University of Alberta

Opponent Modelling and Search in Poker

by

Terence Conrad Schauenberg ©

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
Spring 2006

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-13883-1
Our file Notre référence
ISBN: 0-494-13883-1

Abstract

Poker is a challenging domain that contains both elements of chance and imperfect information. Though progress has been made in the domain, there is still one major stumbling block on the way to creating a world-class calibre computer player. This is the task of learning how an opponent plays (i.e., opponent modelling) and subsequently coming up with a counter-strategy that can exploit that information. The work in this thesis explores this task. A program is implemented that models the opponent through game play and then plans an exploitive counter-strategy using expectimax search. This program is evaluated using two different heads-up limit poker variations: a small-scale variation called Leduc Hold'em, and a full-scale one called Texas Hold'em.

Acknowledgements

I would like to thank my supervisor, Dr. Jonathan Schaeffer. He has always been there to guide the way for me and has been instrumental in helping me accomplish my goals.

I would also like to thank Aaron Davidson and Chris Pinchak. They offered invaluable feedback on my thesis drafts and were always there to listen to my ramblings and offer me encouragement when it was needed.

It has been a great pleasure being a member of the University of Alberta Poker Research Group. I have learned so much from all of you. I especially enjoyed the many good office conversations I had with Darse Billings, Dr. Michael Bowling, and Neil Burch.

Last, and definitely not least, I have to thank my family. They have always been extremely supportive and have always believed in me. They will always be my rock.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Artificial Intelligence and Games

Artificial Intelligence (A.I.) researchers have long used games as research testbeds. Games make excellent testbeds because they have well-defined rules and goals that are easy to represent. Because of these properties, games provide researchers with a concrete problem framework that they can use to explore ideas and easily quantify their results. In fact, because games are used so often as testbeds in A.I. research, they are often called the *drosophila* of A.I.. That is, games are to A.I. research what the fruit fly is to genetics research.

Games that are of interest to researchers are often categorized based on the accessibility of the game state to the players. If all players in a game have complete knowledge of the entire game state, it is called a game of *perfect information.* Otherwise, it is a game of *imperfect information.* Chess, checkers, go, and backgammon are all games of perfect information because all the players have access to the entire game state by just looking at the board. On the other hand, poker, bridge, and Scrabble are examples of imperfect information games because each player has their own cards or tiles that are kept hidden from the other players in the game.

Both perfect and imperfect information games can be further categorized based on whether or not they contain *chance* elements. A chance element, such as the roll of dice in backgammon or the deal of cards in poker, is an event that introduces randomness or nondeterminism into a game. If a game contains elements of chance, it is referred to as a *stochastic* game. Otherwise, it is refereed to as a *deterministic* game.

In the past, most games-related A.I. research focused on decision-making in deterministic perfect information games. Computationally speaking, these games are the easiest to deal with[30] and this research focused mostly on studying, enhancing, and applying brute-force minimax-based search algorithms. This research made outstanding progress and resulted in many programs achieving a level of playing strength that is comparable to or superior to the best humans (e.g. Deep Blue in chess[13], Chinook in checkers[42], and Logistello in Othello[12]).

This research, however, has been criticized for working in an over-simplified domain. The main point of this criticism is that this research has little carry-over value to the much larger class of problems, such as those seen in the real world, that are stochastic and/or have imperfect information.

In recent years, A.I. researchers have addressed this criticism by branching out and focusing on decision-making in domains that contain elements of chance and/or imperfect information. Again, games are an excellent domain for exploring these issues. From the computational decision-making point of view, these games promise great research potential because they offer many new and interesting challenges not present in traditional deterministic perfect information games.

This newer direction of research has already made some great progress both with the development and implementation of various novel A.I.-related ideas. For example,

- Gerry Tesauro successfully tackled the stochastic element of backgammon and his program, *TD-Gammon*, achieved world-class playing strength[54, 55].
- Brian Sheppard's Scrabble program, *Maven*, plays better than the best humans[45].
- Matt Ginsberg's bridge program, *GIB*, reached a strong level of play at the declarer aspect of the game[22, 23].
- The University of Alberta Poker Research Group recently applied a game-theoretic approach to full-scale two-player Texas Hold'em poker creating a program that was competitive with a world-class player[6].

Following this newer direction of A.I. research using games, this thesis continues the University of Alberta Poker Research Group effort of discovering new computer-based decision-making techniques that can be applied to poker[3, 10, 37, 11, 9, 38, 41, 18, 7, 17, 6, 8, 51]. The goal of this research is to eventually discover and implement techniques that enable a poker program to perform at a level that exceeds that of the best humans in the world. The bigger picture, however, is to gain new insights into reasoning with imperfect information in a stochastic domain which is essential when tackling most real-world problems.

1.2 Poker as a Testbed

Poker is a card game that proceeds in *stages*. In each stage, *active* players are dealt cards (some of which are dealt face up for all players to see and some of which are dealt face down to each player to keep private). Players then wager against each other that their *hand* of cards is the best (or will be the best at the end of the game). Because each player knows only their cards and not all their opponents', poker is an example of an imperfect information domain. In addition, because players have no control over which cards are dealt, poker is also an example of a stochastic domain.

To remain active in the game and therefore to proceed through the stages of play, a player must match the amount of money wagered by each of their opponents. If one player makes a wager that no opponent matches, then the game ends immediately with that player winning all the wagered money (the *pot*) regardless of their cards. Otherwise, play continues until there are no stages left in the game and at that point all the active players enter into a *showdown*. During the showdown, all active players reveal their hidden cards and the player with the highest ranked poker hand[7] wins the pot (equally ranked hands split the pot).

Since poker is a gambling game where the goal of a rational player is to maximize their total profit for as long as they play, players typically play in *sessions* of games against the same players. This repeated interaction gives players the opportunity to learn their opponents' playing strategies with the hopes that they can use this knowledge to exploit weaknesses in their opponents' play to maximize their profit. As a result, any decision in any game has the potential to affect a player's overall success rate.

All of the above characteristics make poker a very challenging game that is strategically very complex despite its simple rules. These characteristics make poker an excellent testbed for many important A.I.-related concepts:

- **decision-theory and probabilistic reasoning -** The randomness from the deal of cards and the lack of knowledge of the opponents' cards make poker a noisy and uncertain domain. This forces a player to be adept at both decision-theory and probabilistic reasoning.

- **risk assessment -** Poker is played in stages containing both the deal of additional cards and wagering. To be able to handle this type of domain, a successful player must be able to assess risks.

- **opponent modelling -** In poker there are repeated games against the same adaptive adversaries. This type of environment allows a player to use opponent modelling to learn their opponents' playing strategies and attempt to exploit this knowledge to increase their profit.

Because poker provides an excellent testbed for all these A.I.-related concepts, there is potential for poker research to carry over into other A.I.-related problems, such as:

- **user modelling -** Opponent modelling is a form of user modelling. User modelling is a popular focus of current A.I. research. In typical user modelling research, a modeller observes a user to try to identify patterns of interest in their behavior. The identified patterns can then be used to customize an application to take advantage of the identified user preferences. For example, many online websites use user modelling to observe user browsing patterns so that they can customize their advertisements to target their users' interests.

- **policy-making or negotiation agents -** Game theorists such as John Von Neumann and John Nash long ago realized poker could be used to illustrate fundamental principles of game theory that have subsequently been applied to a variety of fields including law, politics, economics, and the military. The development of automated decision-making in poker could lead to advancements in creating tools for use in these domains.
- **online auction agents -** Poker is a similar domain to online auctions, so it is natural to believe that research into building a successful poker player might carry over into novel ideas for building successful online auction agents.

1.3 Texas Hold'em

The research in this thesis is applied to the specific variation of poker called *Texas Hold'em.* Texas Hold'em is the poker variation used to determine the world champion each year at the World Series of Poker and it is one of the most popular versions of poker played in casinos. It has a higher skill-to-luck ratio than other poker variations and is also one of the most strategically complex. Texas Hold'em is typically played with two to ten players.

In Texas Hold'em, play proceeds in stages where cards are dealt and then a round of betting occurs. In a betting round, each active player (proceeding clock-wise around the table starting at the first active player to the left of the dealer) gets a chance to make a betting decision to wager that they have the best poker hand. A betting decision has to take the form of one of the following three actions:

fold - If a player folds, they are announcing that they are quitting the hand and relinquishing any chance at winning the pot.

check/call - If a player checks or calls, they are performing the logically equivalent action of staying active in a hand by matching the wagers previously made by each of their opponents. If no money needs to be put into the pot at the time of the action, then this action is referred to as a 'check'. Otherwise, it is referred to as a 'call'.

bet/raise - If a player bets or raises, they first match their opponents' wagers, and then they increase the betting level by making their own additional wager. Again, a bet and a raise are logically equivalent in that they both raise the betting level, but the term 'bet' is used to denote the action when it costs no money to match opponent wagers and the term 'raise' is used otherwise. Typically, the number of bets/raises in a betting round are limited to some finite maximum number (in casinos, the maximum is typically four).

In Texas Hold'em the size of a bet allowed depends on the particular variation of the game being played. In *limit* Texas Hold'em, the size of the bet is fixed in each betting round such that a bet/raise in the first two rounds is always the size of a *small bet* and a bet/raise in the last two betting rounds is always the size of a *big bet*. The size of a small bet and big bet are determined and fixed before any game is started. For example in $10-$20 limit Texas Hold'em, the small bet is fixed at $10 and the big bet is fixed at $20 and thus any bet/raise in the first two rounds has to be $10 and any bet/raise in the last two rounds has to be $20. There are more complicated betting variations for Texas Hold'em such as *pot-limit* or *no-limit* where players are given more flexibility in choosing their bet sizes, but the research in this thesis focuses solely on the limit version.

In Texas Hold'em play proceeds sequentially through four stages: *preflop*, *flop*, *turn*, *river*.

1.3.1 Preflop

The first stage in a game of Texas Hold'em is called the *preflop*. Here, the game starts with the two players immediately to the left of the dealer posting forced bets called *blinds*. A blind is a form of *ante* and it is used to stimulate betting by making sure that each game starts with some money in the pot for players to win.

Once the blinds are posted, the dealer deals two *hole* cards face down to each player that are to be kept private from their opponents. After these cards are dealt, a round of betting ensues starting with the player immediately to the left of the players that posted the blinds.

1.3.2 Flop

The second stage in Texas Hold'em is called the *flop*. Here, the dealer first deals three *community cards* face up on the table. Community cards are dealt face up so that they are visible to all players. These cards can be used by all active players to make their five-card poker hands. After these cards are dealt, all active players go through a round of betting.

1.3.3 Turn

The third stage in Texas Hold'em is called the *turn*. Here, the dealer deals one more community card face up and the active players participate in another round of betting.

1.3.4 River

The fourth and final stage in a game of Texas Hold'em is called the *river*. Here, the dealer again deals one community card face up and another round of betting ensues. After this betting round, if there are at least two active players remaining, then those players enter into a *showdown*. In the showdown, the active players reveal their two hole cards to determine the winner of the pot. The winner is the player that has the best five card hand using any combination of their two hole cards and the five community cards according to standard poker hand rankings[7]. If more than one player ties for the best hand, the pot is split equally among the tied winners.

If another game is to be played in the session, the player immediately to the left of the current dealer becomes the dealer for the next game, and a new game is started and played as described above.

1.4 State of the Art

The current state of the art Texas Hold'em computer playing programs are the University of Alberta Poker Research Group's *Poki*[17] and *PsOpti*[6]. Both of these programs have had varying degrees of success, but neither player has reached a level of world-class playing strength.

1.4.1 Poki

Poki is a program designed to play *full-ring* (i.e. 10-player) limit Texas Hold'em. It has been a consistent winner against human competition in "play-money" games on both the Internet Relay Chat (IRC) poker channel and on the University of Alberta Poker Research Group's own poker server[17]. In full-ring games, Poki is believed to play at an intermediate level of playing strength[17].

However, in games with few opponents, Poki's playing strength decreases. Poki's main problem is that it cannot adapt its strategy fast enough to exploit its opponents or prevent its own exploitation. As the number of opponents in a game decreases, the success rate of "tricky" plays (like bluffs) increases. This allows stronger opponents the opportunity to change their strategy to exploit their weaker and slower-adapting opponents. As a result, even though Poki is an intermediate player in full-ring games, it plays *heads-up* (i.e. two-player) Texas Hold'em weakly.

1.4.2 PsOpti

PsOpti is a program designed to play only heads-up limit Texas Hold'em. Its playing strength is considered to be at an advanced level. PsOpti was built using a game-theoretic approach (i.e. building on the ideas of an equilibrium strategy for both players) for an abstract form of poker to make the problem tractable.

In all experiments, PsOpti performed well and even held its own against world-class competition for 7,000 hands (which is small by computer standards, but long by human standards). However, given enough time, strong competition can still eventually find PsOpti's weaknesses and exploit them[6]. The experiments conducted on PsOpti confirmed the presence of two problems inherent to programs that are designed and built using a "pseudo-optimal" game-theoretic approach:

- In pseudo-optimal game-theoretic solutions, there are weaknesses due to approximations and those weaknesses are permanent. This means that once an opponent discovers the program's weaknesses, those weaknesses can be exploited forever.
- Because game-theoretic solutions are non-exploitive, strong players can adopt a style of play that probes for weaknesses without the fear that game-theoretic programs will punish them while they perform this (usually very predictable) activity.

1.4.3 Obstacles to World-class Play

Billings *et al*[7] identified the following attributes that are required for successful poker play: *hand strength*, *hand potential*, *bluffing*, *unpredictability*, and *opponent modelling*. Of these required attributes, the one that remains the biggest obstacle to world-class play is opponent modelling. This was echoed by the world-class opponent, Gautam Rao, who after testing PsOpti said [6]:

> You have a very strong program. Once you add opponent modeling to it, it will kill everyone.

1.5 Thesis Contributions

The work in this thesis focuses solely on the domain of heads-up limit Texas Hold'em. Computationally speaking, the two-player game of Texas Hold'em is more tractable than one with more players, but at the same time it is more strategically complex. In heads-up poker, players can adopt tricky playing styles in order to confuse and exploit their opponent. World-class players have to be deceptive so that their opponent cannot exploit them and at the same time look for opportunities to exploit their opponent.

The main goal of this thesis is to explore techniques to improve both opponent modelling, and decision-making to exploit an opponent based on opponent modelling information. By exploring possible techniques for doing this and implementing the best ones, this thesis advances the state of the art in computer poker playing by creating a fully dynamic adaptive program that tailors its play to exploit its opponent's weaknesses.

All previous attempts at creating poker programs were weak at doing this and this work intends to provide the next "stepping stone" on the road to creating a world-class caliber computer poker playing program.

The remaining structure of the thesis is as follows:

1. Chapter 2 discusses past attempts by other researchers to build poker programs. It is intended to provide a setting which can be used to illustrate the novelty and advantages to the design and implementation of the program described in this thesis.

2. Chapter 3 presents a search-based action-selection procedure that exploits opponent modelling information.

3. Chapter 4 looks at the problem of opponent modelling in poker as it relates to the search-based action-selection decision-making described in Chapter 3.

4. Chapter 5 presents the experimental results of the implementation presented in this thesis and tries to quantify its level of performance.

5. Chapter 6 discusses future work and conclusions.

Chapter 2

Related Work

A.I.-related games research on perfect information games such as chess has made great progress over the years to the point that the problem is understood well enough that there is now a well-defined action-selection framework for tackling similar problems (alpha-beta search). The same cannot be said for imperfect information games such as poker. The technology for understanding a game like poker that is fundamentally based on imperfect information is less mature and researchers are still working towards developing an effective action-selection framework there.

In the 1970's, Nicholas Findler made the first major effort to build a poker program[21]. His program was designed to play 5-card draw poker and he built this program as a testbed to explore computer models of human cognitive processes in the domain. This program was reportedly able to learn, but it did not achieve a strong level of play despite the fact that 5-card draw poker is not as strategically complex as other forms of poker (such as Texas Hold'em).

The next effort may have been in 1984 with Mike Caro's heads-up no-limit Texas Hold'em program *Orac* (Caro spelled backwards). Orac reportedly played a few short exhibition matches against strong players. Unfortunately, there is no good documentation available detailing either the program's technical details or the match results. From a scientific point of view, none of the results were statistically significant. This makes it difficult to critically assess both its success and its technical innovation.

Aside from these two efforts, there have also been hobbyist efforts, such as Greg Wohletz's *R00lbot* that used to play on Internet Relay Chat (IRC), and commercial efforts, such as *Turbo Texas Hold'em*[50]. All of these known efforts probably play at best at an intermediate level of strength and are far from a world-class level of play.

In recent years, computer science researchers have used poker to explore such things as Bayesian decision making[31, 51], game-theoretic analysis[29, 30, 43, 53, 46, 6], reinforcement learning[16], and opponent modelling[31, 17]. This research is summarized here with the intention of illustrating how it relates to creating an effective poker betting strategy. In order to present a structured presentation of this related work, the research is categorized into four approaches based on the type of betting strategy used in the various programs. These approaches are: *heuristic-based*, *simulation-based*, *game-theoretic*, and *heuristic search-based*.

2.1 Heuristic-Based Approach

In a heuristic-based approach to playing poker, a rule-based expert system is used to make a poker betting decision. To make a decision, the expert system first abstracts a complicated game state into some simplified scenario that has a *heuristic* (or multiple weighted-heuristics) associated with it and then it uses that heuristic information to make a decision.

2.1.1 Heuristics in Poker Literature

A heuristic-based approach to playing poker is a crude but possibly effective strategy that is appealing to beginning players because it gives them an easy-to-use starting point. Poker authors have capitalized on this appeal and over the years have generated a wealth of poker literature[47, 48] that aims to help players learn the game. This literature mainly works by describing common game scenarios along with associated heuristics that can be used to help with decision-making in them. For example, most literature contains heuristics for helping a player with:

preflop hand selection - used to help decide which hands are playable in the preflop stage (i.e. *Sklansky and Malmuth's preflop hand groups*[48]).

action-selection - used to help a player decide whether to fold, call, or raise (i.e. *pot odds*, *free card danger*).

deliberate hand misrepresentation - used to help a player disguise their hand (i.e. *bluffing*, *slow-playing*, *check-raising*).

hand reading - used to help a player infer their opponent's private cards based on their actions.

2.1.2 Heuristic Betting Strategies in Programs

The easy-to-use nature of the heuristic-based approach provides a natural starting point for building poker programs. Generally, in this approach, the heuristics or *ad hoc* rules in these programs use various game information, such as a player's table position and betting history, and various computed information, such as their hand strength and hand potential, to generate a *probability triple* that is used to select a fold, call, or raise action.

There are many ways to create the heuristics used to govern betting strategies. For example, they can be knowledge-engineered by a domain expert or they can be derived and refined through trial and error. A heuristic-based approach is problematic because there are so many different decision scenarios and over time the system quickly becomes hard to maintain and test. Despite these problems, the perceived simplicity of this approach has resulted in several implementations.

In the 1970's, Waterman[57] attempted to learn heuristics that were represented as *production rules* to define a betting system for *5-card draw* poker. He reported that this system achieved roughly about the same level of skill as an experienced human player.[1] Smith[49] later reinvestigated the same problem and attempted to learn heuristics through the application of a *genetic algorithm.* He compared his results to Waterman's and reported that comparable performance was achieved despite using less domain knowledge.

Korb *et al*[31] created a computer player for heads-up *5-card stud* poker. Their player used a Bayesian network that incorporated opponent information obtained through playing experience to provide their heuristic betting strategy with a better estimate of the chance of winning a hand. They report that this player won against their two simplistic test opponent programs and lost against an experienced amateur poker player (though the loss was reported not to be statistically significant).

The University of Alberta Poker Research Group's programs, such as *Loki*[37] and the formula-based version of *Poki*[17], also used a heuristic-based betting strategy that incorporated opponent modelling information. In these two programs, opponent modelling information was used to increase the accuracy of the computed *hand strength* and *hand potential* values used by their expert-defined formula-based betting strategy. These programs were consistent winners in the full-ring Texas Hold'em games for which they were designed to play and they reached about an intermediate level in playing strength. Unfortunately, the consensus reached among the researchers responsible for these programs was that programs built using this approach would never scale to world-class play[7]. The main reason for this decision was that they believed that poker simply has too many different decision scenarios to be sufficiently handled by any knowledge-engineered heuristic-based betting strategy. In addition, these programs became difficult to test and debug because they needed a poker expert to identify mistakes with the betting strategy and suggest how to modify it.

2.2 Simulation-Based Approach

In games of imperfect information like poker each player does not know their opponents' cards. This makes it difficult for players to decide which actions will be the most profitable. To overcome this information gap and figure out which action to choose, a player could

[1] 5-card draw poker is considered less strategically complex than Texas Hold'em.

make guesses about their opponents' cards and then forecast possible ways the hand could play out to assess the profitability of their chosen action. This technique is the main idea behind a simulation-based approach to playing poker.

In the simulation-based approach to playing poker, a program chooses among actions[2] at a decision point by running *simulations* for each action. In each simulation a hypothesized action is taken and then future play for all players is simulated through to completion of the hand to see how profitable the action is. Since each simulation is subject to statistical variance, many simulations are performed (often hundreds or thousands) and their results are averaged to estimate an expected value (EV) for each action. Once EVs are obtained for each action, the program can use this information to create a betting strategy for the decision point. For example, the action with the highest EV could be chosen, or a probability distribution could be used to stochastically select actions.

The quality of the EVs obtained from the simulations depends, of course, on the quality of the simulated play. This simulated play, in turn, depends on which cards the opponent is assigned in the simulation and how they play out the hand. Therefore, to generate accurate simulated play, the simulation needs to assign cards to opponents that are consistent with the betting sequence leading up to the decision point. That is, if an opponent only raises with strong cards and they raised in the play leading up to the decision point, then they should be assigned cards that tend to be stronger rather than weaker.

This simulation-based approach to choosing actions in poker is related to the very successful approach of heuristic-search based action-selection (e.g. minimax search with alpha-beta pruning) in perfect information games. Both approaches can be thought of as looking forward in a game tree to identify the most profitable moves. In the simulation-based approach the game tree is explored by following many separate paths that each extend from a decision point all the way to the leaves of the game tree. In contrast, heuristic-search based approaches basically explore the game-tree completely down to some specified depth from the decision point.

The simulation-based approach to poker was used in both Greg Wohletz's *R00lbot* that used to play on Internet Relay Chat (IRC), and the simulation-based version of Poki[17].

Despite the fact that simulations have been used successfully in other imperfect information games such as Scrabble[45] and bridge[22, 23], the simulation-based version of Poki did not result in a substantial performance improvement over its formula-based counterpart[17]. This result is unfortunate because simulations have a couple of advantages over the heuristic-based approach, namely:

- the automatic discovery of a dynamic betting strategy that can automatically adjust

[2] In poker these are the checking/calling and betting/raising actions, but not the folding action because the result of that action is fully known.

to different game conditions, and

- a simple uniform framework that does not rely on heavy knowledge-engineering obtained from a domain expert.

The main reason Poki's simulations did not perform as well as hoped was attributed to the method being dependent on the difficult problem of simulating the opponents' future play in a hand. Also, it was not robust to all the noise and inaccuracies present in trying to do simulations[17].

2.3 Game-Theoretic Approach

Game Theory is a branch of mathematics and economics that is devoted to the analysis of games. Since many real-world decision problems can be modelled as games, game theory has become a powerful tool for understanding various decision-making scenarios involving law, politics, economics, and the military.

Poker has had a long standing connection to game theory. In fact, pioneering game theorists, including John Von Neumann, John Nash and Harold Kuhn, used simplified versions of poker to illustrate what became fundamental principles in their field [56, 32, 36, 35].

In the game-theoretic approach to building a poker program, the specific poker game of interest, in this case two-player Texas Hold'em, is analyzed to find a set of strategies, one for each player, that form a *Nash equilibrium.* A set of strategies are said to be in Nash equilibrium if no player can do better by unilaterally deviating from their strategy. Computing these so-called *game-theoretic optimal* strategies and using them to play is attractive for two reasons:

- the program has a known strategy for achieving the best possible results against its worst-case adversary (i.e. an adversary employing a *best-response* strategy), and

- the program's strategy can be fully known by an opponent and despite this the opponent cannot gain any advantage from this knowledge.

The above two statements imply that a poker program that plays a game-theoretic optimal strategy will be guaranteed to at least break even in the long run no matter what opponent it faces, and it also cannot be exploited to lose money even as its opponents learn its strategy.

Unfortunately though, computing the game-theoretic strategies for a full scale poker variation is almost certainly intractable due to the size of the problem[30].[3] As a result, anybody using this approach will have to settle for approximations which, of course, cannot guarantee the two attractive properties listed above.

[3] For example, the two-player Texas Hold'em imperfect information game tree contains more than 10^{18} nodes[7].

To compute approximate game-theoretic optimal strategies for poker one could look at abstracting the problem and computing an exact solution (for example, via linear programming) for the smaller abstract problem. This approach has been explored for poker by Takusagawa[53], Shi and Littman[46], and the University of Alberta Poker Group[6]. Dahl[16] also explored learning game-theoretic optimal strategies in a simplified game of poker by using reinforcement learning.

Finding approximate game-theoretic optimal strategies could still result in successful play, but the main problem with taking the game-theoretic approach and settling for approximations is that the final result will be a fixed strategy which can theoretically be exploited. Of course, the extent to which the strategy can be exploited in practice depends on whether its weaknesses can be discovered in all the statistical noise and uncertainty that is associated with poker.

It is important to note that game-theoretic optimal strategies do not punish exploitable play no matter how obviously exploitable that play is. An exploitive or *maximal* player, on the other hand, could be made to recognize an opponent's mistakes and take advantage of them. Of course, by trying to exploit an opponent with the intention of increasing their overall profit, a player risks opening themselves up to exploitation. Game-theoretic optimal strategies side step both this risk and this potential extra profit. Intuitively, they are defensive and built to not lose by guaranteeing to at least break even in the long run.

The most successful application of game-theoretic techniques to create a poker program has been the University of Alberta Poker Research Group's PsOpti[6]. PsOpti was built to play two-player Texas Hold'em and plays it at an advanced level of playing strength. In tests against world-class opposition, it held its own for quite awhile before the world-class opposition learned to beat it. Unfortunately, because PsOpti employs a fixed strategy, once its weaknesses are discovered they can be permanently exploited. To address this problem, PsOpti's strategy would have to be recomputed so that it is a better approximation, or alternatively, it would need to be augmented with some dynamic capability that recognizes and patches a weakness an opponent is exploiting.

Research needs to be done to understand how to create a dynamically adapting program that is capable of both exploiting an opponent and preventing itself from being exploited. This knowledge would be useful to create a poker playing program on its own, or to be able to extend a game-theoretic based program so that it can fix its own weaknesses as an opponent discovers and exploits them.

2.4 Heuristic Search Based Approach

The heuristic search-based approach to action-selection tries to plan a betting strategy stemming from a particular game state by performing *backward induction* on the game tree rooted at that game state. Given a particular game state where an action needs to be chosen, this approach works as follows:

- Recursively consider taking each possible action and getting to a new resultant game state until the end of the game is reached.
- At the end of the game, the result of the game is known (i.e. win, loss, draw, etc.).
- Since the action taken leading to that end game state is known, the value of taking that action is then simply the value of that end game state.
- Once all the actions available to a particular player have their values calculated, assumptions are made about which actions that player will choose. These assumptions represent their betting strategy and are used to calculate the value of the decision point itself so that the process can be repeated for earlier decisions.

2.4.1 Minimax Search

One example of this approach is the minimax algorithm with the alpha-beta enhancement. This algorithm has proven itself in the past as a successful means of action-selection in perfect information games such as chess, checkers, and Othello. For the purposes of the following discussion, the alpha-beta enhancement will be ignored since it is not critical to the point being made here and instead the focus will be on the underlying minimax part of the algorithm.

The minimax algorithm's name is based upon the particular assumptions made for each player in the backward induction process:

- The program invoking the algorithm is assumed to want to choose its best available action. To do this, it will only ever choose the highest valued actions. As a result of this action preference, this player could be labelled the "max" player which contributes the "max" part to the algorithm's name.
- The opponent is also assumed to want to choose their best available action. In two-player zero-sum games, this is equivalent to the opponent choosing the action which is the worst for its opponent (which in this case is the program invoking the algorithm). From the perspective of the program invoking the algorithm then, the opponent will only ever choose the lowest valued actions. Because of this action preference, the opponent can be labelled as the "min" player which contributes the "min" part to the algorithm's name.

These *min* and *max* player action-selection assumptions are nice because the resulting minimax algorithm (assuming a full-depth full-width search) computes game-theoretic optimal strategies (i.e. strategies in a best-response equilibrium) in perfect information games.

As mentioned in the previous section, these strategies guarantee a player the best possible result against a worst-case opponent and also mean its opponent cannot exploit its strategy even after learning it.

Games such as chess, checkers, and Othello are unfortunately too large to completely search to the end of the game. Instead, in large games like these, a game is searched as deeply as possible in the time constraints allowed and an *evaluation function* is used to approximate the value of each particular state where the search is stopped early (i.e. the value that would have been backed up for that state had the subtree rooted there been able to be searched all the way to its leaves). This means these programs are likely not playing perfect game-theoretic optimal strategies due to errors introduced by the evaluation function, but nonetheless they are still very successful.

One of the great strengths of using heuristic search for action-selection is that it selectively focuses a program's computation and memory resources only on the decision at hand. It would be desirable to do something analogous in poker.

The addition of stochastic elements can make games substantially larger but their presence does not require abandoning either of the min and max player assumptions. Techniques for doing search in perfect information stochastic games include *expectiminimax*[40] and **-Minimax*[2, 24], which adds pruning to make the search more efficient.

Unfortunately, the minimax algorithm, itself, cannot be used in poker because of the imperfect information caused by the hidden cards. To determine the best action for the players within the search, the likelihood of the cards they could hold must be known. The actions they have taken to get to the decision point where their action must be chosen determine the distribution of cards they could hold. These actions though depend on their strategy which is being determined within the search.

2.4.2 Expectimax Search

One way to incorporate heuristic search into action-selection for poker is to give up having to determine the best actions for the opponent within the search and instead assume that they will choose their actions on their own accord (i.e. their choice of actions is given *a priori* or can be learned by an opponent model).

This different problem formulation now results in a calculation that returns the best way of playing against a specific opponent and not a hypothetical worst-case opponent. In game-theoretic terms, this search process will find a *best-response* playing strategy for one player in response to a particular opponent and not a pair of *best-response equilibrium* playing strategies, one for each player, which guarantees each player some particular value for the game regardless of how their opponent plays.

This heuristic search procedure is typically called the *expectimax* algorithm[33, 40].

Again, as was the case with minimax, assumptions about each player's play are used to derive the algorithm's name:

- The program invoking the algorithm is still assumed to want to choose their best available action. Therefore, they choose only actions with the highest possible value amongst their choices. This action preference contributes the "max" part to the algorithm's name.
- The opponent is assumed to choose their actions according to some predetermined strategy. This assumption effectively turns an opponent decision into an element of chance where there is a probability distribution over their observed actions. As a result, the value of an opponent decision node is calculated as the *expected value* of their possible actions. That is, it is a sum of the values of the various actions available to them weighted by the probability that they will take each of those actions. This expected value calculation contributes the "expecti" part to the algorithm's name.

Using expectimax search and opponent modelling for decision-making in poker was first explored by Aaron Davidson[17]. The work presented in this thesis reimplements his work and extends it. Because the work in this thesis builds heavily upon that work, it will not be discussed here and instead will be discussed in the following chapters.

The overview of this approach presented here is meant to be only enough detail so that other researchers' more distant related work can be discussed and that this approach can be motivated. The remaining chapters of this thesis will provide a more detailed discussion on this approach.

2.4.3 Opponent Modelling in Expectimax Search

The idea of combining opponent modelling information with decision-making in games has been explored in the past. For example, different forms of opponent modelling have been incorporated into various forms of heuristic-search in perfect information games [14, 25, 28]. Opponent modelling has also been used before in poker playing programs. For example, various forms of opponent modelling were present in some of the different poker programs mentioned earlier such as heuristic-based Poki[17], simulation-based Poki[17], and Korb and Nicholson's Bayesian Poker Player[31].

For the interests of using an opponent model in conjunction with an expectimax-based heuristic search framework, the following work is the most relevant and will be summarized here.

Reibman and Ballard's *-Min Search

In their work[39], Reibman and Ballard modified the conventional minimax search procedure to use opponent modelling as a means to account for opponent fallibility. In their approach, conventional minimax backups are done at all nodes in the search tree except for the top-most level of opponent nodes stemming from the decision point. For these nodes the conventional min backup is replaced with an expected value backup where the conventional minimax values of the subtrees are weighted (by the predicted likelihood that the opponent will choose the action that leads to the subtree) and summed. This effectively turns these min nodes into chance nodes.[4]

To predict the probability of the opponent choosing each of their available actions, the opponent is assigned a *predicted strength* value that represents their playing strength. This value is then used as input into a fixed formula that generates an action probability distribution. The formula is designed in such a way that as the opponent's predicted strength decreases, their likelihood for choosing an action with an inferior minimax value increases. When the opponent is given the maximum possible predicted strength value, they only choose their best available action which is the same action chosen with conventional minimax search. When the opponent is given the minimum possible predicted strength value, they are equally likely to choose any available action.

Reibman and Ballard tested the performance of this approach on randomly generated perfect information game trees. According to Donkers[19], there are no known applications of this approach in a practical game setting.

Jansen's Probi-max Search

Jansen[26, 27] explored how play should change from default minimax behavior when a player is in a known losing chess position and facing an opponent known to be fallible. As part of this work, Jansen explored a search procedure where the values of all opponent nodes are calculated via expected value backups, rather than conventional min backups, and the values of the invoking program are calculated via conventional max backups. He called his search procedure "probi-max" search. To generate the probability distribution over opponent actions used in the expected value backups for opponent nodes, Jansen used fixed heuristic-rules that were meant to capture typical fallible play common at that particular decision point.

Sen and Aurora's Maximum Expected Utility Player

Sen and Aurora created a Maximum Expected Utility (MEU) player for perfect information games that exploits its opponents via a learned opponent model[44]. Their work is inspired by the decision-theoretic principle of Maximum Expected Utility[40].

Sen and Aurora's MEU player chooses its actions so as to maximize its expected utility. A probability distribution over opponent actions is used to represent the uncertainty about

[4] Chance nodes are traditionally denoted with asterisks, hence the work's name.

which actions the opponent will choose. Though they do not explicitly mention how their player chooses its actions so as to maximize its expected utility, there is a diagram that hints it uses the expectimax search procedure.

To learn the opponent model used in their experiments, they used a training phase where opponent moves were observed by the MEU player. A tally of move counts was kept in terms of the difference between the minimax value of the best move the opponent could have chosen and the minimax value of the value that the opponent did choose.

Their MEU player was tested using the two-player perfect information game of Connect Four. They presented their results by comparing their player's performance to a conventional minimax player's performance against a simple heuristic player known to have exploitable weaknesses.

PrOM Search

Probabilistic opponent-model (PrOM) search[19, 20] is an extension of opponent-model (OM) search[14, 25]. In OM search, opponent modelling in added to traditional minimax search by allowing the max player to know the min player's evaluation function (which can be different from the max player's). PrOM search extended this idea by allowing the max player to consider a distribution of n different opponent evaluation functions (called *opponent types*) rather than just one.

In PrOM search, for each min player decision, each of the opponent types is used to find the move they consider the best (i.e. the node which appears to give the minimum value to the max player from each opponent type's perspective). The values of these move choices are then weighted by the likelihood of each opponent type according to the max player's distribution of opponent types, and summed to compute the min node's value. Though each move for each opponent type is selected according to a min backup, the weighted sum used to compute the min node's value results in an expected value backup at min nodes.

The quality of PrOM search was evaluated both via theoretical analysis and experimental analysis. As part of the experimental analysis PrOM search was explored using the perfect information game, Lines of Action.

2.4.4 Lessons from RoShamBo

RoShamBo, sometimes called Rock Paper Scissors, is a game where two players simultaneously choose either rock, paper, or scissors and their respective choices result in either one player winning, or the two players tying. In this game, a player's success comes from their ability to "out guess" their opponent to make a winning choice.

In 1999 and 2000, Darse Billings conducted the First and Second International RoShamBo Programming Competitions[5, 4]. The results of these competitions illustrate many key points which are believed to carry over to poker and help justify the potential of an action-selection approach that is able to exploit a learned opponent model.

Determining an Opponent's Weaknesses

To choose the best action in a game of imperfect information, it is essential to first determine how an opponent is weak. In domains of perfect information, information present in the game itself is typically sufficient to identify an objectively correct move independent of the opponent's weaknesses. In domains of imperfect information, the correct move can often only be identified once the opponent's weaknesses are discovered. In RoShamBo, for example, you can only prefer one action over another once you have identified your opponent's own action preferences. This basic idea holds true in poker as well. If your opponent bluffs too much then you need to call more and fold less. If they do not bluff enough, you need to call less and fold more. If they fold too much, then you need to bluff more. If they call too much, you need to bluff less.

Earning as Much As Possible

In poker the goal of a player is to earn as much money as possible. For RoShamBo, this is the same as a program trying to win as many games as possible. Since opponents are often fallible and their weaknesses can be learned through repeated interaction, the most successful RoShamBo programs were the ones that were the best at adopting strategies that attacked their opponents' weaknesses. Overall, the results showed that the best programs were the ones that were able to accurately model their opponent, even if they were changing, so that they could better exploit them.

Defensive Nature of Game-Theoretic Approaches

A game-theoretic strategy sacrifices potential success by not attacking opponents' weaknesses to guard against its own exploitation. Since players in the competition were evaluated in terms of their total winnings, game-theoretic players finished in the middle of the pack because all of their matches were statistical ties regardless of how fallible their opponents were.

It is important to note that despite not being a successful strategy for winning the competition, the defensive characteristic of a game-theoretic strategy still proved useful as a safe defensive strategy that opponents could use to fall back on to stop losing to a better opponent.

Chapter 3

Expectimax Search for Action-selection in Poker

Aaron Davidson first explored using expectimax search for action-selection in poker[17].[1] In that work, he defined two different search procedures, *Miximax* and *Miximix*, which are basically variants of traditional expectimax search modified so that they can be applied to the imperfect information domain of Texas Hold'em. Davidson's work illustrated the potential for these search procedures, but never explored them in depth. The research in this thesis reimplements that work and extends it.

3.1 Imperfect Information Game Trees

Imperfect information games can be represented graphically as a tree whose nodes represent various states of the game and whose edges connecting the nodes represent the possible actions which cause a transition from a state to its successor. In game theory, this representation of a game is referred to as its *extensive form* representation and it provides a useful visualization of the game when doing game-theoretic analysis (i.e. when both players are considered simultaneously so as to assign them both strategies).

In an imperfect information game like poker, game state nodes can be categorized into who is choosing the action at that game state. In two-player Texas Hold'em there are *decision nodes* where each player chooses their actions, *chance nodes*[2] where some random process (often called *nature*) deals cards, and termination states called *leaf nodes* which indicate that the game is over and there are no further actions allowed.

The one important thing that separates imperfect information games from perfect information games is the presence of hidden information. The presence of hidden information is important because it means that for each player, different action nodes become *indistin-*

[1] The concepts underlying the work were conceived jointly by Aaron Davidson and Darse Billings.

[2] Sometimes called *stage nodes* since the betting in Texas Hold'em is split into stages where cards are dealt before any betting occurs.

guishable and the player acting there is required to act the same way in all of them. In game theory, a group of indistinguishable action nodes is referred to as an *information set.*

In Texas Hold'em, the hidden information takes the form of an opponent's private hole cards. This means that in Texas Hold'em, all of a player's action nodes that share both the same public information (i.e. the betting actions and community cards) and the same private information (i.e. the player's hole cards) but differ solely because of the opponent's private hidden information (i.e. the opponent's hole cards) belong to the same information set.

In the extensive form representation of the game, an information set appears as an oval grouping together a set of player action nodes. The use of information sets is not needed in perfect information games because the absence of hidden information means that each information set contains exactly one node.

3.2 Expectimax Search Tree

The tree traversed during expectimax search arises by looking at the imperfect information game tree solely from the perspective of choosing the best strategy for one player (i.e. the best-response player) while assuming some other predetermined strategy for the opponent.[3]

When considering the game from one player's perspective only, the fact that the player has to act the same way in each node of their information sets allows parts of the imperfect information game tree to be merged together into the expectimax tree. More specifically, all of the nodes making up each of that player's information sets and the subtrees rooted at those nodes can be merged together according to which information is observable and not hidden from that player.

After this merging process, each of the new expectimax tree's nodes can be thought of as a single representative for a group of nodes in the imperfect information game tree that are indistinguishable to the player. Likewise, the edges connecting the nodes in the new tree are a single representative of a group of edges that are also indistinguishable.

The effect of merging all indistinguishable nodes and actions into single representatives in the expectimax tree is that at each node in the expectimax tree there is an implicit probability distribution over the information that occurred but was kept hidden and remains unknown.

3.3 Example Kuhn Poker Trees

To illustrate the differences between the trees, an example of each tree is shown for the game of Kuhn Poker. Kuhn Poker is used because it is small enough to present graphically

[3] Rather than from the perspective of simultaneously choosing the best strategies for both players, as would be done in game-theoretic analysis.

and at the same time allows the examples to be focused on poker.

3.3.1 Kuhn Poker Rules

Kuhn poker is described in some detail by Koller and Pfeffer[30], but the essential details are as follows:

- Two players, each of whom has two dollars.
- 3 card deck - Jack (J), Queen (Q), King (K).
- Each player antes one dollar and is dealt one card that remains hidden from the opponent.
- Each player can then make standard poker betting decisions as permissible by their *stack* size (i.e. there is a maximum of one bet since each player only has one dollar remaining in their stack after the ante): *bet* (increase the stakes), *check/call* (stay in the hand), or *fold* (quit and lose all money invested up to that point).

3.3.2 Kuhn Poker Imperfect Information Game Tree

Figure 3.1 illustrates a section of the Kuhn poker imperfect information game tree where the first player is dealt a Queen.[4]

The root node of this tree is a chance node where each player is dealt their hidden cards. The chance node is represented graphically as an octagon labelled with a '*'. The edges coming from this chance node represent the possible deals that can occur. The two edges shown in the image correspond to the only two possible situations where player 1 is dealt a Queen. The leftmost edge, labelled as "Queen, Jack", represents the situation where player 1 is dealt a Queen and player 2 is dealt a Jack. The rightmost edge, labelled as "Queen, King", represents player 1 being dealt a Queen and player 2 being dealt a King.

Below each of these two edges representing the dealt cards are subtrees of the possible legal betting actions. In each betting subtree, player 1's decision nodes are denoted by circles and player 2's decision nodes are denoted by squares. At the leaf node at the end of each path through the betting tree, called a *betting sequence*, player 1's payoff is illustrated.[5]

In the betting subtree, the edges coming from each player's decision nodes are labelled with single characters denoting possible player actions. Checking is represented with a 'k', calling with a 'c', betting with a 'b', and folding with an 'f'. To make things more readable, player 1's actions are shown in lowercase, and player 2's actions are shown in uppercase.

[4] To keep the size of the example small, the other possible deals that can occur in the game are not shown. This corresponds to four possible deals where player 1 and player 2 are respectively dealt a Jack and a Queen, a Jack and a King, a King and a Jack, and a King and a Queen.

[5] Since this game is *zero-sum*, player 2's payoff is simply the value of player 1's payoff negated.

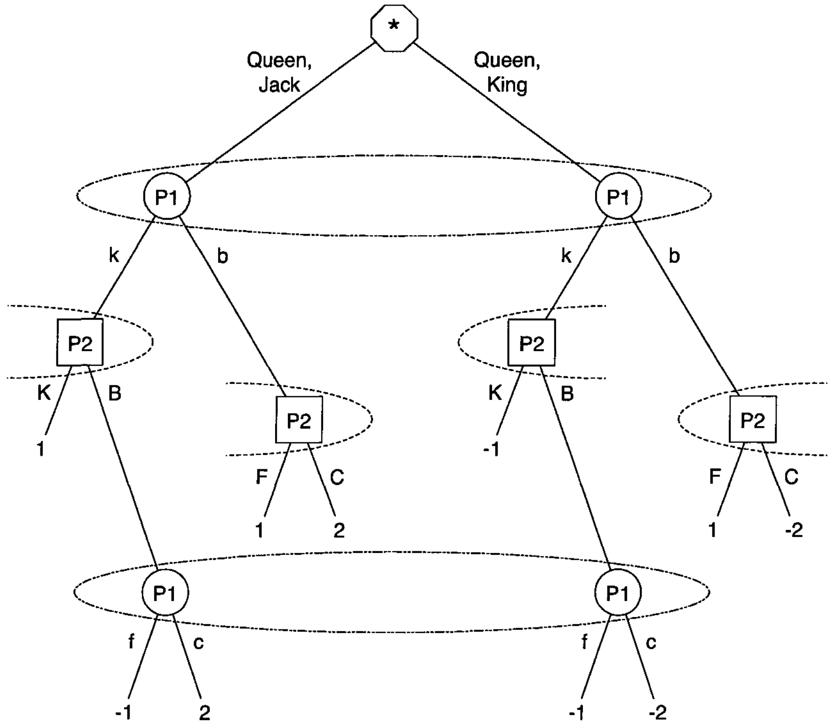

Figure 3.1: Kuhn Poker Imperfect Information Game Tree

The dot-dot-dashed ovals and the dashed ovals in Figure 3.1 illustrate player 1's and player 2's *information sets*, respectively. For example, player 1's first decision node in each betting subtree are grouped together because from player 1's perspective they appear the same; all player 1 knows is that player 2 is holding either a Jack or a King but they cannot distinguish which one. Likewise, player 2's subsequent decision nodes appear as half ovals because the other nodes that would appear in those information sets correspond to nodes in other subtrees that have been left off the diagram. For example, the leftmost player 2 decision node would be grouped with a corresponding decision node in the betting subtree where player 1 is dealt a King and player 2 is dealt a Jack.

3.3.3 Kuhn Poker Expectimax Search Tree

Figure 3.2 illustrates the expectimax search tree for player 1 when they are dealt a Queen and are considering their first action. Many of the nodes in the imperfect information game tree shown in Figure 3.1 are merged together into the expectimax tree. In fact, the two separate betting subtrees in the imperfect information game tree are effectively merged together into a single betting tree. Whereas the imperfect information game tree showed all of the opponent's hidden information explicitly even when it was indistinguishable to player 1, the expectimax search tree shown for player 1 only focuses on information it is allowed to observe.

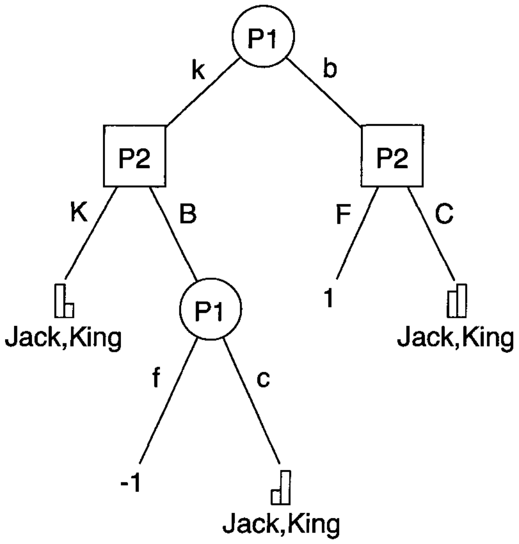

Figure 3.2: Kuhn Poker Expectimax Search Tree

The top-most player 1 decision node in the expectimax search tree merges both of the top-most player 1 decision nodes in the imperfect information game tree. The left-most player 2 decision node in the search tree represents both of the left-most player 2 information sets in each of the betting subtrees in the game tree. Likewise, the right-most player 2 decision node in the search tree represents both of the right-most player 2 information sets in each of the betting subtrees in the game tree. Finally, the bottom-most player 1 decision node in the search tree represents both of the bottom-most player 1 decision nodes in each of the betting subtrees of the game tree.

The effects of merging decision nodes is illustrated at the leaf nodes which go to a showdown. For example, the leftmost leaf node in this figure corresponds to the leftmost leaf nodes in each of two betting subtrees in the extensive form representation in Figure 3.1. In this image, player 1's payoff is no longer shown as a simple integer value and instead is replaced by a distribution representing the likelihood of player 2 holding either a Jack or a King. Showing this distribution rather than the player's payoff is done to illustrate the effects of merging different nodes in the imperfect information game tree into a single representative node in the expectimax tree. Player 1's payoff at this node in the expectimax tree depends on the likelihood of each of player 2's possible holdings (or equivalently, the likelihood of being in each of the different corresponding nodes in the imperfect information game tree that have been merged into the representative node). For example, assuming that player 2 is likely to have a Jack 80% of the time and a King the remaining 20% of the time they go to that particular showdown leaf node, then player 1 would receive a payoff of 1 for the 80% of the time player 2 holds a Jack and they would receive a payoff of -1 for the remaining 20% of the time that player 2 holds a King. For this example, this would result in an expected payoff of (80%)*(1) + (20%)*(-1) = 0.6. Depending on how this distribution changes, player 1's payoff at this leaf node in the expectimax tree falls somewhere in the range of 1 to -1 inclusive.

3.4 Backup Rules in Expectimax Search

The purpose of the expectimax search procedure is to construct an automated player's best-response playing strategy. To do this, the algorithm recursively backs up expected values (EVs) from child decision nodes to parent decision nodes using backup rules that correspond to how the decision-maker would make the decision at that point in the game tree. In the domain of two-player Texas Hold'em, there are three types of game tree decision nodes for which the search procedure needs to define backup rules:

- *opponent decision* - The EV of an opponent decision node is simply the sum of its children's EVs weighted by their probability of occurrence. The opponent's strategy dictates their choice of actions and as a result, the values of these probabilities are specific to it.

- *best-response (or, the program's) decision* - Since the best-response strategy attempts to maximize its EV, the value of a best-response decision node is simply the maximum value of its children's EVs. By recording the maximum valued actions that would be chosen at each decision point as it is reached in the search, a best-response strategy can be constructed.

- *chance event* - In addition to player decision nodes, games can have *chance nodes* which are used to represent random events, such as rolled dice or dealt cards. For these chance nodes, each branch originating from them represents a different possible random outcome and the weight of each branch represents the probability of that outcome occurring. As a result, the backup rule applied to a chance node is a standard expected value calculation where the backed up values of each subtree representing the occurrence of a random event are weighted by their probability of occurrence and then summed together.

3.5 Handling the Effects of the Opponent's Strategy

The opponent's strategy affects three things that have to be accounted for when applying expectimax search to action-selection in poker: leaf node evaluations, opponent action selection, and the probability of community cards being dealt.

3.5.1 Leaf Node Evaluations

To use expectimax search it is necessary to calculate the expected values (which are always defined from the best-response player's perspective since they invoked the search) of leaf nodes. It is important to remember when doing this that a leaf node in the expectimax search tree represents more than one node in the imperfect information game tree.

In poker, there are two types of leaf nodes: those resulting from one player folding, and those where neither player folds and a showdown occurs. For leaf nodes where one player folded, the expected value is simple to calculate: if the best-response player folds they lose all the money they wagered, and if their opponent folds the best-response player wins all the money their opponent wagered. Because a player's folding action is sufficient to determine the value of the leaf node (i.e., the payoffs at every node in the imperfect information game tree represented by this node in the expectimax game tree are exactly the same), it does not matter what distribution of imperfect information game tree nodes is actually represented at the expectimax search tree leaf node.

For leaf nodes that require a showdown, the expected value depends on the distribution of imperfect information game tree nodes represented by that node. At a showdown, each imperfect information game tree node would correspond to each of the possible hands the opponent would hold there. Since each of these leaf nodes has a payoff associated with them, once the distribution of these nodes is known the expected value of the expectimax leaf node can be computed via a weighted sum. An example of this calculation was shown earlier when the Kuhn Poker expectimax tree was described.

Figuring out the value of a showdown leaf node then depends on being able to figure out an opponent's distribution of possible holdings there. This task will be left to the opponent modelling system presented in the next chapter. To abstract this problem away to the opponent modelling system, the expectimax search formulation used here simply assumes that it can ask the opponent modelling system for the best-response player's chance of winning at a showdown[6] which can then be used to estimate an expected value for a showdown leaf node. Pseudocode for a leaf node evaluation is provided in Figure 3.3.

3.5.2 Opponent Action Selection

To perform the expected value backup at opponent decision nodes, the probability of the opponent taking each action at each of their expectimax decision nodes needs to be known. The values of these probabilities, of course, depend on their actual strategy. To handle this, it assumed that an opponent model used in this approach will be able to return a probability distribution over opponent actions at each expectimax opponent decision node.

[6] The chance of winning is the sum of the percentage of the time the player wins outright and half the percentage of the time that the player ties.

```
// Compute the EV at end of game
double getLeafEV(GameState gs) {
   if (!gs.isShowdown()) {
      if(gs.opponentFolded()) {
         pwin = 1.0; // opponent folded
      } else {
         pwin = 0.0; // decision-maker folded
      }
   else {
      // showdown - estimate pwin with opponent model
      pwin = oppModel.estWin(ourHand, gs);
   }

   return (gs.getSizeOfPot() * pwin) - gs.getOurSpent();
}
```

Figure 3.3: Leaf Node Evaluation

It is important to remember that each expectimax opponent decision node can represent more than one opponent information set in the imperfect information game tree. The opponent's strategy would dictate how they act in each of their information sets, and the observed opponent actions at an opponent decision node in the expectimax search tree would then appear as a weighted combination of what the opponent's strategy would do in each information set contained there.

The model is only assumed to be able to provide a probability distribution over what actions the best-response player will see the opponent take at each of their expectimax nodes. This allows the opponent modelling system the flexibility to choose between modelling the opponent's strategy and computing these values or modelling the observed values directly. Pseudocode for an expected value backup at an opponent decision node is shown in Figure 3.4.

3.5.3 Probability of Community Cards Being Dealt

A chance node in the imperfect information game tree is straight forward. Every possible chance outcome occurs with uniform probability (provided the game is fair), and all outcomes which are not possible due to previously dealt cards occur with zero probability.

Because of the way the expectimax tree merges all of the nodes in a best-response player's information set into a single representative, every chance node in the expectimax tree represents a group of chance nodes in the imperfect information game tree.

The distribution of the imperfect information game tree chance nodes merged into a single expectimax search tree representative corresponds exactly to the distribution of opponent hidden card holdings at that point in the game. This is because the opponent hidden card holdings are what defines the nodes in the best response player's information sets in

```
// Compute EV of an opponent decision node
double getOppDecisionEV(GameState gs) {
   // get legal actions
   List actions = getLegalActions(gs);

   for(int i = 0; i < actions.length; i++) {
      // take an action
      GameState temp = new GameState(gs);
      temp.takeAction(actions[i]);

      // use miximix to compute the child EV
      EVs[i] = miximix(temp);

      // estimate the probability of opponent taking action
      probs[i] = oppModel.getActionFreqs(gs, i);
   }

   // return weighted sum of children EVs
   return weightedSum(probs, EVs);
}
```

Figure 3.4: Opponent Decision Node EV Backup

the first place. This means the distribution of imperfect information game tree chance nodes represented by an expectimax search tree chance node depends on the opponent's strategy since their strategy determines the likelihood of each of their holdings at that point in the game.

As a result, the probability on each branch coming from an expectimax search tree chance node is a weighted sum of the probability of that branch for each imperfect information game tree chance node contained there. Intuitively, this means that the more likely the opponent is holding a specific card, the less likely that card will be dealt at that chance node, and vice versa.

To handle this effect of the opponent's strategy, the expectimax search routine presented here assumes that any opponent model used will be able to provide the probability associated with each expectimax chance node branch. Pseudocode for an expected value backup at a chance node is shown in Figure 3.5.

3.6 Miximax and Miximix Search

Two variants of expectimax search have been proposed for action-selection in poker[17]. The first, *Miximax*, corresponds exactly to the backup rules described above. That is, the value of an opponent decision node is computed via an expected value over all their available actions. The value of a decision node for the player invoking the search is defined to be the maximum value of all its available actions. Because the player invoking the search takes the

```
// Computes the EV of a chance node
double getStageEV(GameState gs) {
   // generate cards to deal
   List cards = generateCardsToDeal();

   for(int i = 0; i < cards.length; i++) {
      // update state with dealt board cards
      GameState temp = new GameState(gs);
      temp.updateBoard(cards[i]);

      // use miximix to compute the child EV
      EVs[i] = miximix(temp);

      // estimate chance event probability using opponent model
      probs[i] = oppModel.getChanceFreq(gs, cards[i]);
   }

   // return weighted sum of children EVs
   return weightedSum(probs, EVs);
}
```

Figure 3.5: Chance Node EV Backup

maximum action, the strategy that results from this expectimax search is a *best-response strategy* against the opponent. Pseudocode for a max backup for the player invoking the search is shown in Figure 3.6.

The second variant proposed, *Miximix*, keeps the opponent's backup calculation the same but allows the decision-maker invoking the search to choose their action according to a probability distribution, rather than requiring that player always choose a maximum-valued action. This means that the value of a decision node for the player invoking the search is now an expected value backup just like the one described for the opponent. A probabilistic approach to choosing actions is important in poker since this allows the decision-maker to be less predictable. Pseudocode for a mix backup for the player invoking the search is shown in Figure 3.7.

The implementation of expectimax search used in this thesis for action-selection in poker uses a Miximix variant which dynamically creates a probability distribution for the decision-maker by taking a *softmax*[52] over its available actions. That is, given the values of the actions available to it at a decision node, the player invoking the search uses those values to construct a probability distribution that is biased towards higher-valued actions but does not completely rule out choosing lower-valued actions. This probability distribution is then used to both select actions at the decision node and also to backup values there so that the backward induction process can continue up the tree.

The actual choice of how to construct this probability distribution is still an open research question and has a lot to do with how one wants to address the *exploration/exploitation*

```
// Computes the EV of the decision-maker's node (max backup)
double getMaxDecisionEV(GameState gs) {
   // get legal actions
   List actions = getLegalActions(gs);

   for(int i = 0; i < actions.length; i++) {
      // take an action
      GameState temp = new GameState(gs);
      temp.takeAction(actions[i]);

      // use miximix to compute the child EV
      EVs[i] = miximix(temp);
   }

   // return the EV of the maximum valued action
   return max(EVs[i]);
}
```

Figure 3.6: Decision-maker's Miximax EV Backup

tradeoff[52]. The more biased the distribution is towards maximum-valued actions, the more exploitive the player plays based on its current beliefs about the opponent. This implies that it is less likely to explore taking other actions which it currently believes to be worse just to try and discover if better alternatives exist. Intuitively, addressing the exploration/exploitation tradeoff gives a player the means to take a short-term loss to achieve a long-term gain.

For the implementation used in this thesis, the player's action distribution is constructed as a standard Gibbs, or Boltzmann, distribution[52]. This allows the player's exploration/exploitation tradeoff to be controlled by a single *temperature* parameter. This method is admittedly simple but it provides a simple mechanism for addressing the exploration/exploitation tradeoff while other research issues are being tackled first.

Pseudocode for the Miximax and Miximix algorithms is shown in Figure 3.8. It calls the pseudocode presented earlier in this chapter (i.e., Figure 3.3, Figure 3.4, Figure 3.5, Figure 3.6, Figure 3.7).

3.6.1 Example: Kuhn Poker Miximax Calculation

Figure 3.9 illustrates an example miximax calculation for the situation in Kuhn Poker where player 1 is deciding upon their first action and they hold a Queen. Performing the search in the situations when player 1 is dealt a Jack or King would be a similar calculation in a different part of their larger overall search tree.

In this particular example, player 1 is invoking the search so they are the best-response player and the search is done from their perspective. Player 1's decisions are denoted by circles, and they are choosing maximum-valued actions at each of their decision nodes.

```
// Computes the EV of the decision-maker's node (mix backup)
double getMixDecisionEV(GameState gs) {
   // get legal actions
   List actions = getLegalActions(gs);

   for(int i = 0; i < actions.length; i++) {
      // take an action
      GameState temp = new GameState(gs);
      temp.takeAction(actions[i]);

      // use miximix to compute the child EV
      EVs[i] = miximix(temp);
   }

   // compute the probability distribution for selecting actions
   probs = getMixProbs(gs, EVs);

   // return weighted sum of EVs
   return weightedSum(probs, EVs);
}
```

Figure 3.7: Decision-maker's Miximix EV Backup

Player 2 is the opponent and their decision nodes are denoted by squares. The value of player 2's nodes will be computed via an expected value calculation. To calculate player 2's values, player 1 is assumed to have an opponent model which will specify how they think player 2 will act.

To illustrate this search, player 1's assumed opponent model for player 2 contains the following information:

- the relative frequencies of player 2's choice of actions: player 2 checks 50% of the time ($K = 0.5$) and bets the remaining 50% of time ($B = 0.5$) when they have to act after player 1 checks; player 2 folds 50% of the time ($F = 0.5$) and calls the remaining 50% of the time ($C = 0.5$) after player 1 bets, and

- player 1's chances of winning (pwin) at a showdown leaf node: player 1 wins 80% of the time ($pwin = 0.8$) when player 1 checks and then player 2 checks; player 1 wins 20% of the time ($pwin = 0.2$) when player 1 checks, player 2 bets and then player 1 calls; player 1 wins 0% of the time ($pwin = 0.0$) when player 1 bets and player 2 calls.

To see exactly how the calculation would be done, start at player 1's root decision point and start doing a depth first expansion of subsequent nodes. Player 1's first action to expand is a check, denoted by a 'k' on the branch which points towards the left. Following this action leads to the left-most opponent decision node in the figure. From this opponent decision node, their first action to expand is also a check. Following this action there is a showdown leaf node which player 1's opponent model says they will win 80% of the time.

```
// Compute the EV of a game state using miximix/miximax
double miximix(GameState gs) {
   if (gs.isStageNode()) return getStageEV(gs);
   if (gs.isLeafNode()) return getLeafEV(gs);
   return getDecisionEV(gs);
}

// Compute EV of decision node
double getDecisionEV(GameState gs) {
   if (gs.isOppDecision()) {
      // opponent decision
      return getOppDecisionEV(gs);
   } else {
      // decision for player invoking the search
      if (MIXIMIX) {
         // Miximix: stochastic action-selection
         return getMixDecisionEV(gs);
      } else {
         // Miximax: purely max action-selection
         return getMaxDecisionEV(gs);
      }
   }
}
```

Figure 3.8: Miximax/Miximix Algorithm

At this showdown node, there are 2 betting units in the pot, corresponding to each player's 1 unit antes, which means player 1 has an EV of 0.6 units there.

Now, continuing the depth-first expansion, the betting action, or right-pointing branch, at the left-most opponent decision node is then considered. This action leads to a player 1 decision node. At this decision node, player 1 can fold, which has an EV of -1 betting units since they give up their ante, or they can call which leads to a showdown leaf node. At this showdown leaf node, their opponent model says they have a 20% chance of winning which corresponds to an EV of -1.2 units.

Since player 1 is trying to maximize their earnings, they will choose to take the action which gives them the highest EV. In this case, this would mean they would choose to fold which gives their decision node an EV of -1 units. Since player 2's betting action led directly to this node the value of that action is also -1 units.

Since both of player 2's actions have values, there is now enough information to calculate the value of player 2's decision node following player 1's original check. Player 2's decision node value is calculated as the weighted sum of each of their actions weighted by their likelihood of taking those actions. This means, the value of player 2's decision node after player 1 checks (and therefore the value of player 1 checking since it leads immediately to that node) is then $0.5(0.6) + (0.5)(-1.0) = -0.2$ units.

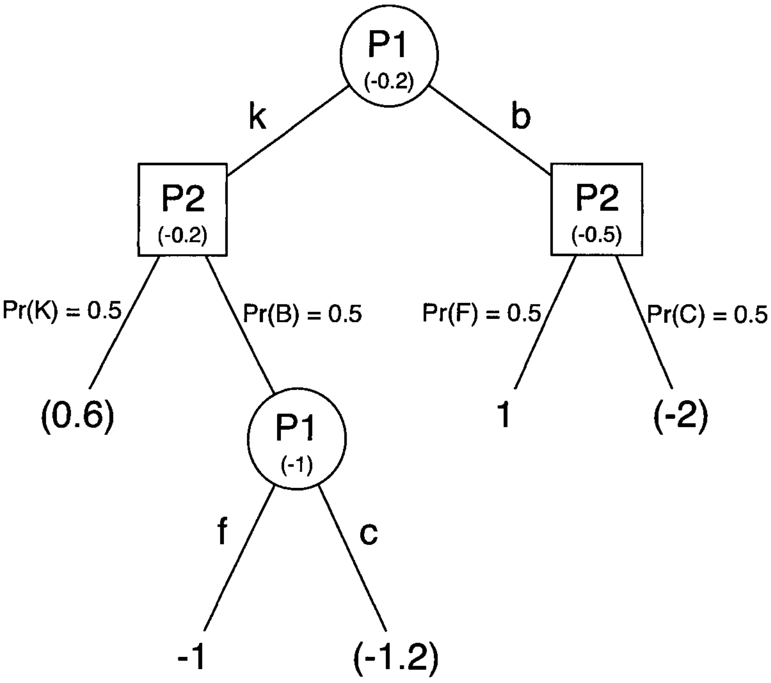

Figure 3.9: Player 1's Best-Response Strategy When Holding a Queen (Miximax)

To complete the depth-first expansion of player 1's original node, player 1's betting action and its subtree need to be expanded. Following the same procedure as was done for player 1's checking action, player 1's betting action has a computed EV of -0.5 units.

Since player 1's check has a higher expected value than a bet (i.e. -0.2 is greater than -0.5), player 1's best-response action would be to check since it is more profitable.

3.6.2 Example: Kuhn Poker Miximix Calculation

Figure 3.10 illustrates a miximix calculation for the same example. In this example, the decision-maker invoking the search will sometimes choose with low probability what it believes to be inferior actions for the sake of exploration and/or unpredictability.

Since the decision-maker's action preferences are now a form of a *softmax*[52] rather than a strict max, the resulting strategy computed during this search is a *soft best-response* rather than a true best-response.

In this example, the probability distribution that player 1 uses to select its actions is derived using a Gibbs distribution with a temperature value of 0.1. In the figure, player 1's probability distributions for choosing actions are denoted along player 1's action branches: since folding is slightly better than calling after player 1 checks and player 2 bets, player 1 computes its action distribution to choose folding $\approx$ 88% of the time and calling $\approx$ 12% of the time; since checking appears better than betting when player 1 first has to act, player 1 chooses checking $\approx$ 95% of the time and betting the remaining $\approx$ 5% of the time.

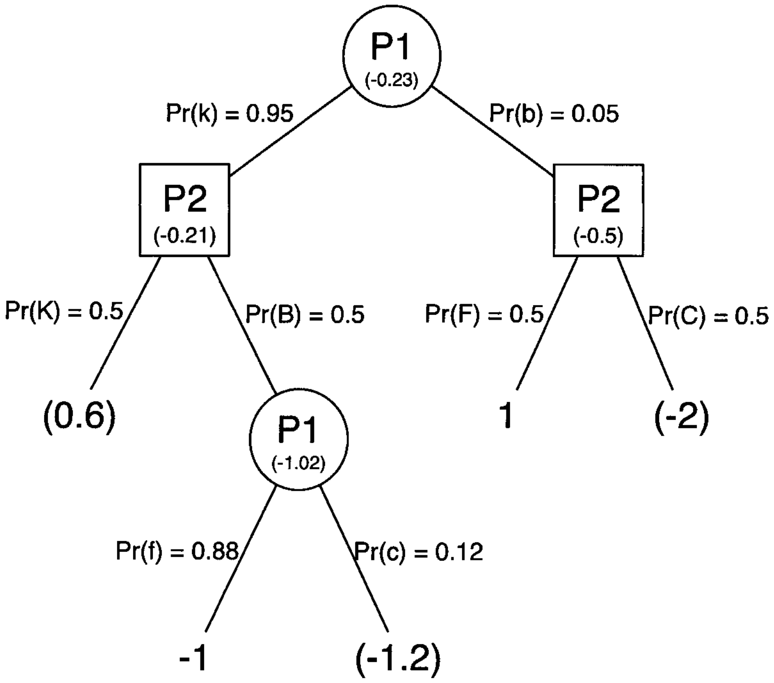

Figure 3.10: Player 1's Soft Best-Response Strategy When Holding a Queen (Miximix)

One thing to note is how acting according to a softmax method explores taking some sub-optimal actions and also how taking those sub-optimal actions affects the expected values of player 1's decision nodes. For example, the expected values are lower all over the tree for player 1 in Figure 3.10 compared to the values for player 1 in the Figure 3.9 because of the non-zero chance of taking exploratory moves which are currently believed to result in somewhat inferior payoffs.

3.7 Practical Search Considerations

Real variations of poker such as Texas Hold'em are obviously much larger than Kuhn Poker. As games get larger issues arise that need to be addressed if expectimax search is to remain practical.

Figure 3.11 illustrates a portion of the expectimax search tree for Texas Hold'em. In this tree there are four distinct game rounds (preflop, flop, turn, and river) where cards are dealt and then players take betting actions. The hexagons in the figure represent the chance nodes where cards are dealt. Each solid triangle represents the betting tree of all possible legal sequences of player betting actions.

The betting tree in our domain of 2-player Texas Hold'em, with a maximum of 4 total bets and raises per round, can be seen in Figure 3.12. The possible player actions are denoted as: 'f' for fold, 'k' for check, 'c' for call, 'b' for bet, and 'r' for raise. To make things more

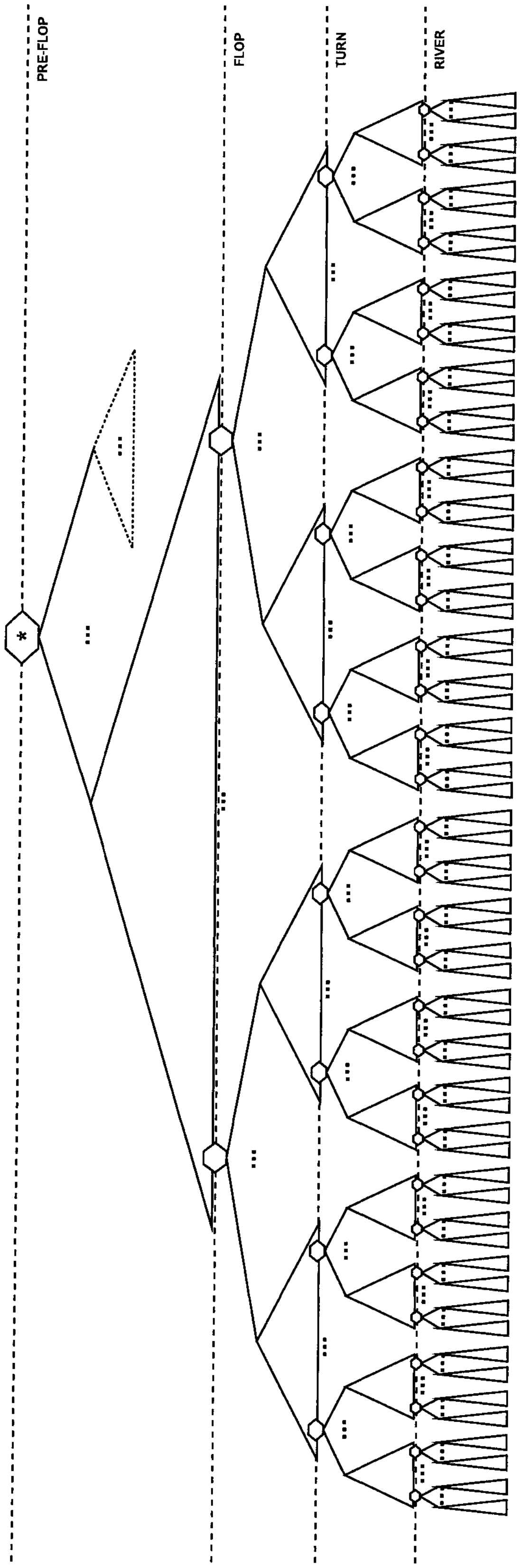

Figure 3.11: Texas Hold'em Expectimax Search Tree

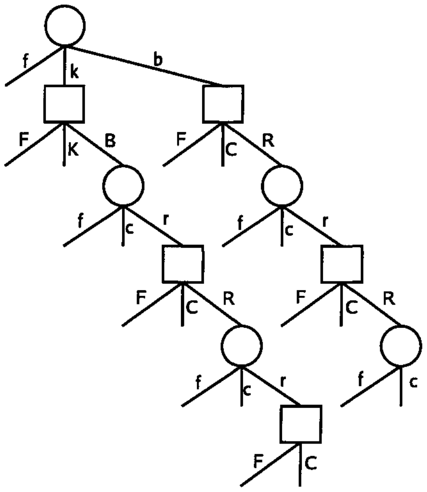

Figure 3.12: 2-player Texas Hold'em Betting Round Tree

readable, lowercase letters denote the first player's actions and uppercase letters denote the second player's actions.

The root of Figure 3.11 is a hexagon which denotes the preflop chance node. This chance node has branches representing each of the possible ${}_{52}C_2 = 1,326$ ways to deal the best-response player's hole cards. The left-most branch originating from this node leads to a subtree following one of these possible deals. The right-most branch leads to a dashed triangle containing ellipses which is simply meant to indicate that the larger subtree structure on the left would again be represented there. The ellipses in between the two branches represent all of the remaining ${}_{52}C_2 - 2 = 1,324$ similar subtrees that exist but are left out for space constraints.

Following each preflop deal branch is a triangle representing a preflop betting tree. As illustrated in Figure 3.12, there are nine betting sequences in the tree that do not end in folds and continue on to the next round. There are eight betting sequences in the tree that end a fold which ends the game.[7]

For each of the nine betting sequences (only two are shown in the diagram and ellipses are used to hint at the presence of the other seven) that continue onto the flop, there is a hexagon denoting the chance node that deals out all possible ${}_{50}C_3 = 19,600$ 3-card flops. Just like in the preflop round, each one of these deals is followed by a betting tree with each of the nine flop betting sequences that do not end in a fold leading to a turn chance node where each of 47 possible turn cards would be dealt. Each one of these turn card deals is then followed by a turn betting tree with the nine turn betting sequences not ending in folds each leading to a river chance node where 46 possible river cards are dealt. Each of these

[7] There are actually ten permissible sequences that end the game but two of them, f and kF, are strictly dominated by the action sequences, k and kK respectively, (since a player might as well never fold when they can check for free) and hence are ignored here.

Decision Point Where Search is Invoked	# of Leaves in Search Tree
start of game before preflop cards are dealt	≈ 697 trillion
after preflop cards dealt	≈ 525 billion
before flop cards are dealt	≈ 58.4 billion
after flop cards are dealt	≈ 2.98 million
before turn card is dealt	331,162
after turn card is dealt	7,046
before river card is dealt	782
after river card is dealt	17

Table 3.1: # of Leaves in 2-player Texas Hold'em Search Tree when First to Act

Decision Point Where Search is Invoked	# of Leaves in Search Tree
after preflop cards dealt and check	≈ 292 billion
after flop cards are dealt and check	≈ 1.656 million
after turn card is dealt and check	3,914
after river card is dealt and check	9

Table 3.2: # of Leaves in 2-player Texas Hold'em Search Tree when Second to Act Following a Check

river cards is likewise followed by a river betting tree where eight sequences end in folds and the remaining nine end in showdowns.

To get an idea of the size of the tree that would need to be searched for a decision at various points in the game, consider Tables 3.1 and 3.2. Table 3.1 shows the size of the complete tree (in terms of the its number of leaf nodes) that would need to be searched when the best-response player is first to act and assuming a worst-case opponent, in terms of search size, where every opponent action choice has to be considered within the search. When the best-response player is first to act, their decision is the first in the round so this search has to consider both their check and bet actions. This results in the worst-case search situation for the best-response player since they need to search almost twice as many nodes as compared to when they are second to act. To illustrate this difference, Table 3.2 shows the size of the complete tree that would need to be searched for some corresponding situations when the best-response player is second to act and assuming that the first player checked (this assumption results in the largest remaining tree to search).

Looking at the numbers in these tables, it is easy to see that the game is too large to search completely to the leaves from early parts in the game in the amount of time that a program would routinely be allowed to make a poker decision (i.e., around one second). Search times start becoming reasonable for a real-time decision once the search is invoked at a decision point after the flop cards are dealt (assuming a relatively efficient search and opponent model).

As with most search algorithms, the search could go to a certain depth and then call an evaluation function to estimate the value that would have been backed up for the subtree

Decision Point Where Search is Invoked	# of Leaves in Search Tree
before flop cards dealt (first to act)	≈ 7.677 billion
after flop cards dealt (first to act)	391,706
after flop cards dealt (second to act after check)	217,614

Table 3.3: # of Leaves in 2-player Texas Hold'em Search Tree for Flop Decisions Sampling River Cards

had it not been cutoff during the search. Properly estimating these values is difficult since each estimate has to implicitly consider all the various ways to win and lose money. This includes factors such as the likelihood of folds and the chance of winning showdowns.

Because implementing this evaluation function properly could potentially be difficult and beyond the scope of the work presented here, the decision was made to always search to a leaf node where it is easier to estimate the values needed to start the backup procedure. For searches invoked on the turn or river, this is not a problem since a full-width and full-depth search finishes very quickly. Unfortunately, for searches invoked on the preflop or flop, the search must be modified to make it possible to both search down to leaf nodes and finish in a reasonable amount of time.

For searches invoked in the flop stage, sampling is used to deal only a subset of the total possible river cards for each dealt turn card. More specifically, the search proceeds normally up to a river chance node where only six of the possible 46 cards are dealt uniformly at random rather than enumerating each of them. This type of sampling is naive but cuts the search tree down significantly as seen in Table 3.3.

For searches invoked in the preflop stage, the search procedure had to be modified even more. The actual implementation of the preflop search procedure was written by Aaron Davidson and is used with his permission. In this search procedure, regular chance nodes that deal cards are replaced by abstract chance nodes that discretize actual cards that could be dealt into a small number of "buckets". These buckets and their associated probabilities were computed via offline simulations and are looked up when needed in the search procedure.

One other thing that is important to note is that the size of the search tree that needs to be searched depends on the opponent. Consider the situation where the best-response player is playing against an opponent that always checks or calls. Against this particular opponent, the best-response player's search tree becomes significantly smaller (i.e. compare Tables 3.4, 3.5, 3.6 to Tables 3.1, 3.2, 3.3, respectively).

Against this opponent, a perfect opponent model would return this opponent's observed relative action frequencies for any particular decision as 0% for fold, 100% for check/call, and 0% for bet/raise. Recalling how the expectimax search backs up the value for an opponent decision node, it is easy to see that the subtrees under the opponent's fold and

Decision Point Where Search is Invoked	# of Leaves in Search Tree
start of game before preflop cards are dealt	≈ 4.55 trillion
after preflop cards dealt	≈ 3.43 billion
before flop cards are dealt	≈ 1.144 billion
after flop cards are dealt	58,374
before turn card is dealt	19,458
after turn card is dealt	414
before river card is dealt	138
after river card is dealt	3

Table 3.4: # of Leaves in 2-player Texas Hold'em Search Tree Against Always Call and First to Act

Decision Point Where Search is Invoked	# of Leaves in Search Tree
after preflop cards dealt and check	≈ 816 million
after flop cards are dealt and check	17,296
after turn card is dealt and check	184
after river card is dealt and check	2

Table 3.5: # of Leaves in 2-player Texas Hold'em Search Tree Against Always Call when Second to Act Following a Check

Decision Point Where Search is Invoked	# of Leaves in Search Tree
before flop cards dealt (first to act)	≈ 149 million
after flop cards dealt (first to act)	7,614
after flop cards dealt (second to act after check)	2,256

Table 3.6: # of Leaves in 2-player Texas Hold'em Search Tree Against Always Call for Flop Decisions Sampling River Cards

raise branches would never have to be searched because no matter what value is returned for those subtrees, it would be multiplied by its probability of occurrence, which in this case would be 0%.

Though this is an idealized opponent and most real-world opponents would lie between this one and the worst-case one where every action has to be considered, this example illustrates that opponent modelling information could provide information to improve the efficiency of the search via opponent specific pruning. This would lead to a search that is much more selective, which in turn would make the search smaller and allow it to be invoked at earlier decision points in the game.

In addition, this opponent modelling information could be used to help decide when to call an evaluation function. If a branch is believed to be extremely rare (i.e. the opponent model says it occurs some very small percentage of the time), it may be appropriate to call an evaluation function at that point to prune the search. Intuitively this seems like a prime candidate for applying an evaluation function because the evaluation function error is bounded by its likelihood within the expectimax search. On top of this, because these situations occur so rarely, the opponent may have a "canned approach" to playing them which could potentially be more easily captured in an evaluation function than other more common situations where the opponent may have a carefully planned out deceptive strategy.

Though these ideas seem like they might offer good potential, they are left for future work. The implementation presented in this thesis only prunes an opponent decision branch within the search when it believes the opponent will never take that action (i.e. its probability according to the model is 0%).

Chapter 4

Opponent Modelling

4.1 Designing a Model for Use in a Poker Program

In poker, though players have the same access to the public game information, such as betting actions and up-turned cards, a player's private hole cards ensures that each player has some game information that only they know. It is the presence of this hidden information that makes poker an interesting game.

Each player's hidden information plays a major role in determining how 1) action will proceed through the hand and 2) who wins if the game proceeds to a showdown. This is reflected in the action probabilities and the leaf node expected values used in the expectimax search for action-selection presented in the previous chapter.

Learning information about the opponent that can be used to infer these two important pieces of information needed for expectimax search is referred to in this thesis as *opponent modelling.* In this section, two different classes of opponent models are characterized that could be used to learn this information.

4.1.1 Strategy Class of Models

The first class of opponent model described here is referred to as a *strategy model* in this thesis, since it tries to learn an opponent's *behavior strategy* directly. The term "behavior strategy" comes from the field of game theory and is defined to mean a mapping of every player decision point (*information set* in the game theory literature) to a probability distribution over the player's choice of available actions allowed there.

Models of this class represent the opponent's decision-making process explicitly. That is, the modelling information maintained tracks how an opponent acts according to the information that they know at the time of their decisions. This known information includes the opponent's hidden cards plus the sequence of publicly observable game actions that include both betting actions and public cards. As an example, in Kuhn Poker this class of model could maintain the information that when the opponent is holding a Jack and facing a bet, they would fold 50% of the time, call the remaining 50% of the time, and never raise.

Because this model focuses on the opponent's strategy, it is inconvenient for games of imperfect information like poker where there is a folding action that allows the opponent's private cards to remain hidden. In these situations where there is a fold and the opponent's cards are never revealed, it is not clear how to update this class of model.

4.1.2 Observation Class of Models

To get around the problem that the folding action presents to the strategy class of models, an alternative model formulation can be considered that is based on modelling *observations* from the decision-maker's perspective. In this alternative class of models, called *observation models* in this thesis, the decision-maker only considers the information accessible to them at each point in the game and tries to model the probability of observing game actions there.

Since the game actions tracked in this class of model are seen from the decision-maker's perspective they are referred to here as observations. These observations correspond to actions taken by other decision-making entities such as nature (or more specifically, the dealer in poker) in the case of chance events and other players in the case of opponent actions (the opponent revealing their cards is included as an observable opponent action as well).

Using Kuhn Poker as an example, this type of model could maintain the following information: given that the decision-maker holds a queen and bet, this model would record what actions the decision-maker observes their opponent making in the resulting situation.

Because observations in this model are based only on the decision-maker's perspective, and therefore correspond only to what they can observe, all the information needed to update the model is always known and can never be kept hidden. For models of this class, folds are no longer problematic. In models of this class, when a fold occurs, the only information needed is the observation that the opponent folded and not what specific cards they folded.

4.1.3 Relationship Between the Two Classes of Models

Since observation class models contain observations from the decision-maker's perspective, they can naturally provide the opponent modelling information needed for the expectimax action-selection search described in Chapter 3. For example, an opponent's actions observed at one of their decision points can be used to predict the frequency of each of their actions at the same decision point within the expectimax search. The cards an opponent is observed revealing at each showdown can be used to predict the chance of the decision-maker winning a showdown in the expectimax search. The cards observed being dealt at each chance node can be used to predict the frequency of the cards being dealt at the same chance node in the expectimax search.

On the other hand, a strategy class model contains information about how an opponent chooses their actions in their information sets. Since the expectimax action-selection search is done from the decision-maker's perspective, some additional work first needs to be done to convert the opponent's strategy information stored in this class of model into the decision-maker's observation information needed in the search. The procedure to do this conversion is described below for the opponent action frequencies, the chance element frequencies, and the cards revealed at a showdown.

Opponent Action Frequencies

As mentioned in Chapter 3, each node in the expectimax search tree is a single representative for a set of nodes in the extensive form game tree. For the case of the expectimax opponent decision nodes, each of these nodes represent a set of opponent decision nodes in the extensive form game tree that share the same public information leading to that decision (i.e. the betting and the community cards), but correspond to the different possible hidden information (i.e. their possible hole cards) that the opponent could have.

This means that the probability of a decision-maker observing the opponent taking a particular action at an expectimax decision node depends on the probability that the opponent will take that action at each of the extensive form nodes represented there. In addition, the probability of the observation also depends on the relative probability of the extensive form node in the set of nodes represented by the expectimax node.

Fortunately, these probabilities can be obtained from the opponent's behavior strategy. The probability that the opponent will take a particular action at an extensive form decision node is specified in their behavior strategy, or likewise in a strategy class model the decision-maker may have of the opponent. The probability distribution over the extensive form nodes represented by an expectimax node can be computed by applying Bayesian updating[40] using the opponent's behavior strategy and all the game information leading to that opponent decision node.

To illustrate how to convert the opponent strategy information into observed action frequencies, the following example is presented. Consider the situation in Kuhn Poker where the decision-maker is first to act and they hold a Queen and bet. The opponent has to make the next decision. In this situation, the opponent has not yet acted so the only information that influences the probability distribution over their hidden information is that the decision-maker holds a Queen.[1] So, assuming the deck is fair and knowing that the opponent cannot hold the Queen, before they act they hold either a Jack or King with equal probability. Now assume, the decision-maker has a strategy class model of the opponent that says that when the opponent holds a Jack they will fold 80% of the time and

[1] The decision-maker's bet does not change the distribution over the opponent's hidden hands.

call the remaining 20% of the time, and when they hold a King, they will call 100% of the time. Given this information, in this example situation, the decision-maker would expect to observe the opponent folding 40% of the time and calling the remaining 60% of the time.

Chance Event Frequencies

Like the expectimax opponent decision nodes, each of the expectimax chance event nodes represent a set of chance nodes in the extensive form game tree that share the same public information leading there but different opponent hidden information. This means that the probability of observing a particular chance outcome in the expectimax tree depends on both the probability of that chance outcome occurring at one of the represented extensive form nodes and the probability of that extensive form node in the set.

Like in the case of the opponent action frequencies, these probabilities can be easily obtained. The probability of a chance outcome occurring at an extensive form node is specified both in the rules of the game and in the extensive form game tree. The probability of an extensive form node in the set can be computed by applying Bayesian updating using the opponent's behavior strategy and the game information leading up to the chance node.

Unfortunately, Kuhn poker does not have any chance nodes other than the initial root chance node where each player is dealt their cards. Since this chance node is right at the beginning of the game, it does not serve as a useful chance node to illustrate how observed chance event frequencies can be obtained from opponent strategy information. As a result, an abstract scenario will be used instead to illustrate this.

Assume that there is a deck consisting of four cards. For example, a Ten, Jack, Queen and King. The decision-maker is dealt the Queen and the opponent is dealt a card face down to be kept hidden from the decision-maker. The players then proceed to do some betting based on their cards and there is now a chance node where another card is dealt from the deck that both players can see.

To calculate the probability of the decision-maker observing each of the possible chance outcomes, the distribution over the opponent's hidden information would first have to be calculated using Bayesian updating. This updating is based on the information that has been revealed up to that point in the hand including any known cards and betting. However, since this is an abstract scenario and no opponent strategy has actually been specified, the Bayesian updating cannot be shown. Instead, assume for the sake of completing the calculation, that at this expectimax chance node there is a 40% chance the opponent holds a Ten, a 50% chance they hold a Jack and a 10% chance they hold a King. Remember there is no chance that they can hold a Queen since the decision-maker was dealt the only Queen in the deck. From the rules of the game, it is also known that in this situation: when the opponent holds a Ten, a Jack or King will be dealt with equal probability; when the opponent holds a Jack, a Ten or King will be dealt with equal probability; and when the opponent holds a King, a Ten or Jack will be dealt with equal probability. Combining this information, at this example expectimax chance node, the decision-maker would expect to observe a Ten being dealt 30% of the time, a Jack 25% of the time, and a King the remaining 45% of the time.

Cards Revealed at a Showdown

Like the other nodes in the expectimax tree, the showdown nodes in the expectimax tree represent a set of showdown nodes in the extensive form game tree. The extensive form nodes represented at each expectimax showdown node all share the same public information leading to that showdown, but each represent the different possible hidden cards that the opponent can have but are kept private from the decision-maker.

As a result, it is this set of hidden cards that the decision-maker observes the opponent revealing at each showdown node. The distribution of elements in this set, and therefore of the actual cards the opponent reveals, can be computed by applying Bayesian updating using the opponent's behavior strategy and the game information that leads to the showdown. To illustrate how this is done, a Kuhn poker example is used.

Consider the showdown in Kuhn Poker where the decision-maker is dealt a Jack and as the first player to act bets and is called by the opponent. After the decision-maker sees their card (the Jack), and assuming a fair deck, by applying Bayesian updating the opponent must hold either a Queen or King with equal probability. The decision-maker's subsequent bet does not provide any information to change these probabilities. On the other hand, the opponent's subsequent choice to call in response to the decision-maker's bet provides information that can change the distribution of cards they hold after choosing that action.

For example, assume that the opponent's strategy is such that when they are facing a bet and holding a Queen, they would fold 50% of time and call 50% of the time, and when they are holding a King they would never fold and would call 100% of the time. Then by applying Bayesian updating, after their call the opponent's hidden card distribution is revised to have them hold a Queen 1/3 of the time and a King the remaining 2/3's of the time. Since this call leads to the showdown, these are also the probabilities of the decision-maker observing the opponent revealing either of those cards there.

4.1.4 Pros and Cons: A Comparison of the Model Classes

Since strategy class models just specify a probability distribution over legal actions at each of the opponent's information sets, it is easy to ensure that models of this type correspond to legal game-playing strategies. As long as the probability distributions are valid and correspond to legal actions and that each reachable information set has a valid distribution associated with it, the model corresponds to a legal strategy. Given that the rules of the game are known, this is relatively trivial.

On the other hand, it is much more difficult to ensure that observation class models are constructed such that they correspond to legal game-playing strategies. The information in a particular part of an observation class model may appear legal (i.e. there is a valid probability distribution over valid observed actions - that is, it sums to one), but despite this, it is possible that the information in the model could never be realized given an actual legal game strategy.

To illustrate this, consider the opponent decision node that leads to the showdown described in the showdown example above. According to the assumed strategy, the opponent is expected to be observed folding 25% of the time at that decision node, and calling the remaining 75% of the time. When they do call, they will reach a showdown, and as shown above they are expected to reveal a Queen 1/3 of the time and a King the remaining 2/3's of the time there.

Now imagine constructing an observation model based on observations of actual game play. It is not hard to imagine observing the opponent call a few times before ever seeing them fold and show a King with each call. These observations might suggest that in that particular situation, the opponent will always call and always show a King and never fold. This seems perfectly reasonable and consistent when just considering the observations.

However, the rules of the game forces them to have a Queen sometimes in that situation. That is, the rules guarantee that they hold a Queen 50% of the time they face the decision to fold or call. Thus, if they never fold they must show Queens some of the time when they call, or if they never show a Queen when they call, then they must fold some of the time. As a result, there is no possible legal strategy they can employ in that situation where they only call and never fold and yet still only show a King (despite the small sample size of observations indicating otherwise).

It is unfortunate that the observation class of models can have these problems since models of this class can be built with much simpler techniques than strategy class models. Since the observation class of models rely strictly on observations that are always accessible, the model can be built by simply keeping track of which observations occurred in each situation. The strategy class of models, on the other hand, have the problem that they are based on the opponent's perspective and in games like poker a folding action sometimes keeps the opponent's private information from ever being revealed.

For the opponent models described and implemented in this thesis, observation class models were used instead of strategy class models. This choice was made for the following three reasons:

- observation class models provide a convenient way of dealing with folding actions that are prevalent in poker and are problematic in strategy class models,

- observation class models can be built using simple techniques, and
- observation class models contain information in a format that can naturally be used in expectimax action-selection search.

4.2 Building Observation Class Models for Poker

For opponent models to be used within the expectimax action-selection search presented in chapter 3, they have to be able to supply three different types of information as seen from the decision-maker's perspective:

- probability distribution over observed opponent actions at an opponent decision node,
- probability distribution over observed chance event outcomes at chance nodes, and
- probability of the decision-maker winning a showdown.

The simplest way to build models that keep this information, and the way it is done in this thesis, is to simply keep counts of the information being modelled as it is observed through actual game play.

Taking this approach, the model keeps track of opponent action frequencies for each of their decision points by maintaining counts of each action observed there. The relative counts can then be used to construct a probability distribution for observed opponent actions.

To illustrate this, consider an example where, at a particular opponent decision point, the opponent might have been observed choosing an action 10 times in the past: 5 folds, 3 calls, and 2 raises. These counts can then be used to construct a probability distribution over observed opponent actions at one of their decision points by taking the counts of each action there and dividing each by the total number of action observations made at that decision point. In this example, the resulting probability distribution constructed from the example counts would suggest that the opponent would be observed folding 50% of the time, calling 30% of the time, and raising the remaining 20% of the time they are faced with choosing an action in that decision situation. Similarly, if the opponent was then later observed calling at that decision point, the updated model would suggest that the opponent folds 45.4% of the time (5 folds observed out of 11 total actions), calls 36.4% of the time (4 calls observed out of 11 total actions), and raises 18.2% of the time (2 raises observed out of 11 total actions).

Modelling chance node frequencies could be done in exactly the same manner. However, to keep the models in the thesis more manageable when applied to Texas Hold'em, the simplifying assumption is made that the chance node frequencies do not need to be modelled and can instead be approximated by assuming that they occur with uniform frequency. Though this assumption will be incorrect for some opponents, it is assumed that the error introduced by this assumption is small compared to errors present in the opponent model.

To model the decision-maker's chance of winning at each showdown leaf node, counts are kept of which hands the opponent has revealed there in past game play. Given the decision-maker's hand, their probability of winning can be estimated by summing up the total number of observed hands that are worse along with half of the hands observed that tie and then dividing that sum by the total number of hands observed.

4.3 Generalizing Observed Data for Texas Hold'em

In Texas Hold'em there are a large number of possible distinct scenarios where opponent modelling information is needed. This makes its difficult to learn an effective opponent model. As the number of situations increase, the amount of data needed to learn about these situations also increase. In the case of trying to build a model during actual poker play, this is particularly unfortunate because it means that many games will need to be played with an ineffective model before an effective model is ever learned. This, of course, may result in potentially substantial losses.

To combat this problem, the opponent modelling implementation used in this thesis for Texas Hold'em attempts to generalize data observed in one situation to other similar situations. The intuition here is that data in one specific game situation can be used to infer data in another related situation, allowing an effective model to be learned more quickly by needing fewer data points.

For example, consider two different poker hands where the players bet exactly the same, but where there is a slight difference in the community cards. It seems reasonable that if the difference amongst the community cards is small between the two different hands, the information about the cards the players had in the first hand could be similar to the cards they had in the second. Similarly, it also seems reasonable that the cards shown in two different hands could be quite similar if the actions were not exactly the same but matched quite closely.

4.3.1 Instance-based Learning

Generalizing observed data is a problem that occurs in all types of machine learning and artificial intelligence applications. In machine learning, the problem of generalizing data can be viewed at as a problem of *function approximation.* In this view, input/output pair examples are viewed as sample points of a function and the goal of the problem is to construct a function using the sample examples that can then accurately map inputs to outputs for any query input.

For the specific problem formulation in this thesis, there are essentially two generalization tasks. The first task needs to map an opponent action node to the actions the decision-maker will observe the opponent making there. The second task needs to map a showdown node to the chance of the decision-maker winning there with a particular hand. To tackle both of these generalization tasks, an instance-based learning method is used.

In instance-based learning methods[34], training examples are simply stored in memory. Each future query is assigned an output value by finding similar query examples previously stored in memory and considering their output values. The k-Nearest Neighbor learning algorithm[15] is a well-known example of this type of method.

An instance-based learning method is particularly convenient for tackling the problem of learning opponent modelling information during actual game play. As each game is played, the information observed in the game is simply stored in memory. This provides an easy way to incrementally add more information to the opponent model as more games are played.

4.4 Instance-based Learning of Opponent Action Frequencies

The method used for learning opponent action frequencies implemented in this thesis is the same as the one used by Davidson[17]. In this method, opponent actions are observed and stored in memory and these observations are used to approximate the opponent's actual frequencies. For any particular opponent decision point, the probability distribution over an opponent's chosen actions can be approximated by taking the observation counts for each of their fold, call, and raise actions there and dividing each by the sum of all three.

In Texas Hold'em there are too many distinct opponent decisions to keep track of observations for each one. To lessen the number of observations that need to be tracked, and at the same time try to speed up learning, only the betting actions made by both players from the start of the current game until the opponent decision point are used as the *context* where observations of the opponent actions are maintained.

The sequence of betting actions that form one of these simplified contexts can naturally be viewed as a string, and is referred to as a *betting string* or *betting sequence* in this thesis. A trie, called a *context tree* by Davidson[17], is then used to store these contexts in a manner that makes it easy to add and retrieve opponent action frequencies. In the context tree, an opponent action context can be found by simply following the path of actions contained in a betting string starting at the root of the context tree.

When an opponent action is observed, a count is incremented on the branch representing that action in the context tree. A probability distribution over observed opponent actions at an opponent decision can then be estimated from the counts on each branch that stems from a particular opponent decision node in the tree.

This method of learning opponent action frequencies can be viewed as a simplistic example of instance-based learning. The distance metric corresponding to this implementation simply uses an exact match of all betting actions to determine which observations are related or unrelated to each other. With this particular distance metric, all related observations have equal weight when generalizing, and all unrelated observations are simply ignored.

This distance function is admittedly simplistic since it does not permit a high degree of generalization and it ignores potentially useful information such as the public board cards. On the other hand, this distance function is conceptually easy to use. It is used since it is believed that a player conveys a lot of information by their betting actions alone in a majority of situations. Though this method of generalization is simplistic, it allowed us to produce a basic implementation that can be built on in future work.

4.5 Instance-based Learning for Estimation of Winning at Showdown

The approach used in this thesis for estimating the chance of the decision-maker winning a showdown is based on similar techniques to those used for estimating opponent action frequencies and is an extension to the approach used by Davidson[17].

In this approach, when an opponent's cards are revealed at a showdown, the hand rank (HR) of those cards is computed. To compute the opponent's HR, the opponent's hand is compared to all the other possible hands that could be made using the five board cards and any two hole cards (other than the ones the opponent holds) to see how often the opponent's cards would win, lose or tie.

Once the win, loss, and tie counts are known, the opponent's HR is simply the number of wins plus half the number of ties divided by the total number of possible hands. This resulting HR represents a percentile ranking of the opponent's two card holding compared to all possible two card holdings given the particular board. This value is between 0 and 1 and it can be stored in a histogram of discretized HRs.

This histogram of discretized opponent HRs is used to estimate the decision-maker's chance of winning a showdown. To do this, decision-maker's own HR is computed and then compared to the histogram of opponent HRs to see how often it would have won, lost, or tied at that showdown in the past.

In Davidson's work, showdown histograms were kept for every possible unique betting string. This provided a mechanism for some generalization (i.e. because the player's hole cards and the community cards were ignored), but the amount of generalization was fixed. This meant that there was no way to increase generalization early in a match when data is sparse and decrease it later in a match when data is more plentiful.

To try and address this problem, a distance function was created to define how similar observations in one showdown context are to observations in another. To keep the distance function simple, only ten levels of similarity were defined. Starting with the highest level of similarity and then successively moving to lower levels, these are:

- context tree - With this distance function, this is the highest level of similarity that showdown observations can have. Showdown observations having this level of similarity correspond to showdowns where the betting actions for both players leading to the showdown match exactly and the players are acting from the same positions. The showdown observations that exhibit this level of similarity correspond exactly to the showdowns automatically grouped together in the context tree. Generalization occurs between observations at this similarity level because the decision-maker's hole cards and the public board cards are ignored.

- s4 (with and without position) - This is the next highest level of similarity defined for showdown observations. Showdown observations having this level of similarity correspond to showdowns which share the same sum total of the bets and raises made by each of the decision-maker and opponent in the preflop and flop rounds, and share the same number of bets and raises made by each player on each of the turn and river rounds. This level of similarity is actually broken into two different sub-levels according to whether the players are in the same relative positions in the showdown observations. Showdowns exhibiting s4 similarity with the players in the same positions are considered more similar than when the players are in opposite positions. By matching on the sum of the bets and raises each player made on the first two rounds, s4 provides more generalization than the context tree.

- s3 (with and without position) - Showdown observations having this level of similarity match the sum of bets and raises made by each player over the preflop, flop, and turn rounds, and also matches exactly the number of bets and raises made by each player on the river round. Like the s4 similarity level, this level is actually two different levels according to the relative positions of the players in the showdowns being compared. By just matching on the sum of bets and raises each player made on the first three rounds, this method provides more generalization than s4.

- s2 (with and without position) - Showdown observations having this level of similarity share the same number of total bets and raises made by each player over the course of the entire hand. Like the s3 and s4 similarity levels, this similarity level is actually two different similarity levels depending on the players' relative positions. This level of similarity is quite general and is simply meant to capture how the opponent's shown aggression and the opponent's perception of the modelling player's shown aggression relates to the opponent's shown HR.

- s1 (with and without position) - Showdown observations having this level of similarity only need to match in terms of the total number of bets and raises the opponent made over the course of a hand. This is the lowest level of similarity defined and it is just meant to capture how the opponent's shown HR relates to their level of aggression. Like the other "s" similarity levels, this level actually specifies two different levels based on the positions of the players.

- unrelated - Showdown observations that do not have match any of the above similarity levels are deemed completely unrelated.

4.5.1 Showdown Similarity Examples

To help illustrate the distance function for showdown contexts, Table 4.1 is provided. The first row in this table contains a target showdown context, denoted by the betting sequence bRc/kK/bC/bRrC. The remaining rows show other possible showdown contexts and their assigned level of similarity to the target context. Examples are provided for each of the context tree, s4, s3, s2, and s1 similarity levels.

The columns of Table 4.1 are defined as follows:

- **Betting Sequence -** The betting sequence leading to the showdown context in the row. The lowercase actions in this sequence are taken by player *p* and the uppercase actions are taken by player p's opponent, player *o*.

- $\mathbf{P}_p$, $\mathbf{F}_p$, $\mathbf{T}_p$, $\mathbf{R}_p$ **-** The total number of bets and raises made by player p in each of the *p*reflop, *f*lop, *t*urn, and *r*iver betting rounds, respectively.

- $\mathbf{P}_o$, $\mathbf{F}_o$, $\mathbf{T}_o$, $\mathbf{R}_o$ **-** The total number of bets and raises made by the opponent o in each of the *p*reflop, *f*lop, *t*urn, and *r*iver betting rounds, respectively.

- **Similarity -** The level of similarity between the showdown context in the row and target showdown context in the top row of the table. The values in this column either belong to the *context tree*, *s4*, *s3*, *s2*, or *s1* similarity levels defined above. In addition, the *s4*, *s3*, *s2*, and *s1* similarity levels have *(sp)* or *(dp)* associated with them to indicate whether the players are acting in the *s*ame relative *p*ositions or in *d*ifferent relative *p*ositions compared to the target showdown context.

To show how the information in Table 4.1 is combined to define similarity levels, Table 4.2, Table 4.3, Table 4.4, and Table 4.5 are also provided.

Betting Sequence	P_p	P_o	F_p	F_o	T_p	T_o	R_p	R_o	Similarity
bRc/kK/bC/bRrC	1	1	0	0	1	0	2	1	
bRc/kK/bC/bRrC	1	1	0	0	1	0	2	1	Ctx. Tree
kK/bRc/bC/bRrC	0	0	1	1	1	0	2	1	s4 (sp)
Kk/KbRc/KbC/KbRrC	0	0	1	1	1	0	2	1	s4 (dp)
BrC/Kk/KbC/KbRrC	1	1	0	0	1	0	2	1	s4 (dp)
bRc/bC/kK/bRrC	1	1	1	0	0	0	2	1	s3 (sp)
kK/bC/bRc/bRrC	0	0	1	0	1	1	2	1	s3 (sp)
BrC/KbC/Kk/KbRrC	1	1	1	0	0	0	2	1	s3 (dp)
Kk/BrC/KbC/KbRrC	0	0	1	1	1	0	2	1	s3 (dp)
bRrRc/bC/bC/kK	2	2	1	0	1	0	0	0	s2 (sp)
kK/kK/bRrC/bRrC	0	0	0	0	2	1	2	1	s2 (sp)
KbRc/KbRrC/Kk/KbC	1	1	2	1	0	0	1	0	s2 (dp)
Kk/Kk/KbRrC/KbRrC	0	0	0	0	2	1	2	1	s2 (dp)
kBc/kBc/kK/kK	0	1	0	1	0	0	0	0	s1 (sp)
bC/bC/bC/bRrRc	1	0	1	0	1	0	2	2	s1 (sp)
Bc/Bc/KbC/KbC	0	1	0	1	1	0	1	0	s1 (dp)
BrRc/Kk/Kk/KbC	1	2	0	0	0	0	1	0	s1 (dp)

Table 4.1: Example Showdown Similarities

In Table 4.1, the second row has the exact same betting sequence as the target context and the players are in the same relative positions so this entry is assigned the context tree level of similarity.

The next three rows are all assigned the s4 level of similarity. As seen in Table 4.2, for each of these entries, the sum of both player p's and his opponent o's preflop and flop bets (i.e. $\mathbf{P}_p$+$\mathbf{F}_p$ and $\mathbf{P}_o$+$\mathbf{F}_o$ in Table 4.2, respectively) match the same information in the target showdown context, and in addition each player's turn (i.e., $\mathbf{T}_p$ and $\mathbf{T}_o$ in Table 4.2) and river bets (i.e., $\mathbf{R}_p$ and $\mathbf{R}_o$ in Table 4.2) match the target context too. In the first of these entries, the players are acting in the same relative positions as in the target showdown context, and in the other two they are acting in different relative positions.

The next four rows in Table 4.1, show example contexts that have the s3 level of similarity. To achieve this level of similarity, each context must match the sum of each player's bets in the first three betting rounds (i.e., $\mathbf{P}_p$+$\mathbf{F}_p$+$\mathbf{T}_p$ and $\mathbf{P}_o$+$\mathbf{F}_o$+$\mathbf{T}_o$ in Table 4.3) and also match each player's bets on the river (i.e., $\mathbf{R}_p$ and $\mathbf{R}_o$ in Table 4.3).

Following the s3 contexts, there are four s2 contexts. For the s2 level of similarity, these contexts must match the target showdown context's sum total of bets and raises put in by each player over all four betting rounds (i.e., $\mathbf{P}_p$+$\mathbf{F}_p$+$\mathbf{T}_p$+$\mathbf{R}_p$ and $\mathbf{P}_o$+$\mathbf{F}_o$+$\mathbf{T}_o$+$\mathbf{R}_o$ in Table 4.4).

The last four entries in Table 4.1 are considered s1 similar to the target showdown context. To exhibit this similarity, these contexts need only match the target context's total number of raises made by the opponent (i.e., $\mathbf{P}_o$+$\mathbf{F}_o$+$\mathbf{T}_o$+$\mathbf{R}_o$ in Table 4.5).

Betting Sequence	P_p+F_p	P_o+F_o	T_p	T_o	R_p	R_o	Similarity
bRc/kK/bC/bRrC	1	1	1	0	2	1	
kK/bRc/bC/bRrC	1	1	1	0	2	1	s4 (sp)
Kk/KbRc/KbC/KbRrC	1	1	1	0	2	1	s4 (dp)
BrC/Kk/KbC/KbRrC	1	1	1	0	2	1	s4 (dp)
bRc/bC/kK/bRrC	2	1	0	0	2	1	s3 (sp)
kK/bC/bRc/bRrC	1	0	1	1	2	1	s3 (sp)
Kk/KbC/BrC/KbRrC	1	0	1	1	2	1	s3 (dp)
Kk/BrC/KbC/KbRrC	1	1	1	0	2	1	s3 (dp)
bRrRc/bC/bC/kK	3	2	1	0	0	0	s2 (sp)
kK/kK/bRrC/bRrC	0	0	2	1	2	1	s2 (sp)
KbRc/KbRrC/Kk/KbC	3	2	0	0	1	0	s2 (dp)
Kk/Kk/KbRrC/KbRrC	0	0	2	1	2	1	s2 (dp)
kBc/kBc/kK/kK	0	2	0	0	0	0	s1 (sp)
bC/bC/bC/bRrRc	2	0	1	0	2	2	s1 (sp)
Bc/Bc/KbC/KbC	0	2	1	0	1	0	s1 (dp)
BrRc/Kk/Kk/KbC	1	2	0	0	1	0	s1 (dp)

Table 4.2: Example S4, S3, S2, and S1 Showdown Similarities

Betting Sequence	$P_p+F_p+T_p$	$P_o+F_o+T_o$	R_p	R_o	Similarity
bRc/kK/bC/bRrC	2	1	2	1	
bRc/bC/kK/bRrC	2	1	2	1	s3 (sp)
kK/bC/bRc/bRrC	2	1	2	1	s3 (sp)
Kk/KbC/BrC/KbRrC	2	1	2	1	s3 (dp)
Kk/BrC/KbC/KbRrC	2	1	2	1	s3 (dp)
bRrRc/bC/bC/kK	4	2	0	0	s2 (sp)
kK/kK/bRrC/bRrC	2	1	2	1	s2 (sp)
KbRc/KbRrC/Kk/KbC	3	2	1	0	s2 (dp)
Kk/Kk/KbRrC/KbRrC	2	1	2	1	s2 (dp)
kBc/kBc/kK/kK	0	2	0	0	s1 (sp)
bC/bC/bC/bRrRc	3	0	2	2	s1 (sp)
Bc/Bc/KbC/KbC	1	2	1	0	s1 (dp)
BrRc/Kk/Kk/KbC	1	2	1	0	s1 (dp)

Table 4.3: Example S3, S2, and S1 Showdown Similarities

Betting Sequence	$P_p+F_p+T_p+R_p$	$P_o+F_o+T_o+R_o$	Similarity
bRc/kK/bC/bRrC	4	2	
bRrRc/bC/bC/kK	4	2	s2 (sp)
kK/kK/bRrC/bRrC	4	2	s2 (sp)
KbRc/KbRrC/Kk/KbC	4	2	s2 (dp)
Kk/Kk/KbRrC/KbRrC	4	2	s2 (dp)
kBc/kBc/kK/kK	0	2	s1 (sp)
bC/bC/bC/bRrRc	5	2	s1 (sp)
Bc/Bc/KbC/KbC	2	2	s1 (dp)
BrRc/Kk/Kk/KbC	2	2	s1 (dp)

Table 4.4: Example S2 and S1 Showdown Similarities

Betting Sequence	$P_o+F_o+T_o+R_o$	Similarity
bRc/kK/bC/bRrC	2	
kBc/kBc/kK/kK	2	s1 (sp)
bC/bC/bC/bRrRc	2	s1 (sp)
Bc/Bc/KbC/KbC	2	s1 (dp)
BrRc/Kk/Kk/KbC	2	s1 (dp)

Table 4.5: Example S1 Showdown Similarities

4.5.2 Combining Data From Different Similarity Levels

For the opponent modelling implementation used in this thesis, the opponent model combines showdown observations of different similarity levels by starting with the most similar observation and then successively moving to more and more distant ones until the total number of observations at a showdown reach a predefined threshold. In addition, when the data is combined, the weight an observation is assigned depends on its similarity with the most similar observations given the most weight and the least similar ones given the least weight.

This method for combining data provides a simple way to automatically tailor the level of generalization used to the amount of data available. For example, when the data is sparse more generalization occurs, and as more data is gathered the amount of generalization decreases.

4.5.3 Default Modelling Information

The type of modelling described in this thesis is problematic when there are no observations. To get around this problem, some default opponent modelling information is used in these cases. Defining default modelling information is tedious, and as a result, the default modelling information used in this thesis is rather simplistic.

Four different opponent HR showdown distributions were defined and used when a showdown node is encountered in the expectimax search that has no data. These four basic opponent HR distributions represent showdowns where the opponent is most likely to hold a very weak, somewhat weak, somewhat strong, or very strong hand. The amount of betting the opponent did leading to a particular showdown is used to decide which of the four distributions is used when the defaults are needed. The more the opponent bets, the stronger the hand they are assumed to have.

When there is no data for observed opponent action frequencies, a simplistic heuristic rule-base is used to assign a default observation probability triple. These heuristics consider the opponent's actions leading to the decision point where the opponent modelling information is needed. If the opponent has been aggressive the default triple is biased towards raising and otherwise it is biased towards calling. The defaults always assume the opponent will never fold.

4.5.4 Handling the Effects of Recency

Since the scope of the work in this thesis focused on opponents that do not change their strategy, little effort was devoted to bias observations so that recent observations are given more consideration than older ones. This issue is expected to be addressed in future work.

Chapter 5

Results

5.1 Leduc Hold'em Results

The purpose of this first set of experiments and results is to show that given enough experience, a computer poker player that implements the ideas presented earlier in the thesis is capable of achieving the best results possible against an opponent whose strategy does not change.

For these experiments, a small-scale variation of Texas Hold'em, called Leduc Hold'em, is used. Leduc Hold'em was designed by the University of Alberta Computer Poker Research Group to create a simple testbed that retains the same strategic elements of the full game, but that is computationally easier to work with.

5.1.1 Leduc Hold'em Rules

A six card deck is used in Leduc Hold'em. This deck consists of two suits that each contain the same three ranks: a Jack, a Queen, and a King. The game is played with two players and starts off with each player posting an ante of one betting unit. After the antes are posted, each player is dealt one hole card face down. There is then a standard round of poker betting (starting with the player that is not the dealer) where players are allowed to fold, check/call, and bet/raise. In the betting round, there is a two-bet maximum, meaning that after one player bets, and the other player raises, no further raising is allowed in the round. In this first betting round, the size of each bet or raise is two betting units. If one player folds during the betting round, the game ends with the player that did not fold winning the pot.

If neither player folds, play proceeds to the second round of the game. This round starts with a community card being dealt face up that both players use to make their two-card poker hand. Following the deal of this community card, a second round of betting takes place. It is the exact same as the first round except that the size of each bet or raise is four betting units. The doubling of the bets mimics the betting structure of Texas Hold'em.

If no player folds during the second betting round, the game goes to a showdown where each player reveals their private hole card. The best two-card poker hand wins the pot and tied hands split it equally. In this game, the two-card poker hands are ranked from best to worst as follows: a pair of Kings, a pair of Queens, a pair of Jacks, one King and one Queen, one King and one Jack, and one Queen and one Jack.

Leduc Hold'em Problem Size

In Leduc Hold'em, the extensive form game tree has 5520 leaves. The root chance node of the game that deals a hole card to each player has 30 possible branches. Following each of these branches is a first round betting subtree. Ignoring the two betting sequences[1] where a player folds when they could check for free (i.e., f and kF), there are four betting sequences that result in leaf nodes where the game ends due to one player folding (i.e., kBf, kBrF, bF, and bRf) and five betting sequences which lead to the second round (i.e., kK, kBc, kBrC, bC, and bRc). Each of the betting sequences that leads to the second round of the game has a chance node that has four branches for each community card that can be dealt there. Following each of these community card branches is the second round betting subtree which when ignoring the betting sequences where a player folds when they could check for free (i.e., f and kF) has nine distinct betting sequences leading to leaf nodes (i.e., kK, kBf, kBc, kBrF, kBrC, bF, bC, bRf, and bRf).

There are 936 information sets in Leduc Hold'em: 468 for the first player and 468 for the second player. Each player can be dealt 6 possible cards to start the game. For each possible hole card they can be dealt, each player has three information sets in the first betting round. For player 1 this corresponds to them having to make the first action of the game plus their two decisions immediately following the kB and bR betting sequences. For player 2, their three information sets in the first round immediately follow the k, kBr, and b betting sequences. In the first betting round, four of the nine possible betting sequences that end the betting round end the game (i.e., kBf, kBrF, bF, and bRf) and the remaining five possible betting sequences that end the betting round lead to the second round of the game (i.e., kK, kBc, kBrC, bC, and bRc). For each of these five betting sequences leading to the second round of the game, there are five distinct community cards which can be dealt which each lead to the second betting round where each player again has three information sets.

In the expectimax search tree, there are 1374 leaf nodes: 750 showdown leaves and 624 fold leaves. The root chance node has six branches for each of the possible hole cards a player can be dealt. These six branches each lead to the first round betting subtree where there are four betting sequences that lead to fold leaf nodes (i.e., kBf, kBrF, bF, and bRf)

[1] The notation used here for betting sequences is the same notation used in Chapter 3 where it was explained.

and five betting sequences that lead to the second round (i.e., kK, kBc, kBrC, bC, and bRc). The five betting sequences that lead to the second round each have a chance node with five possible branches each leading to the second betting round where there are four betting sequences leading to fold leaf nodes and five betting sequences leading to showdown leaf nodes.

5.1.2 Experiment Setup

The purpose of the experiment is to show that a computer player implementing the ideas presented in this thesis is able to learn to play a best-response strategy against an opponent whose strategy is not changing given enough game-playing experience against that opponent.

To do this, the computer player implementing the ideas in the thesis (named BRPlayer since it plays a best-response strategy) will play against three different opponents: one that always checks and calls (CallPlayer), one that always raises (RaisePlayer) when allowed by the rules of the game and calls otherwise, and one that plays according to a Nash equilibrium strategy (NashPlayer) that was solved via linear programming. To give the BRPlayer a chance to build an accurate opponent model, each match consists of 4,000,000 hands. To help lessen the effects of luck on the match results, 20 matches are played and averaged.

Since Leduc Hold'em is small enough, the BRPlayer is implemented with an observation model for its opponent that contains no abstracted or generalized opponent information sets. This would not be possible in Texas Hold'em, where the size of the game makes abstraction necessary. Using Leduc Hold'em in this experiment makes it possible to show results under ideal modelling conditions. In addition, the BRPlayer's opponent model explicitly models the probabilities of each chance node outcome rather than simply assuming possible chance node outcomes happen with uniform probability.

The BRPlayer's opponent model contains simplistic defaults that are used only when no other modelling information is present and are no longer used as soon as a single observation is made. This is a drastic shift in opponent modelling information and in almost all cases a single data point would not provide good opponent modelling information. Since only long term results are the focus of this experiment, more sophisticated opponent modelling is not needed because eventually enough data will be collected to correct problematic information in the opponent model.

For showdown leaf nodes, before any showdown observations are made, the BRPlayer's hand rank is used to estimate its chance of winning at a showdown. For chance nodes, before any observations are made all possible chance node outcomes are assumed to occur with uniform probability. For opponent action nodes, before the opponent is observed making any action at one of their information sets, they are assumed to call 80% of the time, and raise the remaining 20% when they are allowed to raise, or call 100% of the time if they are not allowed to raise. The BRPlayer updates its model of the opponent after every hand, and its model is completely reset at the start of every match trial so that no modelling information carries over between trials.

Within the BRPlayer's action-selection search, the BRPlayer only uses "max backups" to compute values for each of its information sets. This allows the BRPlayer to always know which action and information set currently looks the best according to its opponent model. The action-selection search is invoked the first time the BRPlayer has to choose an action in a hand, and the values of each action are cached so as to save having to perform future searches for future decision points within the hand. When it comes time to actually take an action in the game, the BRPlayer considers the value of each of its available actions and then chooses the maximum valued action 90% of the time and takes an exploratory action the remaining 10% of the time. This action-selection mechanism is a form of soft-max that ensures all actions get tried to make sure an accurate model of the opponent is learned for all possible game situations. For the BRPlayer, its exploratory actions can only ever be a check/call or bet/raise and are never folds (the outcome of folding is always known exactly and never needs to be explored).

Since the purpose of this experiment is to show that the BRPlayer's strategy evolves into a best-response strategy given enough game experience, the BRPlayer outputs what it believes to be its best possible strategy against its current opponent model after every 25,000 hands (and it outputs its strategy before a single hand is played so that its default strategy is recorded as well). To do this, the BRPlayer performs its normal action-selection search that it uses to choose its actions every hand. At every one of its information sets it records which available action has the highest value. These recorded actions then correspond to what the player believes to be its best behavioral strategy against its opponent.

These strategy snapshots for the BRPlayer can then be used in conjunction with the opponent's known behavior strategy to compute the expected value of each strategy when matched up against each other. To do this, the extensive form game tree is used to backup values from leaf nodes at the end of the game, where payoffs are known, to the start of the game. Since the game is zero-sum, the backups can be done strictly from one strategy's perspective and the value of the other strategy can then be obtained by just negating the result. The backup procedure at every player decision node is an expected value over the values of each action possible there, given the likelihood that the strategy employed by that player would choose each of those actions. The backup procedure at the chance nodes is also computed as an expected value over the value of each subtree corresponding to each chance outcome given the likelihood of the chance outcomes (which comes directly from the rules of the game).

The expected values of the BRPlayer's strategy snapshots against its opponent's strategy are then averaged across each of the twenty trials to show a plot of how the BRPlayer's strategy improves after every 25,000 hands of experience. It is important to note that in every 25,000 hands of experience, each player plays 12,500 hands as the first player and 12,500 hands as the second player since each player switches being the dealer (and the consequently, the second to act) after every hand in the match. To separate how the BRPlayer's strategy is evolving when they are first to act (i.e., not the dealer) and when they are second to act (i.e., the dealer), the results of each are plotted separately. In addition, the expected value of a best-response strategy (BRV, in the subsequent graphs) against the opponent's known strategy is also plotted for each playing position. The expected value of the best-response strategy was precomputed prior to the match[2] and graphically represents what the expected value of the BRPlayer's learned strategy should approach and eventually equal given sufficient experience.

5.1.3 Results Against CallPlayer

Figure 5.1 contains the results for the BRPlayer against the CallPlayer, whose strategy is to only check and call. Against the CallPlayer, a best-response strategy has an expected value of 1.466667 bets/hand for both the first to act and second to act positions.

The BRPlayer starts out with a default model for this type of opponent that actually results in a best-response strategy even though the model is not completely accurate. Since the CallPlayer does not exhibit any sort of hand selection, at any showdown they have an equally likely chance to hold any card that is unknown to the BRPlayer at that showdown (i.e., any card that is not the BRPlayer's hole card or the community card). The BRPlayer's use of its own hand rank to estimate its chance of winning a showdown models this exactly. In addition, this lack of hand selection for the CallPlayer also makes the BRPlayer's default assumption correct that at any chance node at the start of the second round, each community card will be observed with equal probability. The BRPlayer's default model for its opponent's observed action frequencies is incorrect since it assumes the CallPlayer will raise 20% of time when a raise is possible even though the CallPlayer will never do that. This systemic error for this particular opponent, however, does not affect the relative values of the different actions that the BRPlayer has to choose between at an information set enough to actually cause the BRPlayer to choose inferior actions.

Looking at Figure 5.1 as the BRPlayer plays more games, the quality of its strategy snapshots first decreases and then increases steadily until it settles on a best-response strategy (first achieved after 200,000 hands and maintained for all subsequent snapshots).

[2] This strategy and its associated value can be computed using an expectimax backup procedure as described in Chapter 3, after converting the opponent's known behavior strategy into an observation model as described in Chapter 4.

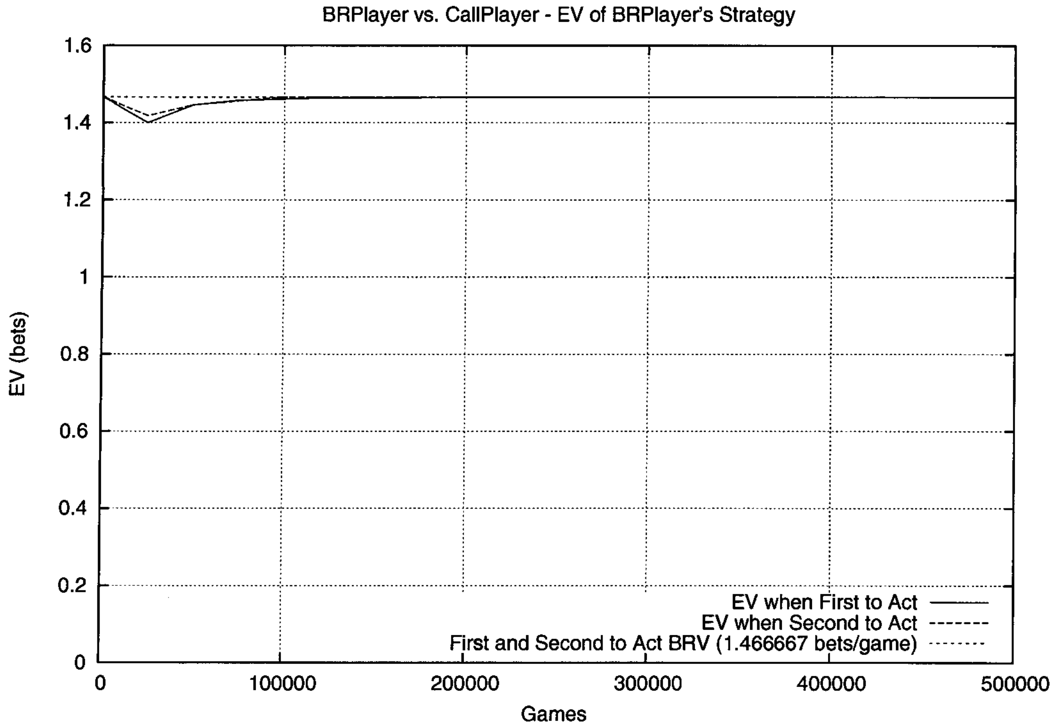

Figure 5.1: BRPlayer vs. CallPlayer

The quality of the BRPlayer's preferred strategy initially starts to decrease since it immediately starts to switch to its opponent modelling information as soon as any information is available. While switching after a single data point is fine for estimating opponent action frequencies in this particular case, it hurts the model's accuracy when predicting the BRPlayer's chances of winning a showdown and when predicting the chance of a chance outcome occurring at a chance node. Eventually, though, as more and more games are played, the accumulation of data in the model increases the model's accuracy which also causes the quality of the BRPlayer's strategy snapshots to increase.

5.1.4 Results Against RaisePlayer

Figure 5.2 contains the results for the BRPlayer against the RaisePlayer, whose strategy is to raise whenever it is legal to do so and call otherwise. The best-response strategy against the RaisePlayer has an expected value against it of 2.366667 bets/hand in both the first to act and second to act positions.

The results in Figure 5.2 are similar to the results in Figure 5.1. In both figures, the initial strategy the BRPlayer chooses in response to its default model is a best-response strategy against its opponent's strategy. Then as the BRPlayer gains more experience, like in the matches against the CallPlayer, the quality of its strategy snapshots first decrease

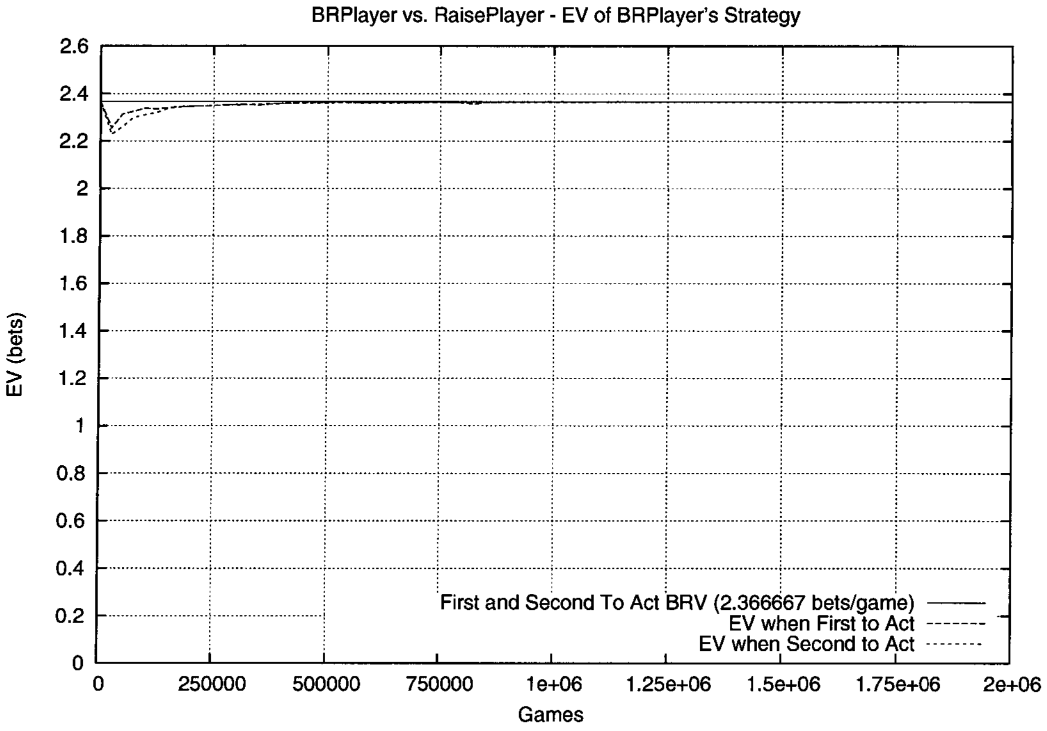

Figure 5.2: BRPlayer vs. RaisePlayer

while relying on its experience-based model that takes some time to become accurate and then increase until the quality reaches the expected value of a best-response strategy.

Though the CallPlayer and the RaisePlayer choose completely different actions, the RaisePlayer's strategy, like the CallPlayer's, is not based on hand selection. As a result, the BRPlayer's default model perfectly predicts both the chance of winning a showdown and the probability of chance outcomes against the RaisePlayer. Also, as was the case with the CallPlayer, the BRPlayer's default model does not perfectly predict the RaisePlayer's observed action probabilities since it assumes the RaisePlayer will call 80% and raise 20% of the time when it will actually always raise when possible. Despite this systemic modelling error, the relative values of the actions the BRPlayer has to choose are still correct enough for a best-response strategy to be chosen.

Despite the similarities in the results, there are also two obvious differences: the experience needed before the BRPlayer adopts a best-response strategy, and the actual best-response value. Comparing the results in Figure 5.2 with Figure 5.1, it takes longer for the BRPlayer to learn a best-response strategy through experience against the RaisePlayer than against the CallPlayer. For example, against the RaisePlayer it took 1,825,000 hands of experience for the value of the BRPlayer's strategy snapshots to converge to the best-response value, while it only took 200,000 hands to do the same against the CallPlayer.

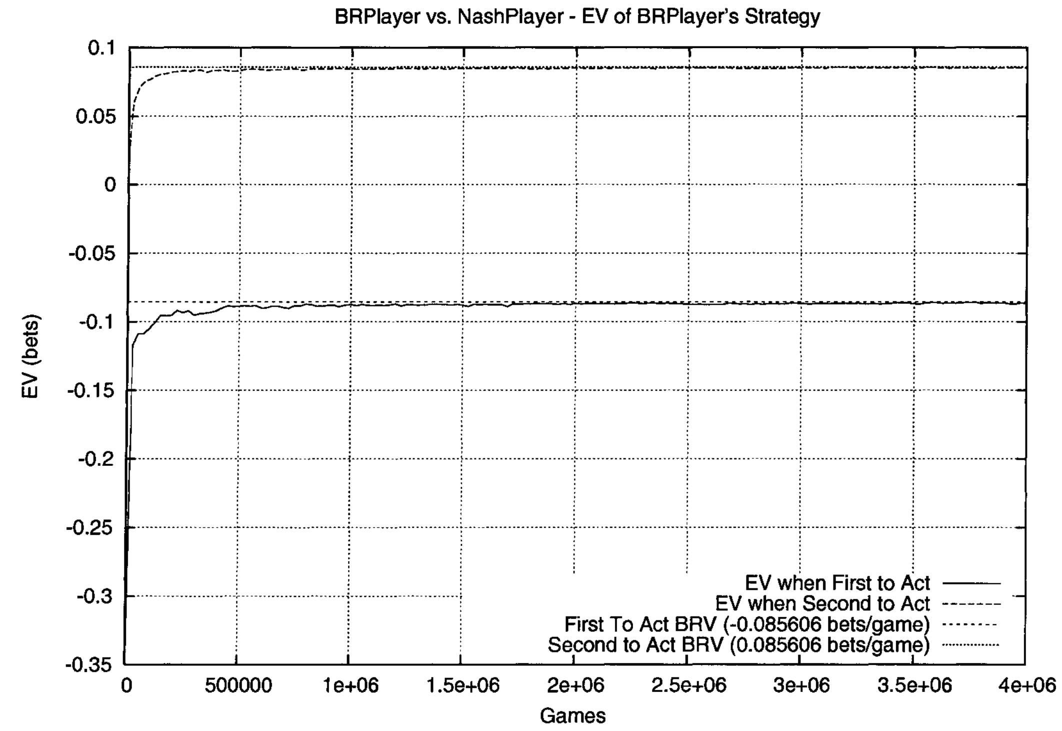

Figure 5.3: BRPlayer vs. NashPlayer

This difference in necessary experience is to be expected since the RaisePlayer's strategy naturally leads to more decisions for the BRPlayer than the CallPlayer's strategy does. For example, anytime the RaisePlayer raises when it could have called, the BRPlayer is faced with making an additional decision that it would never have seen against the CallPlayer. As for the actual best-response value, it is much higher against the RaisePlayer than the CallPlayer since the RaisePlayer will consistently put more money in the pot with its raising.

5.1.5 Results Against NashPlayer

Figure 5.3 shows the BRPlayer's results against the NashPlayer for 20 averaged trials of 4,000,000 hand matches. In addition, Figure 5.4 is also provided to give a more detailed view of the results for the early stages of the matches where the strategy snapshots improve the fastest.

The NashPlayer employs a game-theoretic optimal strategy for the game of Leduc Hold'em that was solved via linear programming using the techniques described in [29]. By employing this strategy, the NashPlayer is guaranteed that the best any opponent could do against them would be to win on average 0.085606 bets/hand when they are the second player to act and to lose on average 0.085606 bets/hand when they are the first player to act.

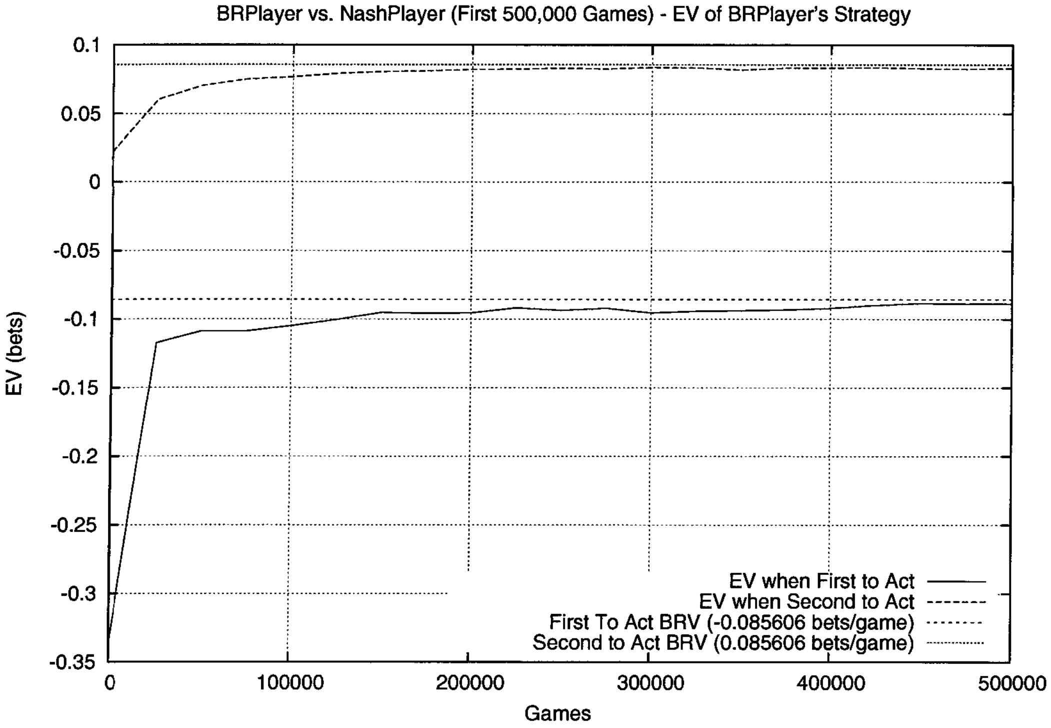

Figure 5.4: BRPlayer vs. NashPlayer (First 500,000 Games)

The BRPlayer uses the same default opponent model for this match as it did in its matches against the CallPlayer and the RaisePlayer. However, unlike in the matches with the CallPlayer and the RaisePlayer, this default model proved to be poor against the NashPlayer as the BRPlayer's default strategy snapshot loses -0.3359 bets/hand when first to act, and wins 0.02178 bets/hand when second to act.

As the BRPlayer's experience increases, its model of the NashPlayer becomes more accurate and the expected value of BRPlayer's strategy snapshots increase towards the best-response values. The final expected value of the BRPlayer's averaged strategy snapshots very closely approached the known best-response values. For example, the final strategy snapshot values taken after 4,000,000 hands of experience and averaged over 20 trials ended up with the BRPlayer losing at a rate of -0.08638 bets/hand when first to act (which is 0.000774 bets/hand worse than the best-response value), and winning at a rate of 0.08512 bets/hand when second to act (which is 0.000486 bets/hand worse than the best-response value).

5.2 Texas Hold'em Results

The main purpose of this second set of experiments is to evaluate the ideas presented in this thesis in the domain of Texas Hold'em. Since Texas Hold'em is a large real-world domain, these experiments are meant to test how practical the ideas presented in this thesis are for real-world problems.

In this set of experiments, a program implementing the ideas in this thesis, the BRPlayer, plays matches against the University of Alberta Computer Poker Research Group's two current state of the art opponents mentioned previously:

- **Poki -** Poki[17] was designed to play full-ring game poker. As a result, it has some weaknesses when playing heads-up.
- **PsOpti -** PsOpti[6] was specifically designed to play heads-up poker. It's strategy was computed using game-theoretic analysis and is designed to be difficult to beat. It's considered to have an advanced level of playing strength.

Because Texas Hold'em is a large domain, the BRPlayer used in these experiments implements the ideas presented in Section 3.7 and in Section 4.3 as a means of coping with the additional computational challenges that appear as a domain gets larger. These same ideas were not implemented in the BRPlayer in the Leduc Hold'em experiments presented earlier since that domain is so much smaller.

The ideas in Section 3.7 were implemented to ensure that the BRPlayer's action-selection search procedure executes quickly enough to allow the BRPlayer to act in a realistic amount of time (approximately one or two seconds per action). This is important because it means that the computer players acting in the matches in these computer experiments are subject to the same time constraints as humans are subject to when playing their matches. The ideas in Section 4.3 were implemented to ensure that the BRPlayer's opponent model is practical for normal computer hardware (i.e. does not take up too much memory and can return necessary information quickly enough) and to also try to help lessen the time it takes for the BRPlayer to build an accurate opponent model.

These matches were setup so that after every hand the player designated as the dealer switches from one player to the next. This simulates how matches are played in a casino. Also, each computer player starts each match with their initial default state. No experience from a previous match in used in a subsequent one. For both the BRPlayer and for Poki, this means they start each match assuming their opponent plays according to their default opponent model.

5.2.1 Results Against Poki

Figure 5.5 shows three different 40,000 hand matches between the BRPlayer and Poki. All three matches were stopped after 40,000 hands because by that point it is clear that the BRPlayer is able to exploit weaknesses in Poki's heads-up strategy. In fact, in matches 1

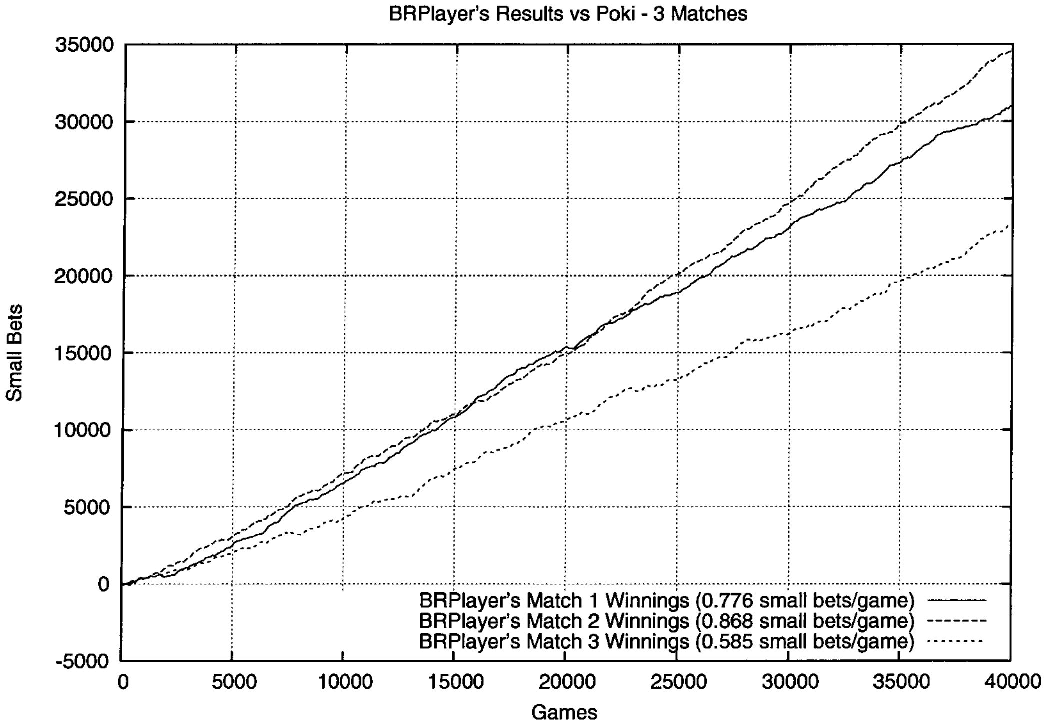

Figure 5.5: BRPlayer vs. Poki - 3 Matches

and 2, over the course of the 40,000 hand matches Poki would have lost less money to the BRPlayer if it had folded every hand.[3]

To show more detail in the early stages of the same matches, Figure 5.6 is also provided. This figure zooms in to show only the results from the first 5,000 hands of each match shown in Figure 5.5. Since poker matches played against humans are always much shorter than matches that can be run between computer opponents, it is important to consider these early results. Though there are not enough samples to draw definitive conclusions, these graphs appear to indicate that the BRPlayer is able to start beating Poki after around 500 games of experience and prior to that slightly loses or breaks even.

Though the three independent matches give a sense of how the BRPlayer does against Poki, more matches would need to be run to accurately determine the BRPlayer's average win rate. In addition, to estimate the BRPlayer's maximum win rate against Poki, these matches would likely have to extend past 40,000 hands in order to see if the win rate is still changing. Since both of these tasks require significant additional computation to complete, they are left to future work.

[3] Folding every hand would correspond to losing at a rate of 0.75 small bets/hand (where a small bet is the size of a bet or raise in the preflop and flop betting rounds in limit Texas Hold'em).

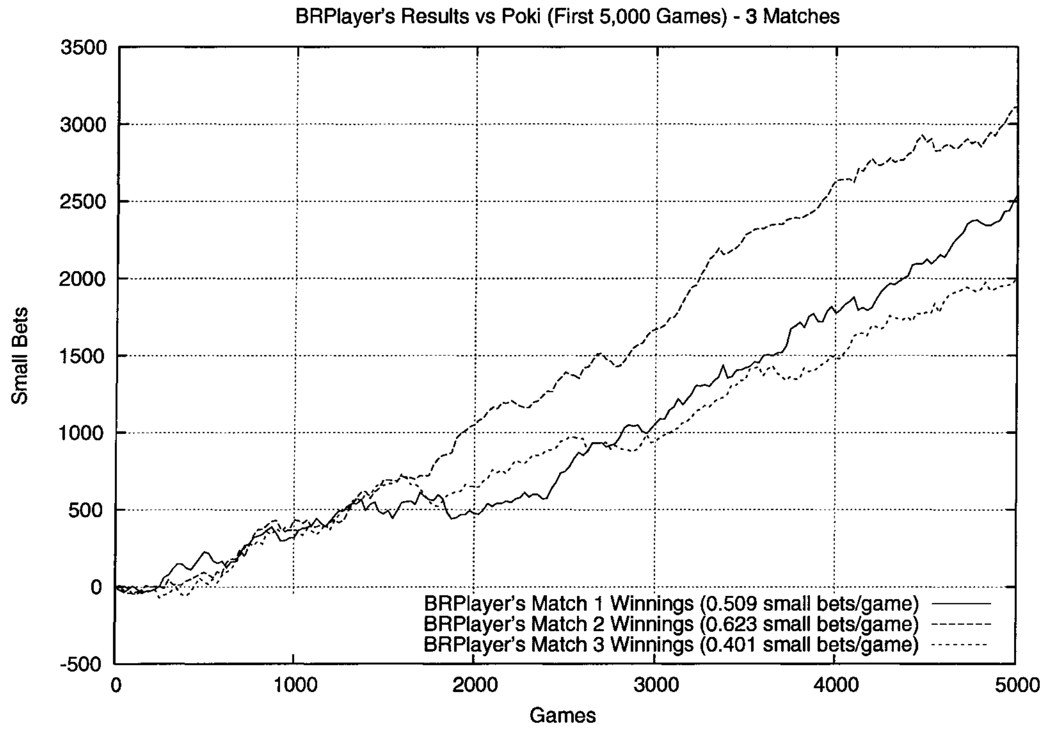

Figure 5.6: BRPlayer vs. Poki (First 5,000 Games) - 3 Matches

5.2.2 Results Against PsOpti

Figure 5.7 shows three different 500,000 hand matches between the BRPlayer and PsOpti4. PsOpti4 is a newer version of the PsOpti programs described to in [6]. PsOpti4 was constructed according to the same process as PsOpti2 but used an expert crafted preflop strategy instead of PsOpti2's preflop strategy that was computed via solving a three round preflop model. The hand crafted preflop strategy for PsOpti4 was designed to be mixed to help disguise its play and make it more difficult to learn against. Like PsOpti2, PsOpti4's preflop strategy is used for both actual preflop play and also to generate the conditional probabilities that serve as input for the computation that solves its different postflop models.

The three matches against PsOpti4 are significantly longer than the matches against Poki. The Poki matches were stopped after 40,000 hands because by then it was clear that the BRPlayer had adopted an effective playing strategy capable of exploiting Poki. In the matches against PsOpti4, the BRPlayer had a much more difficult time learning a winning playing strategy and the matches were allowed to continue for 500,000 hands in order to show that the BRPlayer could eventually adopt a winning playing strategy given enough experience. Though the final average winning rate is quite different in all three matches, the upward curving shape of the BRPlayer's plotted winnings is present in all three matches and show that the BRPlayer's strategy is improving as more game experience is obtained.

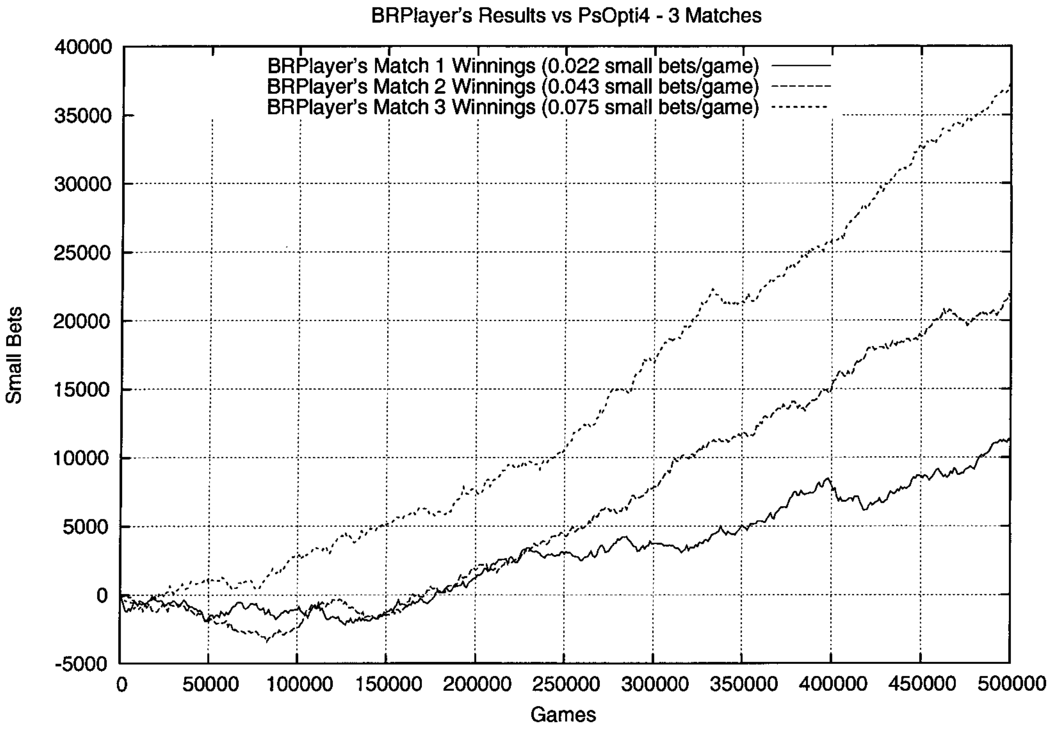

Figure 5.7: BRPlayer vs. PsOpti4 - 3 Matches

It is disappointing that the BRPlayer takes so long to adopt an effective strategy against PsOpti4. After the first 5,000 hands of each of the three matches, as seen in Figure 5.8, the BRPlayer is losing in all three matches. After 50,000 games, as seen in Figure 5.9, the BRPlayer is winning in the third match, but losing in the first two. Having only three matches makes it difficult to make any conclusive statements about how long it actually takes the BRPlayer to finally adopt a winning strategy against a tough opponent like PsOpti4. Given the current implementation of the BRPlayer, it looks like PsOpti4 would have an advantage in a short-term match and that the BRPlayer can eventually exploit PsOpti4 in the long-term.

At the beginning of the match, the BRPlayer only has its default model. When this model is inaccurate and the BRPlayer adopts a poor strategy based on it, the BRPlayer's performance will be poor until it can learn a more accurate model from experience. To explore this further, consider Figure 5.10. This figure shows how the BRPlayer would do in the same matches when it relies only on its default model and none of its actual game experience against PsOpti4. Comparing Figure 5.9 to Figure 5.10, it appears the strategy the BRPlayer adopts with less than 10,000 hands of experience achieves roughly the same performance as the strategy it would adopt if it only used its default model. As more experience is gathered, the BRPlayer's strategy gradually improves.

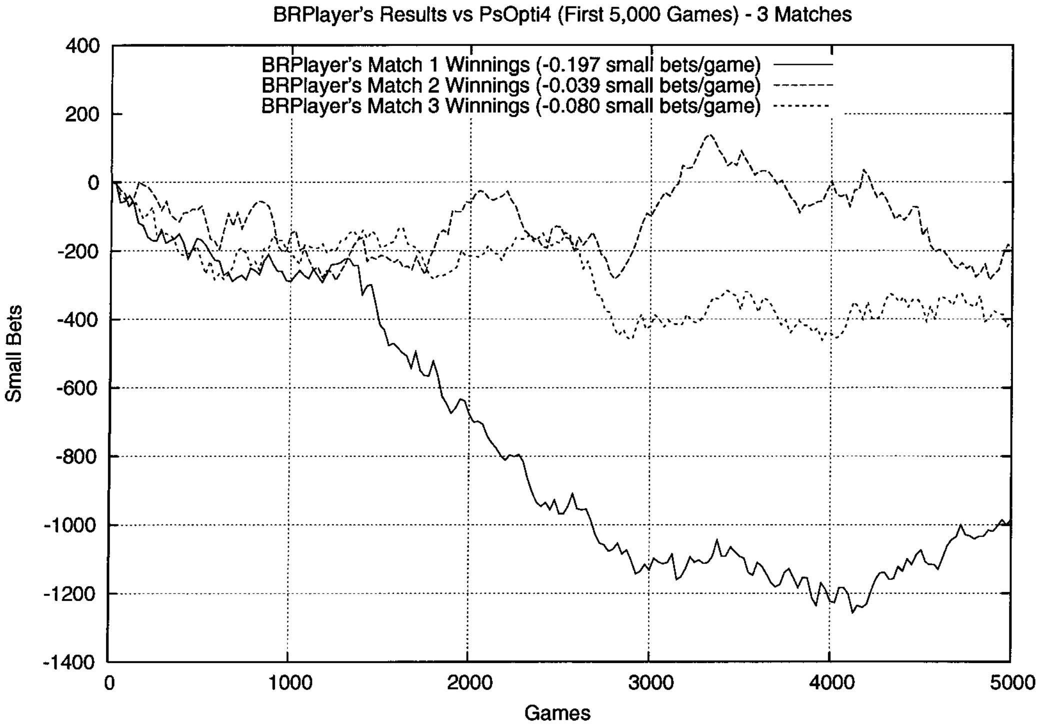

Figure 5.8: BRPlayer vs. PsOpti4 (First 5,000 Games) - 3 Matches

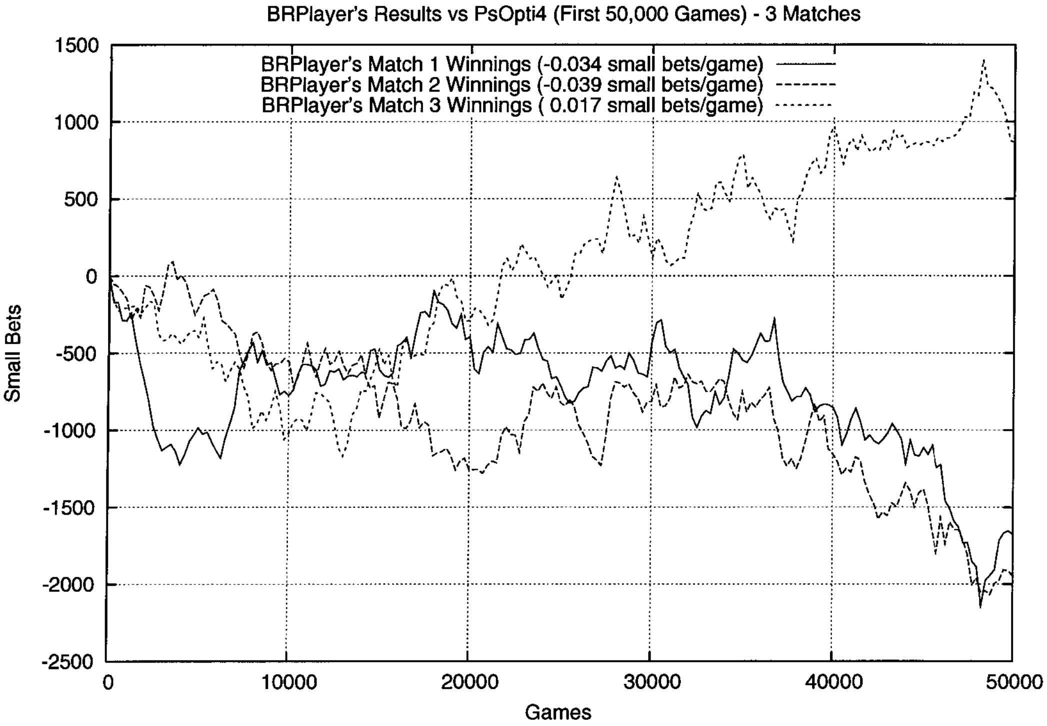

Figure 5.9: BRPlayer vs. PsOpti4 (First 50,000 Games) - 3 Matches

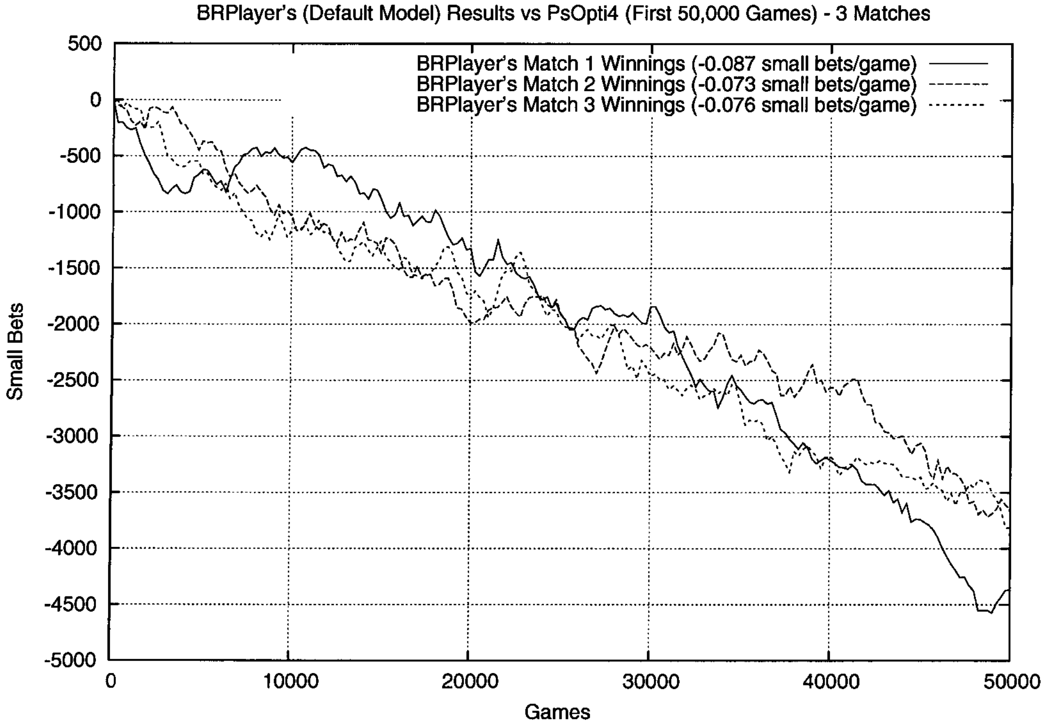

Figure 5.10: BRPlayer (Default Model) vs. PsOpti4 (First 50,000 Games) - 3 Matches

To be more effective, the BRPlayer must be able to adopt a winning strategy more quickly and this weakness definitely has to be addressed in future work. There are many possible reasons why the BRPlayer could have difficulty learning against a tough opponent like PsOpti4 and it is difficult to isolate if it is just one thing or a combination of things.

One possible reason could be due to ineffective exploration. If the BRPlayer does not sufficiently explore different alternative actions against its opponent, it can continue playing according to a strategy that is based on an incorrect model. With the current system, an incorrect model can only be corrected through additional observation of data. On the other hand, if the BRPlayer performs too much exploration, its performance also suffers because every time it explores a bad action, though it learns new information, it gives up the potential profit made available by choosing a better action.

Another possible reason for the BRPlayer's slow learning could be due to the way it uses data to build its model. In the current system, the BRPlayer's model relies on counts of observations to form an opponent model. This approach is powerful since it does not assume much about the opponent (just some basic assumptions about which observations can be generalized together in a naive attempt to speed up learning). However, waiting for enough observations to create an effective opponent model may take too much time for complex opponent like PsOpti4.[4] The model learning process could be potentially sped up if the BRPlayer used the observations as a means of choosing between different opponent types that could be characterized by a few parameters (like, the work in [51]). This, of course, more strongly biases the types of opponent models that can be learned, but this bias could result in faster learning.

5.2.3 Previous Related Results

The BRPlayer used in these experiments closely resembles the program *Vexbot* discussed in [8]. Both of these programs share the same action-selection search procedure. They differ only in the type of opponent model used. Both the BRPlayer and Vexbot use a context tree for modelling their opponent's action frequencies, and they both assume chance node outcomes occur with uniform probability. The difference between the two player's models lies in how they model their opponent's showdown information.

Aside from being separate opponent model implementations,[5] Vexbot's showdown modelling, when considered from a high level, corresponds to the BRPlayer's model with more coarse-grained generalization. Using terms described in Section 4.5, Vexbot's showdown modelling essentially grouped showdown observations into predefined groups according to whether they were deemed to have an s1, s2, or context tree level of similarity. The data from the different groups was then weighted and combined to form the showdown data requested from the opponent model.

It is difficult to directly compare Vexbot's results in [8] to the BRPlayer's results presented here. Both Vexbot and the BRPlayer were able to significantly exploit weaknesses in Poki's play. Comparing the averaged winning rate of the three 40,000 hand matches presented above, 0.743 small bets/hand, to the winning rate of Vexbot reported in [8], 0.601 small bets/hand, it appears that the BRPlayer is able to beat Poki at a higher rate. However, neither experiment performed enough trials of these matchups to be able to conclusively say which player was able to exploit Poki better.

The results against PsOpti4 are even more difficult to compare. The implementation of PsOpti4 used in the experiments presented in this thesis fixed a bug that was present in the implementation used in [8]. This bug fix appeared to improve the playing strength of PsOpti4 because the BRPlayer presented in this thesis went from being able to beat the bugged version of PsOpti4 at an average rate of 0.165 small bets/hand over three matchups of 40,000 hands, to losing at a rate of -0.0086 small bets/hand over the same three matchups of 40,000 hands. Comparing the BRPlayer's average winning rate of 0.165 small bets/hand

[4] In addition, as mentioned in Chapter 4, this problem can be further compounded by the fact that the observation class of models can unfortunately be misled to predict that its opponent can behave in a way that is not actually possible in the game when the model only contains a small random sampling of observations.

[5] Vexbot's model explicitly created predefined classes that grouped similar observations together as they were observed.

against the version of PsOpti4 containing a bug to Vexbot's winning rate of 0.052 small bets/hand suggests the BRPlayer was able to exploit that particular player better, though more matchups are needed to be able to conclusively say that.

Experiments were never setup to test the BRPlayer's performance against human competition. Arranging experiments of this type is difficult to do due to a lack of interested participants, the difficulty of being able to accurately determine the skill level of the participants, and the difficulty of getting participants to play a significant number of hands so that some sort of reliable conclusions can be drawn.

The only experimental evidence that suggests how the BRPlayer might do against human competition comes from a few short matches. Unfortunately, because these matches were so short and so few, they really only serve as anecdotal evidence and should not be used to draw any significant conclusions. In four matches reported in [8], Vexbot beat three intermediate players but lost to an expert player. At the *World Poker Robot Championship* held July, 2005 in Las Vegas, the BRPlayer was a major component in the University of Alberta poker program, *Poki-X*, which lost a short 290 hand match against poker professional, Phil Laak[1]. Setting up matches against humans and seeing how the BRPlayer performs in those matches is an interesting task for the future.

Chapter 6

Conclusions

Poker is a challenging domain that contains both elements of chance and imperfect information. As a result, it serves as a good testbed for exploring decision-theory, probabilistic reasoning, risk assessment, and opponent modelling. The challenges present in poker are also present in many other real-world applications such as user modelling, policy-making or negotiation, and online auctions.

Over the years poker has received some attention from computer scientists. The University of Alberta Computer Poker Research Group has done a lot of work in the domain including the development of two substantial computer poker players:

- **Poki -** Poki[17] was designed to play full-ring game poker. Over the years it has played on the IRC poker channel and the University of Alberta's own public server. Its results while playing in these games indicate that Poki plays full-ring game poker at an intermediate level of playing strength. Unfortunately, Poki's weaknesses increase as the number of players in the game decrease.

- **PsOpti -** PsOpti[6] was specifically designed for heads-up poker play. Its strategy was computed using game-theoretic analysis and is designed to be difficult to beat. PsOpti is considered to have an advanced level of playing strength and was competitive in a 7000 hand match played against a world-class opponent.

Though this prior work has substantially advanced the state of the art of computer poker players, there is still one major difficulty hindering the development of a world-class calibre player. This is the task of of learning how an opponent plays (i.e. opponent modelling) and subsequently coming up with a strategy that can exploit that information. The work in this thesis explores this task and applies the techniques to the domain of heads-up limit Texas Hold'em.

The computer program implemented in this thesis consists of two main components: heuristic search for action-selection, and a model of the opponent's play. The basics for a computer poker player utilizing these components was first described by Davidson in [17] but were only briefly explored there. This thesis presents the ideas in more depth and provides a separate implementation with improved results.

The heuristic search action-selection procedure implemented here is an example of expectimax search. It takes opponent modelling information and plans a strategy that exploits that information. The opponent modelling information takes the form of observations that the computer player observes in actual game play. These observations include the opponent's actions, the cards dealt at chance nodes, and the cards revealed by the opponent at showdowns. These observations are added to the model after each game is played.

To apply the ideas to Texas Hold'em, approximations were necessary to keep them practical. Texas Hold'em is too large to do a full-depth full-width action-selection search, so the search procedure had to be approximated using sampling. The size of Texas Hold'em also means that generalization had to be added to the opponent model. To do this, instance-based learning was introduced to lessen the number of games needed to learn an effective opponent model.

Overall, the resulting computer player implemented using the ideas in this thesis contains minimal expert knowledge. The expert knowledge that is present in the system consists of the initial defaults used when there is no opponent modelling information available, how information is generalized in the opponent model, and how to sample within the action-selection search.

The ideas in this thesis were evaluated in two different sets of experiments. In the first set, the simplified poker domain of Leduc Hold'em was used. This domain is small enough that the ideas could be tested under ideal circumstances. That is, a complete full-depth full-width action-selection search can be used along with an opponent model utilizing no generalization. In the second set, the evaluation was done using Texas Hold'em. Texas Hold'em presents an opportunity to see how the ideas with their approximations perform in a real-world domain.

In the Leduc Hold'em experiments, the player implementing the ideas in this thesis played against three different static opponents to show how the player's strategy improved as the matches proceeded. Against the first two simplistic opponents, one that always called and one that always raised, the player learned a best-response strategy. Against the third opponent, one that played a game-theoretic optimal strategy, the player's learned strategy very closely approached a best-response strategy.

In the Texas Hold'em experiments, the player implementing the ideas in this thesis was evaluated against both Poki and PsOpti. The results against Poki were quite impressive: in two out of three 40,000 hand matches, it beat Poki at a rate worse than if Poki folded every hand. The results against PsOpti were less impressive. Though the player implementing the ideas in the thesis eventually learned to beat PsOpti in each of the three matches played, it took too long for this to occur.

6.1 Future Work

Overall, the ideas presented in this thesis seem to be a promising step forward towards being able to learn and exploit opponent models in heads-up limit Texas Hold'em. There is, however, interesting work that still remains to be done to extend these ideas into a world-class calibre computer poker player. This future work includes:

- **faster opponent modelling** - The current opponent modelling system does not learn fast enough. To compete at a world-class level, an effective model needs to be formed in as little as 100 hands. One potential way to speed up learning is to use the observations to choose from previously constructed or parameterized opponent models rather than building opponent models out of the observations. This type of approach was explored in [51] and appears promising.

- **addressing the exploration/exploitation tradeoff** - The action-selection search implemented in this thesis hardly addresses this interesting tradeoff. Knowing when to explore to improve the current model and when to exploit is a very interesting problem. A proper exploration scheme helps find an effective model while trying to avoid suboptimal results due to excess exploration. Against a learning opponent, the issue of exploiting them so much that they learn to correct their play also becomes an issue.

- **handling changing opponents** - The current implementation does not address how to handle an opponent that changes their strategy. This is certainly a characteristic present in world-class calibre poker players so it will need to eventually be addressed.

- **extending to multi-player** - The current system only plays heads-up Texas Hold'em. To extend these ideas to multi-player poker, many more approximations need to be added to the ideas in this thesis to help them remain computationally practical.

- **better default models** - The current system stands to gain a lot from simply having a better default model. Having a better default model would help the player implementing the ideas in this thesis lose less at the beginning of a match when it hardly has any opponent modelling information.

Bibliography

[1] Results from the World Poker Robot Championship 2005. 2005. http://www.poker-academy.com/wprc/.

[2] Bruce W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[3] D. Billings. Computer poker. Master's thesis, University of Alberta, 1995.

[4] D. Billings. The first international RoShamBo programming competition. *International Computer Games Association Journal*, 23(1):42–50, 2000.

[5] D. Billings. Thoughts on RoShamBo. *International Computer Games Association Journal*, 23(1):3–8, 2000.

[6] 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 661–675, 2003.

[7] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[8] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Computers and Games (CG '04)*, 2004. To appear.

[9] D. Billings, D. Papp, L. Pena, J. Schaeffer, and D. Szafron. Using selective-sampling simulations in poker. In *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, pages 13–18, 1999.

[10] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *AAAI National Conference*, pages 493–499, 1998.

[11] D. Billings, L. Pena, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *AAAI National Conference*, pages 697–703, 1999.

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

[13] M. Campbell, A. J. Hoane, and F. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[14] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *AAAI National Conference*, pages 120–125, 1995.

[15] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[16] F.A. Dahl. A reinforcement learning algorithm applied to simplified two-player Texas Hold'em poker. In *12th European Conference on Machine Learning (ECML'01)*, pages 85-96, 2001.

[17] A. Davidson. Opponent modeling in poker: Learning and acting in a hostile and uncertain environment. Master's thesis, University of Alberta, 2002.

[18] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence (IC-AI'2000)*, pages 1467–1473, 2000.

[19] H.H.L.M. Donkers. *Nosce Hostem - Searching With Opponent Models.* PhD thesis, Universiteit Maastricht, 2003.

[20] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, and H.J. van den Herik. Probabilistic opponent-model search. *Information Sciences*, 135(3–4):123–149, 2001.

[21] N. Findler. Studies in machine cognition using the game of poker. *Communications of the ACM*, 20(4):230–245, 1977.

[22] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *International Joint Conference on Artificial Intelligence*, pages 584–589, 1999.

[23] M. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

[24] T. Hauk. Search in trees with chance nodes. Master's thesis, University of Alberta, 2004.

[25] H. Iida, J. Uiterwijk, J. van den Herik, and I. Herschberg. Thoughts on the application of opponent-model search. In H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk, editors, *Advances in Computer Chess 7*, pages 61–78. Univ. of Limburg, Maastricht, 1994.

[26] P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search.* PhD thesis, Carnegie-Mellon University, 1992.

[27] P. Jansen. KQKR: Speculatively thwarting a human opponent. *International Computer Chess Association Journal*, 16(1):3–17, 1993.

[28] A. Junghanns. Are there practical alternatives to alpha-beta? *ICCA Journal*, 21(1):14–32, 1998.

[29] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *26th Annual ACM Symposium on the Theory of Computing*, pages 750–759, 1994.

[30] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, 1997.

[31] K. Korb and A. Nicholson. Bayesian poker. In *Uncertainty in Artificial Intelligence*, pages 343–350, 1999.

[32] H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 97–103. Princeton University Press, 1950.

[33] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, 1968.

[34] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[35] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[36] J. F. Nash and L. S. Shapley. A simple three-person poker game. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 105–116. Princeton University Press, 1950.

[37] D. Papp. Dealing with imperfect information in poker. Master's thesis, University of Alberta, 1998.

[38] L. Pena. Probabilities and simulations in poker. Master's thesis, University of Alberta, 1999.

[39] A. L. Reibman and B. W. Ballard. Non-minimax search strategies for use against fallible opponents. In *AAAI National Conference*, pages 338–342, 1983.

[40] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2nd edition, 2003.

[41] J. Schaeffer, D. Billings, L. Pena, and D. Szafron. Learning to play strong poker. *Workshop on Machine Learning in Game Playing at the Sixteenth International Conference on Machine Learning (ICML-99)*, 1999.

[42] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–290, 1992.

[43] A. Selby. Optimal heads-up preflop poker. 1999. http://www.archduke.demon.co.uk/simplex/.

[44] S. Sen and N. Arora. Learning to take risks. *Working Papers of the AAAI-97 Workshop on Multiagent Learning*, pages 59–64, 1997.

[45] B. Sheppard. Computer Scrabble. *Artificial Intelligence*, 134(1-2):241–275, 2002.

[46] J. Shi and M. Littman. Abstraction models for game theoretic poker. In *Computer Games'00*, pages 333–345. Springer-Verlag, 2001.

[47] D. Sklansky. *The Theory of Poker.* Two Plus Two Publishing, 1992.

[48] D. Sklansky and M. Malmuth. *Hold'em Poker for Advanced Players.* Two Plus Two Publishing, 2nd edition, 1994.

[49] S. Smith. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI*, pages 422–425, 1983.

[50] Wilson Software. Turbo Texas Hold'em. http://wilsonsoftware.com.

[51] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. In *21st Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, pages 550–558, 2005.

[52] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[53] K. Takusagawa. *Nash Equilibrium of Texas Hold'em Poker.* Undergraduate thesis, Stanford University, 2000.

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

[55] G. Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199, 2002.

[56] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, 2nd edition, 1947.

[57] D. Waterman. A generalization learning technique for automating the learning of heuristics. *Artificial Intelligence*, 1:121–170, 1970.