well; if there are many situations in which we have never observed σ_{opp} acting, then our counter-strategy will perform poorly.

One obvious choice for σ_{train} is σ_{opp} itself. We could observe games of PsOpti4 playing against itself, for example. At first, this seems like a good option, since we would receive more observations (and our opponent model would be more accurate) in the situations that the opponent is likely to play in. However, there are two reasons why self-play data may not be useful as training data for FBR. First, σ_{train} should never fold. If σ_{train} folds, then the game is over and we do not receive any additional observations from σ_{opp}. If we use self-play games, then σ_{train} will fold, and we will not receive as much information as we can. Second, if we only observe situations where σ_{opp} is likely to reach, then we obtain no information about states that it will play into rarely. PsOpti4 is a good example of this. PsOpti4 uses a betting abstraction, in that it only considers at most three bets per round, instead of four. In self-play, PsOpti4 will never reach a four-bet situation. Against other opponents, PsOpti4 will occasionally raise to three bets, and the opponent can raise to four. PsOpti4 ignores this fourth bet, and it will not count its value in the pot. This means that it is not correctly calculating the expected value of future betting decisions. Thus, playing against other opponents reveals *weaknesses* in σ_{opp} that no amount of self-play will uncover.

Instead of using self-play, σ_{train} should be chosen so as to force σ_{opp} to play in as many different information sets as possible. The only way to affect the opponent's information set is through the betting history of the game, so it should try to choose betting actions to evenly cover the possible betting histories. As σ_{train} should never fold, we need to choose a distribution over calling and raising.

4.3.2 Creating the opponent model

The FBR opponent model is very simple. Once we have collected the data, we identify every situation where σ_{opp} chose an action — in other words, every information set faced in every game. For each information set, we will map the card sequence into a bucket sequence, according to the abstract game we intend to use. The training examples we observe are played in the real game, and σ_{opp} has used its own unknown abstraction to choose its actions, but we will instead try to replace this with our own abstraction.

For each information set in our abstract game, we will maintain a set of counters that track the number of times we have observed σ_{opp} fold, call and raise from that information set. We will iterate over the abstracted choice nodes from the training data, and increment these counters according to the actions that σ_{opp} took. Then, our model of the opponent will set the probability of taking action a from information set I to be the number of observations of a from I, divided by the number of observations of I. Thus, the model amounts to frequency counts in our chosen abstraction.

There are still two decisions to be made. What action should the model return in information

sets where there were no observations, and what abstraction should the model and best response be calculated in?

Parameter 3: Choosing the default policy

If we had an unlimited amount of training data, then we would have observations of σ_{opp} acting in every information set. Since we do not have unlimited training data, we need a policy for our model to follow in unobserved situations. We experimented with several simple choices for this default action, by assuming that σ_{opp} will always call, raise, or mix between these options.

Parameter 4: Choosing the Abstraction

FBR can calculate a counter-strategy in any abstraction. If we use a larger (or better) abstraction, we expect the performance of the FBR counter-strategies to improve, since the larger abstraction gives a more precise representation of the hand strength of the counter-strategy's cards. However, we also expect a larger abstraction to require more training data, as there are more information sets that require observations.

4.3.3 Finding a best response to the model

Once we have an opponent model of σ_{opp} defined in our choice of abstraction, we find an abstract game best response to the model, using the best response algorithm described in Algorithm 1. The result is a strategy, $\sigma_{\text{FBR(opp)}}$, that plays in our chosen abstract game. $\sigma_{\text{FBR(opp)}}$ can now play against σ_{opp} in the unabstracted game. Depending on the exploitability of σ_{opp} and the quality of the abstraction chosen for $\sigma_{\text{FBR(opp)}}$, $\sigma_{\text{FBR(opp)}}$ will hopefully be able to defeat σ_{opp} in the unabstracted game.

4.4 Choosing the Parameters

In this section, we will experiment with several settings for each of the four parameters identified in the previous section. Graphs will be provided to show how the choice of each parameter affects the resulting FBR counter-strategies, and we will state and justify our choices for each parameter.

4.4.1 Parameter 1: Collecting Enough Training data

Figure 4.1 shows the effect that the amount of training data observed has on the resulting FBR counter-strategies. In this graph, we have created a series of FBR counter-strategies against the PsOpti4, Smallbot2298, and Attack80, using different amounts of training data. Against all three opponents, increasing the amount of training data results in stronger counter-strategies. Depending on how exploitable σ_{opp} is, we may require considerably more or less training data. Attack80 (a very exploitable strategy) can be defeated by an FBR counter-strategy trained on 10,000 games, although

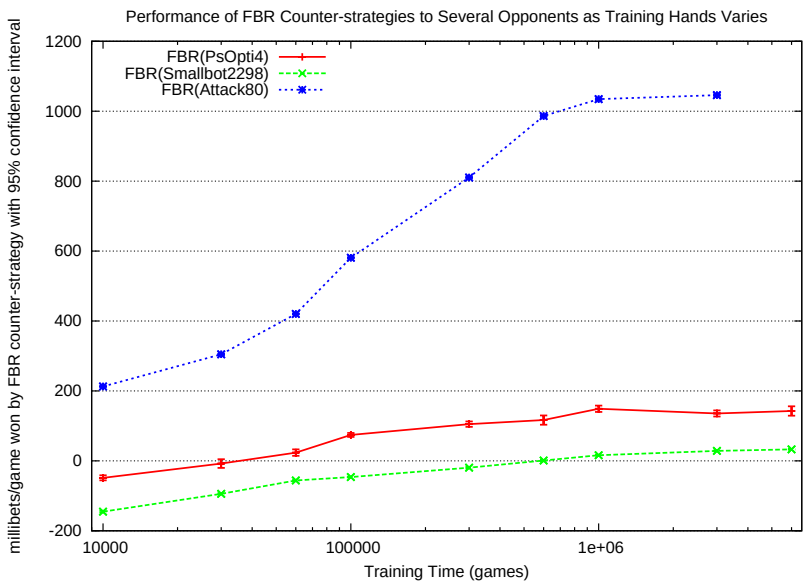


Figure 4.1: Performance of FBR counter-strategies to PsOpti4, Smallbot2298, and Attack80, using different amounts of training data. The x-axis is the number of training games observed, the y-axis is the utility in millibets/game of the FBR strategy, and the error bars indicate the 95% confidence interval of the result.

it can be defeated by a much larger margin when more data is used. Against Smallbot2298 (an ϵ-Nash equilibrium strategy), the FBR counter-strategy requires 600,000 games just to break even.

For our five-bucket abstraction, we typically generate and use one million hands of training data. If we use more training data for this abstraction, the counter-strategies do not improve by an appreciable amount.

4.4.2 Parameter 2: Choosing An Opponent For σ_{opp}

In Figure 4.2, we present the results of several different choices for σ_{train} used to create FBR counter-strategies to PsOpti4. "Probe" is a simple agent that never folds, and calls and raises equally often. "0,3,1" is similar to Probe, except that it calls 75% and raises 25% of the time. "0,1,3" is the opposite, raising 75% and calling 25%. "PsOpti4" indicates an FBR strategy created with self-play data. From this experiment, we find that even a large amount of self-play data is not useful for creating an FBR counter-strategy to PsOpti4. Instead, if we use strategies like Probe for σ_{train}, FBR is able to create effective counter-strategies. Unless otherwise noted, the results presented in this chapter will use Probe as σ_{train}, and we will refer to the Probe strategy as σ_{probe}.

4.4.3 Parameter 3: Choosing the Default Policy

Figure 4.3 shows the effect that the default policy used in unobserved states has on the resulting strategy. "0,1,0" is the normal policy that is used in FBR; in unobserved states, it always calls.

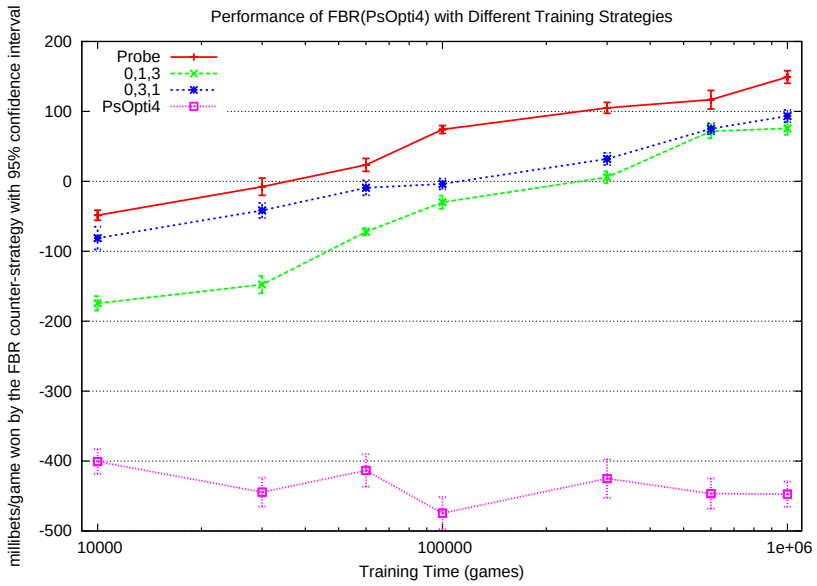


Figure 4.2: Performance of an FBR counter-strategy to PsOpti4, using different training opponents. The x-axis is the number of training games observed, the y-axis is the utility in sb/g of the FBR strategy, and the error bars indicate the 95% confidence interval of the result.

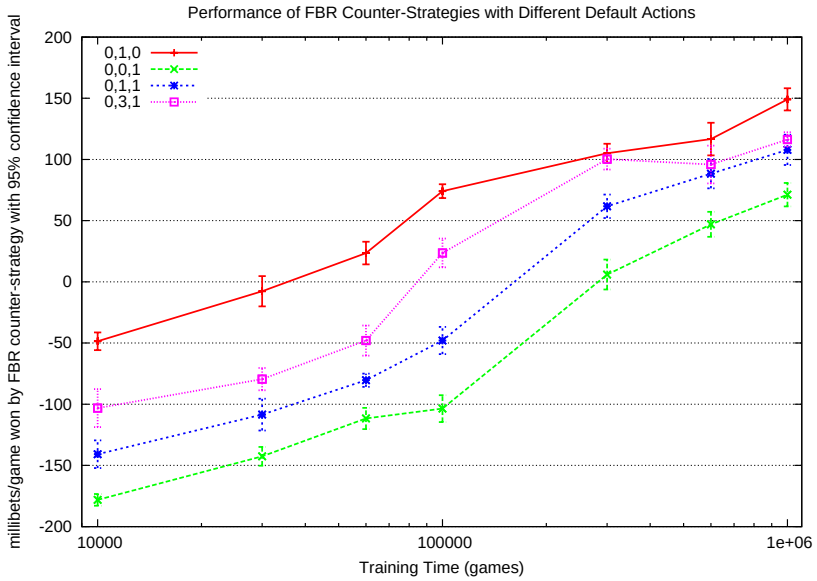


Figure 4.3: Performance of FBR counter-strategies to PsOpti4, using different default actions for unobserved states. The x-axis is the number of training games observed, the y-axis is the utility in sb/g of the FBR strategy, and the error bars indicate the 95% confidence interval of the result.

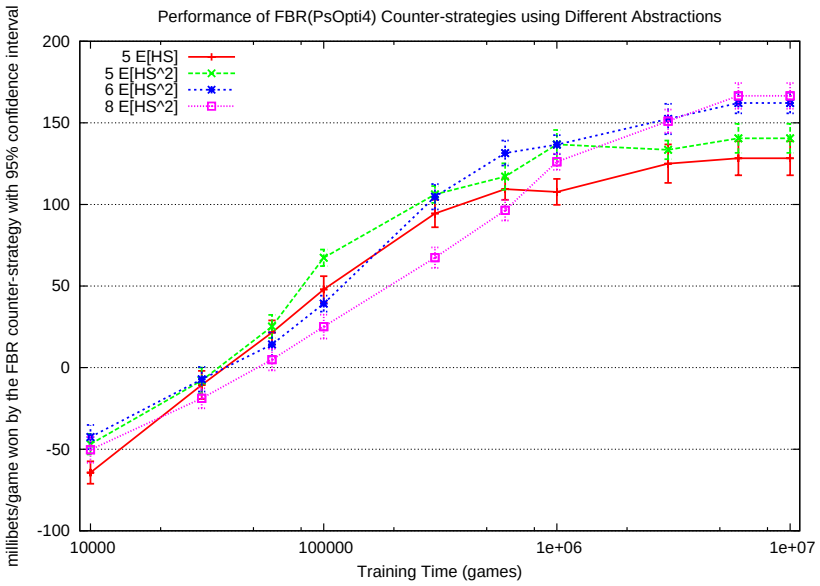


Figure 4.4: Performance of FBR counter-strategies to PsOpti4, using different abstractions. The x-axis is the number of training games observed, the y-axis is the utility in sb/g of the FBR strategy, and the error bars indicate the 95% confidence interval of the result.

"0,0,1" always raises, "0,1,1" calls and raises with equal probability, and "0,3,1" calls 75% and raises 25% of the time.

From the graph, we see that the "always call" default policy consistently performs the best among these options. As we increase the amount of training data, the default policies are used less often, and the difference between the best and worst of the policies diminishes. Unless otherwise noted, the results presented in this chapter will use the "always call" default policy.

4.4.4 Parameter 4: Choosing the Abstraction

Figure 4.4 shows the effect that the choice of abstraction has on the performance of the FBR counter-strategy. In this figure, we have computed several FBR strategies for the 5 bucket $E[HS]$ abstraction and each of the 5, 6, and 8 bucket $E[HS^2]$ abstractions.

First, we notice that the 5 bucket $E[HS^2]$ counter-strategy outperforms the 5 bucket $E[HS]$ counter-strategy at all points on the curve. Both of these abstractions use percentile buckets and history, as described in Section 2.5.5. In that section, we explained that $E[HS^2]$ bucketing represented potential better than $E[HS]$ bucketing, and this graph shows how this representation of potential can be used to produce stronger strategies.

Second, we notice that using the larger 6 and 8 bucket abstractions produces counter-strategies that are better able to exploit the opponent. With 10 million games of training data, all four types of counter-strategies have stopped improving, and are in the order we predicted: the performance of the counter-strategy increases as the abstraction grows.

	PsOpti4	PsOpti6	Attack60	Attack80	Smallbot1239	Smallbot1399	Smallbot2298	Average
FBR	137	330	2170	1048	106	118	33	563
CFR5	36	123	93	41	70	68	17	64

Table 4.1: Results of FBR counter-strategies and an ϵ-Nash equilibrium strategy against a variety of opponent programs in full Texas Hold'em, with winnings in millibets/game for the row player. Results involving PsOpti4 or PsOpti6 used 10 duplicate matches of 10,000 games and are significant to 20 mb/g. Other results used 10 duplicate matches of 500,000 games and are significant to 2 mb/g.

Unless otherwise noted, the results presented in this chapter will use the 5 bucket $E[HS^2]$ abstraction. This choice is made for practical reasons, as the 5 bucket FBR counter-strategies can be produced faster and require a smaller amount of memory to store on disk or use in a match than the 6 or 8 bucket FBR counter-strategies. When we evaluate a new poker program, we typically use the largest abstraction available.

4.5 Results

4.5.1 Comparisons against benchmark programs

To evaluate the FBR algorithm, we considered several of the benchmark programs described in Section 2.3.3 and created an FBR strategy against each. To do this, we collected one million games of training data of each strategy playing against σ_{probe}, used the 'always call' default action, and chose a 5-bucket $E[HS^2]$ game abstraction with the features described in Section 2.5.5. In addition to the benchmark programs, we also evaluated our agents against CFR5, an ϵ-Nash equilibrium agent in the 5 bucket $E[HS^2]$ abstraction, created by the Counterfactual Regret Minimization technique of Chapter 3.

In Table 4.1, we present the results of this experiment. The row labeled FBR indicates the score in average millibets/game for the FBR counter-strategy against the opponent in each column. For comparison, the row labeled CFR5 is an ϵ-Nash equilibrium program in the same abstraction as the FBR counter-strategies. This table shows us that the CFR5 strategy is not exploiting its opponents by anywhere near the maximum that can be attained. Attack60 and Attack80, for example, can be defeated at 2170 and 1048 millibets/game respectively, while CFR5 achieves only 93 and 41 millibets/game. This result shows the value of the FBR counter-strategies: when trained against a particular opponent, the resulting counter-strategy gives us a lower bound on the opponent's exploitability, and is also a powerful strategy to use during a match. This result also shows the necessity of opponent modeling. A hypothetical strategy that can take advantage of the weak opponents could lose to CFR5 and the strong opponents, and win a tournament based on its overall winnings.

However, an FBR counter-strategy is not this hypothetical tournament-winning strategy. in Table 4.2, we present a full cross table of matches between the FBR counter-strategies and all of the opponent programs. This cross table is colour coded to make the results clearer: cells that are green indicate a win for the row player, and red indicates a loss for the row player. When we consider

	PsOpti4	PsOpti6	Attack60	Attack80	Smallbot1239	Smallbot1399	Smallbot2298	CFR5	Average
FBR-PsOpti4	137	-163	-227	-231	-106	-85	-144	-210	-129
FBR-PsOpti6	-79	330	-68	-89	-36	-23	-48	-97	-14
FBR-Attack60	-442	-499	2170	-701	-359	-305	-377	-620	-142
FBR-Attack80	-312	-281	-557	1048	-251	-231	-266	-331	-148
FBR-Smallbot1239	-20	105	-89	-42	106	91	-32	-87	3
FBR-Smallbot1399	-43	38	-48	-77	75	118	-46	-109	-11
FBR-Smallbot2298	-39	51	-50	-26	42	50	33	-41	2
CFR5	36	123	93	41	70	68	17	0	56
Max	137	330	2170	1048	106	118	33	0	

Table 4.2: Results of frequentist best responses (FBR) against their intended opponents and against other opponent programs, with winnings in millibets/game for the row player. Results involving PsOpti4 or PsOpti6 used 10 duplicate matches of 10,000 hands and are significant to 20 mb/g. Other results used 10 duplicate matches of 500,000 hands and are significant to 2 mb/g.

the cross table, we notice that the diagonal from top left to bottom right is where the FBR counter-strategies are playing against the opponents they were designed to defeat. These scores are the same as those in Table 4.1, and they are all positive.

The rest of the table is not nearly as promising; most of the other entries indicate that the FBR counter-strategy lost, and sometimes by a large margin. This indicates that the FBR strategies are **brittle** — they perform well against their intended opponent, but are not good strategies to use in general. Against other opponents, even weak ones, they perform poorly. We also note that some of the strategies in the table (PsOpti6 and PsOpti7, Attack60 and Attack80, Smallbot1239 and Smallbot1399) are very similar, and yet an FBR strategy that defeats one can still lose to the other, or not exploit them for nearly as much as is possible.

4.5.2 Comparisons against BRPlayer

In Section 2.6.4 we discussed BRPlayer, another program that can exploit its opponents. BRPlayer and FBR are difficult to compare, because they are used in very different ways. BRPlayer learns online with only the information a player receives during a match, while FBR counter-strategies are calculated offline with the benefit of hundreds of thousands of full information games.

One task in which BRPlayer and FBR can be compared is in finding a lower bound on the exploitability of an opponent. Before the FBR technique was developed, the highest known exploitability for PsOpti4 was found using BRPlayer. In Figure 5.7 of Schauenberg's MSc thesis [27, p. 70], if we consider the first 400,000 games to be training for BRPlayer and consider only the last 100,000 games, BRPlayer defeats PsOpti4 by 110 millibets/game. In Table 4.4, we calculated an 8-bucket $E[HS^2]$ counter-strategy that was able to defeat PsOpti4 by 167 millibets/game. For this purpose — evaluating new agents that we produce — FBR has taken the place of BRPlayer.

4.6 Conclusion

The Frequentist Best Response technique is an important tool to have available. It allows us to evaluate our own strategies and those of our opponents, and has restrictions that are not nearly as

onerous as those imposed by the best response algorithm. FBR is capable of producing strategies that defeat their opponents, but it is unlikely that such strategies would ever be used by themselves to play poker. Since the strategies are brittle, we need to be sure that our opponent is the one we trained the FBR strategy to defeat. Imagine, for example, that we are doing opponent modeling during a match to find out what strategy to use, and then using an FBR counter-strategy to the suspected opponent. If our opponent modeling is incorrect, either because of modeling error or the opponent changes their strategy, then we can pay a heavy price.

Ideally, we would like a method that allows us to construct *robust* responses: a strategy that can exploit a particular opponent or class of opponents, while minimizing its own exploitability. This confers two advantages. If we are using the strategy on its own to play against an opponent, then we know that if it was trained against this opponent it will win; if it was not trained against this opponent, it will not lose by very much. If we are using opponent modeling to identify our opponent and then select an appropriate robust response, we can be confident that we will not pay a large price if our model is incorrect. An approach for creating these robust responses, called Restricted Nash Response, will be presented in the next chapter.

Chapter 5

Playing to Win, Carefully: Restricted Nash Response

5.1 Introduction

In Chapter 3, we presented Counterfactual Regret Minimization, a method for creating poker agents that do not try to exploit opponents, and instead try to minimize their own exploitability. We showed that these agents are robust: against arbitrary opponents, they will be difficult to defeat. In Chapter 4, we presented Frequentist Best Response, a method for creating counter-strategies that exploit opponents without regard for their own exploitability. We showed that these counter-strategies are brittle: while they perform well against the specific opponents they are designed to defeat, they can lose badly to others, even if those opponents are weak or similar to those trained against. We would like to find a method that is a compromise between these two extremes, so that we can create agents that exploit a particular opponent or class of opponents, while providing a bound on how much they can be defeated by.

We will present such a method in this chapter. In Section 5.2, we will provide a high-level description of the technique, called Restricted Nash Response (RNR). In Section 5.3, we will provide a more formal description. In Section 5.4, we will present results of RNR agents playing against our standard benchmark programs. Finally, we will conclude the chapter by motivating our intended use for these new agents as members of a team of strategies.

5.2 Overview

We will begin with an example that motivates the use of robust counter-strategies. Suppose you have played against an exploitable opponent some time in the past, and will play against a similar opponent — perhaps a new, improved version — in the future. If we would like to beat them by as much as possible, then we could calculate a best response using the Frequentist Best Response approach of Chapter 4. However, these FBR counter-strategies are brittle. If the new opponent has been even slightly improved, the FBR counter-strategy could lose by a large margin. In particular,

the new opponent might display many of the same tendencies as the old one, but sometimes acts so as to punish an agent trying to exploit it. In other words, it might *usually* play like the old agent, but *sometimes* use a counter-counter-strategy to defeat us. Of all the ways the agent could be slightly changed, this would be the worst case for us.

The Restricted Nash Response algorithm produces a strategy that is designed to exploit the old opponent, but it will do so in a robust way. This means that if the new opponent is actually a best response to the RNR strategy, then the RNR strategy will not be very exploitable.

We will construct these robust counter-strategies by finding an ϵ-Nash equilibrium in a restricted game, where the opponent must play according to a fixed strategy with probability p, for our choice of p. This fixed strategy is analogous to the old, known agent from the example. With probability $1 - p$, the opponent is free to play to maximize its utility in the usual game theoretic way. At the same time, our player is unrestricted and so will want to play a game theoretic strategy that can both exploit the fixed strategy that the opponent is forced to play, and defend itself from the game theoretic strategy that the opponent can sometimes adopt.

Since p determines the proportion of the time that player 2 must use the fixed strategy, we can vary $p \in [0, 1]$ to produce a range of strategies for player 1. When $p = 0$, the algorithm produces an ϵ-Nash equilibrium, as the fixed strategy isn't used and the two game theoretic players are only trying to exploit each other. When $p = 1$, the algorithm produces a best response, since the opponent's game theoretic strategy isn't used and all our player has to do is exploit the fixed strategy. At all points in between, our player is choosing a tradeoff of how much it wants to exploit the fixed strategy against how much it wants to defend itself from the game theoretic strategy. Another way of thinking about this tradeoff from our player's perspective is to consider *how confident* we are that the opponent will act like the fixed strategy. If we are very confident, we can exploit it; if we are not very confident, we can concentrate on defending ourselves.

To get the opponent's strategy for use in the RNR algorithm, we first create a model of their play using the Frequentist Best Response technique. This model must be in the same abstraction as the one we wish to create the RNR counter-strategy in.

The Restricted Nash Response approach can be used with any technique for finding an ϵ-Nash equilibrium, including the traditional approach of solving a linear program. However, since the Counterfactual Regret Minimization approach is our most efficient and effective method for finding ϵ-Nash equilibria, we will use that approach to produce RNR agents. In fact, the same program is used to produce both CFR and RNR strategies, as only a minor modification is required. Recall that in the CFR approach, we created two agents who then used self-play to learn to defeat each other, and over millions of games they approached a Nash equilibrium. In the RNR approach, we construct three agents — one for player 1, and a learning and static component for player 2. During millions of games of self-play, player 1 tries to minimize its regret against both the learning and static components of player 2, using p to determine how much weight to put on each part of the

regret.

When we compute RNR agents, we need to choose a value of p, which determines how much of the static strategy we force player 2 to use. Since it determines the tradeoff between our exploitation of the opponent and our own exploitability, one way to set p is to decide how much of our exploitability we are willing to risk. If we are willing to be 100 millibets/game exploitable, for example, then we can experimentally find a value of p that will produce a strategy that is at most 100 millibets/game exploitable. In Section 5.4, we will show how the value of p determines the tradeoff between exploitation and exploitability, as we described above.

In terms of computational complexity, RNR has requirements slightly higher than CFR, but only by a constant factor. When computing an RNR strategy, we are only forcing one player (the dealer or the opponent) to play partially by the static strategy. To form an RNR agent for both seats, then, we need to run the computation twice — once when the dealer plays partially by the fixed strategy, and once with the opponent. These strategies can be computed independently, so the program either takes twice as long, or requires twice as many computers. Also, since the RNR variant of CFR must make six tree traversals instead of four for each training game played, it takes approximately 1.5 times longer to compute one half of an RNR strategy than a CFR strategy. Assuming the strategies are computed in series, an RNR strategy then requires about 3 times as long to compute as a CFR strategy. However, given the low memory requirements of CFR, we typically compute these strategies in parallel. The optimizations described in Section 3.4.3, including the parallel variant of CFR, are also applicable to RNR. Using a 2.4 GHz AMD Opteron, reasonable RNR strategies using the 5-bucket abstraction that are capable of defeating all of our benchmark programs can be produced in under a day. We estimate that competitive strategies that use larger abstractions with 10 or 12 buckets can be computed in less than a month.

In the next section, we will formally define the foundation of the Restricted Nash Response technique.

5.3 Formal Definition

We will start our formal definition of the Restricted Nash Response technique by defining a new class of best response strategies, called ϵ-safe best responses. McCracken and Bowling [20] defined ϵ-safe strategies as the set of strategies $\Sigma^{\epsilon\text{-safe}}$ that have an expected utility no less than ϵ less than the Nash equilibrium value. As we are primarily interested in symmetric games where the value of the Nash equilibrium is 0, these ϵ-safe strategies have an exploitability of no more than ϵ.

We will define the set of ϵ-safe best responses in a similar way. An ϵ-safe best response to a strategy σ_2 is a strategy σ_1 with at most ϵ exploitability that has the best utility against σ_2. In other words, the set of best responses to σ_2 that are exploitable by at most ϵ is:

$$BR^{\epsilon\text{-safe}}(\sigma_2) = \underset{\sigma_1 \in \Sigma^{\epsilon\text{-safe}}}{\operatorname{argmax}} u_1(\sigma_1, \sigma_2) \tag{5.1}$$

Next, we will define the set of strategies that our opponent can choose from. Let us call our model of the opponent's strategy σ_{fix}. We then define $\Sigma_2^{p,\sigma_{fix}}$ to be the set of opponent strategies that are restricted to playing according to σ_{fix} with p probability. Next, we define the set of **restricted best responses** to our strategy $\sigma_1 \in \Sigma_1$ to be:

$$BR^{p,\sigma_{fix}}(\sigma_1) = \underset{\sigma_2 \in \Sigma_2^{p,\sigma_{fix}}}{\operatorname{argmax}} u_2(\sigma_1, \sigma_2) \tag{5.2}$$

Thus, a restricted best response to σ_1 is the best strategy the *restricted* opponent can use, in conjunction with the fixed strategy, against our *unrestricted* player.

We now define the pair of strategies that the algorithm generates. A $(\boldsymbol{p}, \boldsymbol{\sigma_{fix}})$ **restricted Nash equilibrium** is a pair of strategies (σ_1^*, σ_2^*) where $\sigma_2^* \in BR^{p,\sigma_{fix}}(\sigma_1^*)$ and $\sigma_1^* \in BR(\sigma_2^*)$. We call σ_1^* a ***p*-restricted Nash response** to σ_{fix}: it is a robust strategy that is capable of exploiting σ_{fix} that also limits its own exploitability.

In [14, Theorem 1], we prove[1]:

Theorem 6 *For all $\sigma_2 \in \Sigma_2$, for all $p \in (0, 1]$, if σ_1 is a p-RNR to σ_2, then there exists an ϵ such that σ_1 is an ϵ-safe best response to σ_2.*

In other words, by selecting p to generate strategies with exploitability ϵ, the Restricted Nash Response strategies generated are the best responses to σ_2 of any strategies with that exploitability. If we have a desired level of exploitability that we are willing to tolerate in our counter-strategies, then we can experimentally vary p to generate best response strategies with at most that level of exploitability.

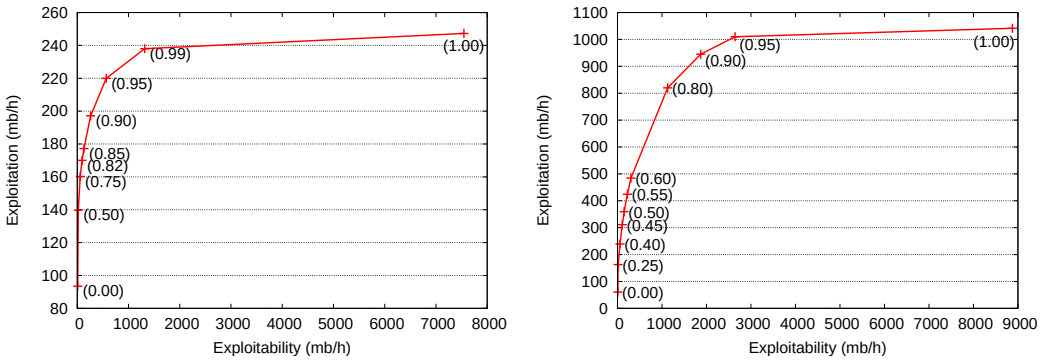
5.4 Results

In this section, we show the effectiveness of the Restricted Nash Response technique. We will begin by exploring settings of p to determine the tradeoff between exploitation and exploitability that our strategies will use. Then, we will generate an RNR counter-strategy against each of a variety of our benchmark agents, and form a crosstable similar to the Frequentist Best Response crosstable shown in Table 4.2.

5.4.1 Choosing p

We begin by demonstrating the tradeoff between exploitability and exploitation. Figure 5.1 shows this tradeoff versus models of two of our benchmark opponents, PsOpti4 and Attack80. On each

[1] The proof of this theorem was found by Zinkevich and Bowling



(a) Versus Opti4

(b) Versus A80

Figure 5.1: The tradeoff between ϵ and utility. For each opponent, we varied $p \in [0, 1]$ for the RNR. The labels at each datapoint indicate the value of p used.

graph, the x-axis is *exploitability* given up by the strategy, while the y-axis is the strategy's *exploitation* of the model of the opponent. The labeled data points are the strategies that were generated; the label indicates the value of p that generated this strategy. Note that while this graph shows the exploitation of the model and not the opponent itself, the strategies remain very effective against the actual agents, as will be shown in Table 5.1.

Note that the curves are highly concave. This means that with very low values of p, we can create strategies that give up a very small amount of exploitability in return for large gains against their intended opponent. Equivalently, by giving up only a small amount of exploitation of the opponent, the strategies obtain a *drastic* reduction in their own exploitability. For example, consider the difference between $p = 1.00$ (a best response) and $p = 0.99$ for PsOpti4 or $p = 0.95$ for Attack80.

A simple alternate method for creating robust responses to an opponent would be to play a mixture between an ϵ-Nash equilibrium strategy and a best response. On each hand, you could choose to play the best response counter-strategy with probability p, and the equilibrium strategy with probability $1 - p$. We plotted this strategy for $p \in [0, 1]$ on Figure 5.2, the same graph that was presented in Figure 5.1. The resulting set of mixture strategies would be a straight line between the $p = 0$ and $p = 1$ datapoints. For any level of exploitability between these endpoints, the RNR strategies indicated by the "RNR" curve perform better than the mixture strategies indicated by the "Mix" line. Because the RNR counter-strategies are ϵ-safe best responses (as presented in Theorem 6), they are the best counter-strategies for some value of ϵ; thus, they will always perform better or equal to a mixture with the same ϵ.

5.4.2 Comparison to benchmark programs

In Table 4.2, we showed a crosstable of the results of Frequentist Best Response agents competing against a collection of the CPRG's benchmark agents. In Table 5.1, we present the same experiment,

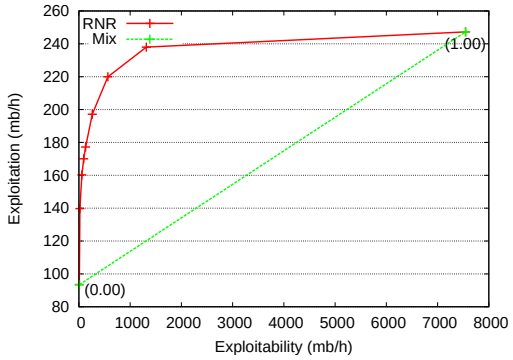


Figure 5.2: A graph showing the tradeoff between exploitiveness and exploitability for Restricted Nash Response agents and a mixture between a best response and an ϵ-Nash equilibrium. By randomly choosing either a best response or a Nash equilibrium, a mixture strategy can be created anywhere along the diagonal line titled "Mix". For any level of exploitability, the RNR counter-strategies with that exploitability exploit the opponent by a larger amount than a mixture with the same exploitability.

	Opponents								
	PsOpti4	PsOpti6	Attack60	Attack80	Smallbot1239	Smallbot1399	Smallbot2298	CFR5	Average
RNR-PsOpti4	85	112	39	9	63	61	-1	-23	43
RNR-PsOpti6	26	234	72	34	59	59	1	-28	57
RNR-Attack60	-17	63	582	-22	37	39	-9	-45	78
RNR-Attack80	-7	66	22	293	11	12	0	-29	46
RNR-Smallbot1239	38	130	68	31	111	106	9	-20	59
RNR-Smallbot1399	31	136	66	29	105	112	6	-24	58
RNR-Smallbot2298	21	137	72	30	77	76	31	-11	54
CFR5	36	123	93	41	70	68	17	0	56
Max	85	234	582	293	111	112	31	0	

Table 5.1: Results of restriced Nash response (RNR) against a variety of opponent programs in full Texas Hold'em, with winnings in mb/h for the row player. Results involving PsOpti4 or PsOpti6 used 10 duplicate matches of 10,000 hands and are significant to 20 mb/h. Other results used 10 duplicate matches of 500,000 hands and are significant to 2 mb/h.

performed using Restricted Nash Response agents. Each of the RNR agents plays in a 5-bucket abstraction, and has had p chosen to produce a strategy that is approximately (and no more than) 100 millibets exploitable in its own abstraction. The agent labeled CFR5 is an ϵ-Nash equilibrium produced by the Counterfactual Regret Minimization technique described in Chapter 3, and it plays in the same abstraction as the RNR agents evaluated here.

From Table 5.1, we note that, similar to the FBR results shown in Table 4.2, the diagonal values are high — this is where the RNR counter-strategy was used against its intended opponent. These values are not as high as the FBR values, since the RNR agents have given up some exploitation in return for a decrease in exploitability. For example, FBR-PsOpti4 defeated PsOpti4 by 137 millibets/game, whereas RNR-PsOpti4 defeated PsOpti4 by 85 millibets/game. For very exploitable opponents, such as Attack60 and Attack80, the exploitation by the RNR strategies is considerably reduced.

The portion of the table not on the diagonal, however, is very different from the FBR table.

We have described the FBR counter-strategies as brittle: while they are very effective against their intended opponents, they can lose by a large margin against even similar strategies. The RNR counter-strategies, on the other hand, are robust: they win against nearly every benchmark program they face. This robustness makes the RNR counter-strategies good candidates for use against an opponent who we *suspect* exhibits an exploitable trait. If we are correct, then the agent will perform well (although not optimally well); if we are incorrect, we know that its exploitability is bounded.

It is also of interest to compare the performance of the RNR agents to that of CFR5, the ϵ-Nash equilibrium in the same abstraction. Equilibrium strategies are the traditional safe strategy to use against arbitrary opponents, because of their worst-case performance guarantees. The RNR counter-strategies, used against the correct opponent, perform better than CFR5. Although CFR5 often outplays the RNR counter-strategies against the other opponents, it does not do so by a large margin.

5.5 Conclusion

Restricted Nash Response counter-strategies are *robust* responses to opponents, unlike traditional best responses, which tend to be *brittle* . In this chapter, we have shown that RNR agents can be computed using the same information as best response agents and in a reasonable amount of time. RNR agents can be computed to achieve nearly the same exploitative power as best responses, but with only a fraction of the exploitability. In their normal use, we have used RNR agents to achieve modest gains in utility against specific opponents, while limiting their worst case performance to within a chosen boundary.

In Chapter 6, we will consider creating a team of RNR agents, from which we will choose strategies for use against unknown opponents. We will show that an adaptive "coach" can learn which RNR strategies are able to exploit opponents *that they were not trained to defeat* ; furthermore, the team as a whole will be shown to be more effective than one ϵ-Nash equilibrium agent. This shows that counter-strategies such as Restricted Nash Responses can play an important role in competitions where the winner is determined by total earnings against several opponents, and not only by winning or losing each individual match.

Chapter 6

Managing a Team of Players: Experts Approaches

6.1 Introduction

In the previous chapters on Frequentist Best Response (Chapter 4), Counterfactual Regret Minimization (Chapter 3) and Restricted Nash Response (Chapter 5), we have presented techniques for creating effective poker agents. However, when the time for a competition arrives and we are facing an unknown opponent, we may not know which of our strategies to use. An ϵ-Nash equilibrium strategy is unlikely to lose, but will not win by a large margin. FBR strategies may win a large amount against the opponents they were trained to defeat, but will lose against other opponents. RNR strategies may win a large amount or lose a small amount, depending on the similarity of the opponent to the opponent the strategy was trained against.

One approach is to use the robust agents created by these techniques to form a team, and use a meta-agent to choose which one to use on each hand. The problem faced by this meta-agent is the one solved by **experts algorithms**. We first explored this idea in Section 2.7, where we presented UCB1. In this chapter, we will explore this method in more detail, and provide the results of experiments performed using teams of agents.

6.2 Choosing the team of strategies

To construct a team of agents, we first need to consider the types of agents that the team will include. The UCB1 algorithm is designed to trade off exploration and exploitation. To find the experimental expected value of using an agent, the allocation strategy must first use the agent to play several hands.

If we are using agents that are exploitable (such as the brittle FBR agents), then we will pay a heavy cost each time they are "explored". However, if the correct FBR counter-strategy to the opponent is in the team, then the allocation strategy also receives a clear signal when the best agent is chosen. The high utility of the correct agent should be quite distinct from the low utilities of the

remaining agents.

If we are using agents that are minimally exploitable (such as the robust RNR agents), then the cost for exploration is much lower. Whenever a suboptimal agent is selected by the allocation strategy, we know that the agent cannot be defeated by very much.

We could also add an ϵ-Nash equilibrium strategy to a team of FBR agents or RNR agents. If none of the agents are suitable for use against the opponent, then the allocation strategy should eventually identify the equilibrium agent as the agent with the highest expected value. However, for the experiments in this chapter, we will consider the use of an equilibrium strategy on its own to be an alternative approach that we will compare the use of teams to.

6.3 Using DIVAT

As we discussed in Section 2.2, poker has a significant amount of variance; each hand has a standard deviation of ± 6 small bets/game. If we use the number of small bets won or lost as the utility value input into UCB1 to choose our strategies, then the variance on each agent's utility will be high and it will take longer to accurately predict their average utility.

Alternatively, we can use a variant of DIVAT called Showdown DIVAT. The normal DIVAT algorithm requires full information to perform its analysis: it requires information about the opponent's cards, even when they fold. Showdown DIVAT is a variant that approximates the true DIVAT score by observing only the player's cards and the opponent's in cases where they are revealed. If the opponent folds, the number of small bets won or lost is used instead.

Similar to DIVAT, Showdown DIVAT has a variance that is much lower than the money reward of the game. The disadvantage of Showdown DIVAT is that it is no longer an unbiased estimator. It is possible that Showdown DIVAT will reward some strategies more than they deserve. However, we find that the variance reduction achieved through Showdown DIVAT is worth the cost of introducing bias.

6.4 Results

In this section, we will consider teams of FBR agents and RNR agents, using the UCB1 algorithm to learn online which agent to use. These two teams, along with an ϵ-Nash equilibrium agent, will compete against several opponents. Some of these (the **training set**), will be used as opponents for which we will create FBR and RNR strategies. Others (the **holdout set**) will not have corresponding FBR or RNR counter-strategies. Our experiments will show the effectiveness of these experts algorithms when confronted with opponents for which they do have effective agents, and when they do not.

Figure 6.1 shows the results of one such experiment. PsOpti4, Smallbot1399, Smallbot2298, and Attack80 were the four strategies that made up the training set, and Bluffbot and Monash-BPP

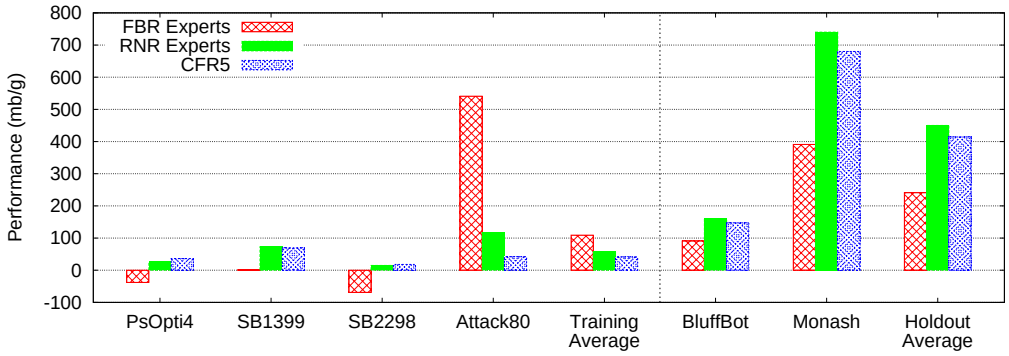


Figure 6.1: Performance of FBR-experts, RNR-experts, and an ϵ-Nash equilibrium strategy (CFR5) against "training" opponents and "hold out" opponents in 50 duplicate matches of 1000 hands.

formed the holdout set. This arrangement was not chosen arbitrarily. The four members of the training set are the strongest strategies from each of the four "families" of poker agents the CPRG has produced. Since they were developed internally and we have the actual strategies, it is simple to collect the data necessary to make FBR models based on them. The two members of the holdout set were competitors in the 2006 AAAI Computer Poker Competition, and are particularly useful in that they are unrelated to the poker strategies we have developed: we have no prior reason to believe that counter-strategies to our agents should work well against them[1].

Starting with the FBR team, we find that the FBR team can perform poorly even against the training set, where one of the strategies in the FBR team is a strong counter-strategy to the opponent. Although UCB1 can find the correct strategy eventually, the heavy exploration cost paid for trying the other three strategies has a costly impact on the team. Against three of the four training opponents, this cost of exploration causes the FBR team to perform very poorly. The one exception is Attack80, which is exploitable for more than 1000 millibets/game, as we learned in Table 4.2. Because of this one result, the FBR team performs better on average than the RNR team or CFR5 against the training opponents. Against the holdout opponents, the FBR team performs by far the worst of the three approaches, although it does still win against Bluffbot and Monash-BPP.

The RNR team's results are more promising. Against PsOpti4, Smallbot1399 and Smallbot2298, the RNR team performs approximately as well as CFR5. Against Attack80, UCB1 is able to learn to use the correct RNR counter-strategy to Attack80, resulting in a significantly higher score than CFR5 is able to obtain. On average, due to this one win, the RNR team places in between FBR and CFR5. Against the holdout opponents, however, the RNR team performs the best of the three approaches. Against both Bluffbot and Monash-BPP, UCB1 is able to find an RNR counter-strategy that performs well, even though the RNR strategies were not designed to defeat these opponents. This experiment

[1]There were two other competitors in the 2006 AAAI Computer Poker Competition, and they were not used in this experiment. The third place competitor in the series competition, GS2, is currently not available on the competition's benchmark server, and so we have no way of evaluating our performance against it. The fourth place competitor in the bankroll competition, Teddy, is a trivial agent that always raises. We did not consider it interesting for use in this experiment.

suggests two hypotheses. First, that the flaws being exploited by the RNR strategies are more general than the flaws being exploited by the FBR strategies, since the RNR strategies were more successful against new opponents. Second, that RNR agents acting as a team can sometimes perform better than an ϵ-Nash equilibrium strategy, motivating their use in tournaments where overall winnings is used to select a winner.

6.5 Conclusion

Exploitation is important in poker. Against new opponents, an ϵ-Nash equilibrium strategy may be a safe strategy, since it is unlikely to be defeated by a large margin, if at all. Restricted Nash Response strategies are able to exploit particular opponents, and when used as members of a team, it may be possible to find a member of that team that can outperform an ϵ-Nash equilibrium strategy against a new opponent.

In a tournament such as the Limit Bankroll event of the AAAI Computer Poker Competition, the winner is selected on the basis of total earnings, and not on the ability to defeat every opponent. In the next chapter, we will present the results of the 2007 AAAI Computer Poker Competition and the First Man-Machine Poker Championship. Although a team of Restricted Nash Response strategies was not used in the Limit Bankroll event, we will show that the CFR ϵ-Nash equilibrium strategy that was entered suffered from this lack of exploitive power.

Chapter 7

Competition Results

7.1 Introduction

The poker agents described in this thesis competed in two poker competitions this year, both presented at the Association for the Advancement of Artificial Intelligence (AAAI) 2007 Conference, held in Vancouver, Canada. These competitions were the 2007 AAAI Computer Poker Competition and the First Man-Machine Poker Championship.

7.2 The 2007 AAAI Computer Poker Competition

This year's event was the second annual Computer Poker Competition, in which teams submit autonomous agents to play Limit and No-Limit Heads-Up Texas Hold'em. This year, 15 competitors from 7 countries submitted 43 agents to three different tournaments. This is the world's only public annual competition for poker programs; the agents in the competition are the strongest known programs in the world. The Computer Poker Competition consisted of three Heads-Up Texas Hold'em tournaments: Limit Equilibrium, Limit Online Learning, and No-Limit.

7.2.1 Heads-Up Limit Equilibrium

Since finding an ϵ-Nash equilibrium strategy is an interesting challenge in its own right, one of the tournaments is designed to determine which player is closest to the Nash equilibrium in the real game. In this competition, every pair of competitors plays a series of several duplicate matches, and the winner of the series receives one point. The players are then ranked by their total number of points. By using this format, we are rewarding players for *not losing* : it does not matter how much they win by, it only matters that they win.

In this competition, the CPRG entered an agent called Hyperborean07EQ. It is an ϵ-Nash equilibrium strategy that plays in a 10-bucket nested abstraction (5 $E[HS^2]$ sets, each split into 2 $E[HS]$ buckets), and it is 2.27 mb/game exploitable in its own abstraction. This strategy was created using the Counterfactual Regret Minimization technique by using four CPUs over 14 days.

The results of the Limit Equilibrium competition are presented in Table 7.1. Hyperborean07EQ took first place in this competition. It did not lose a series of matches against any of its competitors, and had the highest average win rate of any competitor.

7.2.2 Heads-Up Limit Online

In the Heads-Up Limit Online tournament, players are rewarded for exploiting their opponents. Every pair of competitors plays a series of several duplicate matches. The players are then ordered by their total winnings over all of their matches. The bottom $1/3$ of competitors is eliminated to remove extremely exploitable players, and then rank the remaining $2/3$ of the players by their total winnings against the other remaining competitors.

In this competition, the CPRG entered two agents. The first agent was called Hyperborean07OL. It is an ϵ-Nash equilibrium strategy that plays in a 10-bucket non-nested abstraction (10 $[E[HS^2]$ buckets). It is 4.33 mb/g exploitable in its own abstraction. Hyperborean07OL was created by the Counterfactual Regret Minimization technique, using 4 CPUs over 14 days. The other CPRG entry was called Hyperborean07OL-2. It was created by a separate branch of research pursued by Billings and Kan. They describe it as a "quasi-equilibrium" that is better able to exploit weak opponents than regular ϵ-Nash equilibria strategies.

The full results of the Limit Online competition are presented in Table 7.2. As described above, the bottom $1/3$ is removed, resulting in the crosstable shown in Table 7.3. The competitors are ranked according to their winnings in this second, smaller table. The two CPRG entries, Hyperborean07OL-2 and Hyperborean07OL took first and second place respectively, with a statistically insignificant margin between them.

There is an interesting result in this, however. The first place agent, Hyperborean07OL-2, *lost* to the next 3 top ranked agents, and was able to win enough from the remaining opponents to still take first place. The second place agent, Hyperborean07OL, did not lose to any opponent, but only won enough from all opponents on average to narrowly miss first place. This result emphasizes one of the features of poker that make it an interesting game: exploitation is important. In this match, we have shown that an agent can lose to several opponents but still win overall, if it is better at exploiting the weak players. A team of several RNR counter-strategies and Hyperborean07OL, as described in Chapter 6, we may have performed better than either Hyperborean07OL or Hyperborean07OL-2.

7.2.3 No-Limit

In the No-Limit tournament, players are rewarded for exploiting the strongest opponents. Every pair of competitors plays a series of several duplicate matches. The players are then ordered by their total winnings over all of their matches. To find the winner, we repeatedly eliminate the player with

the lowest total winnings against players that have not yet been eliminated.

Before this competition, the CPRG had never created a No-Limit agent. The agent we entered was called Hyperborean07, and it uses another ϵ-Nash equilibrium strategy made by the Counterfactual Regret Minimization technique. It plays in an 8-bucket abstraction (8 $E[HS^2]$ buckets), and considers only four actions — fold, call, pot-raise, and all-in. A pot-raise is a raise of a size equal to the current size of the pot.

The results of the No-Limit competition are presented in Table 7.4. The CPRG entry, Hyperborean07, took third place, losing to BluffBot20NoLimit1 and GS3NoLimit1. Of these top three agents, Hyperborean obtained the highest average score, but was defeated by the top two agents. Once again, we found an interesting pattern in the results. The players SlideRule, Gomel, and Gomel-2 were able to exploit their opponents to a much larger degree than the other competitors. Even though they lost several of their matches, their large wins against the weaker opponents such as PokeMinn and Manitoba-2 meant that their average performance was higher than the top three.

This competition was designed to reward consistent play, but under a different winner determination rule, the ranking could have been very different, putting Gomel or SlideRule into first place.

7.2.4 Summary

In the AAAI 2007 Computer Poker Competition, the agents fielded by the CPRG competed against the world's best artificial poker agents and made a strong showing. In the Limit events that have been our focus since as early as 2003, we took first place twice and second once. One first place and the second place finish went to poker agents created through the techniques described in this thesis. This strong showing provides an experimental basis for our claims as to the applicability and usefulness of these techniques.

The introduction of the No-Limit event and our third-place finish in it affirm that No-Limit is a significantly different style of game than Limit. While our approaches towards finding equilibria and computing counter-strategies are still valid in this game, we believe there are great improvements in performance that can be realized by changing the abstract game in which we compute our strategies. In Section 8.1, we will present our current ideas for future directions that this research can take in order to accommodate the new challenges presented by No-Limit Heads-Up Texas Hold'em.

7.3 The First Man-Machine Poker Competition

The second competition at AAAI was the First Man-Machine Poker Championship, an exhibition match between several of the CPRG's poker agents and two world-class human poker experts, Phil Laak and Ali Eslami. Laak has a degree in Mechanical Engineering and was a competitive backgammon player before turning to poker. Eslami was a computer consultant before discovering his poker talents, and has a strong technical background. These two competitors are not only strong poker players, but technically minded opponents that are familiar with our research.

In order to get statistically significant results, we decided to use duplicate matches for the event. Since humans cannot have their memories erased between matches, this meant that we needed two human experts that would act as a team. In each match, either Laak or Eslami would play against one of our poker agents in the event's main venue. At the same time, in a separate room, the other human player would play against one of our poker agents, with the cards reversed. The two humans usually played against the same poker agent.

Over two days, four duplicate 500 game matches were held, making a total of 4000 hands. After each 500 hand match, the two humans and two computer agents would combine their scores. If the higher scoring team was ahead by more than 25 small bets, then they were declared the winner; any result that was ± 25 small bets was declared a statistical tie. For their efforts, the human players were each given $5000, plus $2500 each for each match they won, or $1250 each for each match they tied. For the purposes of this match, our set of poker agents was called Polaris, and each agent was given a color-coded nickname — Mr. Pink, Mr. Orange, and so on. Photographs and a hand-by-hand blog of the event are available online at www.manmachinepoker.com.

We will now describe each match and present the results from it, along with a DIVAT post-match analysis.

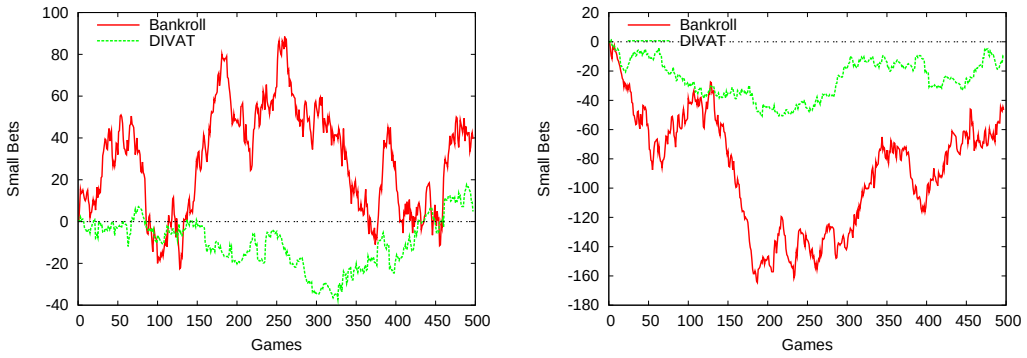
7.3.1 Session 1: Monday July 23rd, Noon

In Session 1, the CPRG team used an ϵ-Nash equilibrium nicknamed "Mr. Pink" to play against Laak and Eslami. In this session, Eslami was on stage in the main room, and provided commentary about the match to the crowd as the game progressed. Meanwhile, Laak was sequestered in a hotel room with two CPRG members. Figure 7.1 shows the progress of the matches that Laak and Eslami were playing. On each graph, both the Bankroll line (number of bets won) and the DIVAT line (an unbiased estimator of the Bankroll, but with less variance) is shown.

Our agent, Mr. Pink, is the largest and strongest CFR ϵ-Nash equilibrium strategy that we have created to date, and the largest known equilibrium strategy for Texas Hold'em. It plays in a nested abstraction with 6 $E[HS^2]$ sets each split into 2 $E[HS]$ buckets, for a total of 12 buckets on each round. We chose to enter an equilibrium strategy as a baseline for comparison against the agents we would use in future matches. Since an ϵ-Nash equilibrium strategy is virtually unbeatable within its own abstraction, observing the performance of the human experts against the equilibrium strategy would give us information about the accuracy of the abstractions we use.

We believe it was a significant factor that, in this session, the human experts did not know the nature of the opponent they were facing. At several times during the match, Eslami described his strategy to the audience, and wondered whether or not he was facing a learning opponent that could adapt to his play, or an equilibrium strategy that would not attempt to exploit him if he played sub-optimally. We saw that the "threat" of learning can in itself be a valuable tool.

Session 1 ended as a statistical tie. At the end of the match, Polaris was ahead by 7 small bets;



(a) On Stage: Ali Eslami

(b) Hotel: Phil Laak

Figure 7.1: Bankroll and DIVAT graphs for Session 1. This session was a statistical tie: the match ended with Polaris ahead by 7 small bets. According to DIVAT analysis, Polaris was expected to win by 5 small bets.

this is within the ± 25 small bet range where a tie is declared.

7.3.2 Session 2: Monday July 23rd, 6pm

In Session 2, the CPRG team used a new, experimental agent developed by the author, nicknamed "Mr. Orange". In this session, Eslami was sequestered in the hotel room and Laak was on stage, providing commentary. The bankroll and DIVAT graphs from this duplicate match is shown in Figure 7.2. Once again, in this match, the human players did not know the nature of the opponent they were facing. They only knew the name (Mr. Orange), and that it was a different agent than they had faced in the first match.

Mr. Orange was created by a method developed by the author which was not discussed in this thesis, due to its experimental nature; nonetheless, it proved to be very effective in this match, and so we will briefly describe it here. Billings, the CPRG's resident poker expert, has often criticized ϵ-Nash equilibrium agents for not being aggressive enough. In response, Mr. Orange was developed to be an aggressive player which is also close to a Nash equilibrium. In poker, an "aggressive" player is one who bets more often and presses harder to win the pot, even in marginal cases where the expected value of such actions may be slightly negative. An aggressive strategy confers several benefits:

- **Punishes suboptimal play**. When playing against an equilibrium strategy, players are usually at liberty to act suboptimally if it allows them to find weaknesses in the strategy. Since the strategy does not attempt to exploit this weakness, the player pays a small price for such actions. Against an aggressive player who increases the size of the pot, the cost of these "probing" actions is increased.

- **Provides more opportunities for opponent error**. By playing more aggressively, the agent forces the human players to make more decisions on each hand, and each game takes longer to

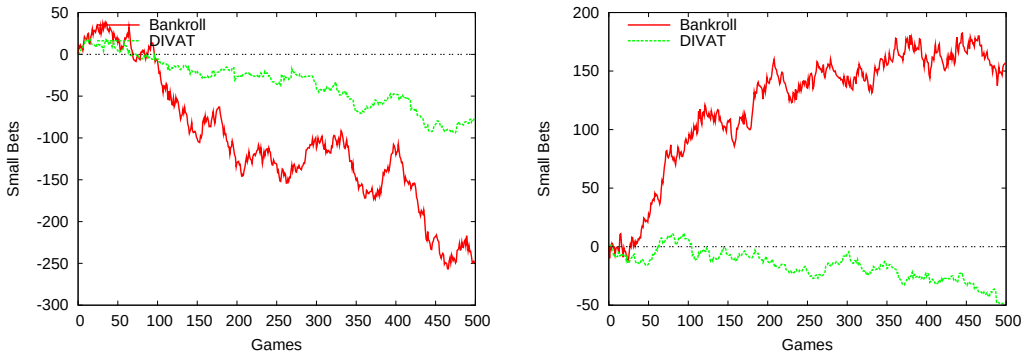
play. Furthermore, the outcome of each hand is more significant, as a larger pot can become a larger win or a larger loss. Every time the human player takes an action, there is the potential that they will make a mistake: perhaps they will form an incorrect model of the agent and fold a winning hand, or raise when they do not have the correct odds to do so, or will simply become more fatigued. As computer agents never get tired, the longer games are only a penalty to the human players.

- **Aggression is not an expensive mistake**. Of the types of suboptimal play an agent can exhibit — too much or too little aggression, folding too many or too few hands, bluffing too often or not often enough — being over or under aggressive is the least costly. Laak and Eslami claimed Mr. Orange was too aggressive, and yet it is only 35 millibets/game suboptimal in its own abstraction. Humans perceive this aggression as a larger weakness than it actually is.

To create an aggressive agent, we used the Counterfactual Regret Minimization technique from Chapter 3, but with one change. On every terminal node of the game tree, we gave a 7% bonus in utility to the winning player, while the losing player did not pay any extra cost. This means that the game is no longer zero sum, as more money is being introduced on each hand. The important benefit of this approach is that the agent always thinks it has better odds than it actually does — it is more willing to fight for marginal pots. For every $1 it invests in a bet, it thinks it will receive $1.07 in return even if the opponent immediately folds, and $2.14 if the opponent calls the bet and the agent wins the showdown. This extra return on investment encourages it to bet in more situations where it might otherwise call or fold. However, since it learns its strategy by playing against an agent trying to exploit it, it learns to express this extra aggression in a balanced way that still effectively hides information.

This agent proved to be very effective against Eslami and Laak. It won Session 2 by a 92.5 small bet margin, a result far beyond the 25 small bet boundary for a tie. The cards dealt to the players in this game were very unbalanced; Laak and the Polaris agent opposing Eslami were consistently dealt stronger cards than their opponents. Although Laak won his side of the match by 157 small bets, Polaris was able to use the same cards to beat Eslami by 249.5 small bets.

After the session had ended, the players described the style of play of Mr. Pink and Mr. Orange, the two poker agents they had faced that day. They said that Mr. Pink played like a calm old man. According to Laak, Mr. Orange "...was like a crazed, cocaine-driven maniac with an ax" [8]. They also suggested what they thought would be a strong combination: a team comprised of Mr. Pink and a slightly less aggressive Mr. Orange. Against this opponent, they said, they would not be able to know if bets were being made because of Mr. Orange's aggression, or because of a strong hand held by Mr. Pink. This corresponds with a conclusion we reached in Section 7.3.1: just the threat of an adaptive opponent can cause the team to perform better than any of its components.



(a) Hotel: Ali Eslami

(b) On Stage: Phil Laak

Figure 7.2: Bankroll and DIVAT graphs for Session 2. This session was a win for Polaris: the match ended with Polaris ahead by 92.5 small bets. According to DIVAT analysis, Polaris was expected to win by 126 small bets.

7.3.3 Session 3: Tuesday July 24th, Noon

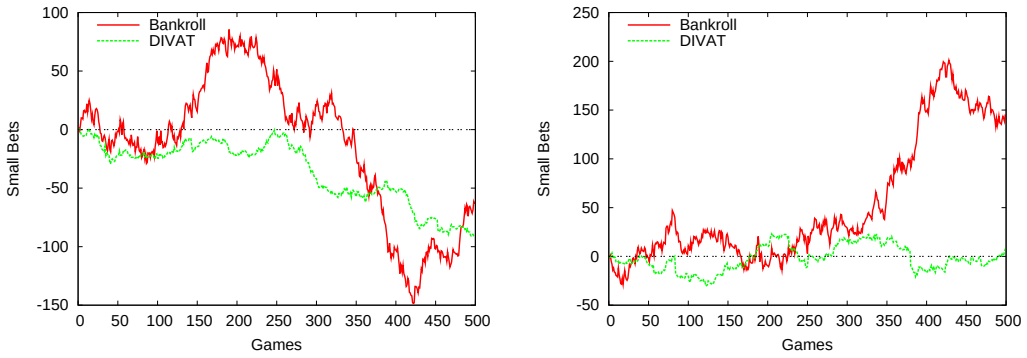
In Session 3, the CPRG fielded different teams of agents against Laak and Eslami. Using the results of the previous day's matches, we used importance sampling to identify 3 out of 10 possible agents that were most likely to perform well against Laak and Eslami. We used UCB1 as the meta-agent to choose between these three agents during the match. The results of this match are presented in Figure 7.3. In this match, the human players were told that they were competing against a team of agents that would adapt to their play, and were told if "Mr. Pink" or "Mr. Orange" were members of the team.

In this match, an ϵ-Nash equilibria agent and two Restricted Nash Response agents were used against Laak, while Eslami once again played against a team of Mr. Pink, another ϵ-Nash equilibria and Mr. Orange, the aggressive agent from session 2. Due to a bug in our program that manages the team, learning only occurred on hands that ended in a showdown, and this is also the only time that the agent in control was changed. If either player folded for several games in a row, a suboptimal agent could be left in control for far longer than was intended. Fortunately, since all of the agents used give guarantees on their worst case performance, the agents did not suffer as much as they could have from this flaw.

Laak in particular was happy with his performance in this match. While Eslami lost 63.5 small bets to the combined strength of Mr. Pink and Mr. Orange, Laak ended the match up 145.5 small bets, causing Polaris' first loss — it ended the session trailing by 82 small bets.

7.3.4 Session 4: Tuesday July 24th, 6pm

In the final session, Eslami was on stage and Laak was sequestered in the hotel room. Now that the match was tied overall with a tie and one win for each team, the CPRG decided to use a low-variance style of play to attempt to close out a tie or perhaps earn a win. The DIVAT analysis



(a) Hotel: Ali Eslami

(b) On Stage: Phil Laak

Figure 7.3: Bankroll and DIVAT graphs for Session 3. This session was a win for Laak and Eslami: the match ended with Polaris behind by 82 small bets. According to DIVAT analysis, Polaris was expected to win by 83 small bets.

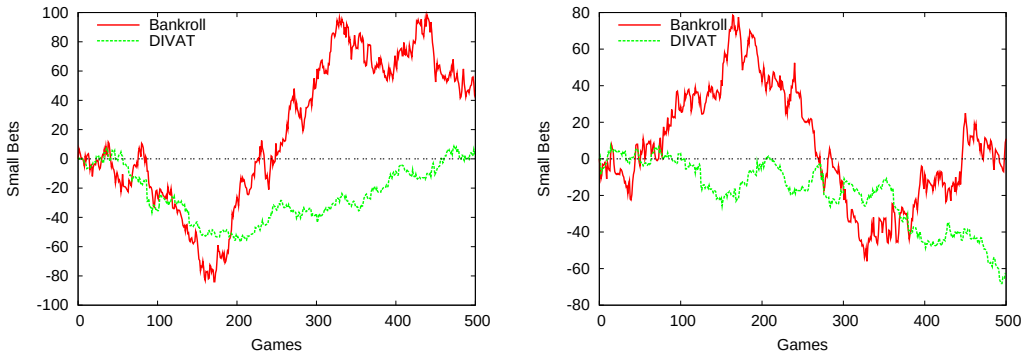
from Session 1 had shown that in the long run, Mr. Pink was expected to tie that match. This confidence in Mr. Pink, along with a fear that Mr. Orange's aggressive style might increase variance and increase the chances of a loss, made the group decide to field Mr. Pink alone for this session. We briefly considered entering Mr. Pink and Mr. Orange as a team, but (as described in Session 3's description), there was a bug in the program that managed the team, and it was not yet fixed. We entered Mr. Pink alone rather than risk encountering new unforeseen problems with the team program. In retrospect, even with the bug in the team management code, using Mr. Pink and Mr. Orange together would have caused uncertainty and the "threat of learning" that we have mentioned previously, and the humans might have performed worse as a result.

Part of the agreement with the human players was that we would tell them a nickname for each opponent, although we were not required to tell them the *nature* of that opponent. This meant that the players knew from the start that their opponent was not a learning agent, and that it was the same Mr. Pink that they had encountered on Session 1.

The results of Session 4 are presented in Figure 7.4. In a match where the CPRG's poker experts were repeatedly impressed by the human players' performance, Laak and Eslami both finished their sides of the match ahead by 11 and 46 small bets respectively, causing Polaris' second loss of the day, down by 57 small bets.

7.3.5 Man-Machine Match Conclusions

When observing the bankroll and DIVAT graphs for each match the reader may notice that, according to the DIVAT scores, Polaris was expected to tie the first match and win the next three matches by more than the 25 small bet boundary for tying the game. It is important to note that although DIVAT reduces variance, it is not free from it. According to the rules of the competition, there is no doubt that the humans won the championship. Throughout each match, we were consistently amazed by



(a) On Stage: Ali Eslami

(b) Hotel: Phil Laak

Figure 7.4: Bankroll and DIVAT graphs for Session 4. This session was a win for Laak and Eslami: the match ended with Polaris behind by 57 small bets. According to DIVAT analysis, Polaris was expected to win by 57 small bets.

their insight into the game, the quality of their play, and their ability to occasionally name the exact cards that Polaris was holding. On the other hand, the post-match DIVAT analysis suggests that the human team's performance may still have had a strong luck factor. If additional games were played, DIVAT suggests that the outcome may have been different.

One conclusion to be drawn from the match and the DIVAT scores, however, is that duplicate matches alone are not enough for reducing the variance of these short games. Part of the reason for this is that human players do not all play by the same style. Eslami describes himself as an aggressive player, and he describes Laak as a cautious and trapping player. Because of their different styles, these players are willing to play different hands in different ways; the variance reduction gained from duplicate matches is most effective in computer-vs-computer matches, where a player uses the same strategy on both sides of the duplicate match. In a future Man-Machine Championship, we should investigate ways to further reduce variance, such as longer matches, playing more than two matches at once (4 humans and 4 computers, for example), or the official use of a tool similar to DIVAT to determine the winner, as opposed to using the noisy bankroll line.

We can also conclude from these matches that our poker agents are quickly approaching world champion levels of play in this game. Eslami and Laak's post game comments were perhaps more telling than any graph in this regard. Eslami was quick to exclaim that the end result was not a win for the human players — he said he had played the best poker of his life, and they just barely won. Across all four matches, the total score for the humans was 39.5 small bets won over 4000 games, or 10 millibets/game. This was a narrower margin of victory than any of the 1st Place vs 2nd Place results from the 2007 AAAI Computer Poker Competition events.

We conclude this section with the observation that the techniques described in this thesis can produce poker agents that become stronger by the hour, while the quality of human professional play is unlikely to improve significantly over the coming year. If there is a Second Man-Machine

Poker Championship, the computer team will be able to field agents that are stronger than any of the agents used in this competition, and they may be ready to challenge the world's very best masters at this particular variant of poker.

	Hyperborean07EQ	IanBot	GS3	PokeMinn	Quick	Gomel-2	DumboEQ	DumboEQ-2	Sequel	Sequel-2	PokeMinn-2	UNCC	Gomel	LeRenard	MonashBPP	MilanoEQ	Average
Hyperborean07EQ		21	32	136	115	110	193	182	165	166	131	454	115	138	465	428	194
IanBot	-21		4	130	99	85	142	119	131	140	142	472	88	130	408	398	164
GS3	-32	-4		150	73	112	160	149	140	148	154	467	107	142	412	445	175
PokeMinn	-136	-130	-150		40	144	80	76	-33	-22	-24	373	265	127	627	421	111
Quick	-115	-99	-73	-40		19	235	135	125	121	134	298	149	15	564	489	131
Gomel-2	-110	-85	-112	-144	-19		206	200	135	150	16	275	232	136	802	859	169
DumboEQ	-193	-142	-160	-80	-235	-206		133	67	64	55	23	300	13	774	672	72
DumboEQ-2	-182	-119	-149	-76	-135	-200	-133		87	82	83	-52	271	54	808	762	74
Sequel	-165	-131	-140	33	-125	-135	-67	-87		19	130	167	-17	92	556	556	46
Sequel-2	-166	-140	-148	22	-121	-150	-64	-82	-19		125	174	-4	74	583	526	41
PokeMinn-2	-131	-142	-154	24	-134	-16	-55	-83	-130	-125		96	123	60	770	748	57
UNCC	-454	-472	-467	-373	-298	-275	-23	52	-167	-174	-96		95	-281	553	503	-125
Gomel	-115	-88	-107	-265	-149	-232	-300	-271	17	4	-123	-95		96	779	993	10
LeRenard	-138	-130	-142	-127	-15	-136	-13	-54	-92	-74	-60	281	-96		478	354	2
MonashBPP	-465	-408	-412	-627	-564	-802	-774	-808	-556	-583	-770	-553	-779	-478		489	-539
MilanoEQ	-428	-398	-445	-421	-489	-859	-672	-762	-556	-526	-748	-503	-993	-354	-489		-576

Table 7.1: Crosstable of results from the 2007 AAAI Computer Poker Competition Limit Equilibrium event. Results are in millibets/game for the row player, and are the average of 10 duplicate matches of 3000 games (60,000 games total). Results involving the top three players (Hyperborean07EQ, IanBot, GS3) are the average of 100 duplicate matches of 3000 games (600,000 games total), to ensure statistical significance. The competitors' rank in this event is the same as the order in which they are presented.

	Hyperborean07OL-2	Hyperborean07OL	GS3	IanBot	Quick	Gomel-2	PokeMinn	Sequel	Sequel-2	LeRenard	DumboOL-2	PokeMinn-2	DumboOL	Gomel	UNCC	MonashBPP	MilanoOL	Average
Hyperborean07OL-2		-37	-27	-37	138	155	172	166	178	170	259	247	280	209	530	631	699	233
Hyperborean07OL	37		21	27	116	108	141	153	175	132	207	136	190	110	466	432	450	181
GS3	27	-21		6	73	112	150	140	148	142	199	154	183	107	467	412	509	175
IanBot	37	-27	-6		99	85	130	131	140	130	157	142	121	88	472	408	422	158
Quick	-138	-116	-73	-99		19	-40	125	121	15	129	134	244	149	298	564	610	121
Gomel-2	-155	-108	-112	-85	-19		-144	135	150	136	123	16	217	232	275	802	1174	165
PokeMinn	-172	-141	-150	-130	40	144		-33	-22	127	-15	-24	75	265	373	627	406	86
Sequel	-166	-153	-140	-131	-125	-135	33		19	92	-1	130	-52	-17	167	556	648	45
Sequel-2	-178	-175	-148	-140	-121	-150	22	-19		74	17	125	-48	-4	174	583	602	39
LeRenard	-170	-132	-142	-130	-15	-136	-127	-92	-74		21	-60	8	-96	281	478	444	4
DumboOL-2	-259	-207	-199	-157	-129	-123	15	1	-17	-21		71	122	146	196	707	898	65
PokeMinn-2	-247	-136	-154	-142	-134	-16	24	-130	-125	60	-71		-49	123	96	770	971	52
DumboOL	-280	-190	-183	-121	-244	-217	-75	52	48	-8	-122	49		285	-4	770	1104	54
Gomel	-209	-110	-107	-88	-149	-232	-265	17	4	96	-146	-123	-285		-95	779	1443	33
UNCC	-530	-466	-467	-472	-298	-275	-373	-167	-174	-281	-196	-96	4	95		553	1129	-126
MonashBPP	-631	-432	-412	-408	-564	-802	-627	-556	-583	-478	-707	-770	-770	-779	-553		762	-519
MilanoOL	-699	-450	-509	-422	-610	-1174	-406	-648	-602	-444	-898	-971	-1104	-1443	-1129	-762		-767

Table 7.2: Crosstable of all results from the 2007 AAAI Computer Poker Competition Limit Online Learning event. Results are in millibets/game for the row player, and are the average of 10 duplicate matches of 3000 games (60,000 games total). The competitors' rank in this event is the same as the order in which they are presented.

	Hyperborean07OL-2	Hyperborean07OL	GS3	IanBot	Quick	Gomel-2	PokeMinn	Sequel	Sequel-2	LeRenard	DumboOL-2	Average
Hyperborean07OL-2		-37	-27	-37	138	155	172	166	178	170	259	114
Hyperborean07OL	37		21	27	116	108	141	153	175	132	207	112
GS3	27	-21		6	73	112	150	140	148	142	199	98
IanBot	37	-27	-6		99	85	130	131	140	130	157	87
Quick	-138	-116	-73	-99		19	-40	125	121	15	129	-6
Gomel-2	-155	-108	-112	-85	-19		-144	135	150	136	123	-8
PokeMinn	-172	-141	-150	-130	40	144		-33	-22	127	-15	-35
Sequel	-166	-153	-140	-131	-125	-135	33		19	92	-1	-71
Sequel-2	-178	-175	-148	-140	-121	-150	22	-19		74	17	-82
LeRenard	-170	-132	-142	-130	-15	-136	-127	-92	-74		21	-100
DumboOL-2	-259	-207	-199	-157	-129	-123	15	1	-17	-21		-110

Table 7.3: Crosstable of results from the 2007 AAAI Computer Poker Competition Limit Online Learning event, after removing the bottom 1/3 of players. Results are in millibets/game for the row player, and are the average of 10 duplicate matches of 3000 games (60,000 games total). The competitors' rank in this event is the same as the order in which they are presented.

	BluffBot20	GS3	Hyperborean07	SlideRule	Gomel	Gomel-2	Milano	Manitoba	PokeMinn	Manitoba-2	Average
BluffBot20		267	380	576	2093	2885	3437	475	1848	2471	1603
GS3	-267		113	503	3161	124	1875	4204	-42055	5016	-3036
Hyperborean07	-380	-113		-48	6657	5455	6795	8697	12051	22116	6803
SlideRule	-576	-503	48		11596	9730	10337	10387	15637	10791	7494
Gomel	-2093	-3161	-6657	-11596		3184	8372	11450	62389	52325	12690
Gomel-2	-2885	-124	-5455	-9730	-3184		15078	11907	58985	40256	11650
Milano	-3437	-1875	-6795	-10337	-8372	-15078		5741	12719	27040	-44
Manitoba	-475	-4204	-8697	-10387	-11450	-11907	-5741		18817	50677	1848
PokeMinn	-1848	42055	-14051	-15637	-62389	-58985	-12719	-18817		34299	-12010
Manitoba-2	-2471	-5016	-22116	-10791	-52325	-40256	-27040	-50677	-34299		-27221

Table 7.4: Crosstable of results from the 2007 AAAI Computer Poker Competition No-Limit event. Results are in millibets/game for the row player, and are the average of 20 duplicate matches of 1000 games (40,000 games total). Results involving the top three players (BluffBot20, GS3, Hyperborean07) are the average of 300 duplicate matches of 1000 games (600,000 games total), to ensure statistical significance. The competitors' rank in this event is the same as the order in which they are presented.

Chapter 8

Conclusion

In this thesis, we have presented three new techniques for creating strong poker agents, as well as showing how they can be combined into a team. These techniques are:

- **Counterfactual Regret Minimization**: a new method for finding ϵ-Nash equilibria strategies in very large stochastic, imperfect information games. It has more favorable memory bounds than traditional approaches, allowing for the calculation of ϵ-Nash equilibria strategies in larger abstractions that are closer to the real game.
- **Frequentist Best Response**: a variant on the standard best response algorithm that has fewer restrictions. It can calculate a "good response" based on observed hands, and can generate counter-strategies in a chosen abstract game.
- **Restricted Nash Response**: an approach that creates a new type of strategy. This approach creates strategies that are a compromise between best responses and Nash equilibria. The strategies are trained to defeat a particular opponent or class of opponents, but can be tuned with a parameter to determine how much emphasis to put on exploiting that opponent as compared to minimizing its own exploitability.
- **UCB1 Teams of Strategies**: an application of the UCB1 experts algorithm to the poker domain. By using teams of Restricted Nash Response strategies, we are able to defeat unseen opponents by a larger margin than an equilibrium strategy.

Each of these techniques has been an improvement over existing technologies, or has created a new niche which did not previously exist.

8.1 Future Work

After the First Man-Machine Championship, the CPRG discussed future directions of this work amongst ourselves and with Phil Laak and Ali Eslami, the two poker experts who we competed against. We have highlighted a range of exciting new ideas resulting from this work that will be pursued over the coming months and years:

8.1.1 Improved Parallelization

The current method of parallelizing the Counterfactual Regret Minimization and Restricted Nash Response code is necessary but crude. In each program, eight computers are used, and each controls a different part of the search tree. This is done for two reasons. The first reason is to make use of the memory of eight computers instead of one; the game tree for the 12-bucket abstraction alone required 16.5 gigabytes of memory, and more overhead was required to store the details of the abstraction. The computers available to us each had 8 gigabytes of memory; the parallelization succeeded in evenly dividing the memory use over the 8 computers. The second reason was to create opportunities for parallel traversal of the game tree fragments. By using 8 CPUs instead of one, we were able to achieve a 3.5x speedup. However, some lines of play are visited much less frequently than others, and the parallelization had poor load balancing. Some CPUs (notably the ones handling the check-call, check-bet-call, and bet-call sequences) were bottlenecks.

This parallelization can be extended in two ways. First, there are still gains to be had from spreading the program over more computers. By using 21 computers or more, we can make use of more distributed memory and solve even larger abstractions. Second, the parallelization can employ better load balancing by spreading the work more evenly over the available computers, so as to increase efficiency.

8.1.2 No Limit Texas Hold'em

Using the Counterfactual Regret Minimization technique described in Chapter 3, we produced an agent that competed in the 2007 AAAI Computer Poker Competition's No-Limit event. As we reported in Chapter 7, it placed third out of ten agents — a respectable showing, but with room for improvement. There are two significant avenues for improvement. First, the abstraction allowed only betting actions that matched the size of the pot or bet the agent's entire stack. By considering finer-grained betting actions, we should be able to improve the performance of the program at the cost of increased time and space requirements. Second, the card abstraction we used was designed for Limit Texas Hold'em, and it may not be suitable for No-Limit.

In Limit Hold'em, the bets on the first two rounds are smaller than on the last two rounds. This means that the agents get to observe 3 buckets before they have to make "expensive" decisions; as the abstraction provided 8 history buckets on each round, that meant the agent knew it was in one of 512 buckets before making such a decision.

In No-Limit Hold'em, an agent may be required to make a decision involving all of their money (the most expensive type of decision possible!) on the very first round, where they are in one of only 8 buckets. An alternate abstraction may shift more buckets from the later rounds to earlier rounds, so that agents can receive more precise information right from the start.

8.1.3 Dynamic Opponent Modeling

In Chapter 6, we considered an approach using several agents working as a team to defeat an opponent. However, this approach assumes that the opponent is stationary, as the historical performance of the agents is used to choose the agent to play the next hand.

A technique that has enjoyed recent success in simpler poker games, such as Kuhn poker [2], performs dynamic opponent modeling. This approach considers not only the strategy employed by the opponent, but ways in which they are changing their strategy over time: do they slowly drift from one part of the strategy space to another, or do they suddenly "change gears" and adopt a new strategy that might defeat the strategy we are currently using?

If a set of strategies can be developed that covers a large "interesting" part of the strategy space, we can try to learn what strategies the opponent is most likely to be using, and how they are changing their play over time. Then, we can employ counter-strategies such as Restricted Nash Responses to defeat them, while retaining the assurance that our counter-strategies will be robust if we have misjudged their movement.

8.1.4 Imperfect Recall Abstractions

The new ϵ-Nash equilibrium finding methods described in Chapter 3 allowed us to solve larger games than were previously possible, but we were soon solving games large enough that we used all of our available memory. We then developed the parallel version of the code to gain access to more memory by using eight computers, and we promptly used all of the distributed memory, as well.

A new approach to solving large games while decreasing our memory use may be to solve imperfect recall games. In the abstractions we have previously discussed, the agents have retained **perfect recall**: they choose their actions based on the entire history of the game. Instead, we can create **imperfect recall** abstractions, where agents "forget" what buckets or actions they observed on rounds previous to this one, and must make their actions based on only some of the history. By "forgetting" unimportant information, we are able to reduce the size of the abstraction. This has an advantage in that we may be able to create strategies in abstract games that have thousands of buckets on each round, instead of our current system of 12 Preflop buckets, 144 Flop buckets, and so on. This approach may be the key to creating strong No-Limit strategies.

8.1.5 Equilibrium Strategies in Perturbed Abstractions

During the second match of the First Man-Machine Poker Championship, we used a highly experimental poker agent that plays a very aggressive style of poker. It won the second match, and was also highly praised by the human experts.

This agent was made more aggressive by "perturbing" the abstraction. Each terminal node in the game tree was modified such that it gave a 7% utility bonus to the winner, while the loser did not pay an extra fee. When we found an ϵ-Nash equilibrium strategy in this new, non-zero sum game, we

found that it was much more aggressive, while still having a low exploitability in the un-perturbed abstract game.

There are several different ways in which we can perturb an abstraction to create a variety of strategies with different characteristics: more aggressive, more passive, tighter (folds more hands), looser (plays more hands), and so on. Each of these strategies is interesting in that it is another style of play that an opponent might be weak against.

Although the strategy generated by this approach performed very well in the competition, it has not yet been examined in detail. After research is performed to ensure the validity of the approach, this ability to form a variety of playing styles may become very useful.

8.1.6 Improved Abstractions

The $E[HS^2]$ metric as described in Section 2.5.5 is useful, but can certainly be improved upon. Instead of bucketing hands based only upon $E[HS^2]$ and $E[HS]$, we would like to find methods that provide a clearer separation between potential hands and normal hands. We would also like to consider hands with negative potential — a hand that is strong, but can become much weaker if certain cards are dealt. There is also a great deal of public information that the current bucketing scheme does not capture. For example, if four cards of the same suit are present on the board, there is a moderate probability that our opponent has a flush. Human players are aware of the "texture" of the board — the features of the board cards that indicate that the opponent may have a high potential hand. By designing our buckets to incorporate this public information, we may be able to create abstractions that are even closer to the real game.

8.2 Concluding Remarks

In this thesis, we have presented three new techniques for creating poker strategies and counter strategies, and shown how to combine them into a team. The Counterfactual Regret Minimization and Restricted Nash Response agents are of particular interest because they are robust: against any opponent, they are unlikely to be defeated by a large margin. Although the Frequentist Best Response strategies are too brittle to be used in a competition, the technique is still very useful for determining an agent's worst case exploitability and for obtaining models for the Restricted Nash Response technique. The value of these techniques was shown by using the agents to compete in an international competition for computer poker programs and to compete against two human professional poker players.

Poker is a challenging and popular example of the domain of stochastic, imperfect information games. Research into deterministic, perfect information games such as chess and checkers has resulted in advances in areas such as heuristic search that have found applicability in other areas of computing science. We feel that research into stochastic, imperfect information games such as poker

will reveal additional applications beyond the games domain, particularly because agents acting in the real world often have to deal with both stochasticity and imperfect information.

Since the AAAI Computer Poker Competition has become an annual event and is entered by teams of academics, research into computer poker promises to be an exciting and fruitful field in the coming years.

Bibliography

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–236, 2002.

[2] Nolan Bard and Michael Bowling. Particle filtering for dynamic agent modelling in simplified poker. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*, pages 515–521, 2007.

[3] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[4] Darse Billings. *Algorithms and Assessment in Computer Poker*. PhD thesis, University of Alberta, 2006.

[5] Darse Billings, Michael Bowling, Neil Burch, Aaron Davidson, Rob Holte, Jonathan Schaeffer, and Terence Schauenberg. Game tree search with adaptation in stochastic imperfect information games. In *International Conference on Computers and Games (CG)*, pages 21–34, 2004.

[6] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1–2):201–240, 2002.

[7] Darse Billings and Morgan Kan. A tool for the direct assessment of poker decisions. In *The International Association of Computer Games Journal*, 2006.

[8] Alan Boyle. Humans beat poker bot... barely. CosmicLog, MSNBC. `http://cosmiclog.msnbc.msn.com/archive/2007/07/25/289607.aspx`.

[9] Michael Buro. The othello match of the year: Takeshi Murakami vs. Logistello. *International Computer Chess Association Journal*, 20(3):189–193, 1997.

[10] Andrew Gilpin and Tuomas Sandholm. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the Twenty-First Conference on Artificial Intelligence (AAAI-06)*, 2006.

[11] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In *AAMAS'07*, 2007.

[12] Andrew Hodges. *Alan Turing: The Enigma*. Vintage, 1992.

[13] F. H. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

[14] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *NIPS07*, 2008. To appear (8 pages).

[15] Morgan Kan. Postgame analysis of poker decisions. Master's thesis, University of Alberta, 2007.

[16] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Annual ACM Symposium on Theory of Computing, STOC'94*, pages 750–759, 1994.

[17] Kevin Korb, Ann Nicholson, and Nathalie Jitnah. Bayesian poker. In *Uncertainty in Artificial Intelligence*, 1999.

[18] Michael Littman and Martin Zinkevich. The aaai computer poker competition. *Journal of the International Computer Games Association*, 29, 2006.

[19] John Markoff. In poker match against a machine, humans are better bluffers. New York Times, July 26, 2006. http://www.nytimes.com/2007/07/26/business/26poker.html?ex=1343275200&en=4a9369b4a4912ebe&ei=5124&partner=permalink&exprod=permalink.

[20] Peter McCracken and Michael Bowling. Safe strategies for agent modelling in games. In *AAAI Fall Symposium on Artificial Multi-agent Learning*, October 2004.

[21] Laszlo Mérö and Viktor Mészáros. *Ways of Thinking: The Limits of Rational Thought and Artificial Intelligence*. World Scientific, 1990.

[22] M. Osborne and A. Rubenstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[23] John W. Romein and Henri E. Bal. Solving awari with parallel retrograde analysis. *IEEE Computer*, 36(26):26–33, 2003.

[24] Teppo Salonen. The official website for bluffbot - poker robot for heads-up limit texas holdem. http://www.bluffbot.com.

[25] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Rob Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 2007.

[26] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK: The world man-machine checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[27] Terence Schauenberg. Opponent modeling and search in poker. Master's thesis, University of Alberta, 2006.

[28] Brian Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134:241–275, 2002.

[29] David Staples. Poker pros out of luck in battle with 'bot. Edmonton Journal, June 11, 2007. http://www.canada.com/edmontonjournal/news/story.html?id=89ee2a49-604c-4906-b716-f96b9087c52e.

[30] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[31] Wikipedia. Texas hold 'em — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Texas_hold_em&oldid=143501436, 2007. [Online; accessed 10-July-2007].

[32] Wikipedia. World series of poker — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=World_Series_of_Poker&oldid=149282484, 2007. [Online; accessed 9-August-2007].

[33] Martin Zinkevich, Michael Bowling, Nolan Bard, Morgan Kan, and Darse Billings. Optimal unbiased estimators for evaluating agent performance. In *American Association of Artificial Intelligence National Conference, AAAI'06*, pages 573–578, 2006.

[34] Martin Zinkevich, Michael Bowling, and Neil Burch. A new algorithm for generating strong strategies in massive zero-sum games. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, 2007.

[35] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. Technical Report TR07-14, Department of Computing Science, University of Alberta, 2007.

[36] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *NIPS07*, 2008. To appear (8 pages).