University of Alberta

ALGORITHMS AND ASSESSMENT IN COMPUTER POKER

by

Darse Billings ©

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

Department of Computing Science

Edmonton, Alberta
Fall 2006

Your file Votre référence
ISBN: 978-0-494-22991-0
Our file Notre référence
ISBN: 978-0-494-22991-0

Abstract

The game of poker offers a clean well-defined domain in which to investigate some truly fundamental issues in computing science, such as how to handle deliberate misinformation, and how to make intelligent guesses based on partial knowledge. In the taxonomy of games, poker lies at the opposite end of the spectrum from well-studied board games like checkers and chess. Poker is a *multi-player* game with *stochasticity* (random events occurring over a known probability distribution), *imperfect information* (some information is hidden from each player), and *partially observable outcomes* (some information might never be known). Consequently, the use of deception, opponent modeling, and coping with uncertainty are indispensable elements of high-level strategic play. Traditional methods for computer game-playing are incapable of handling these properties.

Writing a program to play a skillful game of poker is a challenging proposition from an engineering perspective as well, since each decision must be made quickly (typically about one second). A major theme of this dissertation is the evolution of architectures for poker-playing programs that has occurred since the research began in 1992. Four distinct approaches we address are: knowledge-based systems, simulation, game-theoretic methods, and adaptive imperfect information game-tree search. Each phase of the research explored the strengths and weaknesses of the corresponding architectures, both in theory and in practice. The important problem of obtaining an accurate assessment of performance is also addressed. The creation of a powerful tool for this purpose, called DIVAT, is discussed in detail. The academic papers arising from each of these studies constitute the core chapters of this thesis. The conclusion discusses some of the advantages and shortcomings of each approach, along with the most valuable lessons learned in the process.

Although the goal of surpassing all human players has not yet been attained, the path has been cleared. The best poker programs are beginning to pose a serious threat, even in this most "human" of games. As programs continue to improve, they provide new insights into poker strategy, and valuable lessons on how to handle various forms of uncertainty.

Acknowledgements

I have been truly privileged to work closely with a large team of researchers for many years. In fact, it is fair to say that I have been spoiled, having so much support in developing and implementing my ideas. Consequently, I have many people to thank, and cannot possibly do an adequate job of expressing my gratitude in just a few pages.

I could not have asked for a better thesis committee – their feedback and guidance has simply been outstanding.

Jonathan Schaeffer is the best supervisor I could have ever asked for. He gave me the latitude to create my own research area, to blaze a new path, and to learn from my mistakes along the way. He pushed me when I needed to be pushed. He organized the Computer Poker Research Group (CPRG) to further the research, resulting in more than 20 scientific publications that simply would not have happened otherwise. Apart from his own remarkable achievements, he is a catalyst to the research of all those around him. Although I tease him at every opportunity, it is only because my respect for him runs so deep.

Duane Szafron is one of the finest role models I have ever encountered. He is revered by his students for his patience, wisdom, and diligence to their education. His feedback is invaluable for filling gaps, and greatly improving communication.

Robert Holte is an excellent person to converse with, not only because of his incredible breadth and depth of knowledge, but also for his many interests and anecdotes. His supervision of **Bret Hoehn**'s M.Sc. thesis provided a comprehensive study of opponent modeling in simpler poker domains, illustrating some the inherent challenges and limitations we face.

Mike Carbonaro has been a friend for many years, and has provided much encouragement. I am particularly indebted to him for investigating and lobbying for the paper-based thesis option. Although it is a common format in other faculties and universities, this is the first paper-based Ph.D. thesis in Computing Science at the U of A. This option eliminated months of drudgery in re-hashing work that was in some cases more than ten years old. A considerable amount of redundancy is present as a result, but I believe the benefits far outweigh the disadvantages. I hope future students will benefit from the precedent that Mike has championed.

Michael Littman has been an inspiration to me since 1998, when he followed my AAAI talk with the single best example I've ever seen of a presentation that is both entertaining and informative. His enthusiasm is infectious, and he is a perfect exemplar of how to do good science and have fun doing it. He provided extensive excellent feedback on the dissertation, only some of which I have had time to incorporate.

Martin Müller was originally on my thesis committee, but was on sabbatical in Europe at the time of the defense. He provided a lot of valuable feedback on earlier drafts of the dissertation, as well as some insightful words of wisdom about the process.

Among my peers, **Aaron Davidson** has done more than anyone to make me look good. His coding wizardry has had a huge impact on the CPRG. We had way too much fun developing POKI's online personality (responding to conversation, telling jokes, teaching Latin, and on and on). Aaron's online poker server has provided an important venue for empirical testing and data collection. As the lead developer of POKER ACADEMY (www.poker-academy.com), he has helped produce the finest poker training software in the world, by far. Apart from these things, his friendship has enhanced my life in innumerable ways.

The other major workhorse for the CPRG has been **Neil Burch**. He was responsible for assimilating the many concepts and details needed to produce SPARBOT. Half of our papers would not have been possible without his tireless efforts running countless experiments.

Terry Schauenberg was responsible for merging some vague modeling ideas with a concrete search algorithm to produce our temperamental champion, VEXBOT. He constantly surprises me with his keen insights, and previously unknown areas of expertise.

In the past year, **Morgan Kan** has had an enormous positive impact on my life. He has a knack for asking innocent questions that hit like a punch to the spleen. He is uncompromising in demanding clarity and precision, and equally conscientious in implementation. The DIVAT analysis technique is much stronger as a result of his involvement.

All of the members of the CPRG have made significant contributions to the work. Without the interest expressed by **Denis Papp** and **Lourdes Peña**, the CPRG might never have been formed. **Finnegan Southey**, **Martin Zinkevich**, **Chris Rayner**, **Nolan Bard**, **Mike Johanson**, **Carmelo Piccione**, and other new recruits have created a dynamic, energetic environment for discussing various aspects of the problem. **Michael Bowling** is a brilliant young investigator, and the future of poker research at the U of A is certainly bright under his direction.

I have often said that I am particularly blessed to be surrounded by so many like-minded (mostly iNTp) friends, who insulate me from the cold harsh world, and teach me so much. The value from knowing them is inestimable. Among those not already mentioned are **Roel van der Goot**, **Andreas Junghanns**, **Yngvi Björnsson**, **Michael and Karen Buro**, **Nathan Sturtevant**, **Jason Spencer**, and my lifelong friend, **Greg Huber**.

I have had the good fortune to interact with many other students and faculty, both academically and socially. Getting to know the many members of the GAMES group has been particularly rewarding.

Over the years I have benefited from many interesting discussions with poker players, including Paul Phillips, David Sklansky, Mason Malmuth, Lee Jones, Barry Tanenbaum, Michael Maurer, Steve Brecher, Andy Bloch, Chris Ferguson, Howard Lederer, Annie Duke, Phil Hellmuth, Lou Krieger, Phil Laak, and many others. Numerous late-night conversations with Gautam Rao and Dan Hanson deepened our understanding of the game, and of life.

My parents and family have always provided encouragement and support throughout my life. I love and admire all of them.

Finally, I want to thank the love of my life, **Alexandra Baracu**, for her love, patience, and perpetual wondrousness.

Dedication:

To Howard and Eileen Billings,
for their eternal love and support.

And to Xor, for many years of
unconditional love and friendship.

Preface

It is very common for Ph.D. students to be dissatisfied with their dissertation. The feeling is especially acute in my case, because it could reflect poorly on the excellent work of my colleagues. Numerous people have made significant contributions to this research, but I alone accept the blame for errors, and failures to communicate effectively.

Due to the scope and collaborative nature of this work, this thesis is not nearly as comprehensive as I would like. Expedience had to win over idealism. Many of the current research topics are beyond the scope of the thesis, and are not discussed.

Many excellent comments by my thesis committee (Michael Littman and Robert Holte in particular) are not yet included in this document, due to time constraints. Future revisions (and perhaps additional content) will likely be available in online versions. The reader is also encouraged to read Morgan Kan's upcoming M.Sc. thesis, for an alternate presentation of the DIVAT method discussed in Chapter 5, and some detailed analysis of the Vegas'05 and AAAI'06 competitions (among other things).

Darse Billings, September 27, 2006.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Historical Development

Games have played an important role in Artificial Intelligence (AI) research since the beginning of the computer era. Many pioneers in computer science spent time on algorithms for chess, checkers, and other games of strategy. A partial list includes such luminaries as Alan Turing, John von Neumann, Claude Shannon, Herbert Simon, Alan Newell, John McCarthy, Arthur Samuel, Donald Knuth, Donald Michie, and Ken Thompson [1].

The study of board games, card games, and other mathematical games of strategy is desirable for a number of reasons. In general, they have some or all of the following properties:

- Games have **well-defined rules and simple logistics**, making it relatively easy to implement a complete player, allowing more time and effort to be spent on the actual topics of scientific interest.
- Games have **complex strategies**, and are among the hardest problems known in computational complexity and theoretical computer science.
- Games have a **clear specific goal**, providing an unambiguous definition of success, and efforts can be focused on achieving that goal.
- Games allow **measurable results**, either by the degree of success in playing the game against other opponents, or in the solutions to related subtasks.

Apart from the establishment of *game theory* by John von Neumann, the strategic aspects of poker were not studied in detail by computer scientists prior to 1992 [1]. Poker features many attributes not found in previously studied games (such as checkers and chess), making it an excellent domain for the study of challenging new problems. In terms of the underlying mathematical structure and taxonomy of games, some of the most important properties include the following:

- Poker is a game of **imperfect information**. Various forms of uncertainty are a natural consequence. This property creates a necessity for using and coping with *deception* (specifically, ***bluffing*** and ***trapping***),[1] and ensures a theoretical advantage for the use of *randomized mixed strategies*.

- Poker has **stochastic outcomes**. The element of chance (the random dealing of cards) at several stages of the game introduces uncertainty and uncontrollable outcomes. Among other things, this adds a high degree of variance to the results, and makes accurate assessment of performance difficult.

- Hidden states in poker are **partially observable**. A player can win a game uncontested when all opponents ***fold***, in which case no private information (*i.e.*, the cards held by any of the players) is revealed. Partial observability makes it much more difficult to learn about an opponent's strategy over the course of many games, both in theory and in practice.

- Poker is a **non-cooperative multi-player** game. A wealth of challenging problems exist in multi-player games that do not exist in two-player games. Multi-player games are inherently unstable, due in part to the possibility of coalitions (*i.e.*, teams), but those complexities are minimized in a non-cooperative game [60, 63].

As a general class, stochastic imperfect information games with partial observability are among the hardest problems known in theoretical computer science. This

[1] Technical terms and jargon from poker theory appear in bold face italics throughout this dissertation, and are defined in Appendix A: Glossary of Poker Terms.

class includes many problems that are easy to express but are computationally undecidable [20, 38].

In practice, writing a program to play a legal game of poker is trivial, but designing and implementing a competent poker player (for example, the strength of an intermediate human player) is a challenging task. Writing a program that also adapts smoothly to exploit each opponent's particular playing style, betting patterns, biases and tendencies is a difficult learning problem.

1.2 Major Influences

Since there was no specific research on poker game-playing in the computer science literature prior to 1992, the mathematical models and scientific methodology for the research project were based on other available sources of knowledge. Three major influences were:

1. Classic books on *poker strategy*,
2. Fundamental principles of *game theory*, and
3. Traditional game-playing programs based on *game-tree search*.

1.2.1 Classic Books on Poker Strategy

The single most important book to date for understanding poker strategy is "The Theory of Poker" by David Sklansky [55]. Other books by Sklansky and frequent co-author Mason Malmuth also provide valuable insights [56, 57]. Additional resources, and their utility for scientific research, are discussed in Billings [1].

Although written for human students of the game, the clear exposition in these texts allows a mathematically inclined reader to gain an appreciation for the underlying logical structure of the game. This insight suggests a wealth of algorithmic possibilities to be explored for knowledge-based approaches. Incorporating probabilistic knowledge into a formula-based architecture was the topic of our early research, and is discussed in Chapter 2. The serious limitations of that approach and lessons learned from the research are discussed in Chapter 6.

1.2.2 Fundamental Principles of Game Theory

The game of poker was used as a model of adversarial conflict in the development of mathematical game theory by John von Neumann in the 1920s [69]. The (general probabilistic) *Minimax Theorem* proves that for any two-player zero-sum game, there always exists an *equilibrium strategy*. Using such a strategy ensures (at least) the game-theoretic value of the game, regardless of the opponent's strategy. Thus, playing an equilibrium strategy would guarantee not losing in the long run (assuming the players alternate positions over many games).[2]

John Nash extended the idea of equilibrium strategies to non-zero-sum games and multi-player games, again using poker as an example [41]. A set of strategies are said to be equilibrium when no player can benefit from unilaterally changing their style or strategy [25, 24, 70]. The 1972 book "Winning Poker Systems" by Norman Zadeh attempted to apply game theoretic strategies to a variety of real poker variants, with some degree of success [71, 72, 1].

There are serious limitations to equilibrium strategies in practice, because they are static, are oblivious to the opponent's strategy, and have implicit assumptions that generally give the opponent far too much credit. These inherent limitations have been clearly demonstrated in the simpler imperfect information games of Rock-Paper-Scissors [4, 2] and Oshi-Zumo [17].

Finding an approximation of an equilibrium strategy for real poker is discussed in Chapter 3. Further insights and limitations of applying game-theoretic methods in general are discussed in Chapter 6.

1.2.3 Traditional Game-Tree Search

Many lessons have been learned from traditional high-performance game-playing programs. Research from 1970 to 1990 focused primarily on chess, and other two-player zero-sum games with perfect information. As these programs improved, a recurring theme was an increasing emphasis on computer-oriented solutions, and

[2] In practice, poker is normally a negative constant sum game, because the host (*e.g.*, casino) charges the players a ***rake*** or a ***time charge***. Nevertheless, an equilibrium strategy (or approximations thereof) would be highly useful for playing the real game.

diminishing reliance on human knowledge [46, 34].

In many games with relatively simple logistics but complex strategy, computer programs have now surpassed the best human players by a vast margin. In each case, the formula for success has been the same: deep look-ahead search of the game tree using the highly efficient *alpha-beta search algorithm*, combined with a domain-specific *evaluation function* applied at the nominal leaf nodes of the search frontier [47].

- In 1990, the checkers program CHINOOK earned the right to challenge the human world champion, Marion Tinsley, in a title match. CHINOOK lost narrowly in 1992, but won the return match in 1994, becoming the first computer program to win an official world championship in a game of skill [48]. An effort is now underway to solve the game of checkers, with two of the official tournament openings now proven to be drawn [50].

- In 1997, the Othello program LOGISTELLO defeated the human world champion, Takeshi Murakami, in a six game exhibition match, winning all six games [15, 16].

- In 2000, the Lines of Action program MONA won the *de facto* world championship, defeating all of the top human players, and winning every game it ever played against human opposition [5, 3].

- In 2002, the ancient game of Awari was strongly solved, computing the exact minimax value for every reachable position in the game [68, 45]. Although the best programs already played at a level far beyond any human player, the difference between super-human play and perfect play was shown to be enormous [44].

- In 1997, the chess machine DEEP BLUE won a short exhibition match against the human world champion, Garry Kasparov, scoring two wins, one loss, and three draws [18]. This led to a widely held but premature belief that chess programs had surpassed all human players. Several years later, the programs SHREDDER, FRITZ, and JUNIOR demonstrated performances on par with the best human players. In 2005, the program HYDRA convincingly defeated one of the strongest human players, Michael Adams, scoring five wins, zero losses, and one draw, providing a strong argument for the dominance of chess programs [23].

Similar successes continue to be obtained for this general class of games, usually with the same architecture of alpha-beta search combined with a good heuristic evaluation. The approach has not been successful for the game of Go, however, owing to the high branching factor and vast search space (for 19×19), and the fact that goals and subgoals are very difficult to assess with heuristic evaluation [40].

1.3 Extending Game Tree Representations

Many games admit some element of *random chance*. The traditional game tree representation can be extended to handle stochastic outcomes by incorporating *chance nodes*. Each branch from a chance node represents one of the finite number of random outcomes. From a search perspective, all of these branches must be considered, and combined to obtain an overall ***expected value*** (EV), so the size of the game tree grows multiplicatively with each subsequent chance node. The alpha-beta search algorithm is not applicable to this class of problems, but other search algorithms such as **-Minimax* [28, 27] and simulation methods [67, 54] are able to contend with this form of uncertainty adequately. The property of stochasticity has not been a major impediment to world-class computer performance in practice.

The game of backgammon is a classic example of a perfect information game with an element of stochasticity (the roll of the dice). Excellent evaluation functions have been learned automatically from self-play [66, 67], resulting in several programs that are at least on par with the best human players, without requiring deep search [27].

Multi-player games are much more challenging, both in theory and in practice; but to date they have not received a lot of attention in AI research. Several search algorithms for multi-player game trees are known in the literature, but the potential for guaranteed safe pruning of the search space is much lower than that enjoyed by the alpha-beta algorithm [59, 61, 64]. This fact, combined with the larger branching factor resulting from many players, means that deep search is less feasible, in general.

Moreover, multi-player games are inherently *unstable*, being subject to possible *collusion* between players.[3] In practice, minor differences in search algorithms for multi-player game trees can produce radically different results. For example, two different move choices could be exactly equal in value for Player A, but could dictate whether Player B or Player C wins the game. Due to these volatile conditions, good opponent modeling (for example, knowing each player's method of tie-breaking between equal moves) is necessary to obtain robust and reliable results [60, 62, 63, 65].

However, the major distinguishing factor between poker and other games is the property of *imperfect information*, the effects of which can range from obvious to subtle, from inconsequential to profound. One important consequence is that a complete strategy in poker must include a certain degree of deception, such as bluffing (betting or raising with a weak hand) and trapping (playing a strong hand as though it were weak). This fact was one of the earliest results in game theory [69]. The objective of these deception tactics is to disguise the strength of one's hand (called *information hiding*), and to create uncertainty in the beliefs of the opponent, resulting in greater profitability overall.

The *relative balance* of these deceptive plays (and of responses to the opponent's actions) is of critical importance. Any inappropriate imbalances necessarily imply the existence of statistical biases, patterns, or other weaknesses that are vulnerable to exploitation. Since there may be many ways of obtaining the desired balance of plays in poker, the players have some discretion in how they actually achieve that balance. For example, a particular situation might call for a 10% bluff frequency, but the player is otherwise free to decide when to bluff or not bluff. As a result, there is in general *no single best move* in a given poker situation.

This is in stark contrast to a perfect information game, where there is a single

[3] This is true even for ostensibly non-cooperative games, like poker, since that ethic cannot be enforced, in general. John von Neumann showed that multi-player games become stable only when they devolve into a two-player game between two coalitions [69].

move, or small set of moves, that preserves the game-theoretic value of the game. In backgammon, for example, there is typically only one move in a given position that will maximize the expected value against perfect play.

Furthermore, theoretically correct play in an imperfect information game requires probabilistic mixed strategies, where different moves are chosen some fraction of the time in identical circumstances. In contrast, a *deterministic pure strategy* (always playing one particular move in a given situation) is sufficient to obtain the game-theoretic value in a perfect information game (although the player may choose randomly from a set of equal-valued moves).

Game trees can be further extended to handle imperfect information games, with the inclusion of *information sets*. An information set is a set of decision nodes in the game tree that cannot be distinguished from the perspective of a given player. Since the opponent's cards are hidden in poker, this corresponds to the complete set of all possible opponent holdings in a given situation. Obviously, the same policy (such as a particular mixed strategy) must be applied identically to all of the nodes in the information set, since it is not possible to know precisely which of those states we are in.

The immediate consequence is that nodes of an imperfect information game tree are *not independent*, in general.[4] Thus, a divide-and-conquer search algorithm, such as the alpha-beta minimax technique, is not applicable to this class of problems, since sub-trees cannot be handled independently.

Another characteristic that distinguishes poker from perfect information board games is that it is not enough to simply "play well", while largely ignoring the existence of the opponent. To maximize results, it is absolutely essential to understand the opponent's *style*, and the nature of their errors (such as patterns or tendencies in their play).

As a simple demonstration, consider two opponents, one of whom bluffs far too often, the other of whom seldom bluffs. Both are "weak players", in an objective sense. To maximize our profit against the former, we call (or perhaps raise) more

[4] A perfect information game tree can be thought of as a special case in which all decision nodes belong to their own unique information set.

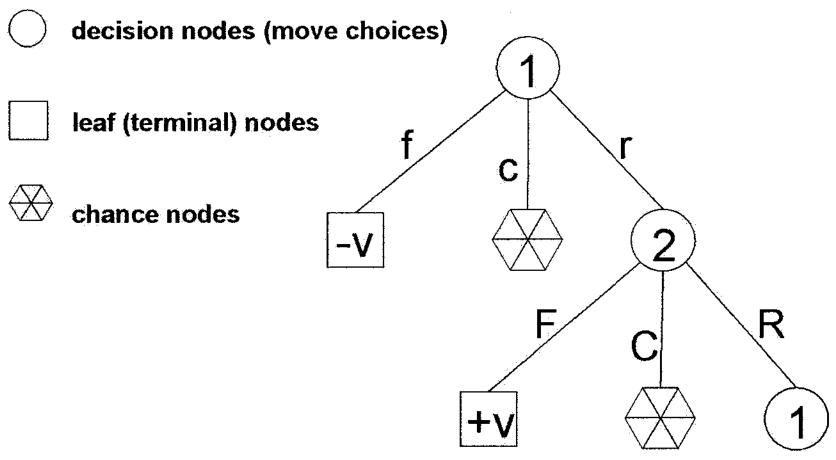


Figure 1.1: A portion of a poker game tree, with chance nodes.

often with mediocre hands. To maximize against the latter, we fold more often with marginal hands. Our strategy adjustments are diametrically opposite, depending on the nature of the opponent's predictable weaknesses.

In perfect information games, simply playing strong moves will naturally punish weaker moves, and it is not necessary to understand *why* an opponent is weak. Opponent modeling has been investigated in chess and other two-player perfect information games, but has not led to significant improvements in performance [33, 31, 32, 19].

An interesting case study is the game of Scrabble. Although Scrabble is technically a game of imperfect information, that property plays a relatively minor role in strategy. Super-human performance has been attained for the two-player game without special consideration for opponent modeling. Relatively simple techniques, such as Monte Carlo simulation and selective sampling, can be used to account for the unknown information adequately [52, 54]. Moreover, the strengths of the computer player, including perfect knowledge of the dictionary and the ability to consider every legal play, are sufficient to surpass all human players in skill [53].

Figure 1.1 shows a small portion of the imperfect information game tree for any Limit poker variant, featuring decision nodes for each player during a betting round. In general, a player will choose one of three possible actions: fold (f), call

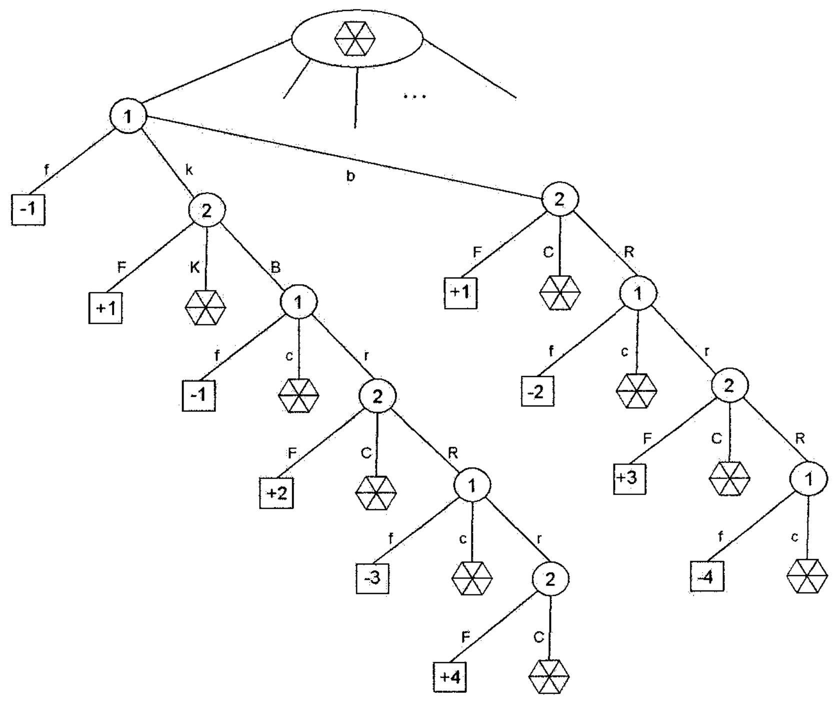


Figure 1.2: A complete betting round in 2-player Limit poker.

(c), or raise (r). By convention, capital letters are used to indicate the actions of the second player.

When the betting round is complete, the game is either over (one player folded, leading to a terminal node), or the game continues with the next chance event (cards being dealt). Figure 1.2 shows the game tree for a complete betting round of 2-player Limit poker (with a maximum of three raises per round).

Figure 1.3 illustrates the notion of information sets in the game of 2-player Limit Texas Hold'em. Only three of the 1,624,350 branches from the initial chance node (*i.e.*, the dealing of the ***hole cards***) are shown. For any given hand, a player will have 1225 indistinguishable states, since the opponent's cards are not known. Naturally, the same decision policy will apply to all states in that information set.

Figure 1.4 shows a high-level view of the structure of Texas Hold'em. Each betting round is depicted with a triangle, and corresponding chance nodes are collected to indicate the stage of the hand. The numbers on the left indicate the branching factors at each stage, leading to more than a quintillion (1,179,000,604,565,715,751)

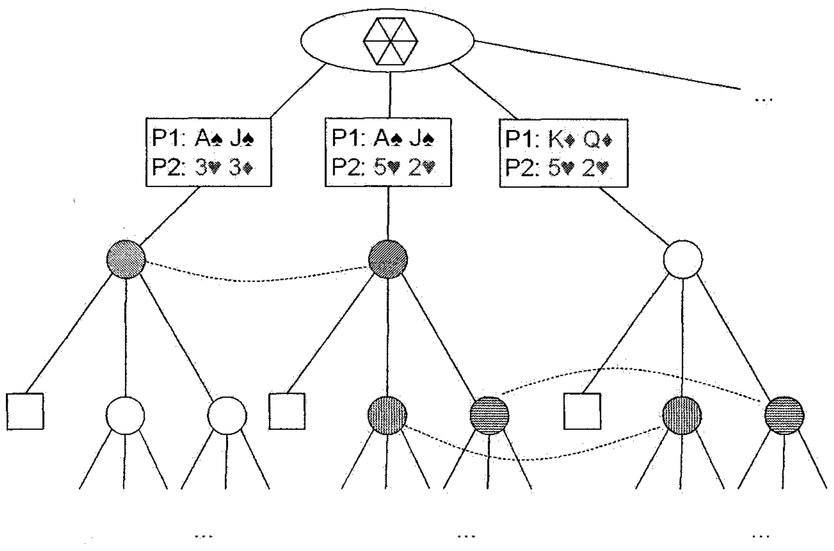


Figure 1.3: Information sets in an imperfect information game tree.

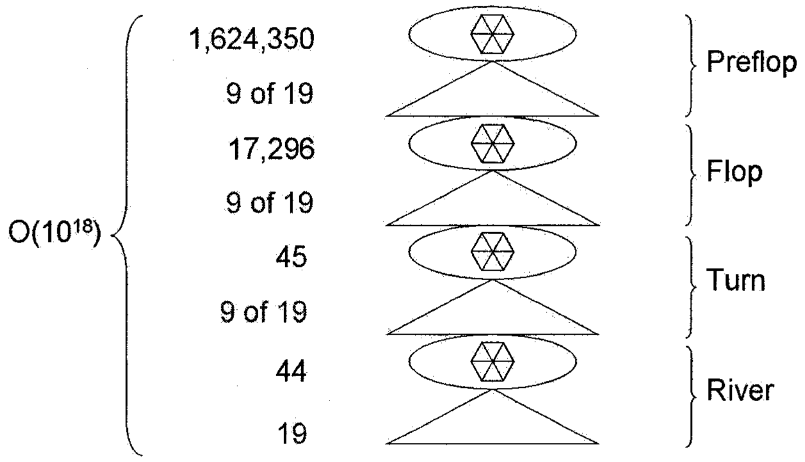


Figure 1.4: The overall structure of the Texas Hold'em game tree.

nodes of all types.

Defining appropriate search algorithms for this fundamentally different mathematical structure of game tree is discussed in Chapter 4. The problems encountered and the necessary modifications for future research are discussed in Chapter 6.

1.4 The University of Alberta Computer Poker Research Group

The University of Alberta Computer Poker Research Group (CPRG) is the major contributor to the academic literature on poker game-playing AI. The purpose of this section is to explain the structure of the research group and the roles of the members. Since it is a collaborative team effort, it is necessary to identify the specific contributions made by this author, distinguishing them from the work of other members, and the group as a whole. All conceptual designs, architectures, and specific component algorithms discussed in this dissertation are attributable to the author unless noted otherwise. The use of the words "our" and "we" in this document refer to the group as a whole.

The research began in 1992 with scientific foundations, methodologies, and research philosophy [1]. This included a complete basic implementation, along with computer-oriented algorithms (rather than knowledge-based methods) for advanced hand assessment, simulation techniques, and other essential functions. The CPRG was formed in 1997 to follow up on this work. The author is the lead architect for the group, and the domain expert.[5]

Dr. Jonathan Schaeffer is a co-founder, scientific advisor, and the administrative head of the CPRG. **Dr. Duane Szafron** is also a co-founder and scientific advisor. **Dr. Robert Holte** joined the group in 2001, contributing expertise in machine learning and linear programming. **Dr. Michael Bowling** joined the group in 2004, adding more knowledge in game theory and learning algorithms. Several M.Sc. students, summer students, and one full-time programmer/analyst have contributed to implementations and experimentation of the resulting systems.

[5] The author played poker professionally from 1996 to 1999, after several years of studying and extending poker theory.

- **Denis Papp** (M.Sc. student) constructed the original LOKI system, in C++, re-implementing the author's Monte Carlo simulation and weighted enumeration algorithms for hand assessment, along with numerous other components (discussed in Chapter 2) [42]. He incorporated the GNU poker library high-speed hand comparators as a core function [39]. He implemented all of the communication protocols to enable LOKI to participate in poker games on the IRC Online Poker Server [14].

- **Lourdes Peña** (M.Sc. student) built on top of the existing system (LOKI II) for the first implementation of selective simulation techniques and the subsequent experiments [43, 11].

- **Aaron Davidson** (M.Sc. student) re-wrote the entire codebase (re-christened POKI), in Java, using native methods where necessary to maintain high-speed performance. He performed code reviews with the author, discovering and correcting numerous errors, and made significant improvements to many components. The neural network approach for opponent modeling was entirely his own design [22, 7, 21]. Aaron developed test suites for conducting experiments, and wrote the University of Alberta online poker server, allowing extensive empirical testing. He also proposed new simulation methods to reduce the problem of compounding errors with sequential actions. Those ideas were refined and reformulated by the author as the *Miximax* and *Miximix* algorithms for imperfect information game-tree search (discussed in Chapter 4). Aaron then implemented and co-developed refinements for those systems [8].

- **Neil Burch** (programmer/analyst) implemented numerous algorithms and support routines, and performed many of the scientific experiments reported in CPRG publications. He developed a system for specifying general poker game definitions and converting them into the *sequence form* linear program encoding described by Koller *et al.* [36, 37]. Neil oversaw all related computations, using a commercial linear program engine (CPLEX) to produce the game-theoretic equilibrium solutions (discussed in Chapter 3) [6]. He also wrote alternate implementations of adaptive architectures (discussed in Chapter 4), for the purposes of testing and comparison [8].

- **Terence Schauenberg** (M.Sc. student) implemented the adaptive *Miximax* algorithm; co-developed the data structures, parameters, and abstractions used in VEXBOT; and performed related experiments (discussed in Chapter 4) [8, 51]. He implemented the author's Expected Value Assessment Tool (EVAT) and Luck Filtering Assessment Tool (LFAT), which were precursors to the Ignorant Value Assessment Tool (DIVAT) performance metric (discussed in Chapter 5) [10]. Terence has also investigated a variety of methods for learning approximations of Nash-equilibrium solutions by means of *fictitious play*.

- **Bret Hoehn** (M.Sc. student) performed an independent study of opponent modeling, under the direction of Dr. Holte. He used the tiny game of Kuhn poker to reduce the complexity of learning an opponent's weaknesses and quickly adopting an appropriate counter-strategy [30, 29]. Serious limitations are encountered despite the large reduction in pertinent variables, demonstrating some of the fundamental impediments to rapid learning and adaptation in partially observable stochastic domains.

- **Morgan Kan** (M.Sc. student) implemented the author's DIVAT method for direct assessment of poker decision quality, and performed numerous experiments during its development that led to deeper insights into the problem (discussed in detail in Chapter 5) [10].

The research group has expanded rapidly in recent years, with the addition of post-doctoral fellows **Finnegan Southey** and **Martin Zinkevich**; M.Sc. students **Chris Rayner**, **Nolan Bard**, and **Mike Johanson**; and research associate **Carmelo Piccione**.

The research has also branched out with several new topics (which are outside of the scope of this thesis), including development of the author's *pdf-cutting algorithm* for creating parameterized probabilistic profiles of the poker strategy space, and new methods for rapid learning using Bayesian inference methods [58].

1.5 Summary of Contents

This thesis identifies four distinct approaches to computer poker-playing, with a corresponding program architecture designed for each technique. Each approach has proven to be highly successful, despite the inherent theoretical limitations. Each generation has superseded the previous one by addressing the most important limitations discovered during the extensive empirical testing, which includes millions of games played. The core chapters of this paper-based thesis are comprised of the academic papers that stemmed from each of these studies.

1.5.1 Knowledge-based Methods and Simulation (1997-2001)

The first two approaches, discussed in Chapter 2, are *formula-based strategies* and *simulation*. Formula-based methods are a generalization of the somewhat intuitive but overly-simplistic method of *deterministic rule-based systems*. Various forms of simulation are an important technique for enhancing the performance of established programs, or for playing the game directly.

The representative paper for the formula-based and simulation methodology is "The Challenge of Poker", published in the journal *Artificial Intelligence* [7]. The paper subsumes most of the previous work by the CPRG [13, 12, 42, 14, 11, 43, 49, 22]. Some of the most important contributions of this work include:

- Expert systems for the (relatively uncomplicated) strategy of the first betting round (the ***pre-flop***), based on values determined by Monte Carlo *roll-out simulations*.
- Exhaustive enumeration algorithms for the assessment of hand quality (*hand strength* and *hand potential*).
- Selective simulation techniques for enhancing and refining ***expected value*** estimates.
- Statistical opponent modeling, and routines for the utilization, maintenance, and updating of relevant belief states.

- Procedures and advanced modules for *post-flop betting strategy*, incorporating general and specific opponent modeling, and including occasional deceptive plays (bluffing and trapping).

- The only poker programs (POKI and its derivatives) that are known to play better than an average human player who plays in low-limit casino games.

In recent years, numerous hobbyists and researchers have referred to these early publications, and based their poker programs on those architectures. They have invariably discovered the advantages and the inherent limitations of knowledge-based systems for themselves.

1.5.2 Game-Theoretic Methods (2002-2003)

The third approach, discussed in Chapter 3, is based on ***game theory***. This addresses the serious short-comings of the formula-based approach in achieving a *well-balanced betting strategy*, with an appropriate ratio of deceptive plays (bluffs and traps) in relation to the frequency of legitimate bets, calls, and folds.

The corresponding paper "Approximating Game-Theoretic Optimal Strategies for Full-scale Poker", won the Distinguished Paper Award at the International Joint Conference on Artificial Intelligence in 2003 [6]. Some of the most important contributions of this work include:

- Abstraction techniques for exact and near-exact reformulation of defined poker games, yielding reductions of the problem size by about two orders of magnitude.

- Crude but powerful abstraction techniques, capable of reductions of the problem size by more than ten orders of magnitude (from 10^{18} states to less than 10^{8} states), but with no guarantees on error bounds. These severe abstractions nevertheless maintain the key properties and relationships of the game, such that exact solutions to the abstract game provide reasonable approximations for use in the full-scale game.

- Poker programs (known collectively as PsOpti or Sparbot) that exhibit a vast improvement in skill for two-player Limit Texas Hold'em.

- The first demonstration of a program that could be competitive with a world-class player.

Several other researchers have recently built on this work, including Andrew Gilpin and Tuomas Sandholm at Carnegie Mellon University [26].

1.5.3 Adaptive Imperfect Information Game-Tree Search (2004)

The fourth approach, discussed in Chapter 4, is based on *imperfect information game-tree search*, with built-in data structures for opponent modeling and adaptive play. This addresses the serious short-comings of the game-theoretic and formula-based approaches in rapidly adapting to the opponent's style of play, exploiting biases and predictable patterns, and making it much more challenging to learn against the program. The *Miximax* and *Miximix* algorithms accommodate the more general class of game trees where some domain information is hidden from one or more players, and where each decision node may be associated with a *randomized mixed strategy*, rather than a single action.

The related paper is "Game-Tree Search with Adaptation in Stochastic Imperfect Information Games", from the 2004 Computers and Games conference [8]. Some of the most important contributions of this work include:

- A generalized framework for stochastic imperfect information games based on generalizations of the (perfect information) *Expectimax* algorithm.

- Refined methods for opponent modeling, with direct applicability to ***expected value*** calculations for each available action.

- Abstraction techniques for partitioning distinct betting sequences into a manageable number of highly correlated situations.

- The experimental poker program Vexbot, which eventually learns to defeat all known programs by a large margin, and can provide a serious threat to world-class players.

1.5.4 Assessment of Performance (2005)

Chapter 5 addresses the difficult issue of *performance assessment* in poker. Unfortunately, measuring the performance of a poker program simply by playing games requires many thousands of trials to produce a single data point, which is then only relevant to that one narrow set of preconditions. Moreover, performance in poker is decidedly non-transitive: "A beats B" and "B beats C" does not imply that "A beats C", nor does it say anything about the relative magnitude of win rates against future opponents. The outcome of any particular match may be governed by a clash of *styles*, rather than the objective strengths of the players. Testing against a wide variety of opponents is essential, but is not guaranteed to be sufficient.

To combat these serious obstacles, the author invented the Ignorant Value Assessment Tool (DIVAT).[6] Similar metrics (called EVAT and LFAT) were developed previously for analyzing experiments and matches, but they had serious shortcomings. DIVAT provides an objective means of accurately assessing decision quality, with a large reduction in the natural variance of outcomes. The tool is based on a hindsight expected value assessment of each decision, comparing the actual equities against a theoretically motivated baseline [9, 35]. The paper "A Tool for the Direct Assessment of Poker Decisions" has been accepted for publication in the International Computer Games Association Journal [10].

1.5.5 Conclusion

Chapter 6 concludes the thesis with a retrospective look at some of the most important lessons that have been learned over the years. A major theme that ties these publications together is the evolution of architectures for poker programs. Each approach has both theoretical and practical limitations. Some of these limitations were known before the system was built, but the full implications can only be understood after many implementations and refinements are tested. Recurring themes include the need for well-balanced betting strategies, better opponent modeling, and faster learning and adaptation.

[6] The 'D' refers to the author's first initial.

For each architecture, program development is often a cyclic process, with each iteration introducing an improved method for handling a particular aspect of the game that had become the limiting factor to performance. In some cases, the cycle was very long and arduous, with some "temporary" components not being re-visited again for years.

There has always been a healthy interplay between theory and practice. Diminishing returns from these refinements help identify fundamental limitations that necessitate a revolutionary change – a new approach and new architecture that does a much better job of addressing some critical strategic aspect of the game. Ultimately, we seek unifying methods that reduce the complexity of the system, and eliminate human intervention, allowing the program to "think for itself".

Although much work remains to be done, poker programs have evolved from very weak players to programs that are a serious threat to world-class players. The past successes and failures suggest what types of solutions are the most viable in general, and which directions of research will be most fruitful in the future.

Bibliography

[1] D. Billings. Computer Poker. Master's thesis, Department of Computing Science, University of Alberta, 1995.

[2] D. Billings. The First International RoShamBo Programming Competition. *The International Computer Games Association Journal*, 23(1):42–50, 2000.

[3] D. Billings. MONA and YL's Lines of Action page. World Wide Web, 2000. `http://www.cs.ualberta.ca/~games/LOA/`.

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

[5] D. Billings and Y. Björnsson. Search and knowledge in Lines of Action. In H. J. van den Herik, H. Iida, and E. A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges, ACG'04*, pages 231–248. Kluwer Academic, 2004.

[6] D. Billings, N. Burch, A. Davidson, T. Schauenberg, R. Holte, J. Schaeffer, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *The Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 661–668, 2003.

[7] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1–2):201–240, January 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 H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04*, LNCS 3846, pages 21–34. Springer-Verlag GmbH, 2004.

[9] D. Billings and M. Kan. Development of a tool for the direct assessment of poker decisions. Technical Report TR06-07, University of Alberta Department of Computing Science, April 2006.

[10] D. Billings and M. Kan. A tool for the direct assessment of poker decisions. *The International Computer Games Association Journal*, 2006. To appear.

[11] D. Billings, D. Papp, L. Peña, J. Schaeffer, and D. Szafron. Using selective-sampling simulations in poker. In *American Association of Artificial Intelligence Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, pages 13–18. American Association of Artificial Intelligence, 1999.

[12] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 493–499, 1998.

[13] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Poker as a testbed for machine intelligence research. In R. Mercer and E. Neufeld, editors, *Advances in Artificial Intelligence, AI'98*, pages 228–238. Springer-Verlag, 1998.

[14] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *American Association of Artificial Intelligence National Conference, AAAI'99*, pages 697–703, 1999.

[15] M. Buro. The Othello match of the year: Takeshi Murakami vs. Logistello. *ICCA Journal*, 20(3):189–193, 1997.

[16] M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1–2):85–99, 2002.

[17] M. Buro. Solving the Oshi-Zumo game. In H. J. van den Herik, H. Iida, and E. A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges*, pages 361–366. Kluwer Academic, 2004.

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

[19] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *American Association of Artificial Intelligence National Conference, AAAI'96*, pages 120–125, 1996.

[20] A. Condon. On algorithms for simple stochastic games. In J. Cai, editor, *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[21] A. Davidson. Opponent modeling in poker. Master's thesis, Department of Computing Science, University of Alberta, 2002.

[22] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI'00*, pages 1467–1473, 2000.

[23] C. Donninger and U. Lorenz. Hydra chess webpage. World Wide Web, 2005. `http://hydrachess.com/`.

[24] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games*. MIT Press, May 1998.

[25] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, August 1991.

[26] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *American Association of Artificial Intelligence National Conference, AAAI'06*, pages 1007–1013, July 2006.

[27] T. Hauk, M. Buro, and J. Schaeffer. *-minimax performance in backgammon. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag GmbH, 2004.

[28] T. Hauk, M. Buro, and J. Schaeffer. Rediscovering *-minimax search. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 35–50. Springer-Verlag GmbH, 2004.

[29] B. Hoehn. The effectiveness of opponent modelling in a small imperfect information game. Master's thesis, Department of Computing Science, University of Alberta, 2006.

[30] B. Hoehn, F. Southey, R. Holte, and V. Bulitko. Effective short-term opponent exploitation in simplified poker. In *American Association of Artificial Intelligence 20th National Conference, AAAI'05*, pages 783–788, July 2005.

[31] H. Iida, J. Uiterwijk, H. J. van den Herik, and I. Herschberg. Potential applications of opponent-model search. *ICCA Journal*, 16(4):201–208, 1993.

[32] H. Iida, J. Uiterwijk, H. J. van den Herik, and I. Herschberg. Thoughts on the application of opponent-model search. In *Advances in Computer Chess 7*, pages 61–78. University of Maastricht, 1995.

[33] P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search*. PhD thesis, School of Computer Science, Carnegie-Mellon University, 1992.

[34] A. Junghanns and J. Schaeffer. Search versus knowledge in game-playing programs revisited. In *The International Joint Conference on Artificial Intelligence, IJCAI'97*, pages 692–697, 1997.

[35] M. Kan. Post-game analysis of poker decisions. Master's thesis, Department of Computing Science, University of Alberta, 2006. In preparation.

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

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

[38] O. Madani, A. Condon, and S. Hanks. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision process problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

[39] M. Maurer, B. Goetz, and L. Dachary. Gnu poker evaluation library. WWW, 1997. `http://sourceforge.net/projects/pokersource/`, `http://pokersource.sourceforge.net/`.

[40] M. Müller. Computer Go. *Artificial Intelligence*, 134(1–2):134–179, 2002.

[41] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

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

[43] L. Peña. Probabilities and simulations in poker. Master's thesis, Department of Computing Science, University of Alberta, 1999.

[44] J. W. Romein and H. E. Bal. Awari is solved. *The International Computer Games Association Journal*, 25(3):162–165, September 2002.

[45] J. W. Romein and H. E. Bal. Solving Awari with parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, October 2003.

[46] J. Schaeffer. *Experiments in Search and Knowledge*. PhD thesis, Department of Computer Science, University of Waterloo, 1986.

[47] J. Schaeffer. The history heuristic and the performance of alpha-beta enhancements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[48] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, 1997.

[49] J. Schaeffer, D. Billings, L. Peña, and D. Szafron. Learning to play strong poker. In *The International Conference on Machine Learning Workshop on Game Playing*. J. Stefan Institute, 1999. Invited paper.

[50] J. Schaeffer, Y. Björnsson, N. Burch, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Solving checkers. In *The International Joint Conference on Artificial Intelligence, IJCAI'05*, pages 292–297, 2005.

[51] T. Schauenberg. Opponent modelling and search in poker. Master's thesis, Department of Computing Science, University of Alberta, 2006.

[52] B. Sheppard. *Toward Perfection of Scrabble Play*. PhD thesis, Computer Science, University of Maastricht, 2002.

[53] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.

[54] B. Sheppard. Efficient control of selective simulations. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag GmbH, 2004.

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

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

[57] D. Sklansky and M. Malmuth. 2+2 website and poker discussion forum. WWW, 1998. `http://www.twoplustwo.com/`.

[58] 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'05)*, pages 550–558, July 2005.

[59] N. Sturtevant. On pruning techniques for multi-player games. In *American Association of Artificial Intelligence National Conference, AAAI'00*, pages 201–207, 2000.

[60] N. Sturtevant. A comparison of algorithms for multi-player games. In J. Schaeffer, M. Müller, and Y. Björnsson, editors, *Computers and Games 2002*, LNCS 2883, pages 108–122. Springer-Verlag, 2002.

[61] N. Sturtevant. Last-branch and speculative pruning algorithms for Maxn. In *The International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 669–675, 2003.

[62] N. Sturtevant. *Multi-Player Games: Algorithms and Approaches*. PhD thesis, Department of Computer Science, University of California, Los Angeles (UCLA), 2003.

[63] N. Sturtevant. Current challenges in multi-player game search. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 285–300. Springer-Verlag GmbH, 2004.

[64] N. Sturtevant. Leaf-value tables for pruning non-zero sum games. In *The International Joint Conference on Artificial Intelligence, IJCAI'05*, pages 317–323, 2005.

[65] N. Sturtevant and M. Bowling. Robust game play against unknown opponents. In P. Stone and G. Weiss, editors, *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'06*, pages 713–719, May 2006.

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

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

[68] R. van der Goot. Awari retrograde analysis. In T. A. Marsland and I. Frank, editors, *Computers and Games 2000*, LNCS 2063, pages 87–95. Springer-Verlag, 2002.

[69] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[70] Wikipedia. Game theory. Wikipedia: The Free Online Encyclopedia. `http://en.wikipedia.org/wiki/Game_theory`.

[71] N. Zadeh. *Winning Poker Systems*. Prentice Hall, 1974.

[72] N. Zadeh. Computation of optimal poker strategies. *Operations Research*, 25(4):541–562, 1977.

Chapter 2

Knowledge-based Methods and Simulation (1997-2001)

The Challenge of Poker [1]

2.1 Introduction

The artificial intelligence community has recently benefited from the positive publicity generated by chess, checkers, backgammon, and Othello programs that are capable of defeating the best human players. However, there is an important difference between these board games and popular card games like bridge and poker. In the board games, players have complete knowledge of the entire game state, since everything is visible to both participants. In contrast, bridge and poker involve *imperfect information*, since the other players' cards are not known. Traditional methods like deep search have not been sufficient to play these games well, and dealing with imperfect information is the main reason that progress on strong bridge and poker programs has lagged behind the advances in other games. However, it is also the reason these games promise greater potential research benefits.

Poker has a rich history of study in other academic fields. Economists and mathematicians have applied a variety of analytical techniques to poker-related problems. For example, the earliest investigations in *game theory*, by luminaries such as John von Neumann and John Nash, used simplified poker to illustrate the

[1] The contents of this chapter originally appeared in the journal *Artificial Intelligence*. Copyright 2001 Elsevier Science B.V. All rights reserved. D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1–2):201–240, January 2002.

fundamental principles [41, 24, 25].

Until recently, the computing science community has largely ignored poker. However, the game has a number of attributes that make it an interesting domain for artificial intelligence research. These properties include incomplete knowledge, multiple competing agents, risk management, opponent modeling, deception, and dealing with unreliable information. All of these are challenging dimensions to a difficult problem.

We are attempting to build a program that is capable of playing poker at a world-class level. We have chosen to study the game of Texas Hold'em, which is one of the most strategically complex and popular variants of poker. Our experiences with our first program, called LOKI, were positive [6, 8]. In 1999, we rewrote the program, christening the new system POKI.

These programs have been playing on Internet poker servers since 1997, and have accrued an impressive winning record, albeit against weak opponents. Early versions of the program were only able to break even against better opposition, but recent improvements have made the program substantially stronger, and it is now winning comfortably in the more difficult games. Although most of these Internet games simulate real game conditions quite well, it would be premature to extrapolate that degree of success to games where real money is at stake. Regardless, analysis of POKI's play indicates that it is not yet ready to challenge the best human players. Ongoing research is attempting to bridge that gap.

Section 2.2 reviews previous work and related research on poker. Section 2.3 provides an overview of Texas Hold'em, including an illustrative example of strategic concepts, and a minimal set of requirements necessary to achieve world-class play. An overview of POKI's architecture is described in Section 2.4. Section 2.5 discusses the program's betting strategy, detailing some of the components of the system. The special problem of opponent modeling is addressed in Section 2.6. Experimental methods and the performance of the program are assessed in Section 2.7. Section 2.8 provides a generalized framework for stochastic games, based on POKI's simulation search strategy. Section 2.9 discusses the challenges that remain for building a world-class poker-playing program.

2.2 Other Research

There are several ways that poker can be used for artificial intelligence research. One approach is to study simplified variants that are easier to analyze. We have already mentioned some of the founding work in game theory, which could only handle extremely simple poker games. An example is Kuhn's game for two players, using a three-card deck, one-card hands, and one betting round, with at most two betting decisions [23]. While this was sufficient to demonstrate certain fundamental principles of game theory, it bears little resemblance to normal competitive poker variations.

Mathematicians have also explored many interesting problems related to poker, and highly simplified variations are again sufficient to provide complex problems (Sakaguchi and Sakai [28] for example).

Another way to reduce the complexity of the problem is to look at a subset of the game, and try to address each sub-problem in isolation. Several attempts have been made to apply machine learning techniques to a particular aspect of poker (some examples include [11, 22, 37, 42]). Similarly, many studies only look at two-player poker games. Multi-player games are vastly more complicated in general, even with the usual assumption of no co-operative behavior between players. One danger with any type of simplification is that it can destroy the most challenging and interesting aspects of the problem.

An alternate approach, which we advocate, is to tackle the entire problem: choose a real variant of poker and address all of the considerations necessary to build a program that performs at a level comparable to or beyond the best human players. Clearly, this is a most ambitious undertaking, but also the one that promises the most exciting research contributions if successful.

Nicholas Findler worked on and off for 20 years on a poker-playing program for 5-card Draw poker [14]. His primary objective was to model human cognitive processes, and he developed a program that could learn. While successful to a degree, the program itself was not reported to be a strong player. Furthermore, the game of 5-card Draw, although quite popular at that time, is not as strategically complex as other poker games, such as 7-card Stud and Texas Hold'em.

Some success in analyzing larger scale poker variants was achieved by Norman Zadeh in the 1970s, and much of this work is still of value today [43, 44]. Other individuals, including expert players with a background in mathematics, have gained considerable insight into "real" poker by using partial mathematical analyses, simulation, and *ad hoc* expert experience (Sklansky [36] is a popular example).

There is a viable middle-ground between the theoretical and empirical approaches. Recently, Daphne Koller and Avi Pfeffer have revived the possibility of investigating poker from a game-theoretic point of view [21]. They presented an algorithm for finding randomized equilibrium strategies in two-player imperfect information games, which avoids the usual exponential blow-up of the problem size when converting it to *normal form*. This algorithm is used in their GALA system, a tool for specifying and solving a greatly extended range of such problems. However, the size of the translated problems is still proportional to the size of the game tree, which is prohibitively large for most common variations of poker. For this reason, the authors concluded "...we are nowhere close to being able to solve huge games such as full-scale poker, and it is unlikely that we will ever be able to do so."

Nevertheless, this does raise the interesting possibility of computing *near-optimal* equilibrium solutions for real poker variants, which might require far less computation to obtain a satisfactory answer. This is analogous to efficient approximation algorithms for certain combinatorial optimization problems that are known to be intractable (NP-hard).

One obvious technique for simplifying the problem is to use abstraction, collecting many instances of similar sub-problems into a single class. There are many states in the poker game tree that are isomorphic to each other (for example, a hand where all relevant cards are hearts and diamonds is isomorphic to two corresponding hands with all spades and clubs). Beyond this, strictly distinct cases might be so similar that the appropriate strategy is essentially identical. For example, the smallest card of a hand being a deuce instead of a trey may have no bearing on the outcome. This is analogous to the approach used by Matt Ginsberg in *partition search*, where he defined equivalence classes for the smallest cards of each suit in a bridge hand [16]. Jiefu Shi and Michael Littman have made some preliminary attempts along these lines to produce near-optimal equilibrium solutions for a scaled-down version of Texas Hold'em [34].

A second method is aimed at constructing a shallower game tree, using ***expected value*** estimates to effectively truncate subtrees. This is similar to the method used so successfully in most perfect information games, where an evaluation function is applied to the leaves of a depth-limited search. However, it is not as easy to accomplish because, unlike perfect information games, the states of a poker game tree are not independent of each other (specifically, we cannot distinguish states where the opponent has different possible hidden cards). Ken Takusagawa, a former student of Koller and Pfeffer, has extended their work by combining this method with abstraction, to produce some approximate equilibrium solutions for particular scenarios of Texas Hold'em [38]. Alex Selby has applied the Simplex algorithm directly to two-player *Pre-flop Hold'em*, and has computed equilibrium solutions for that re-defined game, using expected values in place of the ***post-flop*** phase [30].

Our own empirical studies over the past few years have used similar methods of abstraction and expected value estimation to reduce the computational complexity of the problem, so the approaches are not as different as they may at first appear. It will be interesting to see if these theoretical "hybrid techniques" can be applied directly to a competitive poker program in the future.

2.3 Texas Hold'em

We have chosen to study the game of Texas Hold'em, the poker variation used to determine the world champion in the annual World Series of Poker. Hold'em is generally considered to be the most strategically complex poker variant that is widely played in casinos and card clubs. It is also convenient because it has particularly simple rules and logistics.

We assume the reader is familiar with the ranking of poker hands (if not, many good introductions to poker can be found on the Internet). As mentioned, bold face italics are used to highlight common poker terms, which are defined in Appendix

A: Glossary of Poker Terms.

2.3.1 Rules of Play

A ***game***[2] of Texas Hold'em begins with the ***pre-flop***. Each player is dealt two ***hole cards*** face down, followed by the first round of betting, which is started with two forced ***bets*** called the ***small blind*** and the ***big blind***. Three ***community cards***, collectively called the***flop***, are then dealt face up on the table, and the second round of betting occurs. On the ***turn***, a fourth community card is dealt face up and another round of betting ensues. Finally, on the ***river***, a fifth community card is dealt face up and the final round of betting occurs. The players still active in the game at that time reveal their two hole cards for the ***showdown***. The best five-card poker hand formed from each player's two private hole cards and the five public community cards wins the pot. If a tie occurs, the pot is split.

Texas Hold'em is typically played with 8 to 10 players. Limit Texas Hold'em uses a structured betting system, where the amount of each bet is strictly controlled in each betting round.[3] There are two denominations of bets, called a ***small bet*** and a ***big bet***, which will be \$10 and \$20 in this paper. In the first two betting rounds, all bets and raises are \$10, while in the last two rounds, they are always \$20. In general, when it is a player's turn to act, one of three betting options is available: ***fold***, ***check/call***, or ***bet/raise***.[4] There is normally a maximum of three raises allowed per betting round. The betting option rotates clockwise until each player has matched the current bet, or folded. If there is only one player remaining (all others having folded) that player is the winner and is awarded the pot, without having to reveal their cards.

[2] The term "hand" is often used in place of "game". Thus, the word "hand" is used in two ways: to denote a player's private cards, and to refer to one complete deal, or ***game***. We have tried to avoid the possible ambiguity by using "game" whenever appropriate (although that term also carries some ambiguities of its own). Regardless, the intended meaning of "hand" should be clear from the context.

[3] In No-Limit Texas Hold'em, there are no restrictions on the size of bets; a player may wager any amount, up to their entire stack, at any time.

[4] A ***check*** and a ***call*** are logically equivalent, in that the betting level is not increased. The term ***check*** is used when the current betting level is zero, and ***call*** when there has been a wager in the current betting round. Similarly, a ***bet*** and a ***raise*** are logically equivalent, but the term ***bet*** is used for the first wager of a betting round.

2.3.2 Poker Strategy

To illustrate some of the decisions one must face in Texas Hold'em, we will present a sample game, with some typical reasoning a good player might go through. This game is relatively basic, in order to make the example easier to follow. Many complex interactions can contribute to much more difficult situations, but it is hoped that this example will suffice to demonstrate some of the strategic richness of the game.

The game is \$10-\$20 Limit Hold'em with ten players. We "have the button", meaning that we will be the last to act in each betting round, which is an advantage. The two players to the left of us post the ***small blind*** (\$5) and the ***big blind*** (\$10), and the cards are dealt. The action begins with the player to the left of the big blind, who calls \$10 (we will refer to this player as "EP", for "early position"). The next three players fold (throwing their cards into the discard pile), a middle position player (MP) calls \$10, and the next two players fold.

We are next to act and have 7♢-6♢. A strong poker player would know that this is a reasonably good ***drawing hand***, which should be profitable to play for one bet from late position against several players. This would not be a good hand to call a raise with, or to play against only one or two opponents. From previous games played, we know that EP is a ***tight*** (conservative) player. We expect that EP probably has two big cards, since he called in early position (but did not raise, making large pairs highly unlikely for this particular player). Our opponent modeling has concluded that MP is a ***loose*** player, who sees the flop about 70% of the time, so he could have almost anything (*e.g.*, any pair, any two cards of the same suit, or even a hand like 6-4 of different suits). The small blind is an extremely tight player who will probably fold most hands rather than calling another \$5. The big blind almost always defends her blind (*i.e.*, she will call a raise).

A raise in this situation, for deceptive purposes, is not completely out of the question. However, it would be inappropriate against this particular set of opponents (it might be more suitable in a game with higher limits). We call the \$10, the small blind calls, and the big blind checks.

The ***flop*** is Q♠-7♡-4♢. We have ***second pair*** (connecting with the second largest card on the board) for a hand of moderate strength and moderate potential for improvement. If we do not currently have the best hand, there are five direct ***outs*** (outcomes) that can immediately improve our hand (7♠, 7♣, 6♠, 6♣, 6♡). We also have some indirect flush and straight potential, which will come in about 7% of the time,[5] and can be treated as roughly three direct outs. The ***board texture*** is fairly ***dry***, with only a few possible straight draws, and no direct flush draws. Therefore, any bets by the opponents are likely to indicate a ***made hand*** (*e.g.*, a pair) rather than a ***draw*** (a hand where additional cards are needed), unless they are a chronic bluffer. An expert player would not actually need to go through this thought process – it would simply be known the moment the flop hits the table, through experience and pattern recognition.

Both blinds check, EP bets, and MP folds (see Figure 2.1). There is $60 in the pot, and it will cost us $10 to call. We believe the bettor seldom bluffs, and almost certainly has a Queen, given his early position pre-flop call.[6] The small blind is known to ***check-raise*** on occasion, and might also have a Queen, but is more likely to have a poor match with the board cards, because he is highly selective before the flop. We have never observed the big blind check-raising in the past, so the danger of being trapped for an extra bet is not too high.

If we play, we must decide whether to raise, trying to drive the other players out of the game, or call, inviting others to call also. If there was a good chance of currently having the best hand, we would be much more inclined to raise. However, we feel that chance is relatively small in the current situation. We might also want to drive out other players who are looking to hit the same cards we want, such as 5-3, which needs a 6 to make a straight against our two pair. However, the added equity from having an extra bet in the pot is normally greater than the risk of shared outs, so we are happy to let the blinds draw with us against the bettor.

From our previous study and experience, we know that calling in this situation is a small positive expectation play, but we still cannot rule out the possibility of ***raising for a free-card***. If we raise now, we may induce the bettor to call and then

[5] 73 out of 990 outcomes (43 flushes and 30 straights).

[6] Ironically, reasonably good players are often the most predictable, whereas *very good* players are not.

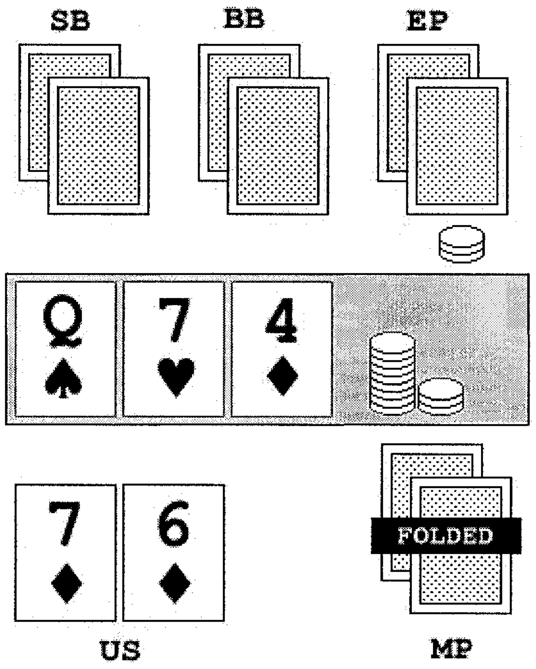


Figure 2.1: Sample game after the flop.

check to us next round, when we can also check and get a second card for "free" (actually for half-price). We need to assess the likelihood that EP will re-raise immediately (costing us two extra bets, which is a very bad result), or will call but then bet into us again on the turn anyway (costing us one extra bet). Since we do not feel we have much control over this particular player, we reject the fancy raise maneuver, and just call the $10. Both of the blinds fold, so we are now one-on-one with the bettor. Despite the many factors to consider, our decision is made quickly (normally within one second when it is our turn).

The turn card is the 5♡ and EP bets. The 5♡ gives us an open-ended draw to a straight, in addition to our other outs. In terms of expected value, this is essentially a "free pass" to the river, as we now have a clearly correct call of $20 to win $90. However, we again need to consider raising. This opponent will probably give us credit for having a very strong hand, since the 5♡ connects for several plausible two pair hands or straights. We could also be ***slow-playing*** a very strong hand, like a ***set*** (three of a kind using a ***pocket pair***, such as 4♠-4♣). Since we are quite certain he has only one pair, this particular opponent might even fold the best hand, especially if his ***kicker*** (side-card) is weak. At the very least, he will probably check to us on the river, when we can also check, unless we improve our hand. Thus, we would be investing the same amount of money as calling twice to reach the showdown, and we would be earning an extra big bet whenever we make our draw. On the other hand, we do not necessarily have to call that last bet on the river (although if we fold too often, we will become vulnerable to bluffing). We decide to make the expert play in this situation, confidently raising immediately after his bet. He thinks about his decision for a long time, and reluctantly calls.

The river card is the 5♠, and our opponent immediately checks. We know that he is not comfortable with his hand, so we can consider bluffing with what we believe is the second-best hand. From our past sessions we know that once this player goes to the river, he will usually see the hand through to the end. In effect, his decision after our raise was whether to fold, or to call two more bets. Since a bluff in this situation is unlikely to be profitable, we stick to our plan and check. He shows Q♣-J♣, we say "good hand", and throw our cards into the discard pile.[7]

Now we consider what effect this game has had on our ***table image***, in anticipation of how the players at the table will react to our future actions. The better players might have a pretty good idea of what we had (a small pair that picked up a good draw on the turn), and will not make any major adjustments to their perception of our play. Our opponent, EP, is more likely to call us down if a similar situation arises, so we might earn an extra bet on a strong hand later. Weaker players may think we are a somewhat wild gambler, so we expect them to call even more liberally against us. This reinforces our plan of seldom bluffing against them, but betting for value with more marginal hands.

[7] The rules for conducting the showdown vary depending on where the game is being played. In games played at casinos, it is standard practice to discard losing hands at the showdown, without revealing them. Players may demand to see a hand that has gone to the showdown if they suspect something suspicious (such as collusion), but it is otherwise considered to be a form of "needling" (harassment), and is socially unacceptable.

In most online games, hands at the showdown are revealed in playing order, one at a time. If a hand cannot beat the best hand shown so far, it is folded without being shown, just as in a live game. However, players have access to a full transcript of the game that includes all hands that went to the showdown. Thus, the common practice for online games is full disclosure of all cards at the showdown. This will be the standard assumption made for games between computer programs.

2.3.3 Requirements for a World-Class Poker Player

We have identified several necessary attributes for an algorithm to play poker at a world-class level. A system may handle some of these requirements indirectly, rather than by explicit design, but all of them must be solved at least satisfactorily if a program is to compete with the best human players. We present one or more ways of solving each requirement, but there are many different approaches that could be just as viable, or possibly much better. Furthermore, these components are not independent of each other. They must be continually refined and integrated as new capabilities are added to the system.

Hand Strength assesses the strength of a hand in relation to the other hands. A simple hand strength computation is a function of the cards held and the current community cards. A better evaluation takes into account the number of players still in the game, the relative position of the player at the table, and the history of betting for the current game. An even more accurate calculation considers the probabilities for each possible opponent hand, based on the likelihood of each hand being played to the current point in the game.

Hand Potential computes the probability that a hand will improve to win, or that a leading hand will lose, after future community cards appear. For example, a hand that contains four cards in the same suit may have a low hand strength, but has good potential to win with a flush as additional community cards are dealt. Conversely, a hand with a high pair might be expected to decrease in strength if many draws are available for opposing hands. At a minimum, hand potential is a function of the cards in the hand and the current community cards. However, a better calculation would use all of the additional factors described in the hand strength computation.

Bluffing makes it possible to win with a weak hand,[8] and creates doubt on the part of the opponent, thereby increasing the amount won on subsequent strong hands. Bluffing is essential for successful play. Game theory can be used to compute a theoretically optimal bluffing frequency in certain situations. A minimal

[8] Other forms of deception such as ***slow-playing*** (only calling with a strong hand) are not considered here.

bluffing system would bluff this percentage of hands, indiscriminately. In practice, other factors (such as hand potential) should also considered. A better system would identify profitable bluffing opportunities by deducing the opponent's approximate hand strength and predicting the probability of a fold.

Unpredictability makes it difficult for opponents to form an accurate model of our strategy. Mixing strategies (occasionally handling a given situation in different ways) hides information about the nature of our current hand. By varying our playing style over time, opponents may be induced to make mistakes based on incorrect beliefs.

Opponent modeling determines a likely probability distribution of the opponent's hand. Minimal opponent modeling might use a single generic model for all opponents. This can be improved by modifying those probabilities based on the personal betting history and collected statistics of each opponent.

Certain fundamental principles of poker, such as ***pot odds***, are taken as a given. There are several other identifiable characteristics that might not be necessary to play reasonably strong poker, but may eventually be required for world-class play. Collectively, these concepts are part of an overall *betting strategy*, which determines whether we fold, call, or raise in any particular situation. The most important of these attributes for poker-playing programs are discussed in greater detail in the following sections.

2.4 POKI's Architecture

A poker game consists of a *dealer* together with multiple *players* that represent either human players or computer players. In our Java implementation, these players are defined as objects. The dealer handles the addition and removal of players from the game, deals the cards to each player at the start of a new game, prompts the players for an appropriate action when it is their turn, broadcasts player actions to other players, and updates a public game context as the game progresses. The game context contains all of the public information about the game, including the names and relative locations of the players, and the board cards.

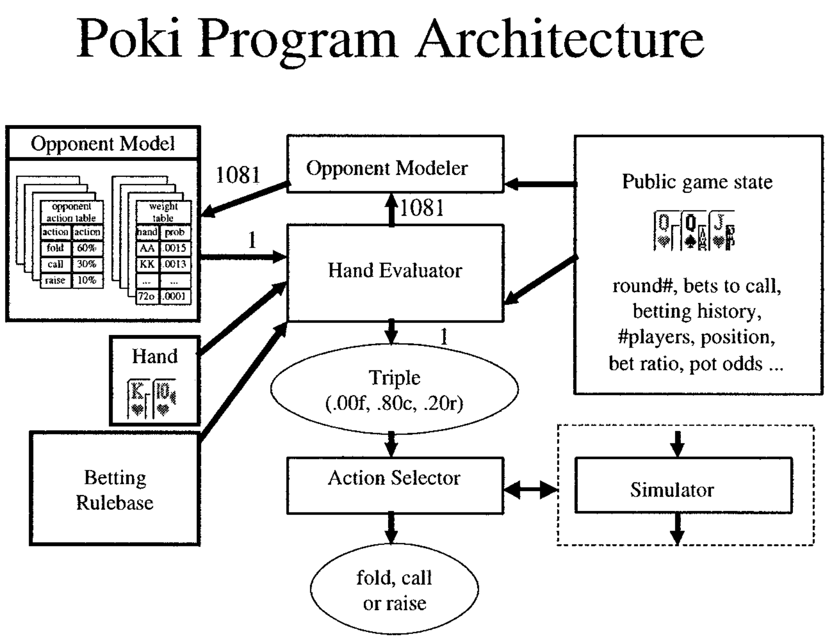


Figure 2.2: The architecture of POKI.

We have implemented several different dealer interfaces: an IRC-Dealer for playing against other players on the Internet Relay Chat poker server, a Tournament-Dealer for internal experiments, and a TCP/IP-Dealer that allows POKI to play against humans using a web browser, and against other programs using a published protocol (see `http://www.cs.ualberta.ca/~games/poker/`).

An overview of POKI's architecture is shown in Figure 2.2. Although each version of the program represents and uses the available information in a different way, all versions share a common high-level architecture.

In addition to the public game context, POKI stores private information: its current hand, and a collection of statistical opponent models. The assessment of the initial two-card hand is explained in Section 2.5.1, and the first-round betting decisions are made with a simple rule-based system. The opponent model (essentially a probability distribution over all possible hands) is maintained for each player participating in the game, including POKI itself, as detailed in Section 2.6. The Opponent Modeler uses the Hand Evaluator, a simplified rule-based Betting Strategy, and learned parameters about each player to update the current model after each opponent action, as described in Section 2.5.2.4.

After the flop, the Hand Evaluator in turn uses the opponent model and the game-state information to assess the value of POKI's hand in the current context, as explained in Sections 2.5.2.1 and 2.5.2.2. Thus, there is a certain amount of cyclic feedback among the core components of the system. The evaluation is used by a more sophisticated rule-based Betting Strategy to determine a plan (how often to fold, call, or raise in the current situation), and a specific action is chosen, as discussed in Section 2.5.2.5 and throughout Section 2.5. The entire process is repeated each time it is our turn to act. For a more advanced decision procedure, the Simulator iterates this process using different instantiations of opponent hands, as discussed in Section 2.5.3.

2.5 Betting Strategy

Betting strategies before the flop and after the flop are significantly different. Before the flop there is little information available to influence the betting decision (only two hole cards and the previous player actions), and a relatively simple expert system is sufficient for competent play. After the flop the program can analyze how all possible opponent holdings combine with the given public cards, and many other factors are relevant to each decision. A post-flop betting strategy uses the full game context, the private hand, and the applicable opponent models to generate an action. Three betting strategies will be described in this paper, one for the pre-flop and two for the post-flop.

2.5.1 Pre-flop Betting Strategy

There are $\{52 \text{ choose } 2\} = 1326$ possible hands prior to the flop. The value of one of these hands is called an *income rate*, and is based on a simple technique that we will call a *roll-out simulation*.[9] This is an off-line computation that consists of playing

[9] The term "roll-out" originates from the game of backgammon (rolling the dice). In the past, strong players would re-play a given position dozens of times in order to obtain a better estimate of

several million games (trials) where all players call the first bet (*i.e.*, the big blind), and then all the remaining cards are dealt out without any further betting. This highly unrealistic *always call assumption* does not necessarily reflect an accurate estimate for the expected value of the hand. However, it does provide a first-order approximation, and the *relative* values of the hands are reasonably accurate for the given situation.

More generally, this method is referred to as the ***all-in equity***. It is a calculation of the percentage expectation for the current hand assuming the player is ***all-in***,[10] and all active hands proceed to the showdown. It can be applied at any phase of the game, and serves as a baseline estimate of the expected value of a hand in any given situation.

2.5.1.1 Comparing Pre-flop Strategies

The best known and most widely respected expert opinion on pre-flop play is that of David Sklansky, a professional poker player and author of the most important books on the game [35, 36]. In "Hold'em Poker for the Advanced Player" [36] he prescribes a hand classification scheme to be used in typical middle-limit games (*e.g.*, \$20-\$40 Limit Hold'em). There is a strong correlation between his rankings and the results of the roll-out simulations.

Before proceeding to a closer comparison of the two ranking systems, a few caveats should be mentioned. First, there is no single ranking of starting hands that applies to all situations. An expert player will make adjustments based on the prevailing conditions (for example, a ***loose game*** (many players seeing the flop), a ***wild game*** (lots of gambling), etc.). Furthermore, the true expectation of each hand will depend on the precise context at the time of each betting decision. For example, a hand is assessed very differently after all previous players have folded than it would be after one or more players have called. The general guidelines must cover a wide variety of situations, so naturally there will be exceptions. Sklansky's recommen-

the true equity. With the advent of strong programs, computer roll-outs (re-playing the game to the end thousands of times) have become the definitive authority on the value of a given position.

[10] Under normal ***table stakes*** rules, a player who does not have enough money on the table to meet the outstanding bet can go ***all-in***, and remains eligible to win the portion of the pot contributed to. The betting continues (toward a ***side-pot***) for the remaining active hands.

dations are also intended for a full game of ten players. A completely different set of hand rankings are necessary for ***short-handed*** games (*e.g.* five players or less), and this is reflected in the different income rates computed by roll-out simulations with fewer players in the hand.

Table 2.1 shows how the roll-out simulations compare to Sklansky's rankings. In the tables, 's' refers to a ***suited*** hand (two cards of the same suit), 'o' refers to an ***offsuit*** hand (two cards of different suits, also called ***unsuited***), and '*' indicates a ***pocket pair*** (two cards of the same rank). Table 2.1 is divided into eight groups, corresponding to Sklansky's rating system, with Group 1 being the best hands, and Group 8 being weak hands that should only be played under special circumstances (*e.g.*, for one bet after many players have called). In general, there is a strong correlation between Sklansky's rankings and the income rates obtained from roll-out simulations.

The simulation values demonstrate a bias in favor of certain hands that play well against many players, known as "good multi-way hands". These are cards that can easily draw to a very strong hand, such as a flush (*e.g.*, ***suited*** hands like A♡-2♡), a straight (*e.g.*, ***connectors*** like 8♡-7♣), or three of a kind (*e.g.*, a ***pocket pair*** like 2♡-2♣). Since all ten players proceed to the showdown in a roll-out simulation, the average winning hand needs to be considerably stronger than in a real ten-player game (where typically half of the players will fold before the flop, and many games are won uncontested before the showdown). By the same reasoning, large pairs may be undervalued, because of the unaccounted potential of winning without improvement against a smaller number of opponents.

Conversely, Sklansky's rankings show evidence of a bias in favor of unsuited connectors, where suited hands should be preferred.[11] Certain small-card combinations, such as 7♠-6♠, may have been given a higher ranking by Sklansky because they add a good balance of deception to the overall play list (for example, one does not want the opposition to conclude that we cannot have a 7 when the flop is 7♡-7♢-3♣). However, the hands intended for information hiding purposes should not extend to the unsuited connectors like 7♣-6♡, which have a much lower overall

[11] The highest valued hands not in Sklansky's rankings are T7s (+231) and Q7s (+209).

Group 1		Group 2		Group 3		Group 4	
+2112	AA*	+714	TT*	+553	99*	+481	T9s [1]
+1615	KK*	+915	AQs	+657	JTs	+515	KQo
+1224	QQ*	+813	AJs	+720	QJs	+450	88*
+935	JJ*	+858	KQs	+767	KJs	+655	QTs
+1071	AKs	+718	AKo	+736	ATs	+338	98s [1]
				+555	AQo	+449	J9s
						+430	AJo
						+694	KTs
Group 5		**Group 6**		**Group 7**		**Group 8**	
+364	77*	+304	66*	+214	44*	-75	87o [2]
+270	87s [1]	+335	ATo	+92	J9o [2]	+87	53s [3] (> 43s)
+452	Q9s	+238	55*	+41	43s [3]	+119	A9o
+353	T8s [1]	+185	86s	+141	75s	+65	Q9o
+391	KJo	+306	KTo	+127	T9o	-129	76o [2]
+359	QJo	+287	QTo	+199	33*	-42	42s [3] (< 52s)
+305	JTo	+167	54s	-15	98o [2]	-83	32s [3] (< 52s)
+222	76s [1]	+485	K9s	+106	64s	+144	96s
+245	97s [1]	+327	J8s	+196	22*	+85	85s
+538	A9s			+356	K8s	-51	J8o [2]
+469	A8s			+309	K7s	+206	J7s
+427	A7s			+278	K6s	-158	65o [2]
+386	A6s			+245	K5s	-181	54o [2]
+448	A5s			+227	K4s	+41	74s
+422	A4s			+211	K3s	+85	K9o
+392	A3s			+192	K2s	-10	T8o
+356	A2s			+317	Q8s		
+191	65s [1]						

Three possible explanations for the differences: [1] small card balancing, [2] bias for ***unsuited connectors***, and [3] logical error (inconsistent).

Table 2.1: Income Rate values versus Sklansky groupings.

expectation.

There are also a few instances of small logical errors in Sklansky's rankings. For example, 43s is ranked in Group 7, ahead of 53s in Group 8, but it can be shown that 53s logically dominates 43s, because it has the same straight and flush potential, with better high-card strength. Similarly, 52s dominates 42s and 32s, but 52s is not ranked in any of the eight groups, whereas the latter are members of Group 8.

Since the differences are not large, it is clear that roll-out simulations provide an acceptable means of quantifying the pre-flop value of each hand. This information is currently used as part of a formula-based expert system for playing before the flop, which is not unlike the guidelines given by Sklansky in the aforementioned text. We prefer to use the computed results, rather than transcribing the Sklansky rules, because (a) we wish to eliminate the use of human knowledge whenever possible, (b) the roll-out simulation information is quantitative rather than qualitative, and (c) the algorithmic approach can be applied to many different specific situations (such as having exactly six players in the game), whereas Sklansky gives only a few recommendations for atypical circumstances.

Future versions of the program should be even more autonomous, adapting to the observed game conditions and making context-sensitive decisions on its own.

2.5.1.2 Iterated Roll-Out Simulations

An interesting refinement to roll-out simulation is to use repeated iterations of the technique, where the previous results govern the betting decision for each player. In the ten-player case, a negative value in the previous simulation would dictate that the hand be folded, rather than calling the big blind. This drastically reduces the number of active players in each game, producing a more realistic distribution of opponents and probable hands. The result is a reduction in the bias toward multi-way hands, and a much better estimation of the hands that can be played profitably when ten players are originally dealt in.

After each round of simulations has reached a reasonable degree of stability, another iteration is performed. This process eventually reaches an equilibrium,

Hand	IR-10	Iterated	Hand	IR-10	Iterated	Hand	IR-10	Iterated
AA*	+2112	+2920	ATs	+736	+640	KQo	+515	+310
KK*	+1615	+2180	99*	+553	+630	QTs	+655	+280
QQ*	+1224	+1700	KQs	+858	+620	QJs	+720	+270
JJ*	+935	+1270	AQo	+555	+560	A9s	+538	+220
TT*	+714	+920	KJs	+767	+480	ATo	+335	+200
AKs	+1071	+860	88*	+450	+450	KTs	+694	+190
AKo	+718	+850	77*	+364	+390	KJo	+391	+160
AQs	+915	+780	AJo	+430	+380	A8s	+469	+110
AJs	+813	+680	JTs	+657	+360	66*	+304	+40

Table 2.2: Iterated income rate (profitable hands).

defining a set of hands that can be played profitably against the blinds and the other unknown hands. The results are most applicable to the "play or don't play" decision for each player. Although much better than a simple roll-out simulation, this technique is still far from perfect, because other important considerations such as betting position and known opponent actions have not been accounted for.

In our experiments, each iteration lasted for 50,000 trials. A diminishing noise factor was added to each income rate, analogous to the cooling factor used in simulated annealing. This gives negative expectation hands a chance to recover as the prevailing context changes. After ten generations, the remaining positive expectation hands were played for another 500,000 trials, to ensure stability. The resulting set of profitable hands, shown in Table 2.2, is in strong agreement with expert opinion on this matter. The table shows a comparison of the income rates for 10-player roll-out simulations (IR-10) and the results refined by iterating (Iterated). The values shown are in *milli-bets* (*e.g.*, a hand with an income rate of +1000 should win an average of one small bet each time it is played). The iterated values are reasonable estimates of actual income rates, unlike the simple roll-out values, which are only used as relative measures.

One of the factors used by Sklansky and other experts is the possibility of a hand being ***dominated***. For example, AQ is said to dominate AJ, because the AQ has a tremendous advantage if they are in a game against each other (an Ace on board does not help the AJ). In contrast, AQ does not dominate the inferior holding of KJ, because they are striving to hit different cards. The role of domination is clearly demonstrated in the results of the iterated roll-out simulations. Examples include the increased value of large pairs and AK unsuited, and the diminished value of KQ (which is dominated by AA, KK, QQ, AK, and AQ).

Iterated roll-out simulations have also been used to compute accurate expected values for two-player *Pre-flop Hold'em*. The resulting betting decisions are in very good agreement with Alex Selby's computation of the game-theoretic equilibrium strategy, in which he used an adaptation of the Simplex algorithm for solving this game directly [30].[12] The small number of cases where the strategies differ are all near the boundary conditions between raise and call, or call and fold. Furthermore, the expected values are always close to the threshold for making the alternate choice, with a difference usually less than 0.1 small bets (100 milli-bets).

2.5.2 Basic Betting Strategy

The basic betting strategy after the flop chooses an action using three steps:

1. Compute the *hand strength* (HS), *positive potential* (PPot), *negative potential* (NPot), and *effective hand strength* (EHS) of POKI's hand relative to the board.

2. Use the game context, a set of betting rules, and formulas to translate the EHS into a *probability triple*: {*Pr(fold), Pr(call), Pr(raise)*}.

3. Generate a random number in the range zero to one, and use it to choose an action from the probability distribution. This contributes to the unpredictability of the program.

EHS is a measure of how well the program's hand stands in relationship to the remaining active opponents in the game. It is a combination of the current hand strength and positive potential for the hand to improve. These are discussed in the following sections.

[12] We are assuming that an optimal equilibrium solution to the re-defined game of *Pre-flop Hold'em* will serve as a near-optimal equilibrium solution to the pre-flop phase of real Hold'em (*i.e.*, that a "perfect" solution to a simpler game will be a "good" solution to the full-scale version).

```
HandStrength(ourcards,boardcards)
{
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   /* Consider all two-card combinations
      of the remaining cards.  */
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank)          ahead += 1
      else if(ourrank==opprank)     tied += 1
      else /* < */                behind += 1
   }
   handstrength = (ahead+tied/2) / (ahead+tied+behind)
   return(handstrength)
}
```

Figure 2.3: Hand Strength calculation.

2.5.2.1 Hand Strength

The *hand strength* (HS) is the probability that a given hand is better than that of an active opponent. Suppose an opponent is equally likely to have any possible two hole card combination.[13] All of these opponent hands can be enumerated, identifying when POKI's hand is better (+1), tied (+$\frac{1}{2}$), or worse (0). Taking the summation and dividing by the total number of possible opponent hands gives the (unweighted) hand strength. Figure 2.3 gives the algorithm for a simple hand strength calculation.

Suppose our hand is A♢-Q♣ and the flop is J♡-4♣-3♡. There are 47 remaining unknown cards and therefore {47 choose 2} = 1,081 possible hands an opponent might hold. In this example, any three of a kind, two pair, one pair, or AK is better (444 cases), the remaining AQ combinations are equal (9 cases), and the rest of the hands are worse (628 cases). Counting ties as one half, this corresponds to a percentile ranking, or hand strength, of 0.585. In other words, there is a 58.5% chance that A♢-Q♣ is better than a random hand.

The hand strength calculation is with respect to one opponent, but can be ex-

[13] This is not true, in general, but simplifies the presentation of the algorithm. This form of unweighted hand strength is also called *hand rank* (HR). We eliminate this assumption and generalize the algorithm in the next section.

A♢-Q♣ hole cards	J♡-4♣-3♡ board cards				
5 Cards	7 Cards				
	Ahead	Tied	Behind	Sum	
Ahead	449,005	3,211	169,504	628x990 =	621,720
Tied	0	8,370	540	9x990 =	8,910
Behind	91,981	1,036	346,543	444x990 =	439,560
Sum	540,986	12,617	516,587	1,081x990 =	1,070,190

Table 2.3: Hand Potential example.

trapolated to multiple opponents by raising it to the power of the number of active opponents.[14] Against five opponents with random hands, the adjusted hand strength, HS_5, is $0.585^5 = 0.069$. Hence, the presence of the additional opponents has reduced the likelihood of A♢-Q♣ being the best hand to only 6.9%.

2.5.2.2 Hand Potential

After the flop, there are still two more board cards to be revealed. On the turn, there is one more card to be dealt. We want to determine the potential impact of these cards. The *positive potential* (PPot) is the chance that a hand that is not currently the best improves to win at the showdown. The *negative potential* (NPot) is the chance that a currently leading hand ends up losing.

PPot and NPot are calculated by enumerating over all possible hole cards for the opponent, like the hand strength calculation, and also over all possible board cards. For all combinations of opponent hands and future cards, we count the number of times POKI's hand is behind, but ends up ahead (PPot), and the number of times POKI's hand is ahead but ends up behind (NPot). The algorithm is given in Figure 2.4, and the results for the preceding example are shown in Table 2.3. In this example, if the hand A♢-Q♣ is ahead against one opponent after five cards, then after 7 cards there is a 449,005 / 621,720 = 72% chance of still being ahead.

Computing the potential on the flop can be expensive, given the real-time con-

[14] This assumes that all of the opponent hands are independent of each other. Strictly speaking, this is not true. To be a useful estimate for the multi-player case, the error from this assumption must be less than the error introduced from other approximations made by the system. More accurate means are available, but we defer that discussion in the interest of clarity.

```
HandPotential(ourcards,boardcards)
{
   /* Hand Potential array, each index represents
      ahead, tied, and behind.  */
   integer array HP[3][3]  /* initialize to 0 */
   integer array HPTotal[3] /* initialize to 0 */

   ourrank = Rank(ourcards,boardcards)
   /* Consider all two-card combinations of the
      remaining cards for the opponent.  */
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank)      index = ahead
      else if(ourrank=opprank) index = tied
      else /* < */             index = behind
      HPTotal[index] += 1

      /* All possible board cards to come.  */
      for each case(turn)
      {
         for each case(river)
         {  /* Final 5-card board */
            board = [boardcards,turn,river]
            ourbest = Rank(ourcards,board)
            oppbest = Rank(oppcards,board)
            if(ourbest>oppbest)       HP[index][ahead]  += 1
            else if(ourbest==oppbest)  HP[index][tied]  += 1
            else /* < */             HP[index][behind]  += 1
         }
      }
   }

   /* PPot:  were behind but moved ahead.  */
   PPot = (HP[behind][ahead] + HP[behind][tied]/2
        + HP[tied][ahead]/2) / (HPTotal[behind]+HPTotal[tied]/2)
   /* NPot:  were ahead but fell behind.  */
   NPot = (HP[ahead][behind] + HP[tied][behind]/2
        + HP[ahead][tied]/2) / (HPTotal[ahead]+HPTotal[tied]/2)
   return(PPot,NPot)
}
```

Figure 2.4: Hand Potential calculation.

straints of the game (about one second per decision). There are $\{45 \text{ choose } 2\} = 990$ possible turn and river cards to consider for each possible two-card holding by the opponent. In practice, a fast approximation of the PPot calculation may be used, such as considering only the next one card to come. Previous implementations have used a fast function to produce a crude estimate of PPot, which was within 5% of the actual value about 95% of the time.

2.5.2.3 Effective Hand Strength

The *effective hand strength* (EHS) combines hand strength and potential to give a single measure of the relative strength of POKI's hand against an active opponent. One simple formula for computing the probability of winning at the showdown[15] is:

$$\begin{aligned} Pr(win) &= Pr(ahead) \times Pr(opponent\ does\ not\ improve) \\ &\quad + Pr(behind) \times Pr(we\ improve) \\ &= HS \times (1 - NPot) \ + \ (1 - HS) \times PPot \end{aligned}$$

In practice, we generally want to bet when we currently have the best hand, regardless of negative potential, so that an opponent with a marginal hand must either fold, or pay to draw. Hence, NPot is not as important as PPot for betting purposes. Since we are interested in the probability that our hand is either currently the best, or will improve to become the best, one possible formula for EHS sets NPot $= 0$, giving:

$$EHS = HS + (1 - HS) \times PPot \tag{2.1}$$

This has the effect of betting a hand aggressively despite good draws being possible for opponent hands, which is a desirable behavior.

For n active opponents, this can be generalized to:

$$EHS = HS^n + (1 - HS^n) \times PPot \tag{2.2}$$

[15] The formula can be made more precise by accounting for ties, but becomes less readable.

assuming that the same EHS calculation suffices for all opponents. This is not a good assumption, since each opponent has a different style (and thus a different distribution over possible hands). A better generalization is to have a different HS and PPot for each opponent i. EHS with respect to each opponent can then be defined as:

$$EHS_i = HS_i + (1 - HS_i) \times PPot_i \tag{2.3}$$

Modifying these calculations based on individual opponents is the subject of Section 2.6.

2.5.2.4 Weighting the Enumerations

The calculations of hand strength and hand potential in Figures 2.3 and 2.4 assume that all two-card combinations are equally likely. However, the probability of each hand being played to a particular point in the game will vary. For example, the probability that an active opponent holds Ace-King is much higher than 7-2 after the flop, because most players will fold 7-2 before the flop.

To account for this, POKI maintains a *weight table* for each opponent. The table has an entry for every possible two-card hand, where each value is the conditional probability of the opponent having played those cards to the current point in the game. To get a better estimate of hand strength, each hand in the enumeration is multiplied by its corresponding probability in the weight table.

In practice, the weights have a value in the range zero to one, rather than absolute probabilities (summing to one), because only the relative sizes of the weights affect the later calculations. When a new game begins, all entries are initialized to a weight of one. As cards become known (POKI's private cards or the public board cards), many hands become impossible, and the weight is set to zero.

After each betting action, the weight table for that opponent is updated in a process called *re-weighting*. For example, suppose an opponent calls before the flop. The updated weight for the hand 7-2 might be 0.01, since it should normally be folded. The probability of Ace-King might be 0.40, since it would seldom be folded before the flop, but is often raised. The relative value for each hand is increased or

```
UpdateWeightTable(Action A, WeightTable WT,
                  GameContext GC, OpponentModel OM)
{
   foreach (entry E in WT)
   {
      ProbabilityDistribution PT[FOLD,CALL,RAISE]

      PT = PredictOpponentAction(OM, E, GC)
      WT[E] = WT[E] * PT[A]
   }
}
```

Figure 2.5: Updating the weight table.

decreased to be consistent with every opponent action.

The strength of each possible hand is assessed, and a ***mixed strategy*** (a probability distribution over available actions) is determined by a formula-based betting strategy. These values are then used to update the weight table after each opponent action. The algorithm is shown in Figure 2.5.

For example, assume that the observed player action is a bet, and that the weight table currently has entries:

[A♠-K♣, 0.40], ..., [Q♢-2♢, 0.20], ...

Further assume that in the given situation, the PredictOpponentAction procedure (Figure 2.5) generates probability distributions $\{$*Pr(fold), Pr(check/call), Pr(bet/raise)*$\}$ of $\{0.0,\ 0.7,\ 0.3\}$ for the hand A♠-K♣, and $\{0.0,\ 0.1,\ 0.9\}$ for the hand Q♢-2♢. After re-weighting, the new weight table entry for A♠-K♣ will be $0.4 \times 0.3\ =\ 0.12$, and $0.2\ \times\ 0.9\ =\ 0.18$ for Q♢-2♢. Had the opponent checked in this situation, the weights would be 0.28 and 0.02, respectively.[16]

Table 2.4 shows a possible game scenario based on the example given in Section 2.3.2 (with the five players that immediately folded in the pre-flop removed). In this game, player EP is assumed to be a default player rather than the well-modeled tight opponent described previously. Figure 2.6 shows POKI's weight table for EP at three stages of the game (pre-flop, flop, and river). In each figure, darker cells

[16] In the parlance of Bayesian (conditional) probabilities, the old weight table represents the *prior distribution* of the opponent's cards, and the new weight table is the *posterior distribution*.

	SB	BB	EP	MP	Poki
Pre-flop					
	small blind	big blind	call	call	call
	call	check			
Flop Q♠ 7♡ 4♢					
	check	check	bet	fold	call
	fold	fold			
Turn 5♡					
			bet		raise
			call		
River 5♠					
			check		check

Table 2.4: Betting scenario for the example game in Section 2.3.2.

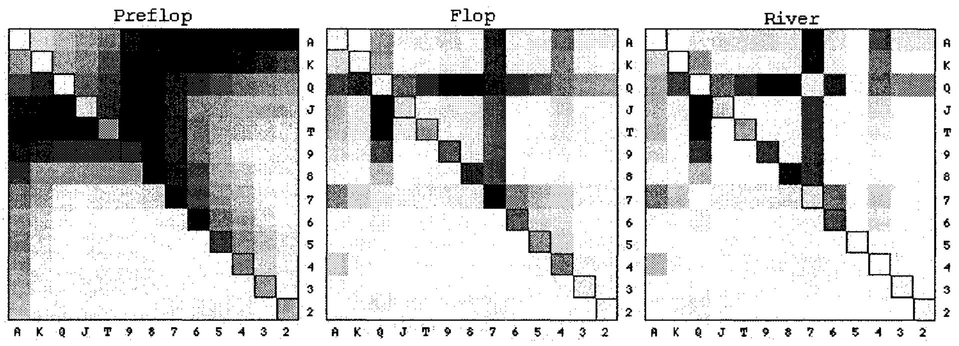


Figure 2.6: Progressive weight tables for opponent EP in the example game.

correspond to higher relative weights. ***Suited*** hands are shown in the upper right portion of the grid, and ***unsuited*** hands are on the lower left. The program gathers more information as the game is played, refining the distribution of hands that are consistent with the betting actions of EP.[17]

2.5.2.5 Probability Triples and Evaluation Functions

A *probability triple* is an ordered triple of values, $PT = \{f, c, r\}$, such that $f + c + r = 1.0$, representing the probability distribution that the next betting action in a given context is a fold, call, or raise, respectively. This representation of

[17] The weight table for the turn is not shown, but is similar to that of the flop (the continuation bet provides little new information).

future actions (analogous to a *randomized mixed strategy* in game theory) is used in three places in POKI:

1. The basic betting strategy uses a probability triple to decide on a course of action (fold, call, or raise).

2. The opponent modeling component (Section 2.6) uses an array of probability triples to update the opponent weight tables.

3. In a simulation-based betting strategy (Section 2.5.3) probability triples are used to choose actions for simulated opponent hands.

Hand strength, hand potential, and effective hand strength are simple algorithms for capturing some of the probabilistic information needed to make a good decision. However, there are many other factors that influence the betting decision. These include things like ***pot odds***, ***implied odds***, relative betting position, betting history of the current game, etc. Hence, the probability triple generation routine consists of *ad hoc* rules and formulas that use EHS, the opponent model, game conditions, and probability estimates to assess the likelihood of each possible betting action. A professional poker player (Billings) defined this system based on crude estimates of the return on investment for each betting decision. We refer to this as either a rule-based or formula-based betting strategy. The precise details of this procedure will not be discussed, as they are of limited scientific interest.

An important advantage of the probability triple abstraction is that most of the expert-defined knowledge in POKI has been gathered together into the triple-generation routines. This is similar to the way that external knowledge is restricted to the evaluation function in alpha-beta search. The probability triple framework allows the "messy" elements of the program to be amalgamated into one component, which can then be treated as a black box by the rest of the system. Thus, aspects like Hold'em-specific knowledge, complex expert-defined rule systems, and knowledge of human behavior are all separated from the engine that uses this input for its calculations. The essential algorithms should be applicable to other poker variants with little or no modification, and perhaps to substantially different domains.

2.5.3 Selective Sampling and Simulation-based Betting Strategy

Having an expert identify all the betting rules necessary to play poker is time consuming and difficult. The game is strategically complex, and decisions must be based on the exact context of the current game and historical information of past sessions. A system based on expert rules is unlikely to produce a world-class level of play, because covering every relevant situation in sufficient detail is not feasible. We believe that dynamic, adaptive, computer-oriented techniques will be essential to compete with the best human players.

As mentioned above, a knowledge-based betting strategy is analogous to a static evaluation function in deterministic perfect information games. Given the current state of the game, it attempts to determine the action that yields the best result. The corresponding analogue would be to add search to the evaluation function. While this is easy to achieve in a game such as chess (consider all possible moves as deeply as resources permit), the same approach is not directly applicable to poker. There are fundamental differences in the structure of imperfect information game trees, and the total number of possibilities to consider is prohibitive.

Toward this end, POKI supports a simulation-based betting strategy. It consists of playing out many likely scenarios, keeping track of how much money each decision will win or lose. Every time it faces a decision, POKI invokes the Simulator to get an estimate of the expected value of each betting action (see the dashed box in Figure 2.2, with the Simulator replacing the Action Selector). A single trial consists of playing out the game from the current state through to the end. Many trials produce a *full-information simulation* (which is not to be confused with the simpler roll-out simulations mentioned in Section 2.5.1).

Each trial is played out twice – once to consider the consequences of a check or call, and once to consider a bet or raise. In each trial, a hand is assigned to each opponent, based on the probabilities maintained in their weight table. The resulting instance is simulated to the end, and the amount of money won or lost is determined. Probability triples are used to determine the future actions of POKI and the opponents, based on the two cards they are assigned for that trial and threshold values determined by the specific opponent model. The average over all trials in which we check or call is the *call EV*, and the average for the matching trials where we bet or raise is the *raise EV*. The *fold EV* can be calculated without simulation, since there is no future profit or loss.

In the current implementation, we simply choose the action with the greatest expectation. If two actions have the same expectation, we opt for the most aggressive one (prefer a raise, then a call, then a fold). To increase the program's unpredictability, we can randomize the selection between betting actions whose EVs are close in value, but the level of noise in the simulation already provides some natural variation for close decisions.[18]

Enumerating all possible opponent hands and future community cards would be analogous to exhaustive game-tree search, and is impractical for poker. Simulation is analogous to a selective expansion of some branches of a game tree. To get a good approximation of the expected value of each betting action, one must have a preference for expanding and evaluating the nodes that are most likely to occur. To obtain a correctly weighted average, all of the possibilities must be considered in proportion to the underlying non-uniform probability distribution of the opponent hands and future community cards. We use the term *selective sampling* to indicate that the assignment of probable hands to each opponent is consistent with this distribution.

At each betting decision, a player must choose a single action. The choice is strongly correlated to the quality of the cards that they have, and we can use the opponent model and formula-based betting strategy to compute the likelihood that the player will fold, call, or raise in each instance. The player's action is then randomly selected based on this probability distribution, and the simulation proceeds. As shown in Figure 2.2, the Simulator calls the opponent model to obtain each of our opponent's betting actions and our own actions. Where two or three alternatives are equally viable, the resulting EVs should be nearly equal, so there is little consequence if the "wrong" action is chosen.

[18] Unfortunately, this simple approach does convey some useful information to observant opponents, in that the strength of our hand and the betting level are too closely correlated. Moving toward a mixed equilibrium strategy would provide better information-hiding, and may be necessary to reach the world-class level.

It is reasonable to expect that the simulation approach will be better than the static approach, because it essentially uses a selective search to augment and refine a static evaluation function. Barring serious misconceptions, or bad luck on a limited sample size, playing out many relevant scenarios will improve the estimates obtained by heuristics alone, resulting in a more accurate assessment overall.

As seen in other domains, we find that the search itself contains implicit knowledge. A simulation contains inherent information that improves the basic evaluation, such as:

- hand strength (fraction of trials where our hand is better than the one assigned to the opponent),
- hand potential (fraction of trials where our hand improves to the best, or is overtaken), and
- subtle considerations that are not addressed in the simplistic betting strategy (*e.g.*, ***implied odds***, extra bets won after a successful draw).

It also allows complex strategies to be *uncovered without providing additional expert knowledge*. For example, simulations produce advanced betting tactics like ***check-raising*** as an emergent property, even if the basic strategy used within each trial is incapable of this play.

At the heart of the simulation is the evaluation function, discussed in Section 2.5.2.5. The better the quality of the evaluation function, the better the simulation results will be. Furthermore, the evaluation system must be compatible and harmonious with the nature of the simulations. Since the formula-based betting strategy was developed and tuned for the original system, it may not be entirely consistent or appropriate for use in the simulation-based version. It is possible that built-in biases that were useful (or compensated for) in the original version are sources of serious systemic error when used as the evaluation function for simulations. It may be the case that a simpler function would be more balanced, producing better results.

One of the interesting results of work on alpha-beta search is that even a simple evaluation function can result in a powerful program. We see a similar situation in poker. The implicit knowledge contained in the search itself improves the basic evaluation, refining the quality of the approximation. As with alpha-beta, there are important trade-offs to consider. A more sophisticated evaluation function can reduce the size of the tree, at the cost of more time spent on each node. In simulation analysis, we can improve the accuracy of each trial, but at the expense of reducing the total number of trials performed in real-time.

Variations of selective sampling have been used in other games, including Scrabble [33, 32], backgammon [39], and bridge [17]. *Likelihood weighting* is another method of biasing stochastic simulations [15, 31]. In our case, the goal is different because we need to differentiate between EVs (for call/check, bet/raise) instead of counting events. Poker also imposes tight real-time constraints (typically a maximum of a few seconds per decision). This forces us to maximize the information gained from a limited number of samples. The problem of handling unlikely events (which is a concern for any sampling-based result) is smoothly handled by the re-weighting system (Section 2.5.2.4), allowing POKI to dynamically adjust the likelihood of an event based on observed actions. An unlikely event with a large payoff figures naturally into the EV calculations.

2.6 Opponent Modeling

No poker strategy is complete without a good opponent modeling system. A strong poker player must develop a dynamically changing (adaptive) model of each opponent, to identify potential weaknesses.

In traditional games, such as chess, this aspect of strategy is not required to achieve a world-class level of play. In perfect information games, it has been sufficient to play an objectively best move, without special regard for the opponent. If the opponent plays sub-optimally, then continuing to play good objective moves will naturally exploit those errors. Opponent modeling has been studied in the context of two-player games, but the research has not translated into significant performance benefits [10, 19, 20].

In poker, the situation is different. Two opponents can make opposite kinds of errors – both can be exploited, but it requires a different response for each. For example, one opponent may bluff too much, the other too little. We adjust by calling more frequently against the former, and less frequently against the latter. To simply call with the optimal frequency would decline an opportunity for increased profit, which is how the game is scored. Even very strong players can employ radically different styles, so it is essential to try to deduce each opponent's basic approach to the game, regardless of how well they play.

2.6.1 RoShamBo

The necessity of modeling the opponent is nicely illustrated in the game of RoShamBo (also known as Rock-Paper-Scissors). This is a well-known "kid's game", where each player chooses an action simultaneously, and there is a cyclic set of outcomes: scissors beats paper, paper beats rock, and rock beats scissors (choosing the same action results in a tie). The game-theoretic equilibrium strategy for this zero-sum game is also well known: one chooses any of the three actions uniformly at random. However, the equilibrium strategy is oblivious to opponent actions, and is not exploitive. The best one can do using the equilibrium strategy is to break even in the long run (an expected value of zero, even if the opponent *always* goes rock). Contrary to popular belief, the game is actually very complex when trying to out-guess an intelligent opponent.

The International RoShamBo Programming Competition[19] is a contest for programs that play Rock-Paper-Scissors [3]. More than 50 entries were submitted from all over the world for each competition. Every program plays every other program in a round-robin tournament, with each match consisting of 1,000 games. Scores are based on total games won, and on the match results (with the match declared a draw if the scores are not different by a statistically significant margin). Since the equilibrium strategy can only draw each match, it consistently finishes in the middle of the pack, and has no chance of winning the tournament.

The authors of the top entries, including some well-known AI researchers, have commented that writing a strong RoShamBo program was much more challenging

[19] See `http://www.cs.ualberta.ca/~games`.

than they initially expected [4, 13]. The best programs do sophisticated analysis of the full history of the current match in order to predict the opponent's next action, while avoiding being predictable themselves. Programs that used a simple rule-base for making their decisions consistently finished near the bottom of the standings. All of the top programs define completely general methods for pattern detection, some of which are remarkably elegant. Given the simple nature of RoShamBo, some of these nice ideas might be applicable to the much more complex problems faced by a poker playing system.

2.6.2 Statistics-based Opponent Modeling

In poker, opponent modeling is used in at least two different ways. We want a general method of deducing the strength of the opponent's hand, based on the betting actions. We also want to predict their specific action in a given situation.

At the heart of an opponent modeling system is a *predictor*. The predictor's job is to map any given game context into a probability distribution over the opponent's potential actions. In Limit poker, this distribution can be represented by a *probability triple* {*Pr(fold), Pr(call), Pr(raise)*}.

One way to predict an opponent action would be to use our own betting strategy, or some other set of rules, to make a rational choice on behalf of the opponent. When we use this type of fixed strategy as a predictor, we are assuming the player will play in one particular "reasonable" manner, and we refer to it as *generic opponent modeling* (GOM).

Another obvious method for predicting opponent actions is to expect them to continue to behave as they have done in the past. For example, if an opponent is observed to bet 40% of the time immediately after the flop, we can infer that they will normally bet with the top 40% of their hands in that situation (including a certain percentage of weak hands that have a good draw). When we use an opponent's personal history of actions to make predictions, we call it *specific opponent modeling* (SOM).

Our first opponent modeling effort was based on the collection of simple statistical information, primarily on the betting frequencies in a variety of contexts.

For example, a basic system distinguishes twelve contexts, based on the betting round (pre-flop, flop, turn, or river), and the betting level (zero, one, or two or more bets). For any particular situation, we use the historical frequencies to determine the opponent's normal requirements (*i.e.*, the average effective hand strength) for the observed action. This threshold is used as input into a formula-based betting strategy that generates a ***mixed strategy*** of rational actions for the given game context (see Section 2.5.2.5).

However, this is a limited definition of distinct contexts, since it does not account for many relevant properties, such as the number of active opponents, the relative betting position, or the texture of the board cards (*e.g.*, whether many draws are possible). Establishing a suitable set of conditions for defining the various situations is not an easy task. There are important trade-offs that determine how quickly the algorithm can learn and apply its empirically discovered knowledge. If a context is defined too broadly, it will fail to capture relevant information from very different circumstances. If it is too narrow, it will take too long to experience enough examples of each scenario, and spotting general trends becomes increasingly difficult. Equally important to deciding how many equivalence classes to use is knowing what kinds of contextual information are most relevant in practice.

Furthermore, there are many considerations that are specific to each player. For example, some players will have a strong affinity for flush draws, and will raise or re-raise on the flop with only a draw. Knowing these kinds of personality-specific characteristics can certainly improve the program's performance against typical human players, but this type of modeling has not yet been fully explored.

Opponent modeling in poker appears to have many of the characteristics of the most difficult problems in machine learning – noise, uncertainty, an unbounded number of dimensions to explore, and a need to quickly learn and generalize from relatively small number of heterogeneous training examples.[20] As well, the real-time nature of poker (a few seconds per betting decision) limits the effectiveness of most popular learning algorithms.

[20] By "heterogeneous" we mean that not all games and actions reveal the same type or amount of information. For example, if a player folds a hand, we do not get to see the cards.

2.6.3 Neural Networks-based Opponent Modeling

To create a more general system for opponent modeling, we implemented a neural network for predicting the opponent's next action in any given context. Guessing the next action is useful for planning advanced betting strategies, such as a ***check-raise***, and is also used in each trial of a full-information simulation (see Section 2.5.3).

A standard feed-forward neural net was trained on contextual data collected from online games against real human opponents. The networks contain a set of nineteen inputs corresponding to properties of the game context, such as the number of active players, ***texture of the board***, opponent's position, and so on. These are easily identified factors that may either influence, or are correlated with a player's next action.

The output layer consists of three nodes corresponding to the fold, call, and raise probabilities. Given a set of inputs, the network will produce a probability distribution of the opponent's next action in that context (by normalizing the values of the three output nodes).

By graphically displaying the relative connection strengths, we are able to determine which input parameters have the largest effects on the output. After observing networks trained on many different opponents, it is clear that certain factors are dominant in predicting the actions of most opponents, while other variables are almost completely irrelevant. The accuracy of these networks (and other prediction methods) is measured by cross-validating with the real data collected from past games with each opponent. Details are available in a previous paper [12].

Figure 2.7 shows a typical neural network after being trained on a few hundred games played by a particular opponent. The inputs are the on the top row, with the activation level ranging from zero (fully white) to one (fully black). The thickness of the lines represent the magnitude of the weights (black being positive, grey being negative). In this example, the connections from input node number twelve (true if the opponent's last action was a raise) are very strong, indicating that it is highly correlated with what the opponent will do next. The bottom row shows the network predicting that the opponent will probably fold, with a small chance of calling.

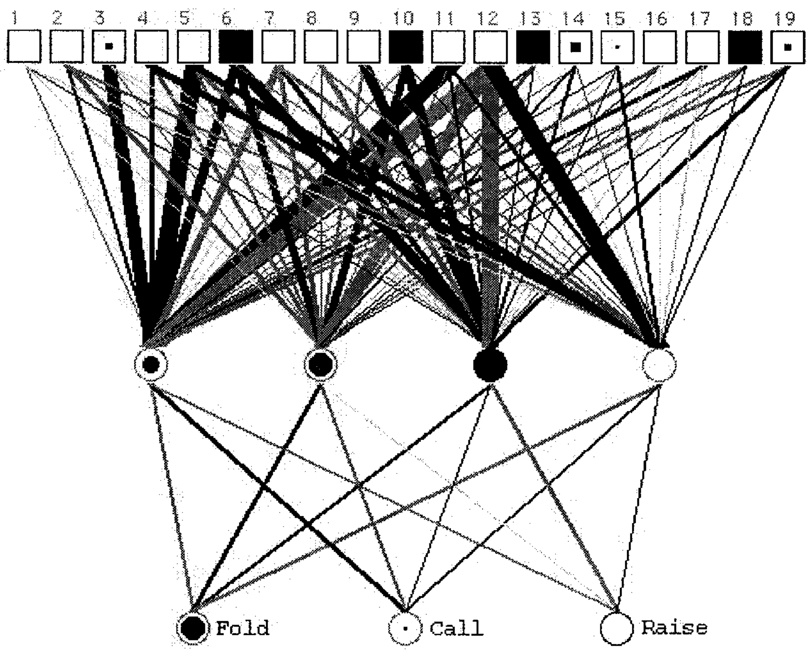


Figure 2.7: A neural network predicting an opponent's future action.

The first result of this study was the identification of new features to focus on when modeling common opponents. This produced a relatively small set of context equivalence classes that significantly improved the statistical opponent modeling reported previously [12]. We are currently experimenting with using a real-time neural network system to replace the frequency table method entirely. Preliminary results from games with both human and computer opponents suggest that this may lead to a dramatic improvement.

2.7 Performance Evaluation

Measuring the performance of a poker-playing program is difficult. POKI is a complex system of interacting components, and changing a single component often has cascading effects, leading to unpredictable and unforeseen behavior. We have employed a variety of methods for assessing the program, but none of them is completely adequate.

2.7.1 Experimental Methodology

Poker is a game of high variance, and the element of luck dominates the outcome of any one game. Among evenly matched players, the effects of good or bad fortune are still significant even after several thousand games. Measurements are always susceptible to high levels of noise and anomalous games. Furthermore, players are constantly adapting during this time, improving their understanding of each opponent, or changing styles to make it more difficult for others to form an accurate model of them.

Internal experiments are a simple way to test new features, by playing older versions of the program against newer versions. This provides an easily controlled closed environment, where many thousands of games can be played quickly.

To reduce variance we use a duplicate tournament system similar to that used in duplicate bridge. Since each game can be played with no memory of preceding games, in a ten-player game, each deal of the cards can be replayed ten times, shuffling the seating arrangement each time so that every player holds each hand once. This reduces the amount of noise considerably, and also reduces the effects of relative seating position (for example, it would be advantageous to always act immediately after a particularly aggressive or unpredictable player). However, this method still admits a lot of variance. For example, one player might choose to fold a marginal hand whereas another might play in that same situation, possibly winning or losing many bets.

Another assessment method attempts to compute an objective measurement of the expected value for each decision, using the perfect information of the actual situation. For example, a weak looking hand might actually win 20% of the time against the current field, and the expected value for making a "loose call" in that situation might be +0.6 bets, compared to -0.6 bets for a more conservative fold, or -0.2 bets for a raise. When comparing two or more players, this kind of specific evaluation of an action can be applied to the first differing decision of each deal, since the subsequent betting actions are not comparable (being different contexts). While this method is attractive in principle, it is somewhat difficult to define a reliable expected value measure for all situations, and consequently it has not been used extensively to date.[21]

The major drawback of internal experiments is that they lack the wide variety of styles and game conditions exhibited by real players. Other researchers have previously commented on the "myopia" of self-play games in chess [2]. The problem is much more acute and limiting for the development of a poker-playing system, because the style of the opponent is of paramount importance to correct play. A program that does very well against normal opponents may be vulnerable to a particular type of erratic or irrational player, even if the play is objectively worse. Although we try to create a variety of computer opponents by varying parameter settings of the players (*e.g.*, percentage of hands played, aggressiveness, advanced betting strategies, etc.), the range of styles is still much more restricted than that of human opponents.

Even with a carefully selected, well-balanced field of artificial opponents, it is important to not over-interpret the results of any one experiment. Often all that can be concluded is the relative ranking of the algorithms amongst themselves. One particular strategy may dominate in an internal experiment, even though another approach is more robust in real games against human opponents.

A good demonstration of this limitation was seen in the testing of early simulation-based betting strategies. The results of internal experiments were very encouraging, and occasionally spectacular. However, this was largely due to the pure aggressiveness of the new strategy, which was particularly effective at exploiting the overly conservative nature of its computer opponents at that time. When testing the new betting strategy in online games, it was much less successful against reasonably strong human opposition, who were able to adapt quickly.

For this reason, playing games against real human opponents is still indispensable for proper evaluation. Unfortunately, this entails other sources of inaccuracy.

A poker program can participate in a real game with willing participants, using a laptop computer on the table. This turns out to be surprisingly difficult, due to the fast pace of a real game and the amount of information to be entered. Even with

[21] Many of these issues were resolved after the publication of this paper, when DIVAT replaced EVAT, as discussed in detail in Chapter 5.

numerous single-character accelerators, text entry is a bottleneck to the process. A well-designed graphical interface might help considerably, and an automatic card-reader (*e.g.*, a bar-code scanner) could prevent the operator from giving away useful information, since only the program would know its hand. However, it may always be more practical to have human players participate in a virtual game, rather than having programs compete in the physical world.

For more than three years, our programs have regularly participated in online poker games against human opposition on the Internet Relay Chat (IRC). Players connect to the IRC poker server and participate in numerous games that are conducted by dedicated software. No real money is at stake, but the accumulated bankroll for each player is preserved between sessions, and a variety of statistics are maintained. There is a hierarchy of games for Limit Hold'em, and a player must win a specified amount in the introductory level games to qualify for the higher tiered games.

These lowest level games (open to everyone) vary from wild to fairly normal, offering a wide variety of game conditions to test the program. The second and third tier games resemble typical games in a casino or card room. Most of these players take the game seriously, and some are very strong (including some professionals). Since POKI has been a consistent winner in these higher tiered games (and is in the top 10% of all players on the server), we believe the program plays better than the average player in a low-limit casino game.

Recently, several online poker servers have begun offering real-money games played over the Internet. The response has been very favorable, and it is normal to have more than 1,000 players logged into a virtual card room at any given time. With the agreement of the entrepreneurs, this might provide a future venue for testing programs in a completely realistic setting.

Another form of online poker is a free Java web applet, where users can play at a table with poker programs and other people. POKI currently hosts such a facility, which provides an interesting hybrid between internal experiments and games against humans.[22]

[22] See `http://www.cs.ualberta.ca/~games`.

While online poker is useful for measuring the progress of a program, it is not a controlled environment. The game is constantly changing, and positive results over a given time-frame can easily be due to playing against a weaker set of opponents, rather than actual improvements to the algorithm. Considering that it may take thousands of games to measure small improvements, it is difficult to obtain precise quantified results. There is also no guarantee that an objectively stronger program will be more successful in this particular style of game. Certain plays that might be good against master players could be inappropriate for the more common opponents in these games. Moreover, regular players may have acquired a lot of experience against previous versions of POKI, making it difficult to achieve the same level of performance.

As a result, it is still beneficial to have a master poker player review hundreds of games played by the program, looking for errors or dubious decisions. Needless to say, this is a slow and laborious method of assessment. A human master can also play against one or more versions of the program, probing for weaknesses or unbalanced strategy. Based on these direct encounters, we believe POKI is an intermediate level player, but has not yet reached the master level.

2.7.2 Experimental Results

The unit of measurement for program performance is the average number of small bets won per game (sb/g). For example, in a game of $10/$20 Hold'em with 40 games per hour, an income rate of +0.05 sb/g translates into $20 per hour. Human players sometimes use this metric in preference to dollars per hour, since it is not dependent on the speed of play, which can vary from 20 to 60 games per hour.

Since no variance reduction methods are available for online games, we generally test new algorithms for a minimum of 20,000 games before interpreting the results. On this scale, the trends are usually clear and stable amid the noise. Unfortunately, it can take several weeks to accumulate this data, depending on the popularity of the online game in question.

Any embellishment resulting in an improvement of +0.05 sb/g in internal experiments against previous versions is considered to be significant. However, this does

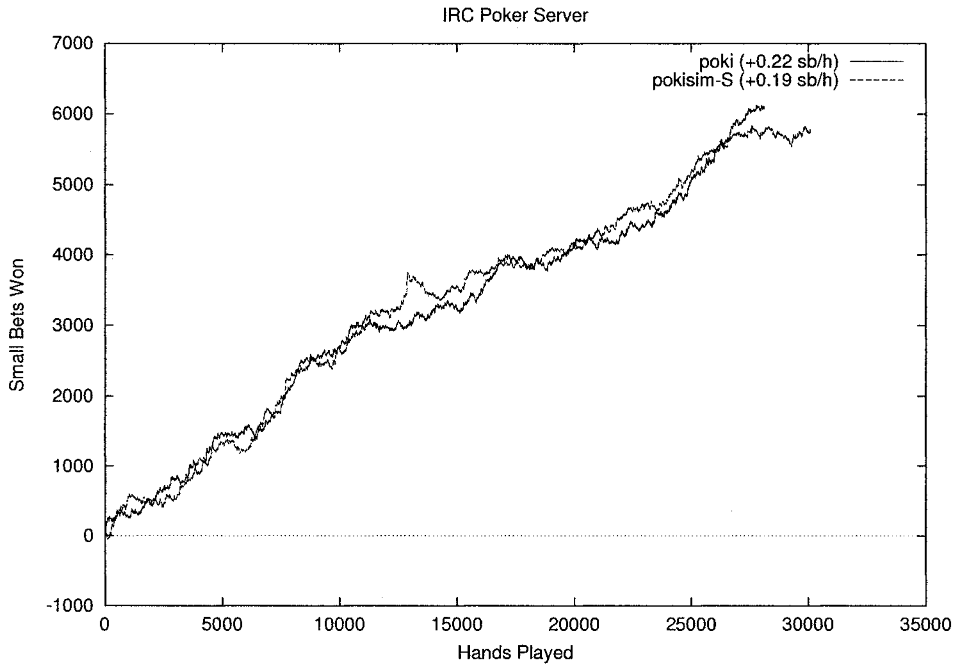


Figure 2.8: POKI's performance on the IRC poker server (introductory level).

not always translate into comparable gains in actual games, as many factors affect the ultimate win rate. Nevertheless, the program has made steady progress over the course of the project. In recent play on the IRC poker server, POKI has consistently performed between +0.10 and +0.20 sb/g in the lowest level games, and between +0.07 and +0.10 sb/g in the higher tiered games against stronger opposition.

The results of simulation-based betting strategies have so far been inconsistent. Despite some programming errors that were discovered later, the earliest (1998) versions of simulation-based LOKI outperformed the regular formula-based version in both internal experiments (+0.10 $\pm$ 0.04 sb/g), and in the introductory level games of IRC (+0.13 sb/g *vs* +0.08 sb/g). However, it lost slowly in the more advanced IRC games, whereas the regular version would at least break even.

The more recent versions are substantially stronger, but a similar pattern is apparent. Figure 2.8 shows that both the regular betting strategy (labeled "poki") and the simulation-based betting strategy (labeled "pokisim-S") win at about +0.20 sb/g in the introductory level games on the IRC poker server. It is quite likely that differences in playing strength cannot be demonstrated against this particular level of opposition, since both may be close to their maximum income rate for this game. In other words, there are diminishing returns after achieving a very high win rate, and further improvement becomes increasingly difficult. However, there is a clear difference in the more advanced games, where the regular betting strategy routinely wins at about +0.09 sb/g, but the simulation-based version could only break even (peaking at +0.01 sb/g after 5,000 games, but returning to zero after 10,000 games.

When the simulation-based versions were introduced, some of the credit for their success was probably due to the solid reputation that the more conservative versions of POKI had previously established. Many opponents required several hundred games to adjust to the more aggressive style resulting from the simulations. However, the stronger opposition was able to adapt much more quickly, and learned to exploit certain weaknesses that had not been detrimental against weaker players.

Figure 2.9 shows some recent results using the online web applet. This game consists of several computer players (some of which are intentionally weaker than the most recent versions of POKI), and at least one human opponent at all times. Since the artificial players are quite conservative, this game is quite a bit ***tighter*** than most IRC games, and the win rate for the regular formula-based betting strategy is +0.13 sb/g. The simulation-based betting strategy performs at +0.08 sb/g, indicating that this particular set of opponents are much less vulnerable to its strategy differences than the players in the introductory IRC games.

A new simulation-based player (labeled "pokisim-A") maintains three different methods for opponent modeling (statistical frequencies, the rule-based method used by POKI, and a real-time neural network predictor), and uses whichever one has been most successful for each opponent in the past. Not surprisingly, it outperforms the single approach, earning +0.12 sb/g, for a 50% overall improvement in this particular game. This is roughly the same degree of success as the formula-based strategy ("poki"), despite the fact that the original system has benefited from much more tuning, and that the underlying evaluation function was not designed for this fundamentally different approach.

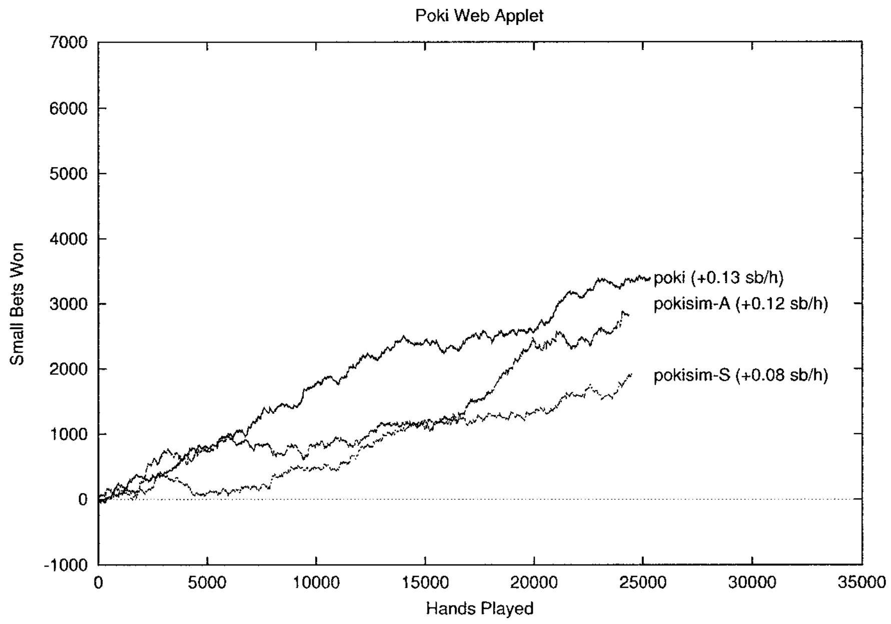


Figure 2.9: POKI's performance on the web applet.

We note that the variance is quite a bit higher in this experiment, which is the more common situation.[23] The results could be quite misleading if interpreted after only 5,000, or even after 15,000 games. The two bottom lines cross over at 15,000 games, but "pokisim-S" is lower before and after that point.

There have been hundreds of internal experiments over the last few years, testing individual enhancements, and the effects of different game conditions. We refer the reader to our previous publications for further details [7, 6, 26, 8, 5, 27, 29, 9, 12].

2.8 A Framework for Stochastic Game-Playing Programs

Using simulations for stochastic games is not new. Consider the following three games:

1. In Scrabble, the opponent's tiles are unknown, so the outcome of future turns must be determined probabilistically. A simulation consists of repeatedly

[23] The relatively low variance in the previous figure may again be a result of both programs being close to maximal gains against that particular level of opposition.

generating a plausible set of tiles for the opponent. Each trial might involve a two ply or four ply search of the game tree, to determine which initial move leads to the maximum gain in points for the program. A simulation-based approach has been used for a long time in Scrabble programs. Brian Sheppard, the author of the Scrabble program MAVEN, coined the term "simulator" for this type of game-playing program structure [33, 32].

2. In backgammon, simulation is used for "roll-outs" of the remainder of a game, and are now generally regarded to be the best available estimates for the equity of a given position. A simulation consists of generating a series of dice rolls, and playing through to the end of the game with a strong program choosing moves for both sides. Gerry Tesauro has shown that relatively simple roll-outs can achieve a level of play comparable to the original neural network evaluation function of TD_GAMMON [39, 40].

3. In bridge, the cards of other players are hidden information. A simulation consists of assigning cards to the opponents in a manner that is consistent with the bidding. The game is then played out and the result determined. Repeated deals are played out to decide which play produces the highest probability of success. Matt Ginsberg has used this technique in GIB to achieve a world-class level for play of the hand [17].

In the above examples, the programs are not using traditional Monte Carlo simulation to generate the unknown information. They use selective sampling, biased to take advantage of all the available information. In each case, and in poker, we are using information about the game state to skew the underlying probability distribution, rather than assuming a uniform or other fixed probability distribution. Monte Carlo techniques might eventually converge on the right answer, but selective sampling allows reduced variance and faster convergence.

In the Scrabble example, MAVEN does not assign tiles for the opponent by choosing from the remaining unknown tiles uniformly at random. It biases its choice to give the opponent a "nice" hand, because strong players usually make plays that leave them with good tiles for future turns (such as letters that may score the 50 point bonus for using all tiles). It also samples without replacement, to ensure that every remaining tile is selected equally often, thereby reducing the natural variance [33]. In backgammon, future dice rolls are generated randomly, but the choice of moves is made by an external player agent. In bridge, the assignment of cards to an opponent is subject to the information obtained from the bidding. If one opponent has indicated high point strength, then the assignment of cards to that opponent reflects this information [17].

The alpha-beta framework has proven to be an effective tool for the design of two-player, zero-sum, deterministic games with perfect information. It has been around for more than 30 years, and in that time the basic structure has not changed much (although there have been numerous algorithmic enhancements to improve the search efficiency). The search technique usually has the following properties:

1. The search is full breadth, but limited depth. That is, all move alternatives are considered, except those that can be logically eliminated (such as alpha-beta cutoffs).

2. Heuristic evaluation occurs at the leaf nodes of the search tree, which are interior nodes of the game tree.

3. The search gets progressively deeper (iterative deepening), until real-time constraints dictate that a choice be made.

The alpha-beta algorithm typically uses integer values for positions and is designed to identify a single "best" move, not differentiating between other moves. The selection of the best move may be brittle, in that a single node mis-evaluation can propagate to the root of the search and alter the move choice. As the search progresses, the bounds on the value of each move are narrowed, and the certainty of the best move choice increases. The deeper the search, the greater the confidence in the selected move, and after a certain point there are diminishing returns for further search.

In an imperfect information game of respectable size, it is impossible to examine the entire game tree of possibilities [21]. This is especially true for poker because of

```
SimulationFramework()
{
   obvious_move = NO
   trials = 0
   while( ( trials <= MAX_TRIALS ) and ( obvious_move == NO ) )
   {
      trials = trials + 1
      position = current_state_of_the_game +
          ( selective_sampling to generate_missing_information )
      for( each legal move m )
      {
         value[m] += PlayOut( position.m, info )
      }
      if( exists i such that value[i] >> value[j] ( forall j ≠ i ) )
      {
         obvious_move = YES
      }
   }
   select move based on value[]
}
```

Figure 2.10: Framework for two-player, zero-sum, imperfect information games.

the many opponents, each making independent decisions. The pseudo-code for the proposed method of selective sampling is shown in Figure 2.10 [5]. This approach has the following properties:

1. The search is full depth, but limited breadth. That is, each line is played out to the end of the game (in poker, to the showdown or until one player wins uncontested).

2. Heuristic evaluation occurs at the interior nodes of the search tree to decide on future moves by the players. Outcomes are determined at the leaf nodes of the game tree, and are 100% accurate.

3. The search gets progressively wider, performing trials consistent with the probability distribution of hidden information, until real-time constraints dictate that a choice be made.

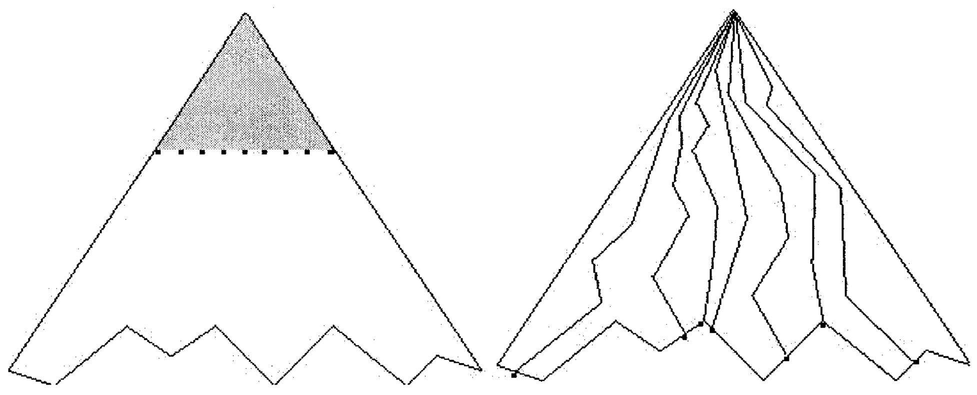


Figure 2.11: Comparing two search frameworks.

The expected values of all move alternatives are computed, and the resulting choice may be a randomized mixed strategy. As the search progresses, the values for each move become more precise, and the certainty of the highest expected value choice increases.[24] The more trials performed, the greater the confidence in the selected move, and after a certain point there are diminishing returns for performing additional trials.

Although the move sequences examined during an alpha-beta search are systematic and non-random, it can be viewed as a sampling of related positions, used as evidence to support the choice of best move. In the case of selective sampling, the evidence is statistical, and the confidence can be measured precisely. The two contrasting methods are depicted in Figure 2.11, with alpha-beta search on the left and simulation-based search on the right.

As noted previously, it is not essential to continue each trial to the end of the game. In stochastic games, the expected value of internal game tree nodes can also be heuristically estimated with a score (as in Scrabble), an evaluation function (as in backgammon), or other methods (such as the roll-out simulations described in Section 2.5.1).

An important feature of the simulation-based framework is the notion of an obvious move. Although some alpha-beta programs try to incorporate an obvious

[24] The "best" move is highly subjective. Here we do not consider deceptive plays that misrepresent the strength of the hand.

move feature, the technique is usually *ad hoc* and based on programmer experience, rather than a sound analytic technique (an exception is the B* proof procedure [1]). In the simulation-based framework, an obvious move is well-defined. If one choice exceeds the alternatives by a statistically significant margin, we can stop the simulation early and take that action, with precise knowledge of the mathematical validity of the decision. Like alpha-beta pruning, this early cut-off may prove to be an effective means for reducing the required amount of search effort, especially if it is applied at all levels of the imperfect information game tree.

The proposed framework is not a complete ready-made solution for stochastic games, any more than alpha-beta search is the only thing required for high-performance in a particular deterministic game. As discussed in Section 2.5.3, there are many trade-offs to be considered and explored. One must find a good balance between the amount of effort spent on each trial, and the total number of trials completed in the allotted time. There are many different ways to create an evaluation function, and as with other strong game programs, speed and consistency may be more important than explicit knowledge and complexity.

2.9 Conclusions and Future Work

Poker is a complex game, with many different aspects, from mathematics and hidden information to human psychology and motivation. To master the game, a player must handle all of them at least adequately, and excel in most. Strong play also requires a player to be adaptive and unpredictable – any form of fixed recipe can and will be exploited by a good opponent. Good players must dynamically alter their style, based on the current game conditions and on historical knowledge (including past sessions). In contrast, traditional games like chess are somewhat homogeneous in nature, where one can focus very deeply on one particular type of strategy.

Like other computer game-playing research, poker has a well-defined goal, and the relative degree of success is measurable – whether the program plays the game well, or does not. We have resisted the temptation of focusing only on the clearly tractable problems, in favor of grounding the research on those topics that actually affect the bottom line the most. As a result, developing POKI has been a cyclic process. We improve one ability of the program until it becomes apparent that another property is the performance bottleneck. Some of the components in the current system are extremely simplistic (such as a constant where a formula or an adaptive method would be better), but do not yet appear to limit overall performance. Others have received much more attention, but are still woefully inadequate.

Human poker players are very good at understanding their opponent, often forming an accurate model based on a single data point (and occasionally before the first hand is dealt!). Programs may never be able to match the best players in this area, but they must at least try to reduce the gap, since they can clearly be superior in other aspects of the game. Although POKI has successfully used opponent modeling to improve its level of play, it is abundantly clear that these are only the first steps, and there are numerous opportunities for improvement.

For example, the current system becomes slower to adjust as more information is collected on a particular opponent. This "build-up of inertia" after thousands of data points have been observed can be detrimental if the player happens to be in an uncommon mood that day. Moreover, past success may have largely been due to opponents staying with a fixed style that does not vary over time (most computer opponents certainly have this property). It is much more difficult to track good players who constantly "change gears" for a relatively brief time. Although recent actions are mixed with the long-term record, a superior historical decay function could allow the system to keep up with current events better.

It is easy to gather lots of data on each opponent, but it is difficult to discern the most useful features. It is possible that simpler metrics may be better predictors of an opponent's future behavior. There are also several techniques in the literature for learning in noisy domains where one must make inferences based on limited data, which have not yet been explored.

For the simulations, the major problem is the high variance in the results. Even with noise-reduction techniques, the standard deviation can still be high. Faster machines and parallel computations might help to base decisions on a larger sample size. This eventually has diminishing returns, and our empirical results suggest that the benefits may be small beyond a necessary minimum number of data points (roughly 500). Once the critical minimum can be attained in real-time, the more important issue is whether the trials are fair and representative of the situation being modeled.

For the game of bridge, simulations have successfully allowed computer programs to play hands at a world-class level [17]. Nevertheless, limitations in the simulation-based approach and the high variance have prompted Matt Ginsberg, the author of GIB, to look at other solutions, including building the entire search tree [18]. We, too, may have to look for new approaches to overcome the limitations of simulations.

The poker project is rich in research opportunities, and there is no shortage of new ideas to investigate. Having explored some fairly straightforward techniques to accomplish a reasonable level of play, we are now contemplating re-formulations that might produce a breakthrough to a world-class level of play. Toward this end, some of our current research has moved toward empirical techniques for deriving game-theoretic equilibrium solutions for betting strategies. We have also given more attention to two-player Hold'em, in which many of the flaws of the current system are emphasized.

However, it is not clear if a single unifying framework is possible for poker programs. Certain abilities, such as the accurate estimation of expected values in real time, will eventually be well solved. However other aspects, like opponent modeling, are impossible to solve perfectly, since even the opponents may not understand what drives their actions!

Acknowledgments

This research was supported by the Natural Sciences and Engineering Council of Canada and the Alberta Informatics Circle of Research Excellence.

We gratefully acknowledge the many comments and suggestions made by the anonymous referees.

Bibliography

[1] H. Berliner. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.

[2] H. Berliner, G. Goetsch, and M. Campbell. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43:7–20, 1990.

[3] D. Billings. The First International RoShamBo Programming Competition. *The International Computer Games Association Journal*, 23(1):42–50, 2000.

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

[5] D. Billings, D. Papp, L. Peña, J. Schaeffer, and D. Szafron. Using selective-sampling simulations in poker. In *American Association of Artificial Intelligence Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, pages 13–18. American Association of Artificial Intelligence, 1999.

[6] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 493–499, 1998.

[7] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Poker as a testbed for machine intelligence research. In R. Mercer and E. Neufeld, editors, *Advances in Artificial Intelligence, AI'98*, pages 228–238. Springer-Verlag, 1998.

[8] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *American Association of Artificial Intelligence National Conference, AAAI'99*, pages 697–703, 1999.

[9] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Learning to play strong poker. In J. Fürnkranz and M. Kubat, editors, *Machines That Learn To Play Games*, volume 8 of *Advances in Computation: Theory and Practice*, chapter 11, pages 225–242. Nova Science Publishers, 2001.

[10] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *American Association of Artificial Intelligence National Conference, AAAI'96*, pages 120–125, 1996.

[11] C. Cheng. Recognizing poker hands with genetic programming and restricted iteration. In J. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford*, 1997.

[12] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI'00*, pages 1467–1473, 2000.

[13] D. Egnor. Iocaine powder. *International Computer Games Association Journal*, 23(1):33–35, 2000.

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

[15] R. Fung and K. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence.* Morgan Kaufmann, 1989.

[16] M. Ginsberg. Partition search. In *American Association of Artificial Intelligence National Conference, AAAI'96*, pages 228–233, 1996.

[17] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *The International Joint Conference on Artificial Intelligence, IJCAI'99*, pages 584–589, 1999.

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

[19] H. Iida, J. Uiterwijk, H. J. van den Herik, and I. Herschberg. Thoughts on the application of opponent-model search. In *Advances in Computer Chess 7*, pages 61–78. University of Maastricht, 1995.

[20] P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search.* PhD thesis, School of Computer Science, Carnegie-Mellon University, 1992.

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

[22] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Annual Conference on Uncertainty in Artificial Intelligence, UAI'99*, pages 343–350, 1999.

[23] 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.

[24] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

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

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

[27] L. Peña. Probabilities and simulations in poker. Master's thesis, Department of Computing Science, University of Alberta, 1999.

[28] M. Sakaguchi and S. Sakai. Solutions of some three-person stud and draw poker. *Mathematica Japonica*, 6(37):1147–1160, 1992.

[29] J. Schaeffer, D. Billings, L. Peña, and D. Szafron. Learning to play strong poker. In *The International Conference on Machine Learning Workshop on Game Playing*. J. Stefan Institute, 1999. Invited paper.

[30] A. Selby. Optimal heads-up pre-flop hold'em. WWW, 1999. `http://www.archduke.demon.co.uk/simplex/`.

[31] R. Shacter and M. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1989.

[32] B. Sheppard. *Toward Perfection of Scrabble Play.* PhD thesis, Computer Science, University of Maastricht, 2002.

[33] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.

[34] J. Shi and M. Littman. Abstraction methods for game theoretic poker. In T. A. Marsland and I. Frank, editors, *Computers and Games 2000*, LNCS 2063, pages 333–345. Springer-Verlag, 2002.

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

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

[37] S. Smith. Flexible learning of problem solving heuristics through adaptive search. In *The International Joint Conference on Artificial Intelligence, IJCAI'83*, pages 422–425, 1983.

[38] K. Takusagawa. Nash equilibrium of Texas Hold'em poker, 2000. Undergraduate thesis, Computer Science, Stanford University.

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

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

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

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

[43] N. Zadeh. *Winning Poker Systems.* Prentice Hall, 1974.

[44] N. Zadeh. Computation of optimal poker strategies. *Operations Research*, 25(4):541–562, 1977.

Chapter 3

Game-Theoretic Methods (2002-2003)

Approximating Game-Theoretic Optimal Strategies for Full-scale Poker [1]

3.1 Introduction

Mathematical game theory was introduced by John von Neumann in the 1940s, and has since become one of the foundations of modern economics [14]. Von Neumann used the game of poker as a basic model for 2-player zero-sum adversarial games, and proved the first fundamental result, the famous *Minimax Theorem.* A few years later, John Nash added results for N-player non-cooperative games, for which he later won the Bank of Sweden Prize in Economic Sciences in Memory of Alfred Nobel [8]. Many decision problems can be modeled using game theory, and it has been employed in a wide variety of domains in recent years.

Of particular interest is the existence of *optimal solutions*, or *Nash equilibria.*[2]

[1] The contents of this chapter originally appeared in the proceedings of *IJCAI'03*. Copyright 2003 International Joint Conferences on Artificial Intelligence, Inc. All rights reserved. D. Billings, N. Burch, A. Davidson, T. Schauenberg, R. Holte, J. Schaeffer, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *The Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 661–668, 2003.

[2] The term "optimal" is over-loaded in computer science, and is highly misleading (overly flattering) in this particular context. The more neutral terms "equilibrium strategy" or "Nash equilibrium" are now preferred. An equilibrium strategy is optimal only in the sense of not being exploitable by a perfect opponent; but since it fails to exploit imperfect opponents, it can perform much worse than a *maximal strategy* in practice. The term "equilibrium" is used in several places where "optimal" appeared in the original publication. However, the term "pseudo-optimal" has been retained.

An equilibrium solution provides a *randomized mixed strategy*, which is basically a recipe of how to play in each possible situation. Using this strategy ensures that an agent will obtain at least the game-theoretic value of the game, regardless of the opponent's strategy. Unfortunately, finding exact equilibrium solutions is limited to relatively small problem sizes, and is not practical for most real domains.

This paper explores the use of highly abstracted mathematical models which capture the most essential properties of the real domain, such that an exact solution to the smaller problem provides a useful approximation of an equilibrium strategy for the real domain. The application domain used is Limit Texas Hold'em.

Due to the computational limitations involved, only simplified poker variations have been solved in the past (*e.g.*, [7, 9]). While these are of theoretical interest, the same methods are not feasible for real games, which are too large by many orders of magnitude [6].

Selby [10] computed an equilibrium solution for the abbreviated game of *Pre-flop Hold'em*. Shi and Littman [11] investigated abstraction techniques to reduce the large search space and complexity of the problem, using a simplified variant of poker. Takusagawa [13] created approximate equilibrium strategies for the play of three specific Hold'em flops and betting sequences.

Using new abstraction techniques, we have produced viable "pseudo-optimal" strategies for the game of 2-player Texas Hold'em. The resulting poker-playing programs have demonstrated a tremendous improvement in performance. Whereas the previous best poker programs were easily beaten by any competent human player, the new programs are capable of defeating very strong players, and can hold their own against world-class opposition.

Although some domain-specific knowledge is an asset when creating accurate reduced-scale models, analogous methods can be developed for many other imperfect information domains and generalized game trees. We describe a general method of problem reformulation that permits the independent solution of sub-trees by estimating the conditional probabilities needed as input for each computation.

This paper makes the following contributions:

1. Abstraction techniques that can reduce a $\Theta(10^{18})$ poker search space to a

manageable $\Theta(10^7)$, without losing the most important properties of the game.

2. A poker-playing program that is a major improvement over previous efforts, and is capable of competing with world-class opposition.

3.2 Game Theory

Game theory encompasses all forms of competition between two or more agents. Unlike chess or checkers, poker is a game of *imperfect information* and *chance outcomes*. It can be represented with an *imperfect information game tree* having *chance nodes* and *decision nodes*, which are grouped into *information sets*.

Since the nodes in this game tree are not independent, divide-and-conquer methods for computing sub-trees (such as the *alpha-beta search algorithm*) are not applicable. More detailed descriptions of imperfect information game tree structure are available elsewhere (*e.g.*, [4]).

A *strategy* is a set of rules for choosing an action at every decision node of the game tree. In general, this will be a *randomized mixed strategy*, which is a probability distribution over the various alternatives. A player must use the same policy across all nodes in the same information set, since from that player's perspective they are indistinguishable from each other (differing only in the hidden information component).

The conventional method for solving such a problem is to convert the descriptive representation, or *extensive form*, into a system of linear equations, which is then solved by a linear programming (LP) system such as the *Simplex algorithm*. The equilibrium solutions are computed simultaneously for all players, ensuring the best worst-case outcome for each player.

Traditionally, the conversion to *normal form* was accompanied by an exponential blow-up in the size of the problem, meaning that only very small problem instances could be solved in practice. Koller [5] described an alternate LP representation, called *sequence form*, which exploits the property of *perfect recall* (wherein all players know the preceding history of the game), to obtain a system of equations and unknowns that is only linear in the size of the game tree. This exponential reduction in representation has re-opened the possibility of using game-theoretic analysis for many domains. However, since the game tree itself can be very large, the LP solution method is still limited to moderate problem sizes (normally less than a billion nodes).

3.3 Texas Hold'em

A ***game*** (or ***hand***) of Texas Hold'em consists of four stages, each followed by a round of betting:

1. ***Pre-flop:*** Each player is dealt two private cards face down (the ***hole cards***).
2. ***Flop:*** Three ***community cards*** (shared by all players) are dealt face up.
3. ***Turn:*** A single community card is dealt face up.
4. ***River:*** A final community card is dealt face up.

After the betting, all active players reveal their hole cards for the ***showdown***. The player with the best five-card poker hand formed from their two private cards and the five public cards wins all the money wagered (ties are possible).

The game starts off with two forced bets (the ***blinds***) put into the ***pot***. When it is a player's turn to act, they must either ***bet/raise*** (increase their investment in the pot), ***check/call*** (match what the opponent has bet or raised), or ***fold*** (quit and surrender all money contributed to the pot).

The best-known non-commercial Texas Hold'em program is POKI. It has been playing online since 1997 and has earned an impressive winning record, albeit against generally weak opposition [3]. The system's abilities are based on enumeration and simulation techniques, expert knowledge, and opponent modeling. The program's weaknesses are easily exploited by strong players, especially in the 2-player game.

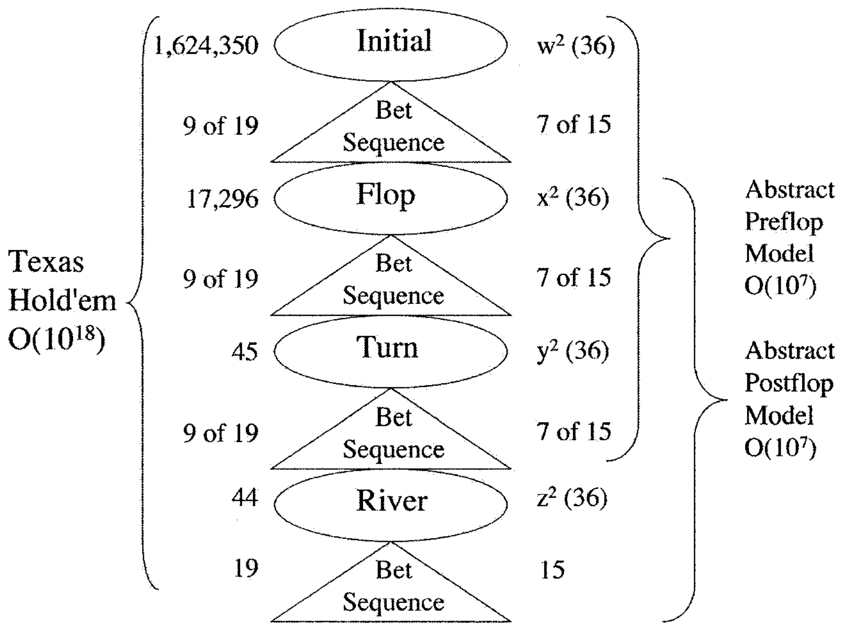


Figure 3.1: Branching factors for Hold'em and abstractions.

3.4 Abstractions

Texas Hold'em has an easily identifiable structure, alternating between chance nodes and betting rounds in four distinct stages. A high-level view of the imperfect information game tree is shown in Figure 3.1.

Hold'em can be reformulated to produce similar but much smaller games. The objective is to reduce the scale of the problem without severely altering the fundamental structure of the game, or the resulting equilibrium strategies. There are many ways of doing this, varying in the overall reduction and in the accuracy of the resulting approximation.

Some of the most accurate abstractions include *suit equivalence isomorphisms* (offering a reduction of at most a factor of $4! = 24$), *rank equivalence* (only under certain conditions), and *rank near-equivalence*. The equilibrium solutions to these abstracted problems will either be exactly the same or will have a small bounded error, which we refer to as *near-optimal* solutions. Unfortunately, the abstractions that produce an exact or near-exact reformulation do not produce the very large reductions required to make full-scale poker tractable.

A common method for controlling the game size is *deck reduction*. Using less than the standard 52-card deck greatly reduces the branching factor at chance nodes. Other methods include reducing the number of cards in a player's hand (*e.g.*, from a 2-card hand to a 1-card hand), and reducing the number of board cards (*e.g.*, a 1-card flop), as was done by Shi and Littman [11] for the game of *Rhode Island Hold'em*.[3] Koller and Pfeffer [6] used such parameters to generate a wide variety of tractable games to solve with their Gala system.

We have used a number of small and intermediate sized games, ranging from eight cards (two suits, four ranks) to 24 cards (three suits, eight ranks) for the purpose of studying abstraction methods, comparing the results with known exact or near-optimal solutions. However, these smaller games are not suitable for use as an approximation for Texas Hold'em, as the underlying structures of the games are different. To produce good playing strategies for full-scale poker, we look for abstractions of the real game which do not alter that basic structure.

The abstraction techniques used in practice are powerful in terms of reducing the problem size, and subsume those previously mentioned. However, since they are also much cruder, we call their solutions *pseudo-optimal*, to emphasize that there is no guarantee that the resulting approximations will be accurate, or even reasonable. Some will be low-risk propositions, while others will require empirical testing to determine if they have merit.

3.4.1 Betting Round Reduction

The standard rules of Limit Hold'em allow for a maximum of four bets per player per round.[4] Thus, in 2-player Limit poker there are 19 possible betting sequences, of which two do not occur in practice.[5] Of the remaining 17 sequences, 8 end in a fold (leading to a terminal node in the game tree), and 9 end in a call (carrying forward

[3] Recently, Gilpin and Sandholm introduced an automated abstraction technique called *gameshrink*, which was used to solve the game of *Rhode Island Hold'em* (see Chapter 6).

[4] Some rules allow unlimited raises when only two players are involved. However, occasions with more than three legitimate raises are rare, and do not greatly alter an equilibrium strategy.

[5] Technically, a player may fold even though there is no outstanding bet. This is logically dominated, and therefore does not occur in an equilibrium strategy, and is seldom seen in practice.

to the next chance node). Using k = check, b = bet, f = fold, c = call, r = raise, and capital letters for the second player, the tree of possible betting sequences for each round is:

```
kK kBf kBc kBrF kBrC kBrRf kBrRc kBrRrF kBrRrC
    bF  bC  bRf  bRc  bRrF  bRrC  bRrRf  bRrRc
```

We call this local collection of decision nodes a *betting tree*, and represent it diagramatically with a triangle (see Chapter 1 Figure 1.2).

With *betting round reduction*, each player is allowed a maximum of three bets per round, thereby eliminating the last two sequences in each line. The effective branching factor of the betting tree is reduced from nine to seven. This does not appear to have a substantial effect on play, or on the ***expected value*** (EV) for each player. This observation has been verified experimentally. In contrast, we computed the corresponding *post-flop models* with a maximum of two bets per player per round, and found radical changes to the equilibrium strategies, strongly suggesting that that level of abstraction is not safe.

3.4.2 Elimination of Betting Rounds

Large reductions in the size of a poker game tree can be obtained by *elimination of betting rounds*. There are several ways to do this, and they generally have a significant impact on the nature of the game. First, the game may be *truncated*, by eliminating the last round or rounds. In Hold'em, ignoring the last board card and the final betting round produces a 3-round model of the actual 4-round game. The solution to the 3-round model loses some of the subtlety involved in the true equilibrium strategy, but the degradation applies primarily to advanced tactics on the turn. There is a smaller effect on the flop strategy, and the strategy for the first betting round may have no significant changes, since it incorporates all the outcomes of two future betting rounds. We use this particular abstraction to define an appropriate strategy for play in the first round, and thus call it a *pre-flop model* (see Figure 3.2).

The effect of truncation can be lessened through the use of *expected value leaf nodes*. Instead of ending the game abruptly and awarding the pot to the strongest hand at that moment, we compute an average conclusion over all possible chance outcomes. For a 3-round model ending on the turn, we *roll-out* all 44 possible river cards, assuming no further betting (or alternately, assuming one bet per player for the last round). Each player is awarded a fraction of the pot, corresponding to the probability of winning the game. In a 2-round pre-flop model, we roll-out all 990 2-card combinations of the turn and river.

The most extreme form of truncation results in a 1-round model, with no foresight of future betting rounds. Since each future round provides a refinement to the approximation, this will not reflect a correct strategy for the real game. In particular, betting plans that extend over more than one round, such as deferring the raise of a very strong hand, are lost entirely. Nevertheless, even these simplistic models can be useful when combined with expected value leaf nodes.

Alex Selby computed an equilibrium solution for the game of *Pre-flop Hold'em*, which consists of only the first betting round followed by an EV roll-out of the five board cards to determine the winner [10]. Although there are some serious limitations in the strategy based on this 1-round model, we have incorporated the Selby pre-flop system into one of our programs, PsOpti1, as described later in this section.

In contrast to truncating rounds, we can *bypass* certain early stages of the game. We frequently use *post-flop models*, which ignore the pre-flop betting round, and use a single fixed flop of three cards (see Figure 3.2).

It is natural to consider the idea of *independent betting rounds*, where each phase of the game is treated in isolation. Unfortunately, the betting history from previous rounds will almost always contain contextual information that is critical to making appropriate decisions. The probability distribution over the hands for each player is strongly dependent on the path that led to that decision point, so it cannot be ignored without risking a considerable loss of information. However, the naive independence assumption can be viable in certain circumstances, and we do implicitly use it in the design of PsOpti1 to bridge the gap between the 1-round pre-flop model and the 3-round post-flop model.

Another possible abstraction we explored was *merging* two or more rounds into a single round, such as creating a combined 2-card turn/river. However, it is not clear what the appropriate bet size should be for this composite round. In any case, the solutions for these models (over a full range of possible bet sizes), all turned out to be substantially different from their 3-round counterparts, and the method was therefore rejected.

3.4.3 Composition of Pre-flop and Post-flop models

Although the nodes of an imperfect information game tree are not independent in general, some decomposition is possible. For example, the sub-trees resulting from different pre-flop betting sequences can no longer have nodes that belong to the same information set.[6] The sub-trees for our post-flop models can be computed in isolation, provided that the appropriate preconditions are given as input. Unfortunately, knowing the correct conditional probabilities would normally entail solving the whole game, so there would be no advantage to the decomposition.

For simple post-flop models, we dispense with the prior probabilities. For the post-flop models used in PsOpti0 and PsOpti1, we simply ignore the implications of the pre-flop betting actions, and assume a uniform distribution over all possible hands for each player. Different post-flop solutions were computed for initial pot sizes of two, four, six, and eight bets (corresponding to pre-flop sequences with zero, one, two, or three raises, but ignoring *which* player initially made each raise). In PsOpti1, the four post-flop solutions are simply appended to the Selby pre-flop strategy (Figure 3.2). Although these simplifying assumptions are technically wrong, the resulting play is still surprisingly effective.

A better way to compose post-flop models is to estimate a set of conditional probabilities, using the solution to a pre-flop model. With a tractable pre-flop model, we have a means of estimating an appropriate strategy at the root, and thereby determine the consequent probability distributions.

In PsOpti2, a 3-round pre-flop model was designed and solved. The resulting

[6] To see this, each decision node of the game tree can be labeled with all the cards known to that player, and the full path that led to that node. Nodes with identical labels differ only in the hidden information, and are therefore in the same information set. Since the betting history is different for these sub-trees, none of the nodes are inter-dependent.

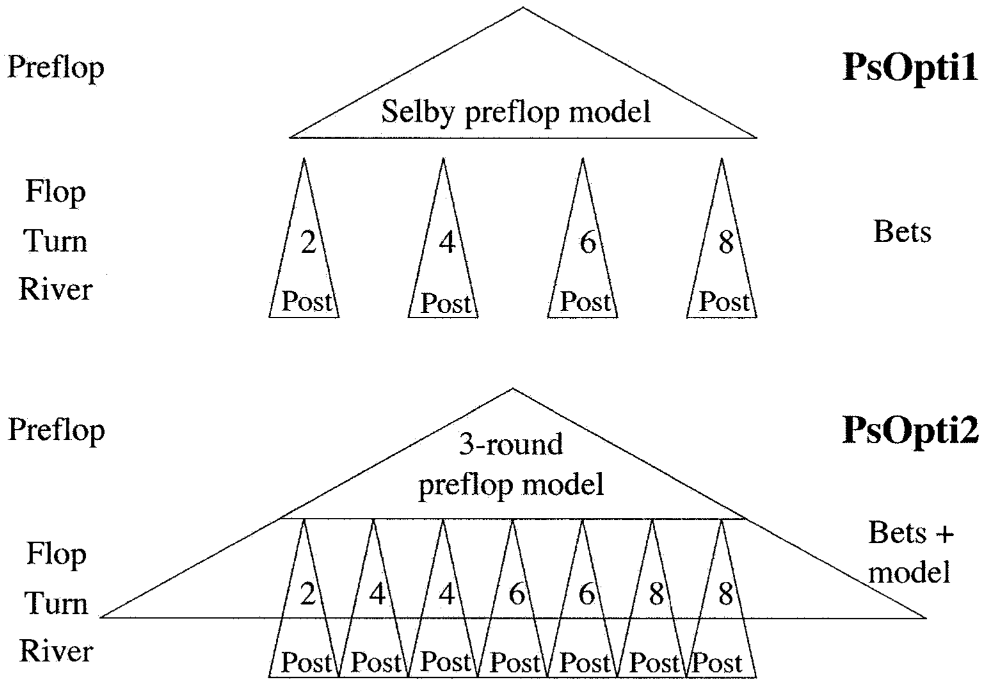


Figure 3.2: Composition of PsOpti1 and PsOpti2.

pseudo-optimal strategy for the pre-flop (which was significantly different from the Selby strategy) was used to determine the corresponding distribution of hands for each player in each context. This provided the necessary input parameters for each of the seven pre-flop betting sequences that carry over to the flop stage. Since each of these post-flop models has been given (an approximation of) the perfect recall knowledge of the full game, they are fully compatible with each other, and are properly integrated under the umbrella of the pre-flop model (Figure 3.2). In theory, this should be equivalent to computing the much larger tree, but it is limited by the accuracy and appropriateness of the proposed pre-flop betting model.[7]

[7] Actually, this is only true if there is a *unique* equilibrium strategy, whereas most interesting games have an *infinite* number of equilibria. This caveat has serious implications on the theoretical and practical value of this approach. Please see Chapter Endnote 3.7.1 for more information on this important limitation.

3.4.4 Abstraction by Bucketing

The most important method of abstraction for the computation of our pseudo-optimal strategies is called *bucketing*. This is an extension of the natural and intuitive concept that has been applied many times in previous research (*e.g.*, [12, 13, 11]). The set of all possible hands is partitioned into equivalence classes (also called *buckets* or *bins*). A *many-to-one mapping function* determines which hands will be grouped together. Ideally, the hands should be grouped according to *strategic similarity*, meaning that they can all be played in a similar manner without much loss in EV.

If every hand was played with a particular *pure strategy* (*i.e.*, only one of the available choices), then a perfect mapping function would group all hands that follow the same plan, and 17 equivalence classes for each player would be sufficient for each betting round. However, since a *mixed strategy* may be indicated for equilibrium play in some cases, we would like to group hands that have a similar probability distribution over action plans.

One obvious but rather crude bucketing function is to group all hands according to strength (*i.e.*, its *rank* with respect to all possible hands, or the probability of currently being in the lead). This can be improved by considering the roll-out of all future cards, giving an (unweighted) estimate of the chance of winning the game.

However, this is only a one-dimensional view of hand types, in what can be considered to be an N-dimensional space of strategies, with a vast number of different ways to classify them. A superior practical method would be to project the set of all hands onto a two-dimensional space consisting of (roll-out) hand strength and hand potential (similar to the hand assessment used in POKI [3]). Clusters in the resulting scattergram suggest reasonable groups of hands to be treated similarly.

We eventually settled on a simple compromise. With n available buckets, we allocate $n - 1$ to *roll-out hand strength*. The number of hand types in each class is not uniform; the classes for the strongest hands are smaller than those for mediocre and weak hands, allowing for better discrimination of the smaller fractions of hands that should be raised or re-raised.

One special bucket is designated for hands that are low in strength but have *high* *potential*, such as good draws to a flush or straight. This plays an important role in identifying good hands to use for bluffing (known as ***semi-bluffs*** [12]). Comparing post-flop solutions that use six strength buckets to solutions with five strength plus one high-potential bucket, we see that most bluffs in the latter are taken from the special bucket, which is sometimes played in the same way as the strongest bucket. This confirmed our expectations that the high-potential bucket would improve the selection of hands for various betting tactics, and increase the overall EV.

The number of buckets that can be used in conjunction with a 3-round model is very small, typically six or seven for each player (*i.e.*, 36 or 49 pairs of bucket assignments). Obviously this results in a very coarse-grained abstract game, but it may not be substantially different from the number of distinctions an average human player might make. Regardless, it is the best we can currently do given the computational constraints of this approach.

The final requirement to sever the abstract game from the underlying real game tree are the *transition probabilities*. The chance node between the flop and turn represents a particular card being dealt from the remaining stock of 45 cards. In the abstract game, there are no cards, only buckets. The effect of the turn card in the abstract game is to dictate the probability of moving from one pair of buckets on the flop to any pair of buckets on the turn. Thus, the collection of chance nodes in the game tree is represented by an $(n \times n)$ to $(n \times n)$ transition network as shown in Figure 3.3. For post-flop models, this can be estimated by walking the entire tree, enumerating all transitions for a small number of characteristic flops. For pre-flop models, the full enumeration is more expensive (encompassing all $\{48 \textit{ choose } 3\} = 17296$ possible flops), so it is estimated either by sampling, or by (parallel) enumeration of a truncated tree.

For a 3-round post-flop model, we can comfortably solve abstract games with up to seven buckets for each player in each round. Changing the distribution of buckets, such as six for the flop, seven for the turn, and eight for the river, does not appear to significantly affect the quality of the solutions, better or worse.

The final linear programming solution produces a large table of mixed strategies (probabilities for fold, call, or raise) for every reachable scenario in the abstract

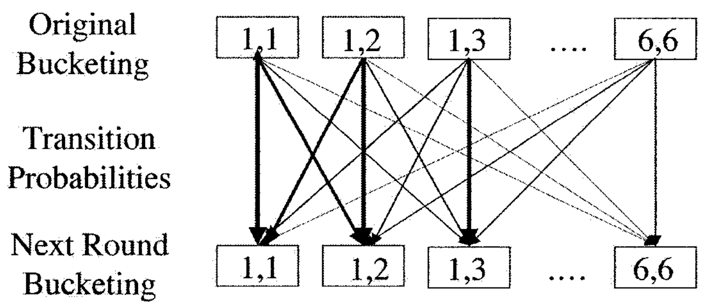


Figure 3.3: Transition probabilities (six buckets per player).

game. To use this, the poker-playing program looks for the corresponding situation based on the same hand strength and potential measures, and randomly selects an action from the mixed strategy.

The large LP computations typically take less than a day (using CPLEX with the barrier method), and use up to two Gigabytes of RAM. Larger problems will exceed available memory, which is common for large LP systems. Certain LP techniques such as *constraint generation* could potentially extend the range of solvable instances considerably, but this would probably only allow the use of one or two additional buckets per player.

3.5 Experiments

3.5.1 Testing Against Computer Players

A series of matches between computer programs was conducted, with the results shown in Table 3.1. Win rates are measured in small bets per game (sb/g). Each match was run for at least 20,000 games (and over 100,000 games in some cases). The variance per game depends greatly on the styles of the two players involved, but is typically +/- 6 sb. The standard deviation for each match outcome is not shown, but is normally less than +/- 0.03 sb/g.

The "bot players" were:

PsOpti2, composed of a hand-crafted pre-flop model, which was solved by linear programming to provide conditional probability distributions to each of seven

Program	1	2	3	4	5	6	7	8
1 PsOpti1		+0.090	+0.091	+0.251	+0.156	+0.047	+0.546	+0.635
2 PsOpti2	-0.090		+0.069	+0.118	+0.054	+0.045	+0.505	+0.319
3 PsOpti0	-0.091	-0.069		+0.163	+0.135	+0.001	+0.418	+0.118
4 Aadapti	-0.251	-0.118	-0.163		+0.178	+0.550	+0.905	+2.615
5 Anti-Poki	-0.156	-0.054	-0.135	-0.178		+0.385	+0.143	+0.541
6 Poki	-0.047	-0.045	-0.001	-0.550	-0.385		+0.537	+2.285
7 Always Call	-0.546	-0.505	-0.418	-0.905	-0.143	-0.537		=0.000
8 Always Raise	-0.635	-0.319	-0.118	-2.615	-0.541	-2.285	=0.000	

Table 3.1: Computer *vs* computer matches (small bets per game).

3-round post-flop models (Figure 3.2). All models in this prototype used six buckets per player per round.

PsOpti1, composed of four 3-round post-flop models under the naive uniform distribution assumption, with seven buckets per player per round. Selby's equilibrium solution for *Pre-flop Hold'em* is used to play the pre-flop [10].

PsOpti0, composed of a single 3-round post-flop model, wrongly assuming uniform distributions and an initial pot size of two bets, with seven buckets per player per round. This program used an always-call policy for the pre-flop betting round.

Poki, the University of Alberta poker program. This older version of Poki was not designed to play the 2-player game, and can be defeated rather easily, but is a useful benchmark.

Anti-Poki, a rule-based program designed to beat Poki by exploiting its weaknesses and vulnerabilities in the 2-player game. Any specific counter-strategy can be even more vulnerable to exploitation by adaptive players.

Aadapti, a relatively simple adaptive player, capable of slowly learning and exploiting persistent patterns in play.

Always_Call, a very weak benchmark strategy.

Always_Raise, a very weak benchmark strategy.

It is important to understand that a game-theoretic equilibrium player is, in principle, *not designed to win.* Its purpose is to *not lose.* An implicit assumption is that the opponent is also playing an equilibrium, and nothing can be gained by observing the opponent for patterns or weaknesses.

In a simple game like RoShamBo (also known as Rock-Paper-Scissors), playing the equilibrium strategy ensures a break-even result, regardless of what the opponent does, and is therefore insufficient to defeat weak opponents, or to win a tournament ([2, 1]). Poker is more complex, and in theory an equilibrium player *can* win, but only if the opponent makes *dominated errors*. Any time a player makes any choice that is part of a randomized mixed strategy of any game-theoretic equilibrium policy, that decision is not dominated. In other words, it is possible to play in a highly sub-optimal manner, but still break even against an equilibrium player, because those choices are not strictly dominated.

Since the pseudo-optimal strategies do no opponent modeling, there is no guarantee that they will be especially effective against very bad or highly predictable players. They must rely only on these fundamental strategic errors, and the margin of victory might be relatively modest as a result.

The critical question is whether such errors are common in practice. There is no definitive answer to this question yet, but preliminary evidence suggests that dominated errors occur often enough to gain a measurable EV advantage over weaker players, but may not be very common in the play of very good players.

The first tests of the pseudo-optimal solutions were done with PsOpti0 playing *Post-flop Hold'em*, where both players agree to simply call in the pre-flop (thereby matching the exact pre-conditions for the post-flop solution). In those preliminary tests, the author played more than 2000 games, and was unable to defeat the pseudo-optimal strategy. In contrast, Poki had been beaten consistently at a rate of over 0.8 sb/g (which is more than would be lost by simply folding every hand).

Using the same no-bet pre-flop policy, PsOpti0 was able to defeat Poki at a rate of +0.144 sb/g (compared to +0.001 sb/g for the full game including pre-flop), and defeated Aadapti at +0.410 sb/g (compared to +0.163 sb/g for the full game).

All of the pseudo-optimal players play substantially better than any previously existing computer programs. Even PsOpti0, which is not designed to play the full game, earns enough from the post-flop betting rounds to offset the EV losses from the pre-flop round (where it never raises good hands, nor folds bad ones).

Player	Games	Posn 1	Posn 2	sb/g
Master-1 early	1147	-0.324	+0.360	+0.017
Master-1 late	2880	-0.054	+0.396	+0.170
Experienced-1	803	+0.175	+0.002	+0.088
Experienced-2	1001	-0.166	-0.168	-0.167
Experienced-3	1378	+0.119	-0.016	+0.052
Experienced-4	1086	+0.042	-0.039	+0.002
Intermediate-1	2448	+0.031	+0.203	+0.117
Novice-1	1277	-0.159	-0.154	-0.156
All Opponents	15125			-0.015

Table 3.2: Human *vs* PsOpti2 matches.

It is suspicious that PsOpti1 outperformed PsOpti2, which in principle should be a better approximation. Subsequent analysis of the play of PsOpti2 revealed some programming errors, and also suggested that the bucket assignments for the pre-flop model were flawed. This may have resulted in an inaccurate pseudo-optimal pre-flop strategy, and consequent imbalances in the prior distributions used as input for the post-flop models. We expect that this will be rectified in future versions, and that the PsOpti2 design will surpass PsOpti1 in performance.[8]

3.5.2 Testing Against Human Players

While these results are encouraging, none of the non-pseudo-optimal computer opponents are better than intermediate strength at 2-player Texas Hold'em. Therefore, matches were conducted against human opponents.

More than 100 participants volunteered to play against the pseudo-optimal players on our public web applet (`www.cs.ualberta.ca/~games/poker/`), including many experienced players, a few masters, and one world-class player. The programs provided some fun opposition, and ended up with a winning record overall. The results are summarized in Table 3.2 and Table 3.3.[9] (Master-1 is author

[8] Although the overlayed architecture did overtake PsOpti1 in later incarnations, the side-effects of having a specific pre-flop model are a serious impediment to top-level play, as discussed in Chapter Endnote 3.7.1.

[9] Note that the overall averages in these two tables should not be compared directly to each other, as they are based on a different mixture of opponents and different circumstances. For example, some players may have played against PsOpti2 only after gaining experience against PsOpti1, thus playing better.

Player	Games	Posn 1	Posn 2	sb/g
thecount	7030	-0.006	+0.103	+0.048
Master-1	2872	+0.141	+0.314	+0.228
Master-2	569	-0.007	+0.035	+0.014
Master-3	425	+0.047	+0.373	+0.209
Experienced-1	4078	-0.058	+0.164	+0.053
Experienced-2	511	+0.152	+0.369	+0.260
Experienced-3	2303	-0.252	+0.128	-0.062
Experienced-5	610	-0.250	-0.229	-0.239
Intermediate-1	16288	-0.145	+0.048	-0.049
Intermediate-2	478	-0.182	+0.402	+0.110
Novice-1	5045	-0.222	-0.010	-0.116
Novice-2	485	-0.255	-0.139	-0.197
Novice-3	1017	-0.369	-0.051	-0.210
Novice-4	583	-0.053	-0.384	-0.219
Novice-5	425	-0.571	-0.296	-0.433
All Opponents	46479			-0.057

Table 3.3: Human *vs* PsOpti1 matches.

Darse Billings, Experienced-1 is Aaron Davidson).

In most cases, the relatively short length of the match leaves a high degree of uncertainty in the outcome, limiting how much can be safely concluded. Nevertheless, some players did appear to have a definite edge, while others were clearly losing.

A number of interesting observations were made over the course of these games. It was obvious that most people had a lot of difficulty learning and adjusting to the computer's style of play. In poker, knowing the basic approach of the opponent is essential, since it will dictate how to properly handle many situations that arise. Some players wrongly attributed intelligence where none was present. After losing a 1000-game match, one experienced player commented "the bot has me figured out now", indicating that its opponent model was accurate, when in fact the pseudo-optimal player is oblivious and does no modeling at all.

It was also evident that these programs do considerably better in practice than might be expected, due to the emotional frailty of their human opponents. Many players commented that playing against the pseudo-optimal opponent was an exasperating experience. The bot routinely makes unconventional plays that confuse and confound humans. Invariably, some of these "bizarre" plays happen to coincide with a lucky escape, and several of these ***bad beats*** in quick succession will often cause strong emotional reactions (sometimes referred to as "going on tilt"). The level of play generally goes down sharply in these circumstances.

This suggests that a perfect game-theoretic equilibrium poker player could perhaps beat even the best humans in the long run, because any EV lost in moments of weakness would never be regained. However, the win rate for such a program could still be quite small, giving it only a slight advantage. Thus, it would be unable to exert its superiority convincingly over the short term, such as the few hundred games of one session, or over the course of a world championship tournament. Since even the best human players are known to have biases and weaknesses, opponent modeling will almost certainly be necessary to produce a program that surpasses all human players.

3.5.3 Testing Against a World-class Player

The elite poker expert was Gautam Rao, who is known as "thecount" or "CountDracula" in the world of popular online poker rooms. Mr. Rao is the #1 all-time winner in the history of the oldest online game, by an enormous margin over all other players, both in total earnings and in dollar-per-game rate. His particular specialty is in ***short-handed games*** with five or fewer players. He is recognized as one of the best players in the world in these games, and is also exceptional at 2-player Hold'em. Like many top-flight players, he has a dynamic ultra-aggressive style.

Mr. Rao agreed to play an exhibition match against PSOPTI1, playing more than 7000 games over the course of several days. The graph in Figure 3.4 shows the progression of the match.

The pseudo-optimal player started with some good fortune, but lost at a rate of about -0.2 sb/g over the next 2000 games. Then, there was a sudden reversal, following a series of fortuitous outcomes for the program. Although "thecount" is renown for his mental toughness, an uncommon run of bad luck can be very frustrating even for the most experienced players. Mr. Rao believes he played below his best level during that stage, which contributed to a dramatic drop where he lost

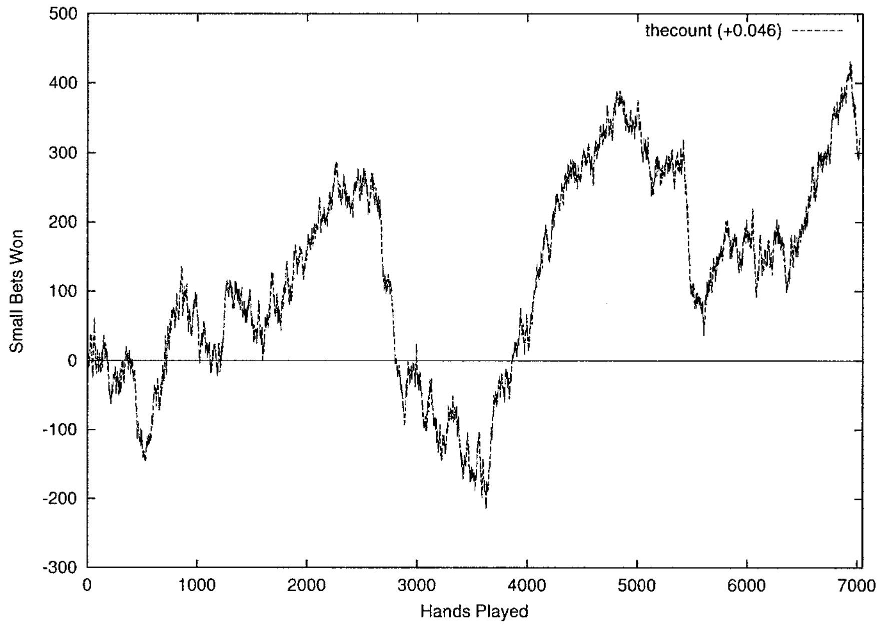


Figure 3.4: Progress of the "thecount" *vs* PsOpti1.

300 sb in less than 400 games. Mr. Rao resumed play the following day, but was unable to recover the losses, slipping further to -200 sb after 3700 games. At this point, he stopped play and did a careful reassessment.

It was clear that his normal style for maximizing income against typical human opponents was not effective against the pseudo-optimal player. Whereas human players would normally succumb to a lot of pressure from aggressive betting, the bot was willing to call all the way to the showdown with as little as a Jack or Queen high card. That kind of play would be folly against most opponents, but is appropriate against an extremely aggressive opponent. Most human players fail to make the necessary adjustment under these atypical conditions, but the program has no sense of fear.

Mr. Rao changed his approach to be less aggressive, with immediate rewards, as shown by the +600 sb increase over the next 1100 games (some of which he credited to a good run of cards). Mr. Rao was able to utilize his knowledge that the computer player did not do any opponent modeling. Knowing this allows a human player to systematically probe for weaknesses, without any fear of being punished for playing in a methodical and highly predictable manner, since an oblivious opponent does not exploit those patterns and biases.

Although he enjoyed much more success in the match from that point forward, there were still some "adventures", such as the sharp decline at 5400 games. Poker is a game of very high variance, especially between two opponents with sharp styles, as can be seen by the dramatic swings over the course of this match. Although 7000 games may seem like a lot, Mr. Rao's victory in this match was still not statistically conclusive.

We now believe that a human poker master can eventually gain a sizable advantage over these pseudo-optimal prototypes (perhaps +0.20 sb/g or more is sustainable). However, it requires a good understanding of the design of the program and its resulting weaknesses. That knowledge is difficult to learn during normal play, due to the good information hiding provided by an appropriate mixture of plans and tactics. This "cloud of confusion" is a natural barrier to opponent learning. It would be even more difficult to learn against an adaptive program with good opponent modeling, since any methodical testing by the human would be easily exploited. This is in stark contrast to typical human opponents, who can often be accurately modeled after only a small number of games.

3.6 Conclusions and Future Work

The pseudo-optimal players presented in this paper are the first complete approximations of a game-theoretic equilibrium strategy for a full-scale variation of real poker.

Several abstraction techniques were explored, resulting in the reasonably accurate representation of a large imperfect information game tree having $\Theta(10^{18})$ nodes with a small collection of models of size $\Theta(10^{7})$. Despite these massive reductions and simplifications, the resulting programs play respectably. For the first time ever, computer programs are not completely outclassed by strong human opposition in the game of 2-player Texas Hold'em.

Useful abstractions included betting tree reductions, truncation of betting rounds combined with EV leaf nodes, and bypassing betting rounds. A 3-round model anchored at the root provided a pseudo-optimal strategy for the pre-flop round, which in turn provided the proper contextual information needed to determine conditional probabilities for post-flop models. The most powerful abstractions for reducing the problem size were based on bucketing, a method for partitioning all possible holdings according to strategic similarity. Although these methods exploit the particular structure of the Texas Hold'em game tree, the principles are general enough to be applied to a wide variety of imperfect information domains.

Many refinements and improvements will be made to the basic techniques in the coming months. Further testing will also continue, since accurate assessment in a high variance domain is always difficult.

The next stage of the research will be to apply these techniques to obtain approximations of Nash equilibria for N-player Texas Hold'em. This promises to be a challenging extension, since multi-player games have many properties that do not exist in the 2-player game.

Finally, having reasonable approximations of equilibrium strategies does not lessen the importance of good opponent modeling. Learning against an adaptive adversary in a stochastic game is a challenging problem, and there will be many ideas to explore in combining the two different forms of information. That will likely be the key difference between a program that can *compete* with the best, and a program that *surpasses* all human players.

Quoting "thecount":

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

Acknowledgments

The authors would like to thank Gautam Rao, Sridhar Mutyala, and the other poker players for donating their valuable time. We also wish to thank Daphne Koller, Michael Littman, Matthew Ginsberg, Rich Sutton, David McAllester, Mason Malmuth, and David Sklansky for their valuable insights in past discussions.

This research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Alberta Informatics Circle of Research Excellence (iCORE), and an Izaak Walton Killam Memorial postgraduate scholarship.

3.7 Chapter 3 Endnotes

3.7.1 Overlaying Models

Following the original publication of this paper, a serious flaw was discovered with the proposition of overlaying a 3-round pre-flop model with a 3-round post-flop model, as in the architecture of PsOpti2. Although this technique can produce systems that play reasonably well in practice, it is not well-grounded in theory.

In general there are an *infinite* number of possible equilibrium solutions to a complex game. This is certainly true in Texas Hold'em poker, which gives rise to the wide variety of playing styles that are completely viable.

The equilibrium solution to any one pre-flop model may be inappropriate for use as a model of any specific opponent, or as a generic opponent model. Combining a pre-flop equilibrium strategy with a post-flop equilibrium strategy is *not* guaranteed to produce an equilibrium strategy to the whole game, because the two strategies may not be consistent with each other.[10] In general, there is no reason to assume that two independently derived approximations will be harmonious with each other. Moreover, using *any* particular pre-flop strategy as a rigid model of one specific opponent can lead to seriously incorrect beliefs about the distribution of possible hands they can hold after the flop.

This inconsistency creates a noticeable "schism" or "gap" between the pre-flop and post-flop play in the pseudo-optimal players. For example, the program may re-raise in the pre-flop, building a large pot, but then fold on the flop to a single small bet. This is almost always incorrect, even if the flop was of no help whatsoever. In practice, the play in the pre-flop often provides key clues that make it easier to accurately deduce the computer player's range of hands later in the game.

[10] This observation was formally proven by Neil Burch, using much simpler imperfect information domains to generate counterexamples.

In our experience with both game-theoretic and adaptive poker algorithms, it has repeatedly been demonstrated that assuming one particular model of the opponent's play can be far worse than having no model at all. That is, the naive assumption of a uniform distribution over opponent holdings is often safer and superior to having a plausible but counter-indicated model of the opponent.

3.7.2 Reverse-Mapping Strategies

Once an equilibrium strategy for an abstracted game has been obtained, there are many ways it can be used to play the real game. The most obvious is to simply do the *reverse mapping* of the abstraction to translate the strategy back onto the full game. However, numerous embellishments are worth consideration. One method in particular, called *bucket splitting*, was implemented but was not discussed in the original paper, due to space limitations.

Suppose that for a particular context we have a bucket 4 hand, and the equilibrium strategy gives us a mixed strategy of $\{0.00, 0.60, 0.40\}$ for fold, call, and raise, respectively. The straightforward way to use this information is to spin a spinner (*i.e.*, generate a random number in the range 0.0 - 1.0) and play the corresponding action. However, this treats all hands in the bucket identically. That was a necessary assumption for accomplishing the coarse-granularity abstraction, but when the time comes to choose an action for the real game, it might be advantageous to distinguish between hands within that wide bucket range.

In simple bucket splitting, we distinguish between the top half and the bottom half of the hands in the bucket, and then bias our actions in a way that should produce better results. In the given example, we might call 100% of the time with hands from the bottom half, and raise 80% of the time with hands from the top half. Thus, we maintain the same recommended game-theoretic frequency (ratio) of actions overall, but we improve our *hand selection* for those designated actions. More generally, we can split the class into as many subdivisions as we like, up to the total number of distinct hands in the class. We call this more general process *bucket splintering*. In the extreme case, almost all hands in the class can be assigned a pure strategy, while the mixed strategy is still maintained for the class as a whole.

Technically, aligning the actions with an ordering of hands by strength could violate the equilibrium, and could potentially leak information. However, in practice the differences might be almost impossible to detect. Ironically, we can take advantage of the stochastic noise and partial observability of the game (which are usually impediments to us), to provide more than adequate obfuscation of our slightly biased actions. Moreover, in view of the crudeness of the entire approach, we can only claim that pseudo-optimal playing strategies are "in the spirit" of equilibrium strategies. Going to any length to preserve the integrity of the abstracted mixed strategies would almost certainly be needlessly pedantic.

Bibliography

[1] D. Billings. The First International RoShamBo Programming Competition. *The International Computer Games Association Journal*, 23(1):42–50, 2000.

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

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

[4] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, 1992.

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

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

[7] 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.

[8] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[9] M. Sakaguchi and S. Sakai. Solutions of some three-person stud and draw poker. *Mathematica Japonica*, 6(37):1147–1160, 1992.

[10] A. Selby. Optimal heads-up pre-flop hold'em. WWW, 1999. `http://www.archduke.demon.co.uk/simplex/`.

[11] J. Shi and M. Littman. Abstraction methods for game theoretic poker. In T. A. Marsland and I. Frank, editors, *Computers and Games 2000*, LNCS 2063, pages 333–345. Springer-Verlag, 2002.

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

[13] K. Takusagawa. Nash equilibrium of Texas Hold'em poker, 2000. Undergraduate thesis, Computer Science, Stanford University.

[14] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1944.

Chapter 4

Adaptive Imperfect Information Game-Tree Search (2004)

Game-Tree Search with Adaptation in Stochastic Imperfect Information Games [1]

4.1 Introduction

Modeling the preferences and biases of users is an important topic in recent AI research. For example, this type of information can be used to anticipate user program commands [14], predict web buying and access patterns [8], and automatically generate customized interfaces [23]. It is usually easy to gather a corpus of data on a user (*e.g.*, web page accesses), but mining that data to predict future patterns (*e.g.*, the next web page request) is challenging. Predicting human strategies in a competitive environment is even more challenging.

The game of poker has become a popular domain for exploring challenging AI problems. This has led to the development of programs that are competitive with strong human players. The current best program, PSOPTI (also known as SPARBOT), is based on approximating a game-theoretic *Nash equilibrium solution* [4]. However, there remains a significant obstacle to overcome before programs can

[1] The contents of this chapter originally appeared in the proceedings of *Computers and Games 2004*. Copyright 2004 Springer-Verlag GmbH. All rights reserved. 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 H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04*, volume 3846 of *Lecture Notes in Computer Science*, pages 21–34. Springer-Verlag GmbH, 2004.

play at a world-class level: opponent modeling. As the world-class poker player who played against PsOpti insightfully recognized [4]:

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

This issue has been studied in two-player perfect information games (*e.g.*, [16, 10, 15]), but has not played a significant role in developing strong programs. In poker, however, opponent modeling is a critical facet of strong play. Since a player has imperfect information (does not know the opponent's cards), any information that a player can glean from the opponent's past history of play can be used to improve the quality of future decisions. Skillful opponent modeling is often the differentiating factor among world-class players.

Opponent modeling is a challenging learning problem, and there have been several attempts to apply standard machine learning techniques to poker. Recent efforts include: neural nets [5], reinforcement learning [11], and Bayesian nets [18], which have had only limited success. There are a number of key issues that make this problem difficult:

1. Learning must be rapid (within 100 games, preferably fewer). Matches with human players do not last for many thousands of games, but the information presented over a short term can provide valuable insights into the opponent's strategy.

2. Strong players change their playing style during a session; a fixed style is predictable and exploitable.

3. There is only partial feedback on opponent decisions. When a player ***folds*** (a common scenario), their cards are not revealed. They might have had little choice with a very weak hand; or they might have made a very good play by folding a strong hand that was losing; or they might have made a mistake by folding the best hand. Moreover, understanding the betting decisions they made earlier in that game becomes a matter of speculation.

This paper presents a novel technique to automatically compute an exploitive counter-strategy in stochastic imperfect information domains like poker. The program searches an imperfect information game tree, consulting an opponent model at all opponent decision nodes and all leaf nodes. The most challenging aspects to the computation are determining: 1) the probability that each branch will be taken at an opponent decision node, and 2) the ***expected value*** (EV) of a leaf node. These difficulties are due to the hidden information and partial observability of the domain, and opponent models are used to estimate the unknown probabilities. As more games are played, the opponent modeling information used by the game-tree search generally becomes more accurate, thus improving the quality of the evaluations. Opponents can and will change their style during a playing session, so old data needs to be gradually phased out.

This paper makes the following contributions:

1. *Miximax* and *Miximix*, applications of the Expectimax search algorithm to stochastic imperfect information adversarial game domains.

2. Using opponent modeling to refine expected values in an imperfect information game-tree search.

3. Abstractions for compressing the large set of observable poker situations into a small number of highly correlated classes.

4. The program VEXBOT, which convincingly defeats strong poker programs including PSOPTI, and is competitive with strong human players.

4.2 Texas Hold'em Poker

Texas Hold'em is generally regarded as the most strategically complex poker variant that is widely played in casinos and card clubs. It is the game used in the annual World Series of Poker to determine the world champion.

A good introduction to the rules of the game can be found in [5]. The salient points needed for this paper are that each player has two private cards (hidden from the opponent), some public cards are revealed that are shared by all players (community cards), and each player is asked to make numerous betting decisions: either ***bet/raise*** (increase the stakes), ***check/call*** (match the current wager and stay in the game), or ***fold*** (quit and lose all money invested thus far). A game ends when all but one player folds, or when all betting rounds are finished, in which case each player reveals their private cards and the best poker hand wins (the ***showdown***). The work discussed in this paper concentrates on two-player Limit Texas Hold'em.

Computer Poker Programs

The history of computer poker programs goes back more than 30 years to the initial work by Findler [13]. Some of the mathematical foundations go back to the dawn of game theory [22, 19]. Recently, most of the AI literature has been produced by the Computer Poker Research Group at the University of Alberta. Those programs – LOKI, POKI, and PSOPTI – illustrate an evolution of ideas that has taken place in the quest to build a program capable of world-class play:

Rule-based [6]: Much of the program's knowledge is explicit in the form of expert-defined rules, formulas, and procedures.

Simulations [7, 5]: Betting decisions are determined by simulating the rest of the game. Likely card holdings are dealt to the opponents and the game is played out to completion. A large sample of hands is played, and the betting decision with the highest expected value is chosen.

Game theory [4]: Two-player Texas Hold'em has a search space size of $O(10^{18})$. This was abstracted down to a structurally similar game of size $O(10^7)$. Linear programming was used to find a Nash equilibrium solution to that game (using techniques described in [17]), and the solution was then mapped back onto real poker. The resulting solution – in effect, a strategy lookup table – has the advantage of containing no expert-defined knowledge. The resulting *pseudo-optimal* poker program, PSOPTI, plays reasonably strong poker and is competitive with strong players. However, the technique is only a crude approximation of equilibrium play. Strong players can eventually find the seams in the abstraction and exploit the resulting flaws in strategy. Nevertheless, this represented a large leap forward in the abilities of poker-playing programs.

As has been seen in many other game-playing domains, progressively stronger programs have resulted from better algorithms and *less* explicit knowledge.

4.3 Optimal versus Maximal Play

In the literature on game theory, a Nash equilibrium solution is often referred to as an *optimal strategy*. However, the adjective "optimal" is dangerously misleading when applied to a poker program, because there is an implication that an equilibrium strategy will perform better than any other possible solution. "Optimal" in the game theory sense has a specific technical meaning that is quite different, so the term *equilibrium strategy* is preferred.

A Nash equilibrium strategy is one in which no player has an incentive to deviate from the strategy, because the alternatives *could* lead to a worse result. This simply maximizes the minimum outcome (sometimes referred to as the *minimax solution* for two-player zero-sum games). This is essentially a defensive strategy that implicitly assumes the opponent is perfect in some sense (which is definitely not the case in real poker, where the opponents are highly fallible).

A Nash equilibrium player will not necessarily defeat a non-optimal opponent. For example, in the game of rock-paper-scissors, the equilibrium strategy is to select an action uniformly at random among the three choices. Using that strategy means that no one can defeat you in the long term, but it also means that you will not win, since you have an expected value of zero against *any* other strategy.

Unlike rock-paper-scissors, poker is a game in which some strategies are *dominated*, and could potentially lose to an equilibrium player. Nevertheless, even a relatively weak and simplistic strategy might break even against a Nash equilibrium opponent, or not lose by very much over the long term. There are many concrete examples of this principle, but one of the clearest demonstrations was seen in the game of Oshi-Zumo [9].

In contrast, a *maximal* player can make moves that are non-optimal (in the game-theoretic sense) when it believes that such a move has a higher expected value. The *best response* strategy, which computes the maximizing counter-strategy to a static strategy, is an example of a maximal player.

Consider the case of rock-paper-scissors where a opponent has played "rock" 100 times in a row. A Nash equilibrium program is completely oblivious to the other player's tendencies, and does not attempt to punish predictable play in any way. A maximal player, on the other hand, will attempt to exploit perceived patterns or biases. This always incurs some risk (the opponent *might* have been setting a trap with the intention of deviating on the 101st game). A maximal player would normally accept this small risk, playing "paper" with a belief of positive expectation [2, 1].

Similarly, a poker program can profitably deviate from an equilibrium strategy by observing the opponent's play and biasing its decision-making process to exploit the perceived weaknesses.

If PsOpti was based on a true Nash equilibrium solution, then no human or computer player could expect to defeat it in the long run. However, PsOpti is only an approximation of an equilibrium strategy, and it will not be feasible to compute a true Nash equilibrium solution for Texas Hold'em in the foreseeable future. There is also an important practical limitation to this approach. Since PsOpti uses a fixed strategy, and is oblivious to the opponent's strategy, a strong human player can systematically explore various options, probing for weaknesses without fear of being punished for using a highly predictable style. This kind of methodical exploration for the most effective counter-strategy is not possible against a rapidly adapting opponent.

Moreover, the key to defeating all human poker players is to exploit their highly non-optimal play. This requires a program that can observe an opponent's play and adapt to dynamically changing conditions.

4.4 *Miximax* and *Miximix* Search

Expectimax search is the counterpart of *perfect information minimax search* for domains with a stochastic element [20, 21]. Expectimax combines the minimization and maximization nodes of *minimax search* with the addition of chance nodes, where a stochastic event happens (for example, a dice roll). The value of a chance node is the sum of the values of each of the children of that node, weighted by the probability of that event occurring (1/6 for each roll in the case of a die).

For perfect information stochastic games such as backgammon, an opponent decision node is treated as a normal max (or min) node. However, this cannot be done for imperfect information games like poker, because the nodes of the tree are not independent. Several opponent decision nodes belong to the same information set, and are therefore indistinguishable from each other. In poker, the information set is comprised of all the possible opponent hands, and our policy must be the same for all of those cases, since we do not know the opponent's cards. Furthermore, a player can, in general, use a *randomized mixed strategy* (a probability distribution over the possible actions), so taking the maximum (or minimum) value of the subtrees is not appropriate.

To extend Expectimax for poker, we handle all of the opponent decision nodes within a particular information set as a single node, implicitly maintaining a probability distribution over the range of possible hands they might hold. We cannot treat all possible combinations as having equal probability (for example, weak hands might be folded early and strong hands played through to the end). Imperfect information adds an extra challenge in evaluating leaf nodes in the search, since we can only estimate the relative probabilities for the opponent's possible holdings, rather than having exact values.

We have implemented two variants of Expectimax for search on poker game trees, which we call *Miximax* and *Miximix*. These algorithms compute the expected value at decision nodes of an imperfect information game tree by modeling them as chance nodes with probabilities based on the information known or estimated about the domain, and the specific opponent [12].

The algorithm performs a full-width depth-first search to the leaf nodes of the imperfect information game tree. For two-player poker, the leaf nodes are terminal nodes that end the game – either at a showdown, or when one of the players folds. The search tree is the set of all possible betting sequences from the current state to the end of the game, over all possible outcomes of future chance nodes. At the showdown leaf nodes, the probability of winning is estimated with a heuristic evaluation function, and the resulting EV is backed-up the tree. This tree can be a fairly large (millions of nodes), but with efficient data structures and caching of intermediate calculations, it can be computed in real-time (about one second). In general, the search can be stopped at any depth, and the evaluation function used to estimate the EV of that subtree, as is done in traditional game-playing programs.

The EV calculation is used to decide which action the program should perform: bet/raise, check/call, or fold. Given the EV for each of our three possible actions, one could simply select the option with the maximum value. In that case, the tree will contain mixed nodes for the opponent's decisions and max nodes for our own decisions. Hence we call this algorithm *Miximax*.

However, always taking the maximum EV could lead to predictable play that might be exploited by an observant opponent. Instead, we could choose to use a mixed strategy ourselves. Although we (presumably) know the randomized policy we will use, it can be viewed as both players having mixed nodes, and we call this more general algorithm *Miximix*. (Thus, *Miximax* is a special case of *Miximix* in which all of our own decision nodes use a *pure strategy*, choosing one action 100% of the time).

There are two unresolved issues:

1. How to determine the relative probabilities of the opponent's possible actions at each decision node. This is based on frequency counts of past actions at corresponding nodes (*i.e.*, given the same betting sequence so far).

2. How to determine the expected value of a leaf node. At fold nodes, computing the EV is easy – it is the net amount won or lost during the game. At showdown nodes, a probability density function over the strength of the

opponent's hand is used to estimate our probability of winning. This histogram is an empirical model of the opponent, based on the hands shown in corresponding (identical or similar) situations in the past.

In summary, the search tree consists of four types of nodes, each with different properties:

Chance nodes: For chance nodes in the game tree, the EV of the node is the weighted sum of the EVs of the subtrees associated with each possible outcome. In Texas Hold'em, chance outcomes correspond to the dealing of public ***board cards***. To be perfectly precise, the probability of each possible chance outcome is dependent on the cards that each player will likely hold at that point in the game tree. However, since that is difficult or impossible to determine, we currently make the simplifying (but technically incorrect) assumption that the chance outcomes occur uniformly at random.[2] Thus, the EV of a chance node is simply the average EV over all of the expansions.

Let $Pr(C_i)$ be the probability of each branch i of chance node C, and let n be the number of branches. The EV of node C is:

$$EV(C) = \sum_{1 \leq i \leq n} Pr(C_i) \times EV(C_i) \tag{4.1}$$

Opponent decision nodes: Let $Pr(O_i)$ be the estimated probability of each branch i (one of fold, call, or raise) at an opponent decision node O. The EV of node O is the weighted sum:

$$EV(O) = \sum_{i \in \{f,c,r\}} Pr(O_i) \times EV(O_i) \tag{4.2}$$

Program decision nodes: At decision node U, we can use a mixed policy as above (*Miximix*), or we can always take the maximum EV action for ourselves (*Miximax*), in which case:

$$EV(U) = \max(EV(U_f), EV(U_c), EV(U_r)) \tag{4.3}$$

[2] To see why the remaining deck does not have a uniform density over all cards, notice that the opponent has had previous opportunities to selectively fold with very weak cards. Since the opponent has elected to not fold, the cards in the opponent's hand are stronger than the uniform average, and the deck is slightly richer in low-rank cards than in high-rank cards.

Leaf nodes: Let L be a leaf node, P_{win} be the probability of winning the pot, $L_{\$pot}$ be the size of the pot, and $L_{\$cost}$ be the cost of reaching the leaf node (normally half of $L_{\$pot}$). At terminal nodes resulting from a fold, P_{win} is either zero (if we folded) or one (if the opponent folded), so the EV is simply the amount won or lost during the game. The net EV of a showdown leaf node is:

$$EV(L) = (P_{win} \times L_{\$pot}) - L_{\$cost} \tag{4.4}$$

4.5 EV Calculation Example

For each showdown leaf node of the game tree, we store a histogram of the *hand rank* (HR, a percentile ranking between 0.0 and 1.0, broken into 20 cells with a range of 0.05 each) that the opponent has shown in previous games with that exact betting sequence. We will use 10-cell histograms in this section to simplify the explanation.

For example, suppose we are Player 1 (P1), the opponent is Player 2 (P2), and the pot contains four ***small bets*** (sb) on the final betting round. We bet (2 sb) and are faced with a raise from P2. We want to know what distribution of hands P2 would have in this particular situation. Suppose that a corresponding 10-cell histogram has relative weights of [1 1 0 0 0 0 0 4 4 0], like that shown in Figure 4.1. This means that based on our past experience, there is a 20% chance that P2's raise is a ***bluff*** (a hand in the HR range 0.0-0.2), and an 80% chance that P2 has a hand in the HR range 0.7-0.9 (but not higher).

The histogram for the showdown node after we re-raise and P2 calls (bRrC) will be related, probably having a shape like [0 0 0 0 0 0 0 5 5 0], because P2 will probably fold if he was bluffing, and call with all legitimate hands. The action frequency data we have on this opponent will be consistent, perhaps indicating that after we re-raise, P2 will fold 20% of the time, call 80%, and will not re-raise (because it is not profitable to do so). The probability triple of action frequencies is $Pr(F, C, R) = \{0.2, 0.8, 0.0\}$.

To decide what action to take in this situation, we compute the expected value

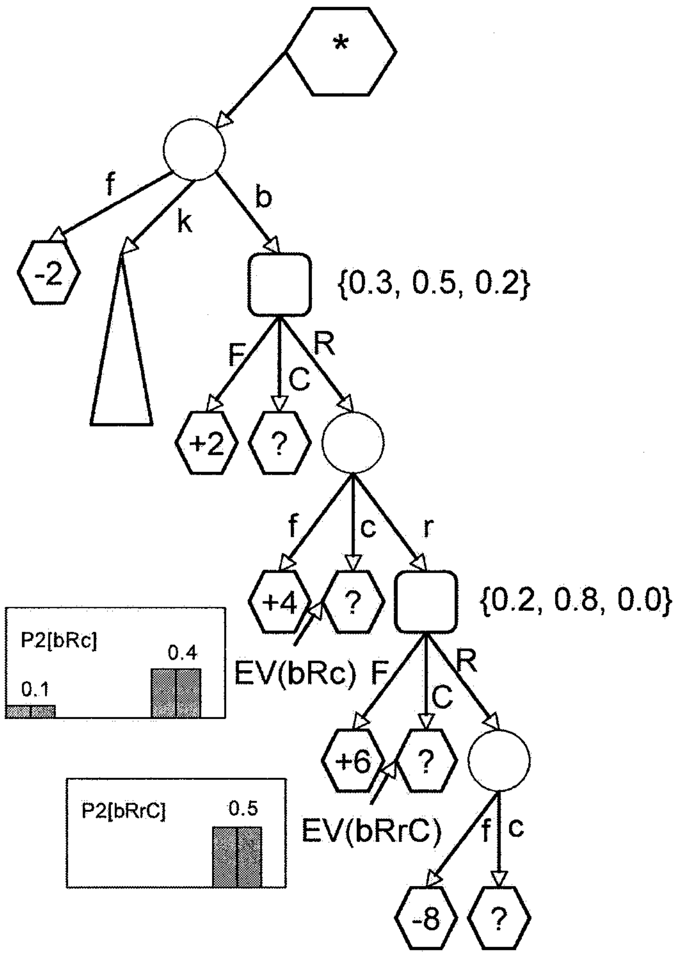


Figure 4.1: Betting tree for the EV calculation example.

HR	Pr(w | c)	EV(c)	Pr(w | rC)	EV(r)	Action
0.70	0.2	-3.6	0.0	-5.2	call
0.75	0.4	-1.2	0.25	-2.0	call
0.80	0.6	+1.2	0.5	+1.2	c or r
0.85	0.8	+3.6	0.75	+4.4	raise
0.90	1.0	+6.0	1.0	+7.6	raise

Table 4.1: Expected values for call or raise for selected hand ranks.

for each choice: EV(fold), EV(call), and EV(raise). EV(fold) is easily determined from the betting history – the game will have cost us -4 small bets.

EV(call) depends on our probability of winning, which depends on the strength of our hand. If our hand rank is in the range 0.2-0.7, then we can only beat a bluff, and our chance of winning the showdown is $Pr(win|bRc) = 0.20$. Since the final pot will contain 12 sb, of which we have contributed 6 sb, the net EV(call) = -3.6 sb. Therefore, we would not fold a hand in the range 0.2-0.7, because we expect to lose less in the long run by calling (0.4 sb less).

If our hand rank is only HR = 0.1 (we were bluffing), then EV(call) = -4.8 sb, and we would be better off folding. If hand rank is HR = 0.8, then we can also beat half of P2's legitimate hands, yielding an expected profit of EV(call) = +1.2 sb. Table 4.1 gives the EV for calling with selected hand ranks in the range 0.7-0.9.

To calculate the expected value for re-raising, we must compute the weighted average of all cases in that subtree, namely: bRrF, bRrC, bRrRf, and bRrRc. Since the probability assigned to a P2 re-raise is zero, the two latter cases will not affect the overall EV and can be ignored. The share from bRrF is $0.2 \times 6 = +1.2$ sb, and is independent of our hand strength. The probability of winning after bRrC is determined by the histogram for that case, as before. Thus, in this example EV(raise) $= 1.2 + 0.8 \times (16 * Pr(win|rC) - 8)$, as shown in Table 4.1 for the same selected hand ranks.

As a consequence of this analysis, if we are playing a strictly maximizing strategy, we would decide to fold if our hand is weaker than HR = 0.167, call if it is in the range 0.167 to 0.80, and re-raise if it is stronger than HR = 0.80. Computing the viability of a possible bluff re-raise is done similarly.

4.6 Abstractions for Opponent Modeling

After each hand is played against a particular opponent, the observations made during that game are used to update our opponent model. The action decisions made by the opponent are used to update the betting frequencies corresponding to the sequence of actions during the game. When showdowns occur, the hand rank (HR) shown by the opponent is used to update a leaf node histogram, as illustrated in the previous section.

The *context tree* is an explicit representation of the imperfect information game tree, having the same skeletal structure with respect to decision nodes. Chance nodes in the tree are represented implicitly (all possible chance outcomes are accounted for during the EV calculation).

A leaf node of the context tree corresponds to all of the leaves of the game tree with the same betting sequence (regardless of the preceding chance nodes). Associated with this is an efficient data structure for maintaining the empirically observed action frequencies and showdown histograms for the opponent. For this we use a trie, based on the natural prefix structure of related betting sequences. Hash tables were used for low-overhead indexing.

4.6.1 Motivation for Abstractions

The *Miximax* and *Miximix* search algorithms perform the type of mathematical computation that underlies a theoretically correct decision procedure for poker. For the game of two-player Limit Texas Hold'em, there are $9^4 = 6561$ showdown nodes for each player, or 13122 leaf-level histograms to be maintained and considered. This fine level of granularity is desirable for distinguishing different contexts and ensuring a high correlation within each class of observations.

However, having so many distinct contexts also means that most betting sequences occur relatively rarely. As a result, many thousands of games may be required before enough data is collected to ensure reliable conclusions and effective learning. Moreover, by the time a sufficient number of observations have been made, the information may no longer be current.

This formulation alone is not adequate for practical poker. It is common for top human players to radically change their style of play many times over the course of a match. A worthy adversary will constantly use deception to disguise the strength of their hand, mask their intentions, and try to confuse our model of their overall strategy. To be effective, we need to accumulate knowledge very quickly, and have a preference toward more recent observations. Ideally, we would like to begin applying our experience (to some degree) immediately, and be basing decisions primarily on what we have learned over a scope of dozens or hundreds of recent games, rather than many thousands. This must be an ongoing process, since we may need to keep up with a rapidly changing opponent.

Theoretically, this is a more challenging learning task than most of the problems studied in the machine learning and AI literature. Unlike most Markov decision process (MDP) problems, we are not trying to determine a static property of the domain, but rather the dynamic characteristics of an adversarial opponent, where historical perspective is essential.

In order to give a preference toward more recent data, we gradually "forget" old observations using exponential history decay functions. Each time an observation is made in a given context, the previously accumulated data is diminished by a history decay factor, h, and the new data point is then added. Thus, for $h = 0.95$, the most recent event accounts for 5% of the total weight, the last $1/(1-h) = 20$ observations account for approximately $(1 - 1/e) = 0.63$ of the total, and so on.[3]

4.6.2 Abstraction

In order to learn faster and base our inferences on more observations, we would like to combine contexts that we expect to have a high mutual correlation. This allows us to generalize the observations we have made, and apply that knowledge to other related situations. There are many possible ways of accomplishing these abstractions, and we will address only a few basic techniques.

An important consideration is how to handle the *zero-frequency problem*, when

[3] The exact value for this case is $1 - h^{20} = 0.6415$, but the approximation is true in general for $1/(1-h)$ observations.

there has yet to be any observations for a given context; and more generally, how to initialize the trie with good default data. Early versions of the system employed somewhat simplistic defaults, which resulted in rather unbalanced play early in a match. More recent implementations use default data based on rational play for both players, derived in a manner analogous to Nash equilibrium strategies.

The finest level of granularity is the context tree itself, where every possible betting sequence is distinct, and a different histogram is used for each. The opponent action frequencies are determined from the number of times each action was chosen at each decision node (again using a history decay factor to favour recent events). Unfortunately, having little data in each class will result in unreliable inferences.

One coarse-grained abstraction groups all betting sequences where the opponent made an equal number of bets and raises throughout the game, ignoring what stage of the game they were made. A finer-grained version of the same idea maintains an ordered pair for the number of bets and raises by each player.

However, testing reveals that an even coarser-grained abstraction may be desirable. Summing the total number of raises by both players (no longer distinguishing which player initiated the action) yields only nine distinct classes. Despite the crudeness of this abstraction, the favorable effects of grouping the data is often more important than the lower expected correlations between those lines of play.

Another similar type of coarse-grained abstraction considers only the final size of the pot, adjusting the resolution (*i.e.*, the range of pot sizes) to provide whatever number of abstraction classes is desired.

An abstraction system can be hierarchical, in which case we also need to consider how much weight should be assigned to each tier of abstraction. This is based on the number of actual observations covered at each level, striving for an effective balance, which will vary depending on the opponent.

Our method of combining different abstraction classes is based on an exponential mixing parameter (say $m = 0.95$) as follows. Let the lowest-level context tree (no abstraction) be called A0, a fine-grained abstraction be called A1, a cruder amalgam of those classes be called A2, and the broadest classification level be called A3. Suppose the showdown situation in question has five data points that match the context exactly, in A0. This data is given a weight of $(1 - m^5) = 0.23$ of the total. If the next level of abstraction, A1, has 20 data points (including those from A0), it is assigned $(1 - m^{20}) = 0.64$ of the remaining weight, or about 50% of the total. The next abstraction level might cover 75 data points, and be given $(1 - m^{75}) = 0.98$ of the remainder, or 26% of the total. The small remaining weight is given to the crudest level of abstraction. Thus, all levels contribute to the overall profile, depending on how relevant each is to the current situation.

4.7 Experiments

To evaluate the strength of VEXBOT, we conducted both computer *vs* human experiments, and a round-robin tournament of computer *vs* computer matches.

The field of computer opponents consisted of:

1) **SPARBOT**,[4] the publicly available version of PSOPTI4, which surpassed all previous programs for two-player Limit Hold'em by a large margin [4].

2) **POKI**, a formula-based program that incorporates opponent modeling to adjust its hand evaluation. Although POKI is the strongest known program for the ten-player game, it was not designed to play the two-player game, and thus does not play that variation very well [5].

3) **HOBBYBOT**, a slowly adapting program written by a hobbyist, specifically designed to exploit POKI's flaws in the two-player game.

4) **JAGBOT**, a simple static formula-based program that plays a rational, but unadaptive game.

5) **ALWAYS_CALL** and

6) **ALWAYS_RAISE**, extreme cases of weak exploitable players, included as a simple benchmark.

The results of the computer *vs* computer matches are presented in Table 4.2. Each match consisted of at least 40,000 games of poker. The outcomes are statisti-

[4] The name SPARBOT refers generally to any program in the PSOPTI family of game-theoretic programs. PSOPTI4 through to the most recent version PSOPTI7, are all based on the architecture of PSOPTI2, with overlaying 3-round models, described in Chapter 3. PSOPTI4 is publicly available on the University of Alberta online poker server (`www.cs.ualberta.ca/~games/`), and in the commercial program POKER ACADEMY (`www.poker-academy.com/`).

Program	Vexbot	Sparbot	Hobbot	Poki	Jagbot	A.Call	A.Raise
Vexbot		+0.052	+0.349	+0.601	+0.477	+1.042	+2.983
Sparbot	-0.052		+0.033	+0.093	+0.059	+0.474	+1.354
Hobbybot	-0.349	-0.033		+0.287	+0.099	+0.044	+0.463
Poki	-0.601	-0.093	-0.287		+0.149	+0.510	+2.139
Jagbot	-0.477	-0.059	-0.099	-0.149		+0.597	+1.599
Always Call	-1.042	-0.474	-0.044	-0.510	-0.597		=0.000
Always Raise	-2.983	-1.354	-0.463	-2.139	-1.599	=0.000	

Table 4.2: Computer *vs* computer matches (small bets per game).

cally significant, with a standard deviation of approximately ± 0.03 small bets per game (sb/g).

VEXBOT won every match it played, and had the largest margin of victory over each opponent. VEXBOT approaches the theoretical maximum exploitation against ALWAYS_CALL and ALWAYS_RAISE. No other programs came close to this level, despite those opponents being perfectly predictable.

Against SPARBOT, the strongest previous program for the two-player game, VEXBOT was able to find and exploit flaws in the pseudo-optimal strategy. The learning phase was much longer against SPARBOT than any other program, typically requiring several thousand games. However, once an effective counter-strategy is discovered, VEXBOT will continue to win at that rate or higher, due to the oblivious nature of the game-theoretic player.

The testing against humans (Table 4.3) involves a smaller number of trials, and should therefore be taken only as anecdotal evidence (the outcomes being largely dominated by short-term swings in luck). Most humans players available for testing did not have the patience to play a statistically significant number of games (especially when frustrated by losing). However, it is safe to say that VEXBOT easily exploited weaker players, and was competitive against expert level players. The results also consistently showed a marked increase in its win rate after the first 200-400 games of the match, presumably due to the opponent-specific modeling coming into effect.

A more recent implementation of the *Miximax* algorithm was able to improve considerably on the VEXBOT results against computer opponents. That version

Num	Rating	sb/g	Games
1	Expert	-0.022	3739
2	Intermediate	+0.136	1507
3	Intermediate	+0.440	821
4	Intermediate	+0.371	773

Table 4.3: VEXBOT *vs* human matches.

defeated SPARBOT by +0.145 sb/g – three times the win rate of the previous version.[5] Moreover, its win rate against SPARBOT was more than three times the win rate achieved by a world-class player [4], but still considerably less than that of the most successful human player (the author).

Revised versions of both programs competed in the 2003 Computer Olympiad, with VEXBOT again dominating SPARBOT, winning the gold and silver medals respectively [3].

4.8 Conclusions and Future Work

Limit poker is primarily a game of mathematics and opponent modeling. We have built a program that "does the math" in order to make its decisions. As a result, many sophisticated poker strategies emerge without any explicit encoding of expert knowledge. The adaptive and exploitive nature of the program produces a much more dangerous opponent than is possible with a purely game-theoretic approach.

One limitation not yet adequately addressed is the need for effective defaults, to ensure that the program does not lose too much while learning about a new (unknown) opponent at the beginning of a match. If good default data is not easily derivable by direct means, there are several ways that existing programs can be combined to form hybrids that are less exploitable than any of the component programs in isolation.

With a smoothly adapting program and a good starting point, it may be possible to use self-play matches and automated machine learning to constantly refine the

[5] Although all versions of VEXBOT can be challenging and dangerous opponents, the play can also be highly volatile and suspect, especially early in a match, when it must depend on the wholly inadequate default seeding as its model of real play.

default data. In principle, that process could eventually approach a game-theoretic near-optimal default that is much closer to a true Nash equilibrium strategy than has been obtained to date.

The performance of VEXBOT can be improved further in numerous ways. While we believe that the modeling framework is theoretically sound, the parameter settings for the program could be improved considerably. Beyond that, there is a lot of room for improving the context tree abstractions, to obtain higher correlations among grouped sequences.

Only the two-player variant has been studied so far. Generalization of these techniques to handle the multi-player game should be more straightforward than with other approaches, such as those using approximations for game-theoretic solutions.

Refinements to the architecture and algorithms described in this paper will undoubtedly produce increasingly strong computer players. It is our belief that these programs will have something to teach all human poker players, and that they will eventually surpass all human players in overall skill.

Acknowledgments

This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), Alberta's Informatics Circle of Research Excellence (iCORE), and by the Alberta Ingenuity Center for Machine Learning (AICML). The author was supported in part by an Izaak Walton Killam Memorial Scholarship and by the Province of Alberta Ralph Steinhauer Award of Distinction.

Bibliography

[1] D. Billings. The First International RoShamBo Programming Competition. *The International Computer Games Association Journal*, 23(1):42–50, 2000.

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

[3] D. Billings. Vexbot wins poker tournament. *International Computer Games Association Journal*, 26(4):281, 2003.

[4] D. Billings, N. Burch, A. Davidson, T. Schauenberg, R. Holte, J. Schaeffer, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *The Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 661–668, 2003.

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

[6] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 493–499, 1998.

[7] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *American Association of Artificial Intelligence National Conference, AAAI'99*, pages 697–703, 1999.

[8] P. Brusilovsky, A. Corbett, and F. de Rosis, editors. *User Modeling 2003*. Springer-Verlag, 2003.

[9] M. Buro. Solving the Oshi-Zumo game. In H. J. van den Herik, H. Iida, and E. A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges*, pages 361–366. Kluwer Academic, 2004.

[10] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *American Association of Artificial Intelligence National Conference, AAAI'96*, pages 120–125, 1996.

[11] F. Dahl. A reinforcement learning algorithm to simplified two-player Texas Hold'em poker. In *European Conference on Machine Learning*, pages 85–96, 2001.

[12] A. Davidson. Opponent modeling in poker. Master's thesis, Department of Computing Science, University of Alberta, 2002.

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

[14] E. Horvitz, L. Breese, D. Heckerman, D. Hovel, and K. Rommeke. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Annual Conference on Uncertainty in Artificial Intelligence, UAI'98)*, pages 256–265, 1998.

[15] H. Iida, J. Uiterwijk, H. J. van den Herik, and I. Herschberg. Potential applications of opponent-model search. *ICCA Journal*, 16(4):201–208, 1993.

[16] P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search.* PhD thesis, School of Computer Science, Carnegie-Mellon University, 1992.

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

[18] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Annual Conference on Uncertainty in Artificial Intelligence, UAI'99*, pages 343–350, 1999.

[19] 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.

[20] D. Michie and R. Chambers. Boxes: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 125–133. Elsevier / North-Holland, 1968.

[21] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[22] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[23] D. Weld, C. Anderson, P. Domingos, O. Etzioni, T. Lau, K. Gajos, and S. Wolfman. Automatically personalizing user interfaces. In *The International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 1613–1619, 2003.

Chapter 5

Assessment of Performance (2005)

A Tool for the Direct Assessment of Poker Decisions [1]

5.1 Introduction

The game of poker has many properties that present new challenges for Artificial Intelligence research, making it distinct from all previously studied games. Traditional games like chess and Go are two-player perfect information games with no element of random chance. Poker is at the opposite end of the spectrum: a multi-player imperfect information game with partial observability and stochastic outcomes. Deception, opponent modeling, and coping with uncertainty are indispensable elements of high-level strategic play.

The property of stochasticity compounds many of the complex problems in the domain. It also has a highly detrimental effect on performance assessment: on quantifying how well a player is playing, or simply determining who the better players are. The "signal to noise ratio" is low – a player can play extremely well and still lose over an extended period of time, just by bad luck. The normal variance in the game is so high that many thousands of games need to be played before obtaining even a modest degree of confidence in the result. Without reliable feed-

[1] The contents of this chapter have been accepted for publication in The International Computer Games Association Journal. A previous version of this work appeared in Technical Report TR06-07, Copyright 2006 University of Alberta. All rights reserved. D. Billings and M. Kan. Development of a tool for the direct assessment of poker decisions. Technical Report TR06-07, University of Alberta Department of Computing Science, April 2006.

back on how effective a solution is, it is difficult to identify problem areas and make steady forward progress. Moreover, the problem will become more serious over time. As programs become stronger, and closer to each other in skill level, the number of games, n, required to distinguish between them with a particular statistical confidence grows at a rate of $\Theta(n^2)$.

Some techniques can help ameliorate the problem of high variance. For example, *duplicate tournament systems* use independent tournaments with the same series of cards, shuffling the players to different positions for each replay [1]. Since games between computer programs can be re-played with no memory, the total number of good and bad situations is equalized, at least to some degree. However, once the players deviate in their corresponding actions (particularly if one player folds while another continues with the hand), the subsequent actions are no longer directly comparable, and are subject to random outcomes. The problem of accurate assessment persists, even if other sources of noise are kept under control. Furthermore, duplicate systems cannot be used for assessing an individual human player, since they will recognize previously played hands and outcomes. Developing a tool to perform an objective and insightful analysis of the decisions made by each player would be of great value, because the effects of randomness could largely be factored out. Unfortunately, this turns out to be difficult to construct, both in theory and in practice, due to the implications of imperfect information in poker.

It is useful to compare poker to perfect information games that have a stochastic element. Two well-studied examples are backgammon and blackjack. In backgammon, both players have full access to the state of the game, but randomness is introduced by the roll of the dice. In blackjack, the dealer's face-down card has no effect on the player's decision making, and is drawn from the same probability distribution as the rest of the unknown cards. Since the dealer's decisions are completely determined by the rules of the game, blackjack can be viewed as a single-player stochastic game. In all perfect information games, whether stochastic or deterministic, the notion of a *best move*, or *set of maximal actions* is well-defined. In backgammon and blackjack, each choice has an objective game-theoretic ***expected value*** (EV), and usually there is exactly one choice that has a maximum EV. In deterministic perfect information games like chess and checkers, there is a set of moves that preserves the game-theoretic value of the game. Of these, there may be a "best practical move" that maximizes the probability of success in a contest between typical imperfect players.

For these games, the quality of each player decision can be compared to the best available choice, using an *oracle* or *near-oracle* to determine the quantitative value of every possible move. A perfect oracle is based on exact enumeration over all reachable game states. In blackjack, simulation or enumeration using the exact composition of the remaining deck provides a sound basis for direct assessment of EV decisions, as shown by Wolfe [15].

A near-oracle can be based on Monte Carlo simulations, using a strong program to provide accurate estimates of the objective value of each move. In backgammon, Monte Carlo *roll-outs* using a strong program like TD_GAMMON are considered to be the definitive assessment of game equity [13, 14]. In the game of Scrabble, the program MAVEN has surpassed all human players in skill level, and is used in a similar fashion [11, 10]. Again, these are good baselines for excellent play because they are strongly correlated with the maximum EV play available in each situation. Poker programs are not yet strong enough to serve as a near-oracle; but the problem goes much deeper than that, because the concept of a single maximum EV play is not applicable.

In contrast to perfect information games, there is generally no single best choice in a given poker situation. The concept of a perfect oracle is not well-defined (or at least, is not very discriminating between the possible options). A sound strategy is dependent on the *relative balance* of deceptive plays (such as ***bluffing*** with a weak hand, or ***trapping*** with a strong hand). A balanced approach is essential, but the player has considerable flexibility in *how* to obtain that balance. In both theory and practice, radically different strategies can have the same objective EV and can be equally viable. Analogously, the notion of a best move is not meaningful in the imperfect information game of Rock-Paper-Scissors – the most effective choice is entirely dependent on the preceding history between the players.

In practice, a good poker player's strategy is strongly dependent on the beliefs about the opponent's weaknesses and vulnerabilities. For example, if Alice bluffs too much, an appropriate counter-strategy is to call (or raise) more often with mediocre holdings. If Bob seldom bluffs, the appropriate response is to fold more often with mediocre holdings. We see the opposite response depending on the perception of the opponent's style; and a player must continue to adapt as the opponent changes their strategy over time. In other words, in chess it is possible and appropriate to simply "play the board" (ignoring the opponent), whereas in poker it is necessary to "play the player" to maximize the win rate. Since there cannot be a "perfect" way to do this in general, it introduces a speculative element to the game that is not consistent with defining an objective assessment system.

For these and other reasons (some of which are outlined in the next section), it becomes evident that developing a "perfect" assessment tool is essentially impossible. In fact, an expert might argue that the assessment problem is "POKER-complete", meaning that it is at least as hard as any other problem in the domain, including playing perfectly (if such a notion is even well-defined).

However, it is possible to design an imperfect assessment system that is conceptually simple, objective, and highly effective at reducing the variance due to stochastic outcomes. The Ignorant Value Assessment Tool (DIVAT)[2] is one such system. The term "ignorant" refers to the fact that it implicitly *ignores* much of the context in each situation. The expected value estimates may not be especially well-informed, but they are *consistent*, and are applied in an egalitarian fashion to all players. In the two-player case, the same criteria are applied equally to both players, with the intention of obtaining an unbiased assessment, even if the specific criteria could be criticized as being somewhat arbitrary. After we developed the tool empirically, it was formally proven to be statistically unbiased by Zinkevich et al. [16]. This means that the long-term expected value from the DIVAT assessment is guaranteed to match the long-term expected value of money earnings.

In Section 5.2, we examine the strengths and limitations of perfect information hindsight analysis, provide some background on previous attempts to use this approach, and present a concrete example of a game in order to ground the discussion.

[2] The first letter of the acronym corresponds to the author's first initial.

In Section 5.3, we define the terminology and explain the metrics used in each of the components, and re-visit the example to perform a detailed quantitative analysis using the DIVAT system. In Section 5.4, we illustrate some of the uses of the tool, and demonstrate its power in variance reduction, while providing an unbiased estimate of the difference in skill between two players playing a series of games. We conclude with a discussion of future work and generalization of the technique to other domains.

5.2 Motivation

In this section we will focus our attention on a specific example of a game of poker, in order to identify some of the considerations and obstacles an analysis tool will need to cope with. We examine the reasons that an imperfect information domain cannot simply be broken down into a collection of perfect information instances. Then, we look at our previous attempts to use perfect knowledge hindsight analysis and identify their major failings, which leads us to the design of the DIVAT system.

5.2.1 Example Game

We now look at an example of one complete game of Limit Texas Hold'em. We will call Player 1 (P1) Alfred, and Player 2 (P2) Betty. Although we will try to motivate some of their decisions, the objective analysis will, of course, be completely free of any such subjective interpretation.

A succinct summary of the game is:

Alfred: A♣-K♣ Betty: 7♡-6♡ Board: K♠-5♡-3♢ T♣ 4♡

Betting: SlRrC/kBrC/bC/bRc

All monetary units will be expressed in terms of ***small bets*** (sb), which is the size of all bets and raises in the first two rounds of play (the bet size doubles for the last two rounds of play). With the ***reverse-blinds*** format of two-player Limit Texas Hold'em, P2 posts a ***small blind*** bet of 0.5 sb, P1 posts a ***big blind*** bet of 1 sb, and P2 must make the first betting decision – to either fold, call 0.5 sb, or raise another 1 sb. The betting notation is: 's' for small blind, 'l' for large blind, 'k' for check, 'b' for bet, 'f' for fold, 'c' for call, 'r' for raise, and '/' is used as a delimiter for the four betting rounds.[3] Upper-case letters are used to more easily distinguish the actions of the second player (Betty).

Betty is dealt the **7♡-6♡**, which is not a particularly strong hand, but is certainly worth calling to see the ***flop***. In this game, Betty elects to raise with the hand, for purposes of deception (called a ***semi-bluff***). Alfred has a very strong hand, the **A♣-K♣**, and re-raises, to Betty's detriment. Betty calls.

The flop is the **K♠-5♡-3♢**, giving Alfred a strong hand with a pair of Kings. Alfred decides to try for a ***check-raise trap***, perhaps based on the belief that Betty bets too aggressively after a check (and is not as aggressive when it comes to raising a bet). Betty bets (another semi-bluff) and Alfred completes his plan by raising. Betty has a weak hand, but the ***pot*** is now large (nine small bets) so she calls in the hope of improving (with a **4**, **6**, or **7**; or any heart, **8**, or **9**).

The ***turn*** card is the **T♣**, and Alfred simply bets his strong hand, since he believes that trying for another check-raise trap is unlikely to succeed. Betty still has nothing, and cannot be certain whether or not making a pair with a **6** or **7** will give her the best hand. If those outcomes will win (10/44 in total), then she has a correct call (2 sb to win 12 sb); if not, then she is better off folding. She elects to call, perhaps because Alfred has frequently followed this betting pattern in previous games without having a strong hand.

The ***river*** is a perfect **4♡** for Betty, giving her the best possible hand (she could tie, but she cannot lose). Of course, Alfred has no way of knowing that this card was a disaster for him, and bets with what he believes to be the best hand. Betty raises, and Alfred elects to only call, perhaps based on his prior experience where Betty's raises on the final betting round are usually meaningful. The final pot size is 22 sb, giving Betty a net of +11 sb, and Alfred a net of -11 sb.

As an outside observer, we would like to know if one of these players played better than the other. We would like to filter out the stochastic luck element, and have an objective means of quantifying the perceived difference in skill. For a

[3] A ***check*** is functionally equivalent to a ***call***, but "check" is used when the current amount to call is zero. Similarly, a ***bet*** is functionally equivalent to a ***raise***, but "bet" is used when the current amount to call is zero.

rational and accurate assessment, we would like to know how each player played *in general*, not just in hindsight, with all of the cards known. In this example game, most poker players would agree that Alfred played his hand well. However, most of his decisions were relatively easy,[4] so it is difficult to say how competent he is based on this one game. From the perspective of an average player, Betty might appear to be a wild gambler; but to an expert observer, she might appear to be either a world-class player *or* a weak player. Much of the assessment is subjective opinion, and is neither right nor wrong. Moreover, the opinions cannot be well-informed without a full historical context – Betty's decisions will depend very strongly on Alfred's style of play, and vice-versa. Real poker is highly non-Markovian in nature (*i.e.*, is not memoryless). A single game taken in isolation does not provide sufficient context for a well-informed assessment. However, for practical purposes, an analysis tool will likely have to ignore this fact (or temper it with some form of approximation).

5.2.2 EVAT and LFAT

Since 2001, we have used an assessment procedure called the Expected Value Assessment Tool (EVAT) to try to gain a more accurate picture of the results of short matches. The EVAT is based on hindsight analysis, where each player's decisions are compared to what the maximal action would have been if all players had known all of the cards. Thus, we are comparing imperfect information decisions to the actions that would have been taken in the perfect information variant of the game. This provides a perspective that is quite distinct from the money line, and can occasionally provide useful insights. Unfortunately, it also suffers from some serious drawbacks, and tends to be rather unreliable as a predictor of future outcomes. Because of these limitations, the tool has not found its way into regular use.

The EVAT view is analogous to what poker author David Sklansky [12] calls "The Fundamental Theorem of Poker" (FToP), which states:

> Every time you play a hand differently from the way you would have played it if you could see all your opponents' cards, they gain; and

[4] In general, it is easier to make good decisions with a strong hand than it is with a hand near the borderline between fold and not fold.

> every time you play your hand the same way you would have played it if you could see all their cards, they lose. Conversely, every time opponents play their hands differently from the way they would have if they could see all your cards, you gain; and every time they play their hands the same way they would have played if they could see all your cards, you lose.

Sklansky is suggesting that the long-term expected value in poker is equivalent to the differences in perfect information decisions. Similar assertions have been made in the past for many domains involving imperfect or incomplete information. However, strictly speaking this is not true, and the FToP is not a theorem. An imperfect information game cannot be expressed merely as a collection of perfect information instances. Some of the key differences between these classes of problems were exemplified for the game of bridge by Frank and Basin [7], and by Ginsberg [8].[5] The FToP is also qualitative in nature, not quantitative.[6] It is, however, a useful heuristic that can guide human players to formulating better decisions.

The EVAT is essentially a quantitative comparison between actual decisions and the ideal perfect information counterparts. Each time the actions are different, the player is assessed a penalty, equal in magnitude to the difference in expected value. The sum of all such "misplays" for each player are compared, yielding a quantitative estimate of the difference in skill.

Unfortunately, the EVAT is based on a highly unrealistic baseline for comparison, because the players are implicitly expected to have omniscient knowledge, without regard to the actual conditions that exist in each instance. As an extreme example, a player with the second-best possible hand on the river would be expected to *fold* whenever the opponent happens to hold the best possible hand, and raise otherwise. Conversely, a player with the second-worst possible hand would be expected to fold *except* when the opponent happens to hold the worst possible

[5] A simple example is the case of a two-way finesse: the declarer can win a key trick and secure the contract in every perfect information lay of the cards (a 100% chance of success), but is forced to *guess* in the real imperfect information game (for a 50% chance of success).

[6] As stated, the FToP is also untrue for a number of mundane reasons, nor does it hold for multi-player situations, but that is of little concern to this discussion.

hand.

In the example game, Alfred's entirely reasonable bet on the river is a technical misplay, incurring an EVAT penalty, since his action is different from what he would have done if he had known all the cards. Moreover, he is expected to play the hand differently in otherwise identical situations, based only on the hidden information. This goes against fundamental principles of imperfect information games, in that the same policy must be employed within the same *information set* of the *game tree*. The perfect hindsight view is largely irrelevant to the imperfect information reality of the situation, and illustrates one of the biggest flaws with the EVAT policy: when a player has a moderately strong second-best hand, then with reasonable play they *should* lose money, in expectation.

Another way of looking at the problem of variance is in terms of *opportunities*. An immediate consequence of a high-variance stochastic game is that it can take a very long time before each player has had their "fair share" of good and bad situations. One of the reasons that the money line is a weak estimator of the long-term EV of the players is that the opportunities for each player do not balance out quickly.

To a large extent, the EVAT analysis suffers from the same limitation. In contrast, the DIVAT system will automatically account for certain imbalances in opportunities, by distinguishing between good and bad situations, and scaling the penalties accordingly. As a result, the DIVAT hindsight view converges much more rapidly than the comparatively naive EVAT assumptions.

The Luck Filtering Analysis Tool (LFAT) is a complementary system that was developed to address some of the shortcomings of the EVAT. Using the same methods to compute the expected value of each situation, the LFAT compares each player's ***pot equity***[7] before and after each chance event (*i.e.*, cards being dealt). Thus, the LFAT analysis occurs between the betting rounds, while EVAT occurs between the chance events, so the analysis is split into the natural alternating phases of chance outcomes and player actions. The LFAT quantifies the effects of the stochastic outcomes alone, without regard to implications of betting decisions. Conversely,

[7] The concept of equity is explained in Section 5.3.1.2.

the EVAT method considers only the betting decisions themselves, and simply disregards the fluctuations due to luck, which is a highly desirable feature for reducing variance. The DIVAT system is similar to EVAT in this regard.

However, the separation is not perfect. Although the stochastic luck has been eliminated from consideration, the situations that arise truly are a consequence of all previous chance events that have occurred. Thus, the two are not independent, and treating them as such introduces some degree of error.

5.3 Development

In this section, we define terminology and explain the components of the Ignorant Value Assessment Tool. We present specific details of how the DIVAT system is constructed, and then work through the concrete example, illustrating its use.

5.3.1 Definitions and Metrics

In order to analyze the play of poker games, we will need metrics to assess the value of a hand. This is a complex problem, and good methods are often multi-dimensional. For example, *adjusted hand strength* is a combination of *hand strength*, which is a weighted probability of currently holding the best hand, and *hand potential*, which is a measurement of future outcomes [4]. For our purposes, we want to develop a *one-dimensional metric of hand goodness*, on a scale from zero to one.

For a given situation in the middle of a game, it will be necessary to estimate the ***equity*** for each player – the net amount that will be won or lost by the end of the game, on average. Most simplistic metrics do not consider the impact of future betting rounds (known as ***implied odds*** in the terminology of poker theory). While those methods tend to be easy to compute, and are sufficient for some purposes, we will also need to develop better-informed metrics that take future betting rounds into account.

5.3.1.1 IHR, 7cHR, and EHR

The Immediate Hand Rank (IHR) is the relative ranking of a hand, on a scale from zero to one, compared to all other possible holdings at the current point in a game.

IHR considers only the number of opponent hands that are currently ahead, behind, or tied with the player's hand, and ignores all future possibilities. For two-player games, the formula is: IHR = (ahead + tied/2) / (ahead + tied + behind). Before the flop, there are 1225 possible two-card combinations for opponent hands. After the flop cards are known, there are 1081 possible opponent hands. There are 1035 combinations on the turn, and 990 combinations on the river. No inferences or assumptions are made about the opponent's cards, so all hands implicitly have uniform probability in hand rank calculations.

The 7-card Hand Rank (7cHR) performs a complete enumeration of all possible future ***board cards***, computing the hand rank on the river for each instance, and reporting the overall average outcome. On the river, IHR and 7cHR are equivalent, since there are no future cards to be dealt. On the turn, there are 46 possible cases for the river card (before considering the possible opponent holdings). Thus, 7cHR is an enumerated average of one-card futures with respect to 6-card Hand Rank (6cHR). On the flop, there are 1081 two-card futures to consider (order is not important in this case, since no folds are possible with this metric). Thus, 6cHR is an enumerated average of one-card futures with respect to 5-card Hand Rank (5cHR), and 7cHR is an enumerated average of two-card futures with respect to 5cHR. Before the flop, there are 2118760 five-card boards (ignoring order) to reach the river stage.

7cHR amounts to an unweighted enumeration of unweighted hand ranks. It can be computed efficiently through the use of clever caching techniques [4]. Much greater efficiency was obtained by pre-computing look-up tables for all ***pre-flop***, flop, and turn possibilities, mapping to and from canonical representatives to take advantage of suit isomorphisms to reduce storage requirements.

Since 7cHR is an average over all possible future chance outcomes, it contains a mixture of *positive potential* (the chance of improving to the best hand when behind), and *negative potential* (the chance of falling behind when ahead). However, it is not a particularly good balance of the two, because it implicitly assumes that all hands will proceed to a ***showdown***, which is patently false. In practice, 7cHR tends to overestimate the value of weak no-pair hands. For example, **P1: 3♣-**

2♣ Board: K♠-T♡-7♢, has 7cHR = 0.1507, based on two chances of making a small pair. Since the hand could be forced to fold on the turn, it is incorrect to assume a two-card future. In general, we should only be looking forward to the next decision point, which is a one-card future.[8] One possibility is to use 6cHR as an estimate of hand value on the flop. In practice, a better measure for Limit Hold'em is to take the *average* of 5cHR and 7cHR, because that captures the extra value from good two-card combinations. For example, **P1: 3♡-2♡ Board: K♠-T♡-7♢**, has 6cHR = 0.0729, but 7cHR = 0.1893 due to the possibility of two running hearts. Since hitting the first heart will provide a flush draw that almost certainly has enough value to proceed to the river, these indirect combinations have definite value. With respect to the DIVAT folding policy on the flop, we define the Effective Hand Rank (EHR) to be the average of IHR and 7cHR: EHR = (IHR + 7cHR) / 2.

With respect to the DIVAT betting policies (when the hand is sufficiently strong), we define EHR to be the *maximum* of IHR and 7cHR: EHR = max (IHR, 7cHR). The reason the maximum is a better measure of hand value for this purpose is that a hand with high negative potential generally has a greater *urgency* to bet. In the terminology of poker theory, a hand that degrades in strength as the game progresses (*i.e.*, 7cHR < IHR) is said to have a high ***free-card danger***. This means that there is a relatively high risk in *not betting* (or raising), because allowing a free draw could be potentially disastrous in terms of EV (such as losing the pot when a bet would have forced the opponent to fold).

Since 7cHR does not distinguish cases where the opponent does or does not bet, the assessment is somewhat optimistic in practice, even for one-card futures from the turn forward. For example, **P1: 3♣-2♣ Board: K♠-T♡-7♢ 4♣**, has 7cHR = 0.0620, but could easily be ***drawing dead*** against a one-pair hand or better. Making a small pair on the river will not be strong enough to bet, but will be strong enough to warrant a call. Thus, the hand stands to lose an additional two small bets whenever the opponent has a strong hand, but gain nothing when it is the best. In the terminology of poker theory, 7cHR does not adequately account for the ***reverse***

[8] This is especially imperative in No-Limit poker, where the opponent can apply much greater leverage with a large subsequent bet.

implied odds in most situations.

Despite these drawbacks, 7cHR is a convenient metric that is simple to compute, and the shortcomings can be dealt with easily enough. The use of these metrics is illustrated in the detailed example of Section 5.3.4.

5.3.1.2 AIE and ROE

The ***all-in equity*** (AIE) is measured with the fraction of all future outcomes each player will win, given perfect knowledge of the cards held by all players.[9] This fraction can be multiplied by the current pot size as a simple measure of (gross) ***pot equity*** – the "fair share" of the current pot that each player is entitled to, based on the complete enumeration of future outcomes. For two-player Texas Hold'em, there are 44 cases to consider for turn situations, 990 cases for the flop, and 1712304 cases for the pre-flop.

To convert the gross amount to the net gain or loss, we subtract the total amount invested from the current pot equity: Net = (AIE * $potsize$) - $invested$. For example, if the pot size after the turn betting is 10 sb (5 sb from each player), and 11 out of 44 outcomes will win, zero will tie, and 33 will lose (written as +11 =0 -33), then our current net pot equity is Net = 0.25 * 10 - 5 = -2.50 bets. All further discussion of equity will be in terms of net pot equity.[10]

In general, AIE is an unrealistic measure of true equity because it does not consider the effects of future betting. For example, a ***pure drawing hand*** like **P1: 3♡-2♡ P2: A♠-T♠ Board: K♠-T♡-7♢** A♡, has AIE = 0.2045 (9/44) for P1, but has very favourable implied odds, since it stands to win several bets if a heart lands, and need not lose any additional bets if it does not improve. Conversely, a hand that will be mediocre in most of the outcomes will have reverse implied odds, and the

[9] With the normal (***table stakes***) rules of poker, a player is said to go ***all-in*** when they commit their last chips to the pot. The player cannot be forced to fold, and is eligible to win the portion of the pot they have contributed to if they win the ***showdown***. (If more than one active player remains, betting continues toward a ***side pot***). Hence, from the perspective of the all-in player, there is no future betting, and the AIE is an accurate calculation of their true equity.

[10] Note that measures of equity require the perfect knowledge hindsight view to assess the fraction of future outcomes that are favourable. In contrast, measures of hand strength are with respect to a player's imperfect information view, and are used for realistic assessment, such as establishing baseline betting sequences.

true equity will be less than the AIE metric would indicate. In cases where a player may be forced to fold to a future bet, using AIE can result in a gross over-estimate of hand value.

In contrast, *roll-out equity* (ROE) is based on an explicit enumeration of each future outcome, accounting for the betting in each case according to some standard policy. A simple estimate might assume exactly one bet per round contributed by each player. A better measure would account for folding of the worst hands, and raising of the best hands. The tools we will develop in the following sections establish a more refined betting policy that can be used for all future betting rounds, providing a much more accurate estimate of the true equity of each situation. We refer to the basic implementation as AIE DIVAT, whereas the well-informed roll-outs give rise to the superior ROE DIVAT system.

5.3.2 DIVAT Overview

Poker decisions are extremely context sensitive. Each decision depends on every decision that preceded it, throughout the history of previous games, and especially with respect to the previous actions in the current game. The scope of the DIVAT analysis is a single complete betting round, encompassing all of the player decisions made between chance events. Since it is essential to keep the frame of reference as simple as possible, we simply ignore much of the relevant context of each situation. This is not as serious a limiting factor as it might seem, as will be shown in Section 5.4.

In particular, each player's betting actions (and responses to the opponent) in previous betting rounds are treated as being essentially information free, and no inferences are made with respect to their likely holdings. Since the EHR metric is based on unweighted hand rank calculations, the distribution of opponent hands is implicitly assumed to be uniform. In a sense, each new round is taken as an independent event, with no memory of how the current situation arose, apart from the current size of the pot. However, the implications of bets and raises within a single betting round *are* taken into account. By largely ignoring the history of how each situation arose, we lose some relevant context, and possibly introduce error to the assessment. Fortunately, this does not completely undermine the analysis, because those errors apply to both players in roughly equal proportions, and thus the effects largely cancel out.

The core component of the Ignorant Value Assessment Tool is the DIVAT *baseline betting sequence*. The baseline sequence is determined by *folding policies* and *betting policies*, applied symmetrically to both players. Each of the DIVAT policies is designed to use the simple metrics, while also compensating for some of their liabilities.

To understand DIVAT on an intuitive level, we need to consider the implications of deviations between the baseline sequence and the sequence that actually took place in the game. The baseline sequence represents a sane line of play between two "honest players", free of past context, and devoid of any deceptive plays. If the total number of bets that actually went into the pot was not equal to the baseline amount, then the *net difference in the resulting equities* can be measured.

For example, when Alfred decided to try for a ***check-raise trap*** on the flop, he was taking a calculated risk in the hope of a higher overall payoff. If his deceptive play is successful, then more money will go into the pot than in the baseline, and he is awarded the equity difference on that extra money. If the ploy fails, then he will receive a penalty based on the lost equity (which happens to be the same amount in this case). Thus, his judgement in attempting a non-standard play is immediately endorsed or criticized from the net difference in equities.[11] Although the obvious limitations in measurement and scope can produce a somewhat uninformed view of the ideal play in some cases, the net *difference* in equities is a highly meaningful characteristic most of the time.

Figure 5.1 provides pseudo-code for how the DIVAT Difference is computed. A step-by-step example is provided in Section 5.3.4, using all-in equity. Figure 5.2 gives a recursive definition of the alternative roll-out equity calculation.

[11] Since the net difference is relative, there is actually no distinction between a bonus for one player and a penalty for the other. The assessment is symmetric and zero sum in this regard.

```
/* map each hand strength onto Make level 0-4,
   return kK kBf kBc bF bC bRc bRrC or bRrRc */
Baseline = Derive_Baseline_Sequence (P1HS, P2HS)

/* play forward to the end of the betting round,
   using baseline sequence and actual sequence */

Play_Sequence (Current_State, Baseline)
Baseline_Equity = Rollout_Equity (P1_hand, P2_hand,
                     Board, round, baseline_potsize)

Play_Sequence (Current_State, Actual)
Actual_Equity = Rollout_Equity (P1_hand, P2_hand,
                     Board, round, actual_potsize)

DIVAT_Difference = Actual_Equity - Baseline_Equity
```

Figure 5.1: DIVAT Difference pseudo-code.

5.3.3 DIVAT Details and Parameter Settings

The DIVAT *folding policy* is used to decide when a player should fold to an opponent's bet. The folding policy is based on a *quasi-equilibrium* strategy, motivated by game-theoretic equilibrium strategies. Since the 7cHR metric is in the range zero to one, and is roughly uniformly distributed across that range on the river, the 7cHR is compared directly to the game-theoretic optimal frequency for folding. The game-theoretic equilibrium fold frequency is an invariant that depends only on the size of the bet in relation to the size of the pot. If the bet size is bs and the pot size is ps at the beginning of the river betting round, and the opponent then bets, the optimal fold frequency is $bs/(ps + bs)$. For example, if the pot size on the river is 8 sb and the bet size is 2 sb, the DIVAT policy would fold all hands with 7cHR below $2/(8 + 2) = 0.20$, as an estimate of the bottom 20% of all hands.[12]

Prior to the river round, this simple method is not applicable. First, some weak hands may be worth calling for their ***draw potential***. One can imagine a situation where it is correct to call a bet with *any* hand, because the pot is much larger than the

[12] Folding more often than this would be immediately exploitable by an opponent who always bluffs, showing a net profit overall. Conversely, folding less often than the equilibrium value would be exploitable by betting with slightly weaker hands. Employing the game-theoretic optimal fold policy makes a player indifferent to the opponent's strategy.

```
Rollout_Equity (P1_hand, P2_hand, Board, round, potsize)

/* base case: game ended with a fold */

if (P2 folded) then
   return (net_outcome)
else if (P1 folded) then
   return (-net_outcome)

/* base case: game ended with a showdown */

if (round == river) {
   net = potsize / 2
   if (P1_hand > P2_hand) then
      return (net)
   else if (P1_hand < P2_hand) then
      return (-net)
   else /* tie */
      return (0)
}

/* general case: average future roll-out equity */

sum = 0
cases = 0
round++
for each chance outcome {
   cases++
   Update (Board)
   P1HS = GetHandStrengh (P1_Hand, Board)
   P2HS = GetHandStrengh (P2_Hand, Board)
   Baseline = Derive_Baseline_Sequence (P1HS, P2HS)
   Play_Sequence (Current_State, Baseline)
   value = Rollout_Equity (P1_hand, P2_hand,
               Board, round, baseline_potsize)
   sum += value
}
expected_value = sum / cases
return (expected_value)
```

Figure 5.2: Roll-out Equity (ROE) pseudo-code.

size of the bet (giving extremely high ***pot odds***). Clearly it is not correct in principle to fold at the same rate when there are still cards to be dealt. Secondly, as an average of future outcomes, 7cHR does not have a uniform distribution. Very low values (*e.g.*, below 7cHR = 0.15 on the flop) do not occur, due to the ubiquitous possibility of improvement. Very high values are also under-represented, being vulnerable to a certain fraction of unlucky outcomes. Thirdly, the 7cHR metric is not especially well-balanced with respect to the extra expense of mediocre hands (reverse implied odds), as mentioned previously.

To account for these factors, we add an offset to the game-theoretic folding frequency, and use that as a fold threshold for the EHR metric. Thus, the formula for pre-river fold policies is: Fold Threshold = $bs/(ps + bs) + \mathit{offset}$. The offsets were first estimated on a theoretical basis, then verified and tuned empirically. The empirical data based on millions of simulated outcomes is omitted in the interest of brevity. On the turn, the normal offset is +0.10, so the fold threshold in the previous example would be increased to hands below 7cHR = 0.30. On the flop, the normal offset is +0.075, thus hands would be folded below 7cHR = 0.275 when the initial pot is four bets in size. Prior to the flop, all hands have sufficient draw odds to see the three-card flop, so no offset is required.[13]

The DIVAT *betting policy* is used to decide how many bets and raises a hand is worth. It is strictly a ***bet-for-value*** policy, meaning that all bets and raises are intended to be positive expected value actions, with no deceptive plays (*i.e.*, no bluffing with a weak hand, nor trapping with a strong one). Betting in direct proportion to hand strength is a very poor poker strategy in general, because it conveys far too much reliable information to the opponent. Nevertheless, it serves as a reasonable guideline of how many bets each player should wager in a given situation, all else being equal.

Although bluffing is an absolutely essential component of any equilibrium strategy, the benefits of bluffing are exhibited as a side-effect, increasing the profitability of strong hands. The actual bluffing plays themselves have a net EV of zero against

[13] The pre-flop offset could be negative, but it would make little difference in practice, since almost all cases will proceed to the flop.

an equilibrium strategy (neither gaining nor losing equity). Since the DIVAT fold policies and betting policies are applied in an oblivious manner, the highly predictable behavior of the bet-for-value policy is not being exploited. In effect, the deceptive *information hiding* plays are taken as a given, and all players simply employ the ideal bet-for-value policy without consideration to the opponent's possible counter-strategies. Since the DIVAT policy provides a realistic estimate of how many bets should be invested by each player, and is applied in an unbiased fashion to all players, the weighted gains and losses in hindsight EV are a low-variance estimate of the long-term differentials in decision quality.

As an example, if Player 1 holds a hand of value EHR = 0.70 and Player 2 holds an EHR = 0.90 hand, then the bet-for-value betting sequence for the round would be bet-Raise-call (bRc), indicating that each player will normally invest two bets on that betting round. In the reverse case, with P1 = 0.90 and P2 = 0.70, the expected sequence would be bet-Call (bC), an investment of one bet each, because the second player does not have a strong enough hand for a legitimate raise. This demonstrates a natural asymmetry of the game, and reflects the inherent advantage enjoyed by the player in second position.

The *Make1 threshold* is the strength needed to warrant making the first bet of a betting round. The *Make2 threshold* specifies the strength required to raise after the opponent bets. The *Make3 threshold* is the strength required for a re-raise. The *Make4 threshold* governs re-re-raises (called ***capping*** the betting, because no more raises are allowed).[14] The betting thresholds are derived from the equilibrium points ensuring positive expectation. On the river, the betting thresholds must account for the fact that the opponent will only call with a hand that has some minimum value. A bet cannot earn a profit if the opponent folds, and the hand must win more than half of the time *when it is called* to have positive expectation. For example, if the opponent will call at the game-theoretic optimal frequency, then the range above that point is the relevant interval. A Make1 threshold of 0.64 approximately

[14] Some Limit games permit a fifth level, or even unlimited raises in two-player (***heads-up***) competition. This does not greatly affect the analysis, as situations with more raises are increasingly uncommon, and can be handled if necessary, even for the infinite case.

	Fold Offset	Make1	Make2	Make3	Make4
Pre-flop	0.000	0.580	0.825	0.930	0.965
Flop	0.075	0.580	0.825	0.930	0.965
Turn	0.100	0.580	0.825	0.930	0.965
River	0.000	0.640	0.850	0.940	0.970

Table 5.1: Standard (moderate) DIVAT settings.

corresponds to a policy of betting with the top 41.4% of holdings in that interval.[15] Prior to the river, there is tangible value in forcing the opponent to correctly fold a hand that has a small chance of winning (sometimes called the ***fold equity***). We handle this case by considering the appropriate interval to be the entire window from zero to one, lowering the thresholds by the corresponding ratio. A Make1 threshold of 0.58 approximately corresponds to a policy of betting the top 41.4% of all possible holdings.

The standard betting thresholds for each round of play are listed in Table 5.1.

5.3.4 Basic AIE DIVAT Analysis of the Example Game

We now re-visit the example game presented in section 5.2.1 to illustrate the quantitative DIVAT assessment of all the decisions made by each player. This analysis uses a basic version of DIVAT, based on immediate hand rank, seven-card hand rank, and all-in equity. Although roll-out equity is clearly superior (being better-informed and provably unbiased), the all-in equity is simpler to compute and easier to follow. Table 5.2 presents the pertinent values in the analysis of each round of betting.

In the first round of play, Betty chose to make a deceptive play by raising with a hand that is unlikely to be the best. She may not have done this with the expectation of Alfred folding immediately, but the raise could create a misrepresentation that will have lasting effects much later in the game. Moreover, if she never raised with weaker hands, she would potentially be conveying too much information to her opponent. If Alfred had only called the raise, then there would be no net difference

[15] The zero EV equilibrium points for a 4-bet maximum are derived recursively. The fractions corresponding to the Make1 - Make4 thresholds are the top 12/29, 5/12, 2/5, and 1/2 of the interval, respectively. For unlimited raises, the zero EV equilibrium point is the top $\sqrt{2} - 1$ at all levels. The derivations are omitted for brevity.

Board: <none>			
	IHR	7cHR	EHR
P1: A♣-K♣	0.9376	0.6704	0.9376
P2: 7♡–6♡	0.1604	0.4537	0.4537
AIE = 0.6036 (+1029832 =7525 -674947)			
LFAT change = +0.2073			
DIVAT Baseline = SlCrC	Actual Seq = SlRrC		
Round EV = +0.2073 [SlCrC]			
Actual Equity = +0.6218 [SlRrC]			
Baseline Equity = +0.4145 [SlCrC]			
DIVAT Difference = +0.2073 sb			

Pre-flop Analysis

Board: <K♠-5♡-3♢>			
	IHR	7cHR	EHR
P1: A♣-K♣	0.9685	0.8687	0.9685
P2: 7♡–6♡	0.0634	0.3798	0.2216
AIE = 0.7636 (+756 =0 -234 of 990)			
LFAT change = +0.9600			
DIVAT Baseline = bC	Actual Seq = kBrC		
Round EV = +0.5273 [bC]			
Actual Equity = +2.6364 [kBrC]			
Baseline Equity = +2.1091 [bC]			
DIVAT Difference = +0.5273 sb			

Flop Analysis

Board: <K♠-5♡-3♢ T♣ >			
	IHR	7cHR	EHR
P1: A♣-K♣	0.9411	0.8902	0.9411
P2: 7♡–6♡	0.0662	0.2146	0.2146
AIE = 0.9091 (+40 =0 -4 of 44)			
LFAT change = +1.4545			
DIVAT Baseline = bF	Actual Seq = bC		
Round EV = 0.0000 [bF]			
Fold Equity = 0.9091 [bF]			
Actual Equity = +5.7273 [bC]			
Baseline Equity = +5.0000 [bF]			
DIVAT Difference = +0.7273 sb			

Turn Analysis

Board: <K♠-5♡-3♢ T♣ 4♡ >			
	IHR	7cHR	EHR
P1: A♣-K♣	0.8576	0.8576	0.8576
P2: 7♡–6♡	0.9955	0.9955	0.9955
AIE = 0.0000 (+0 =0 -1 of 1)			
LFAT change = -12.7273			
DIVAT Baseline = bRc	Actual Seq = bRc		
Round EV = -4.0000 [bRc]			
Actual Equity = -11.0000 [bRc]			
Baseline Equity = -11.0000 [bRc]			
DIVAT Difference = +0.0000 sb			

River Analysis

Table 5.2: Round-by-round analysis of the example game.

in equity compared to the DIVAT baseline, since the same amount of money would be invested by the two players. In the actual game sequence, Alfred correctly re-raised, and is thus awarded a skill credit for the round, at the expense of Betty. Since Alfred's hand will win about 60% of the future outcomes, he earns a small fraction of the two extra bets that went into the pot (Net = 0.60 * 2 - 1 = +0.20 sb).[16]

The flop was favourable for Alfred, giving him a strong hand, and increasing his chance of winning a showdown to 76%. He tried for a ***check-raise trap***, which was successful, netting him another 0.53 sb in equity. If Betty had simply checked, she would have earned that amount instead of Alfred, and his gamble would have backfired.

[16] In reality, the equity difference is not that large, because Betty will usually not continue with the hand when it does not connect with the flop, but stands to win several extra bets in the future if the flop is favourable. These positive implied odds are reflected in the roll-out equity, which indicates that Alfred's advantage in this situation is actually much smaller, as shown in Table 5.3.

The turn brings another good card for Alfred, increasing his expected share of the pot to 91%. Since his check-raise on the flop may have revealed the true strength of his hand, he opts for the straightforward play of betting. Now Betty is faced with a difficult decision between calling with a weak hand, or folding when the pot is relatively large. Betty might have supposed that there was still a chance that Alfred had no pair, and that she could perhaps win if a **6** or **7** landed. The necessity of making informed guesses is the very essence of poker. In this particular case, that belief would be wrong, and folding would have had a higher expected return for Betty. The magnitude of the resulting misplay is accurately reflected by the DIVAT Difference for the round. [17] When a fold is involved, the DIVAT Difference is not the same as the baseline EV for the round. Here Betty loses a net of 1.64 sb from the actions of the round (bet-Call), but calling only loses 0.73 sb more than folding. The difference is due to the ***fold equity*** that Betty refused to abandon (1/11 of the pot prior to the betting), which gives her partial compensation for the calling error.

The perfect **4**♡ for Betty on the river erases all of the good luck Alfred enjoyed up to that point. The LFAT swing indicates that the card cost him 12.73 sb, based solely on his pot equity after the turn betting. However, the damage is actually much greater, because he naturally continues to bet his strong hand and walks into a raise, losing an additional 4 sb.

Here we witness the dramatic difference between the perfect knowledge hindsight view of EVAT, and the more realistic DIVAT view. The EVAT baseline sequence for the final round of betting would be check-Bet-fold, which is quite absurd with respect to the imperfect information reality. The DIVAT baseline of bet-Raise-call is much more reflective of real play. Since Alfred did not compound the problem with a re-raise, he loses only the expected amount from the river betting, and thus does not receive any penalty. If he had chosen to play check-Bet-call (perhaps based on a telling mannerism by Betty), he would in fact *gain* in the view of DIVAT, because he lost less than was expected.

Not only is the DIVAT baseline more reasonable, it is also clear that the EVAT

[17] We again see the effects of favourable implied odds when a **4** lands, which out-weigh the ***reverse implied odds*** when a **6** or **7** lands. The slightly better prospects after the calling error are captured by the roll-out equity analysis.

A♣-K♣ 7♡-6♡ K♠-5♡-3◇ T♣ 4♡
Bet sequence: SlRrC/kBrC/bC/bRc

Round	LFAT	RndEV	DIVAT
Pre-flop	+0.207	+0.207	+0.207
Flop	+0.960	+0.527	+0.527
Turn	+1.455	+0.909	+0.727
River	-12.727	-4.000	0.000

Total DIVAT Difference = +1.462 sb

Example Game AIE DIVAT summary

A♣-K♣ 7♡-6♡ K♠-5♡-3◇ T♣ 4♡
Bet sequence: SlRrC/kBrC/bC/bRc

Round	LFAT	RndEV	DIVAT
Pre-flop	+0.207	+0.029	+0.029
Flop	+0.960	+0.523	+0.523
Turn	+1.455	+0.909	+0.636
River	-12.727	-4.000	0.000

Total DIVAT Difference = +1.189 sb

Example Game ROE DIVAT summary

Table 5.3: Full-game analysis of the example game (AIE and ROE).

analysis is statistically biased, because it is preferential toward one particular *style* of play over another, apart from EV considerations. As we have seen, it is simply impossible to play the river betting round without frequent misplays, with respect to the EVAT perfect information baseline. This means that a player who employs a conservative style (folding in neutral or marginal EV situations) will be viewed more favourably than a player with a liberal style (calling in those same situations), simply because the conservative player gets to the river round with a marginal hand less often. In fact, if one wanted to maximize the EVAT opinion of one's play, it would be wise to sacrifice slightly positive expectation situations on the flop and turn, simply to avoid being measured against an impossible standard on the river. The irrational EVAT baseline has the same undesirable effect on earlier rounds, but it is most pronounced on the river, where the effect is not dampened by averaging over future outcomes. In contrast, the DIVAT baseline provides one standard way to play the hand for *all* cases, without regard to the opponent's actual (hidden) holding.

The round-by-round LFAT transitions cannot easily be combined to determine a net effect on the game as a whole. They cannot simply be summed, since instances of early good luck can be wiped out by a final reversal, as in the example game. The LFAT value can, however, provide some relevant context when interpreting the round-by-round analyses.

The DIVAT analyses of each round are treated as being essentially independent of each other, and can be summed to give the net difference for the complete game. Indeed, the sum is generally more meaningful than the individual rounds in isolation, because a player may make a deliberate misplay on an early betting round in order to induce larger errors on the turn or river. For example, a player may ***slow-play*** by only calling with a strong hand on the flop, intending to raise on the turn when the bet size is larger. Conversely, a player may ***raise for a free-card*** with a drawing hand, intending to check on the turn after the opponent checks. All else being equal, the success or failure of the tactic will be reflected in the total DIVAT Difference for the game.

Table 5.3 gives a summary for the complete example game. The corresponding summary for the better-informed and theoretically sound ROE DIVAT is given for direct comparison. The ROE DIVAT procedure has been formally proven to be statistically unbiased by Zinkevich et al. [16].

Overall, we can conclude that Alfred did better than expected with the cards and situations that arose in this game. In hindsight, he made better choices than Betty, worth a total of about +1.46 small bets in terms of long-term expected value. The fact that Alfred actually lost 11 sb on the hand is not particularly relevant, and only serves to occlude the most important considerations.

Of course, this is only a single game, and as previously noted, Alfred's hand was somewhat easier to play well than Betty's. The non-independence of consecutive games should also be kept in mind. In the wider scope, Betty's play might have been entirely appropriate, and might earn some compensation in future games. Long-term plans taking effect over many games are also captured by the running total of DIVAT Differences over the course of a match. Given many examples over a series of games, the accumulated DIVAT Differences provide a much more accurate, low-variance estimate of the eventual win rate for one player over another.

5.3.5 Smoothing and Using Equilibrium Baselines

Further refinement to DIVAT can be obtained by averaging the results of several runs, using different parameter settings for each. For this purpose, we defined nine sets of folding parameters that reflect styles of play ranging from very loose to very tight; and defined nine sets of betting parameters that reflect styles of play ranging from very aggressive to very conservative. In our experiments, we normally run a sweep of five settings from loose/aggressive to tight/conservative.[18]

This provides a natural *smoothing* of the DIVAT Differences over the short term. For example, a situation might arise that is close to the borderline between bet-Call and bet-Raise-call. Over the five separate runs, the baseline might be bet-Call four times and bet-Raise-call once. Thus, instead of a single threshold with a jump from 2.0 bets to 4.0 bets, we have a smoother transition with five smaller steps, yielding a jump of 2.4 bets on average.[19]

In effect, the average of several runs is a hedge, being less committal in situations where the best line of play is more debatable. This can be especially useful over the range of fold thresholds, because the difference in EV between folding and continuing with the hand can be very large (since the whole pot is at stake, rather than fractions of a bet). However, over the course of many games the average of several runs will simply converge on the DIVAT Difference line for the median (moderate style) parameter settings. Thus, smoothing is primarily useful for short matches.

In imperfect information games where an equilibrium solution is known (or can be accurately estimated), the ideas behind DIVAT can be used in a manner analogous to smoothing. For example, in the case of poker, suppose we have a weak hand on the final round of betting, which would be checked in the DIVAT baseline. If we know that the bluff frequency in this situation should be 10%, then the corresponding baselines for betting would be computed, and would contribute 10% of the total weight for actions made with a weak hand. The pertinent regions

[18] Although the fold parameters and betting parameters could be varied independently, the number of runs would grow multiplicatively, without much benefit. For the same reason, we keep the parameter settings for each of the four betting rounds consistent, and apply the same settings for the actions of both players, rather than mixing them. The intention is to aim for a range of distinct perspectives over a span of reasonable styles, rather than an exhaustive enumeration of many possibilities.

[19] The weight of each run need not be equal. Since the median value is deemed to be the most reasonable, the weights for five runs could be 1-2-3-2-1 instead of 1-1-1-1-1; or could follow the binomial distribution 1-4-6-4-1 to reflect a Gaussian dispersion of styles over the specified range.

of the strategy space are not continuous (unlike smoothing, which was done over a continuous range of values), but that is of no consequence for determining the average EV outcome for a given situation. Thus, knowing an equilibrium strategy provides us with an appropriate way to compute a weighted average of possible actions (or action sequences).

We have not attempted to use a full quasi-equilibrium strategy to obtain a collection of DIVAT baselines to be averaged. However, it remains an interesting possibility for future work.

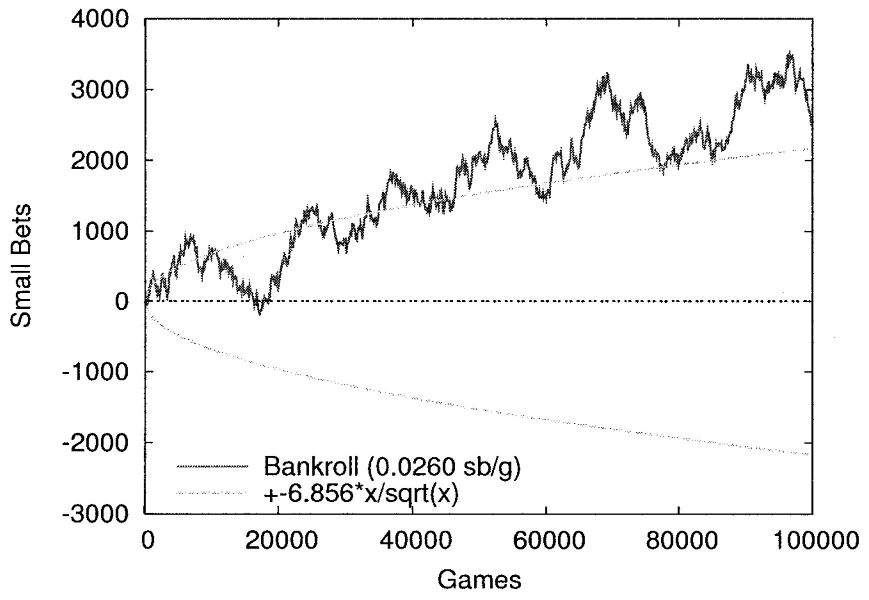
5.4 Experiments

The Ignorant Value Assessment Tool is based on relatively simple methods for hand strength measurement, betting policies, and folding policies. During development, each method was empirically tested and refined iteratively, since the components are mutually dependent to some degree.

For example, the folding policy was tuned to be consistent with the strengths and liabilities of the simplistic IHR and 7cHR hand strength metrics. In the case of 7cHR, it was observed that reverse implied odds were not being given enough consideration, resulting in folding policies that were too liberal when cards were still to be dealt. The correction was to add an offset to the game-theoretic threshold value.[20] The value of this offset was determined empirically using roll-out simulations to find the best practical setting for achieving a neutral outcome, thereby estimating the game-theoretic equilibrium point. The experimentally derived offset was consistent with the predicted offset based on expert calculations and intuition.

Each series of experiments was repeated on at least seven different matches, involving a wide range of playing styles and skill levels. This is necessary because the conditions of the match dictate the frequency and type of situations that will arise. The computer players ranged from simple rule-based algorithms to the strongest known game-theoretic and adaptive programs. The human players ranged from intermediate strength to world-class experts.

[20] Actually, the linear adjustment involved both a multiplier and an offset ($y = mx + b$), but a multiplier of 1.0 was found to be sufficient, leaving only the constant offset as a correction factor.



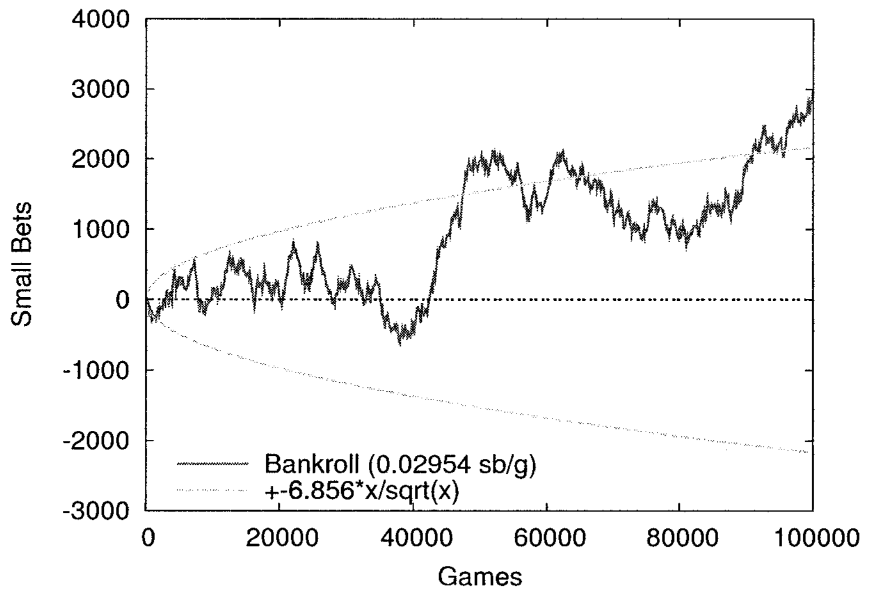


Figure 5.3: High variance in two ALWAYS_CALL *vs* ALWAYS_RAISE matches.

To make the presentation of experimental results easier to follow, we will limit ourselves to only five types of matches. The first type is between two extremely simple players: ALWAYS_CALL and ALWAYS_RAISE. Although both algorithms are extremely weak as poker players, there are several advantages to studying this contest, because it holds many variables constant. The exact betting sequence is always known: if ALWAYS_RAISE is the first player, the betting sequence is bet-Call (bC) for every round; if ALWAYS_CALL is the first player, the betting sequence is check-Bet-call (kBc) for every round. There is never a fold, so every betting round is involved in every game. The final pot size is always 14 sb, and the net outcome for one player is either +7 sb, -7 sb, or zero (a tie will occur approximately 4.06% of the time). The long-term expected value for each player is exactly zero, since neither player exhibits superior skill. The outcome of each game is similar to coin-flipping, with occasional ties. The natural variance can easily be determined for these conditions. Simulation experiments were run for 25 million games, showing a variance of 47.010, for a standard deviation of $\pm$ 6.856 small bets per game (sb/g). The empirical variance for each 100,000-game match agreed, also showing $\pm$ 6.856 sb/g.

This gives us a frame of reference for the natural fluctuations on the amount won or lost during the match (the "money line", labeled as "Bankroll"). The $\pm$ 6.856*x/sqrt(x) guide-curves indicate the one standard deviation boundary on the total outcome after x games. The 95% confidence interval would correspond to roughly two standard deviations. The variance in real matches strongly depends on the styles of the two players involved, but $\pm$ 6 sb/g is typical.[21] When complete data is available, we use the actual measured variance during the match for more accurate guide-curves.

Figure 5.3 shows two separate matches of 100,000 games between ALWAYS_CALL and ALWAYS_RAISE. The money line in these two matches nicely illustrates just how pernicious the problem of variance can be. In the first match, there is every appearance that one player is outplaying the other, by an average of more than +0.025

[21] Many games will end early when one player folds, but those frequent small net outcomes (n) are offset by the occasional large outcomes, which carry greater weight toward total (n^2) variance.

small bets per game (sb/g). Although there are some fluctuations along the way, they are small enough to give the impression of a steady increase. This is extremely misleading, since we know that the long-term average is exactly zero. With the aid of the guide-curves, we can see that the money line strays outside the boundary of one standard deviation, but is still easily within the realm of possibility. In other words, this is not a severely abnormal outcome, and it is not statistically significant at the 95% confidence interval.

In the second match (generated with an independent random number generator, due to our own suspicions), we again see a high-variance event with a potentially misleading conclusion.[22] Another feature to notice about this second match is the dramatic rise from about -650 sb at 38,000 games, to almost +2000 sb by 48,000 games. During that 10,000-game span (which is much longer than most competitive matches) one player demonstrates an enormous win rate of +0.25 sb/g that is entirely due to random noise.

The second type of match we will examine is a self-play match, with the game-theoretic equilibrium-based program PsOpti4 playing both sides. This player is *stationary* (*i.e.*, uses a strategy that is randomized but static, is oblivious to the opponent, and does no learning or adaptation), so clearly the long-term expectation is exactly zero. However, unlike the Always_Call *vs* Always_Raise match, the play is realistic – the player folds bad hands, raises good hands, bluffs, and attempts to trap the opponent.

The third match type is between PsOpti4 and PsOpti6. The newer PsOpti6 plays a substantially different style, but loses to the older version in head-to-head competition.[23] Both players are static, so learning effects are eliminated. Since we want to compare the DIVAT estimate to the long-term EV, this match and the self-play match were extended to 400,000 games, concatenating the first 100,000-

[22] These matches were not selected after the fact – they were the first two matches generated.

[23] Note that this does **not** mean that PsOpti4 is a better player than PsOpti6. The result of any one match-up is not necessarily indicative of overall ability. By analogy, consider a pseudo-optimal Rock-Paper-Scissors player that chooses Rock 30% of the time, Paper 40%, and Scissors 30%. The maximum exploitation best response (+0.10 units per game) is achieved by Always_Scissors, but that is one of the worst possible strategies in general. Poker results are highly non-transitive (*e.g.*, A beats B beats C beats A) in practice. Nor is there a simple cycle – it is a complex *wreath* of dominance relationships between styles.

game series above with three new 100,000-game series (each generated with a different random seed). The same series of cards was then used to play a *duplicate match*, with the players occupying the reverse seats (note that the self-play match is self-duplicate). Thus, 800,000 distinct games were played between PsOpti4 and PsOpti6, with each 400,000-game match being directly comparable to the other. The hand-by-hand results were averaged to obtain the *Duplicate Money Average* and the *Duplicate DIVAT Average* lower-variance estimators, for comparison with the single-sided DIVAT.

The fourth type of match we examine here is a man versus machine match between world-class expert "thecount" (Gautam Rao), and the first game-theoretic equilibrium-based program PsOpti1. This match was featured in our 2003 paper [3]. The 7030-game contest again featured large swings, which we now know were almost entirely attributable to stochastic noise, with each player having alternating phases of extreme luck.

A match involving real players is significantly different from a match played under the sanitized conditions of the laboratory. Here we witness a clash between radically different approaches to the game. Top-flight experts change their style rapidly and frequently, as the game conditions dictate. The human player avoids being predictable, explores new tactics, and learns over time how best to exploit the opponent. In this match, "thecount" started with a hyper-aggressive style that is highly effective against most human opponents, but was counter-indicated against the program. After shifting to a more patient approach, he enjoyed more success. Much of that was due to fortuitous cards and situations that coincidentally occurred at around the same time; but the DIVAT analysis is able to extract signal from the wash of noise, revealing the overall *learning curve* and the positive impact of that major shift in style.

The final match we will examine is between PsOpti4 and the adaptive program Vexbot. Vexbot is the strongest poker program to date, having defeated every computer opponent it has faced, and often provides a serious threat to top-flight human players [2, 5]. However, the learning systems embedded in the Vexbot architecture are slow and imperfect. They often require many thousands of games to train, and frequently lead to local minima that are well below the maximum exploitation level of the given opponent. The DIVAT analysis reveals much more about the changes of VEXBOT over time, and again shows how misleading the basic money line can be.

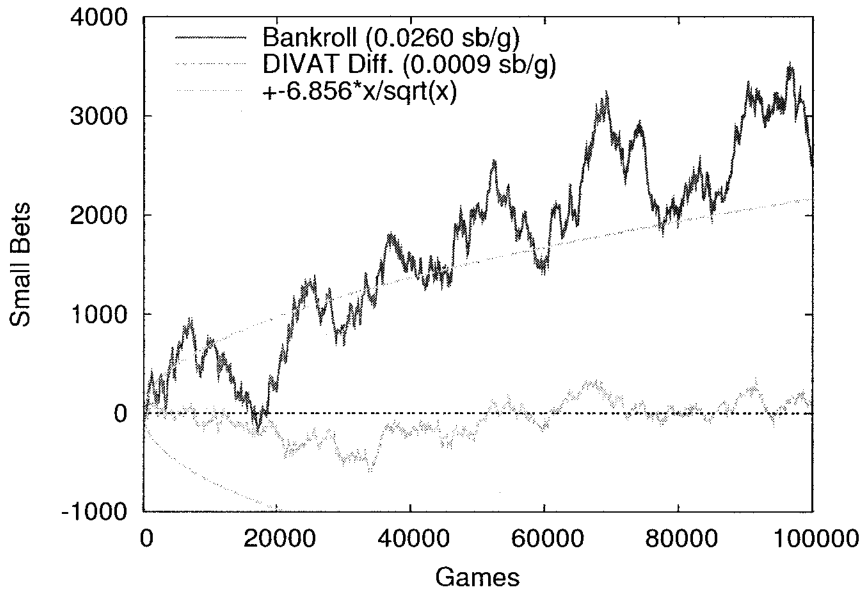
The experiments shown in this section address: (1) the correctness and variance-reduction properties of DIVAT, (2) the ability to reveal learning curves and changes in style for *non-stationary* players, (3) the robustness of the DIVAT Difference line, and (4) the usefulness of the round-by-round analysis to gain extra insights into match results. Many other experiments were conducted during the development of the DIVAT system. In the interest of brevity and focus, we do not show experimental results pertaining to: (1) system components and parameter tuning (other than robustness), (2) the significant statistical bias of the (perfect information) EVAT view, (3) other reduced-variance estimators arising out of the DIVAT analysis (such as *Money minus Dcash*), (4) asymmetric assignments of credit and blame to each player (such as *RedGreen points*), (5) comparison of ROE and AIE formulations of DIVAT (ROE DIVAT is used throughout), and (6) *smoothing* by averaging over a range of parameter settings. Some of these experiments may be addressed in future updates of the full technical report [6], and the M.Sc. thesis of Morgan Kan [9].

5.4.1 Correctness and Variance Reduction

We now present a series of experiments to verify empirically that the DIVAT Difference is an unbiased estimate of the long-term expected value between two players. We will measure the overall reduction in variance in each case.

5.4.1.1 ALWAYS_CALL versus ALWAYS_RAISE Match

Figure 5.4 shows the ROE DIVAT Difference line for the two 100,000-game matches between ALWAYS_CALL and ALWAYS_RAISE. In each case, the DIVAT Difference line hovers near the zero line, showing much better accuracy than the money line, and showing no apparent bias in favour of one player over the other. The standard deviation for the money line is ± 6.856 sb/g in each case (as expected), whereas the measured standard deviation for the DIVAT Difference lines are ± 2.934 sb/g and



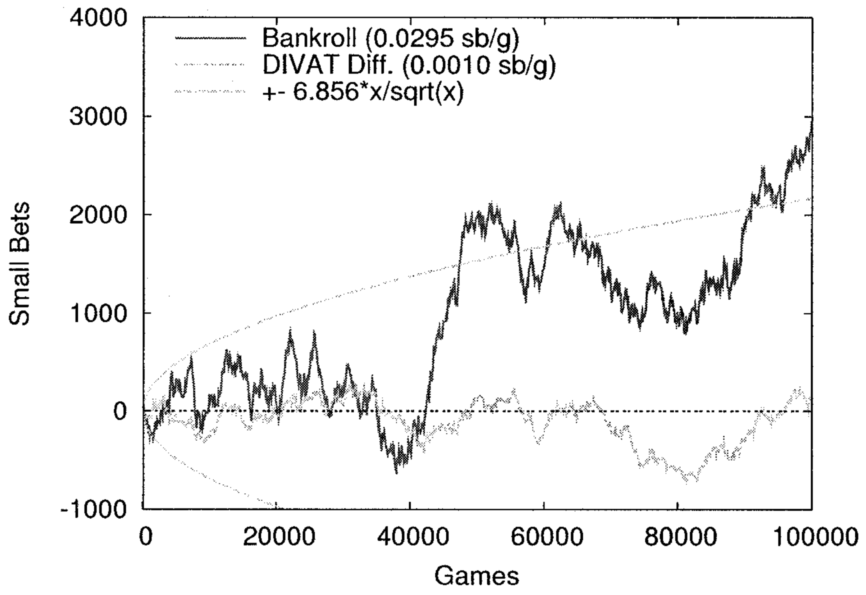


Figure 5.4: DIVAT Difference analysis of two ALWAYS_CALL *vs* ALWAYS_RAISE matches.

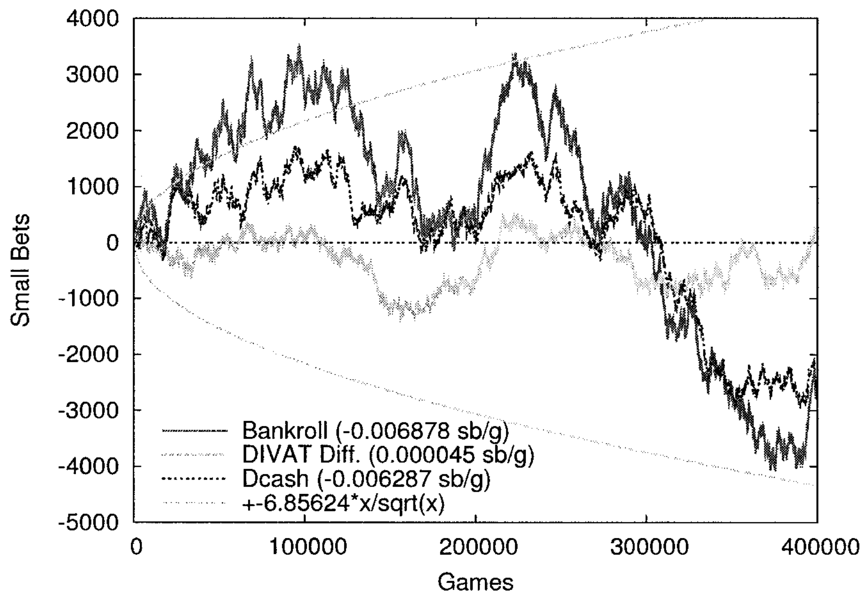


Figure 5.5: DIVAT Difference analysis and DIVAT Cash baseline for an extended ALWAYS_CALL *vs* ALWAYS_RAISE match.

± 2.917 sb/g, respectively, for an overall reduction in variance by a factor of 5.50.

Figure 5.5 shows the results after extending the first match for an additional 300,000 games. Following the steady climb over the first 100,000 games, the money line displays a natural regression toward the mean, but later shows a relentless down-swing, reaching about -1.4 standard deviations. This graphically demonstrates that the full statistical interval is indeed used over the course of normal stochastic events.

There appears to be a positive correlation between the money line and the DIVAT Difference line (particularly over the last half of the second 100,000-game match, and in the middle regions of the 400,000-game series). This is expected, because the two measures are not completely independent. In general, strong hands are easier to play correctly than weak hands, because the EV consequences of an incorrect raise (or lack thereof) are generally smaller than the EV consequences of an incorrect fold (or lack thereof).

In this match, ALWAYS_RAISE makes frequent errors by betting instead of checking with weak hands. The ALWAYS_CALL player makes frequent errors by checking instead of betting strong hands (in first position), but those errors are effectively forgiven when the opponent bets. The ALWAYS_CALL errors of calling instead of raising with especially strong hands are distinct, as are the large EV errors of calling instead of folding with very weak hands. We know that the weighted sum of all misplays will exactly balance out, because the overall EV difference between these two players is zero.

Also in Figure 5.5, we show the DIVAT Cash (Dcash) baseline, which is the amount that would be won or lost if both players followed the DIVAT baseline policy for the entire game. Since Dcash is a reasonable estimate of the normal outcomes, we can see that *Money minus Dcash* would be a decent lower-variance estimator of the difference in skill, and is certainly much better than the money line alone. However, this estimate has a higher variance than the DIVAT Difference, is more strongly dependent on the stochastic luck of the card outcomes, and can only be applied to complete games. The Dcash line is also biased in favour of styles that are similar to the (overly honest) DIVAT baseline, whereas the ROE DIVAT Difference is a provably unbiased estimator.

The identical sequence of 400,000 deals was used for subsequent experiments, to eliminate sampling differences. We include the Dcash line in those graphs to provide a common frame of reference.

5.4.1.2 PSOPTI4 Self-play Match

Figure 5.6 shows the results for the PSOPTI4 self-play match. Here the Bankroll line represents the actual amount of money won or lost (in sb) when this particular player plays both sides of each hand. We can observe that the self-play money line is much closer to the Dcash line than it was for the essentially random ALWAYS_CALL *vs* ALWAYS_RAISE match, because the play is much more realistic. Nevertheless, the final Bankroll difference is about 35% greater in magnitude. This is also to be expected, because the baseline reflects a highly conservative "boring" style of play. PSOPTI4 has a relatively low-variance style itself, as evidenced by the measured standard deviation of "only" ± 4.691 sb/g. The measured standard deviation of the *Money minus Dcash* estimator is ± 2.825 sb/g in this match. The measured standard

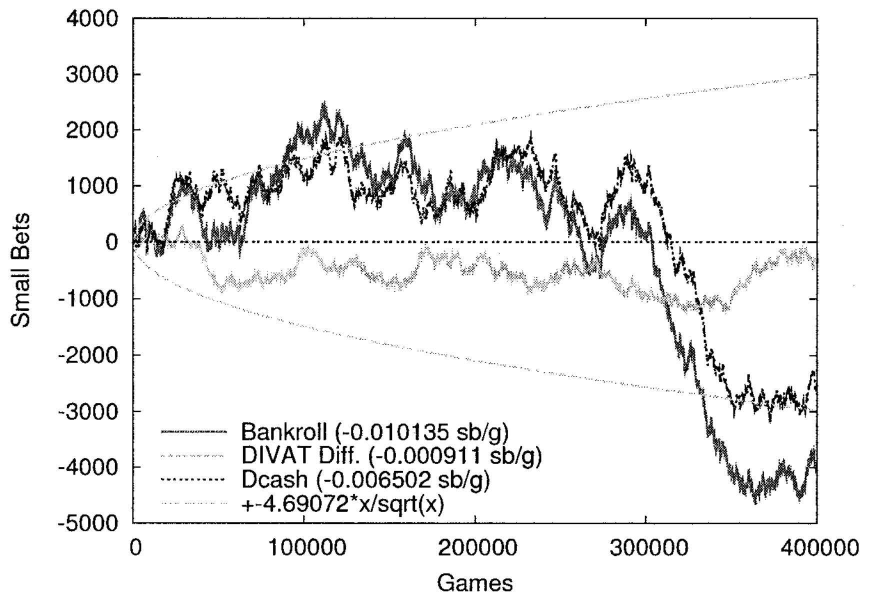
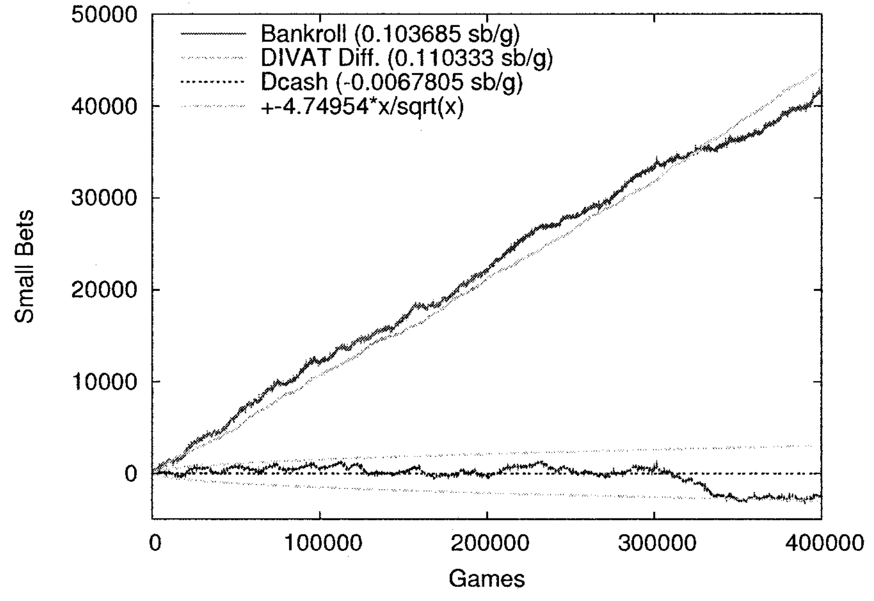


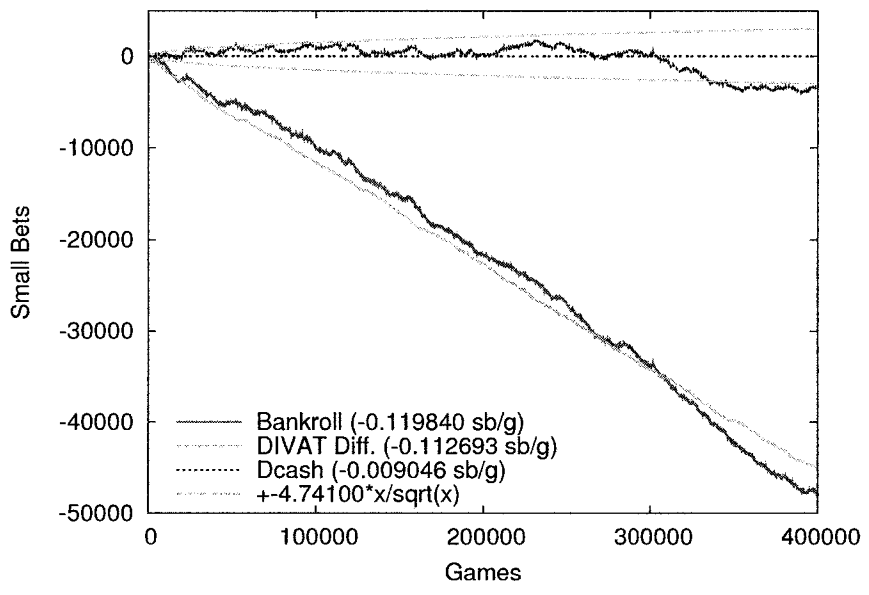
Figure 5.6: DIVAT Difference analysis of PsOpti4 self-play match.

deviation of the DIVAT Difference is ± 1.863 sb/g, which is a 6.34-fold reduction in variance from the self-play money line.[24]

In practical terms, this means that routine 40,000-game matches could be replaced with 6000-game matches having comparable variance. More to the point, the same 40,000-game matches can discern between two players who are nearly three times closer to each other in skill level, instead of requiring matches of more than a quarter million games. With 95% confidence, a 40,000-game match distinguishes differences of roughly 0.06 sb/g, which represents a substantial difference in skill (such as a professional player against a mediocre player). Being able to discern a difference of 0.02 sb/g (such as the difference between a good player and a very good player) is much more useful in practice.



(a) PsOpti4 (North) *vs* PsOpti6 (South)



(b) PsOpti6 (North) *vs* PsOpti4 (South)

Figure 5.7: DIVAT Difference analysis of PsOpti4 *vs* PsOpti6 duplicate match.

5.4.1.3 PsOpti4 versus PsOpti6 Duplicate Match

Figure 5.7 shows the results for the duplicate match between PsOpti4 and PsOpti6 over the same 400,000 deals. All graphs are shown from the perspective of the player in the North seat,[25] so PsOpti4 wins both sides of this duel by a convincing margin of +0.112 sb/g. In both passes, the money line dips below the DIVAT Difference line at about the 300,000-game mark, because of the huge swing of luck in South's favour at that stage in the match.

The Bankroll difference is not a very accurate measure of skill difference even after 400,000 games. On the North side of the cards PsOpti4 wins at +0.104 sb/g, compared to +0.120 on the (stronger) South side; whereas the DIVAT estimate is within $\pm$ 0.0012 sb/g in either case. Using the Dcash line as a correction of the money line would considerably improve the accuracy. However, the DIVAT Difference is strictly more powerful for predicting future outcomes.

The measured standard deviations for Bankroll are $\pm$ 4.750 and $\pm$ 4.747 sb/g respectively. For *Money minus Dcash* they are $\pm$ 2.948 and $\pm$ 2.945 sb/g. For the DIVAT Difference, they are $\pm$ 1.929 and $\pm$ 1.932 sb/g respectively, for an overall reduction in variance by a factor of 6.05.

Note that the measurements for the two halves of this duplicate match are very consistent, because 400,000 games is sufficiently large for the number and variety of opportunities to be reasonably well balanced. Over a shorter duration, one side could enjoy more good opportunities, while facing relatively few difficult situations.

Figure 5.8 shows the results after combining the outcomes of each North and South pair of duplicate games. Both the Duplicate Money Average and the Duplicate DIVAT Average are good low-variance predictors of future outcomes, and match each other almost exactly over the long term.[26] The measured standard de-

[24] The DIVAT Difference line does not cling tightly to the zero line in this graph, but the meaningful indicator of EV correctness and low variance is simply the levelness (*i.e.*, horizontal flatness) of the line. Once it drifts away from zero, it should persist at the same constant difference. Obviously the DIVAT Difference line is much flatter than the money line overall.

[25] The North seat still alternates between first and second position (large and small blind). By convention, the first named player is North, having the big blind on all odd numbered hands and the small blind on all even numbered hands.

[26] The duplicate Dcash line shown in the figure is not exactly zero because of the way it is computed. In cases where a player folds, the roll-out equity is computed by applying the DIVAT

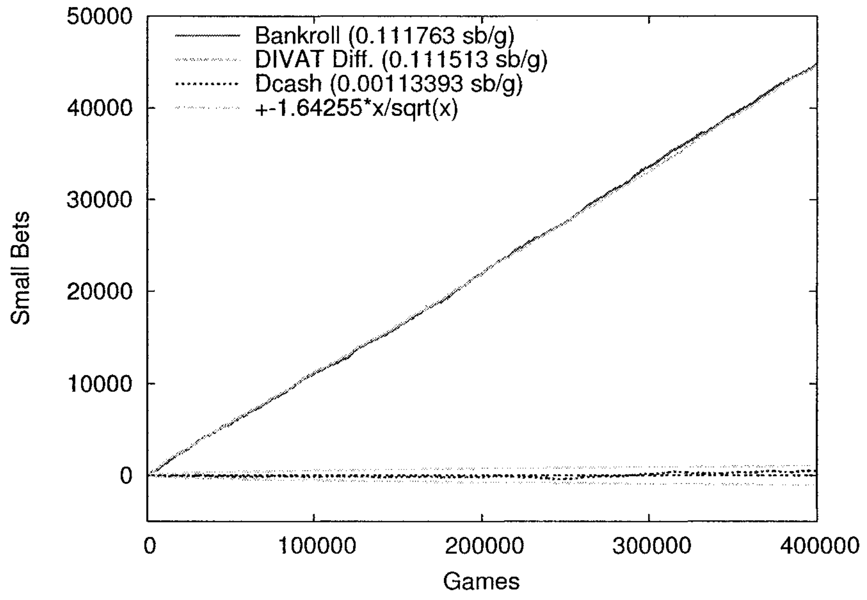


Figure 5.8: Duplicate Money Average and Duplicate DIVAT Average.

viation for Duplicate Money Average is ± 1.643, for a reduction in the normal variance between these two (fairly conservative) players by a factor of 8.36. The measured standard deviation for Duplicate DIVAT Average is ± 1.178, for a 16.25-fold reduction in variance, making it the lowest variance estimator we currently have.

Figure 5.9 zooms in on the first 1000 games of the pairwise duplicate games in Figure 5.8. Here, we can see the limits of the discrimination power for each metric. The guide-curves show the one standard deviation bound for the Duplicate Money Average (at ± 1.643), and the narrower bound for the Duplicate DIVAT Average (at ± 1.178).

After 200 games, neither technique has detected a difference in skill between the players, with a net difference close to zero. It is fair to say that PsOpti6 had some "luck", in that the most telling skill differences were not yet exposed during that phase.

After 300 games, the Duplicate DIVAT Average is beginning to favour PsOpti4

baseline to all future chance outcomes, and taking the average. Thus, if one player folds while the other continues with the hand, their Dcash lines will usually be slightly different.

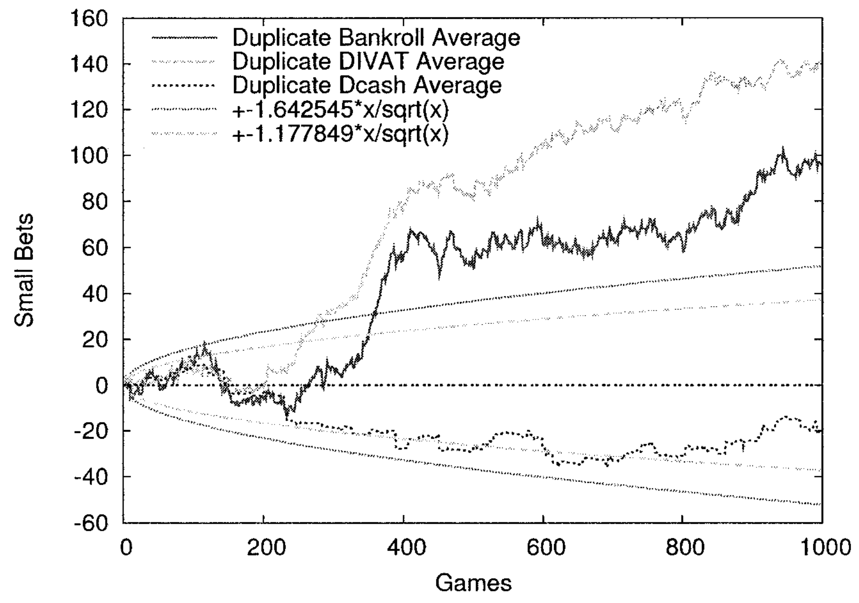


Figure 5.9: Duplicate Money Average and Duplicate DIVAT Average over the first 1000 games.

(about +1.5 standard deviations), whereas the Duplicate Money Average is still inconclusive. There is a sharp increase in both metrics between 300 and 400 games, where PsOpti4 is able to exhibit its superior skill quite a bit faster than usual. The non-zero duplicate Dcash line gives us a hint that some of that difference may be due to the folding behaviors of the two players (either PsOpti6 folds too easily to bluffs, or does not fold correctly when it is behind).

After 400 games, the Duplicate DIVAT Average is at about +3.4 standard deviations, and would conclude with more than 99% confidence that PsOpti4 is the better player heads-up against PsOpti6. The Duplicate Money Average cannot make the same conclusion even at the 95% confidence level. The same statements are still true after 1000 games.

This example happened to be quite favourable to the DIVAT method. In general, to reach the 95% confidence interval for this highly unbalanced (+0.112 sb/g) contest, the Duplicate Money Average would require about 870 games, while the Duplicate DIVAT Average would need about 450 games. For a live match, the normal single-sided DIVAT Difference would require about 1200 games, whereas the simple Bankroll difference would need about 7260 games on average.

A natural question is whether the two approaches are somewhat orthogonal to each other, and could be combined for an even sharper resolution. Unfortunately, we can easily deduce that there must be a substantial amount of overlap between the two techniques. As mentioned previously, the single-sided DIVAT baseline is especially useful for discerning the "normal" outcome when a strong second-best hand loses many bets to an even stronger hand. The Duplicate Money Average achieves a similar neutralization, because both players are given the opportunity to play the weak side and strong side of that situation.

Moreover, if one player gained on the DIVAT scale by losing less on the weaker side, that superior skill could also be reflected in the Duplicate Money Average by showing a net profit over the pair of duplicate games. In the case of a successful or unsuccessful bluff, we can see that the player is given full credit or full blame with either the DIVAT measure or the duplicate result.

For a pair of duplicate games, noise is introduced into the result after the actions of the two players diverge. In particular, when one player folds while the other continues with the hand, all chance outcomes from that point forward are subject to the usual effects of stochastic noise.[27] Directly measuring the loss or gain of equity from each decision yields a more stable estimate.

5.4.2 Learning Curves for Non-stationary Players

We now show how the DIVAT analysis can provide powerful insights into the changing behavior of non-stationary players, which includes virtually all strong human players, and the most advanced programs.

5.4.2.1 The 2003 "thecount" versus PsOpti1 Match

Figure 5.10 compares the money line to the ROE DIVAT Difference line for the 2003 match between "thecount" and PsOpti1.[28] We observe a similar improve-

[27] We see a similar source of variance in the *Money minus Dcash* estimator, whenever the assumed fold policy differs from actual events.

[28] Note that a complete-knowledge log is required for this analysis, including all cards folded by either player. The match was played on our online poker server, which maintains complete-knowledge logs of all games played.

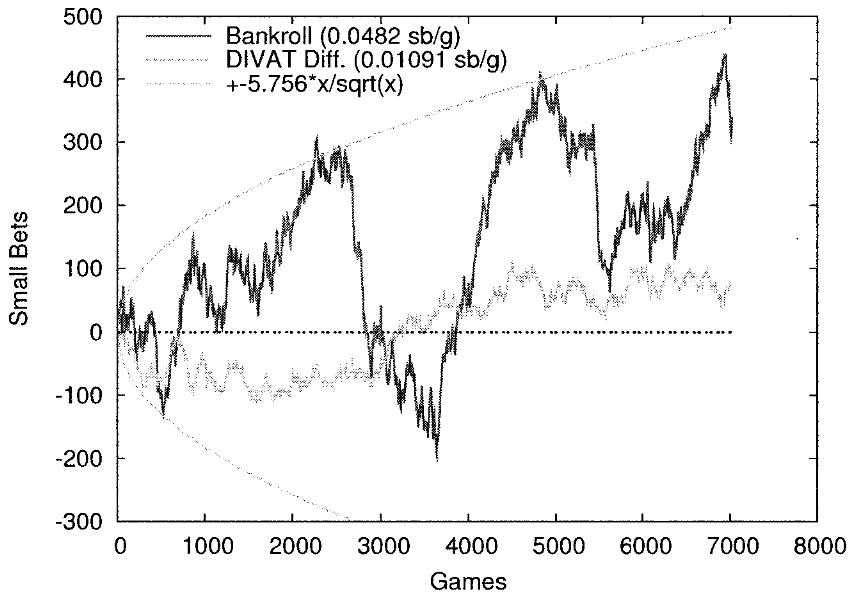


Figure 5.10: DIVAT Difference analysis of "thecount" *vs* PsOpti1.

ment from ± 5.756 sb/g standard deviation for the money line to ± 2.109 sb/g for the DIVAT Difference, for a 7.45-fold reduction in variance. The *Money minus Dcash* estimate had a standard deviation of ± 3.417 sb/g.

Overall, the DIVAT Difference line indicates that "thecount" exhibited a small advantage against PsOpti1 over the course of this match. However, it also suggests that the actual win rate should have been less than one quarter of what the money line indicates. In view of the high noise element, and given the estimated difference of only +0.01 sb/g, it is not difficult to imagine that the final result could have been dominated by short-term luck.

If we trust the DIVAT analysis, it tells a very different story of how the match progressed. First, it appears that PsOpti1 held a slight advantage in play early in the match. This is consistent with the comments "thecount" made after the match. He used an extremely aggressive style, which is effective against most human opponents (who can be intimidated), but turned out to be counter-indicated against this program. PsOpti1 won many games during that phase by calling to the end with relatively weak hands, frequently beating a bluff.

The human expert then went on an extended winning streak, but the DIVAT line suggests that the play was actually close to break-even during that phase. The dramatic collapse that occurred before game 3000 was almost entirely due to bad luck. However, that turnaround did cause "thecount" to stop playing, and do a complete reassessment of the opponent. He changed tactics at this point of the match, toward a more conservative style.

The DIVAT analysis indicates that this was a good decision, despite the fact that he continued to lose due to bad luck. Then the cards broke in favour of the human again, but his true advantage does not appear to have changed by much. Toward the end of the match, the two players appear to be playing roughly on par with each other. Regardless of whether this is a perfectly accurate synopsis of the true long-term expected values, one point is irrefutable: that it is almost impossible to discern the truth from the money line alone.

Interestingly, the DIVAT Difference line also appears to reveal the learning curve of "thecount" as the match progresses. The general upward bend of the DIVAT win rate suggests that the human expert was continuously adapting to the program's style, and learning how to exploit certain weaknesses. PsOpti1 is not an easy opponent to learn against, because it employs a healthy mixture of deceptive plays (bluffing and trapping). Nevertheless, it is a static strategy that is oblivious to the opponent, and is vulnerable to systematic probing and increasing exploitation rates over time.[29] The exposition of learning curves is one of several unplanned bonuses from the DIVAT analysis technique.

5.4.2.2 The Vexbot versus PsOpti4 Match

Figure 5.11 shows the DIVAT Difference line for the match between Vexbot and PsOpti4. The measured standard deviation is $\pm$ 5.611 sb/g for the money line, $\pm$ 3.671 sb/g for *Money minus Dcash*, and $\pm$ 2.696 sb/g for the DIVAT Difference, reducing variance by a factor of 4.33.

In this experiment, the adaptive program took a particularly long time to find a winning counter-strategy, and the strategy it finally adopted secured only a modest

[29] As a point of reference, the author's win rate was approximately +0.3 sb/g against PsOpti1, after extensive experience. Against subsequent pseudo-optimal computer opponents, the author's win rate has been in excess of +0.4 sb/g.

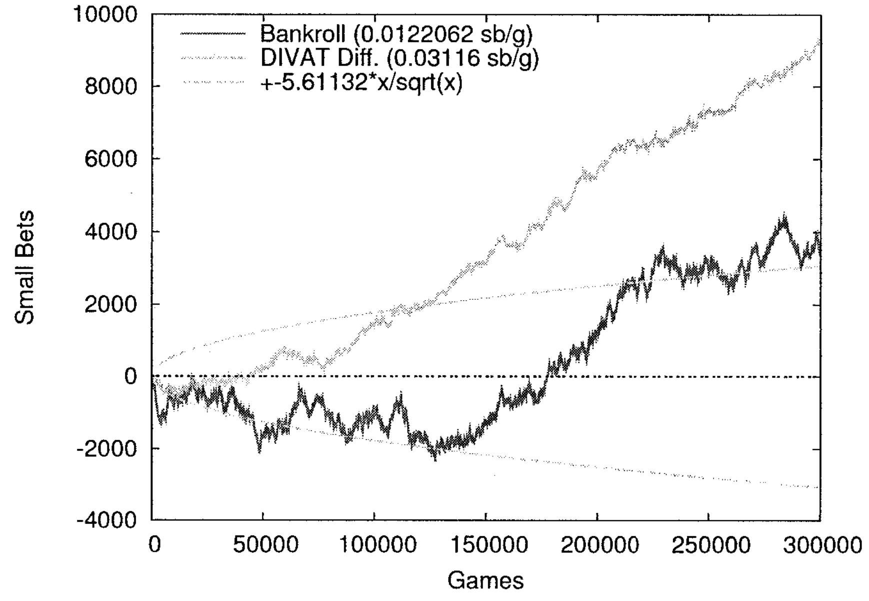


Figure 5.11: DIVAT Difference analysis of VEXBOT *vs* PSOPTI4.

win rate. In other runs, VEXBOT discovered a much more effective exploitation much more quickly (often after only a few thousand hands).[30]

After 300,000 games, the money line is inconclusive (a little over one standard deviation), and close to meaningless (having come from minus one standard deviation, which could be due to normal drift, as we have seen in previous matches). In contrast, the DIVAT analysis is quite certain that VEXBOT is exhibiting an advantage. Moreover, the DIVAT line indicates that VEXBOT found the counter-strategy after about 80,000 games, whereas the money line does not start to run parallel until about 130,000 games. The 50,000-game lag phase is yet another demonstration of the misleading nature of high-variance stochastic outcomes.

Figure 5.12 ignores the early learning phase of VEXBOT by removing the first 130,000 games of the match. This re-calibrated view overlays the two lines, and shows that the win rate was over +0.03 sb/g from that point forward, while the

[30] This illustrates another limitation of the duplicate match system. VEXBOT is capable of displaying many different personalities, from wildly aggressive and over-optimistic to passive and sullen. Since the sequence of cards on the opposing side provides a very different learning experience, there is no way of predicting which VEXBOT will show up in each run, thus nullifying some of the beneficial effects of opportunity equalization.

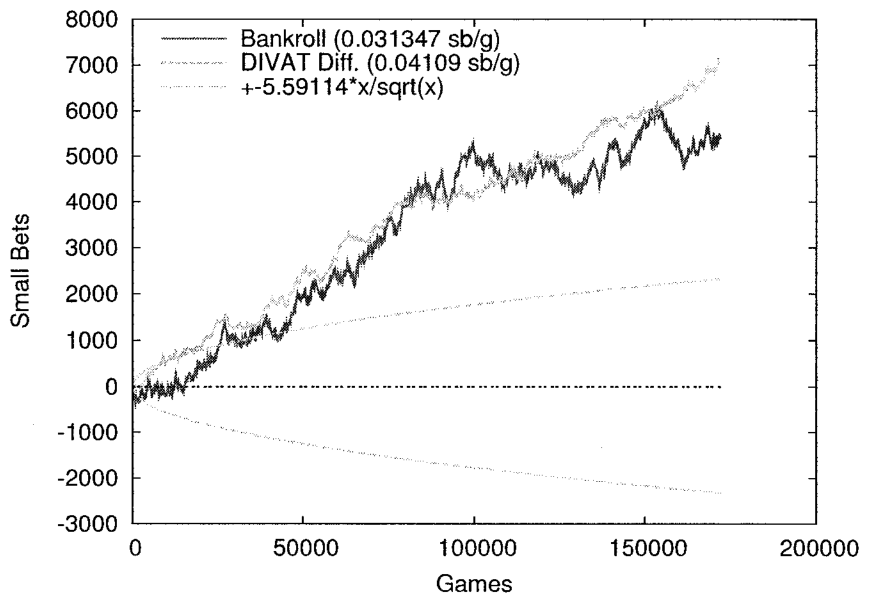


Figure 5.12: DIVAT Difference analysis over the last 170,000 games.

DIVAT total was over +0.04 sb/g.[31] Furthermore, there is evidence of a reversal at about 80,000 games (210,000 games in total), where VEXBOT appears to slip into a less effective counter-strategy.[32] VEXBOT continues to learn throughout a match, albeit at a decreasing rate because of "inertia" from the full match history. We can see from the slope of the DIVAT line that VEXBOT was winning at about +0.05 sb/g during the middle stages, but fell off to about +0.033 sb/g over the final 90,000 games.

5.4.3 Robustness of DIVAT Difference

Over the course of many experiments, it was observed that even radical changes to the underlying policies and parameters had relatively little effect on the DIVAT Difference line. This indicates that it is a robust measurement – it is not overly sensitive to the exact construction of each component, nor to the precise settings of

[31] Note that we have committed the statistical crime of selecting our endpoints, but only to serve an illustrative purpose.

[32] As pointed out by Martin Müller, this speculation could be verified by taking snapshots of VEXBOT's complete strategy at various points in the match, and running them against the static PSOPTI strategy to determine the actual win rate.

parameters.

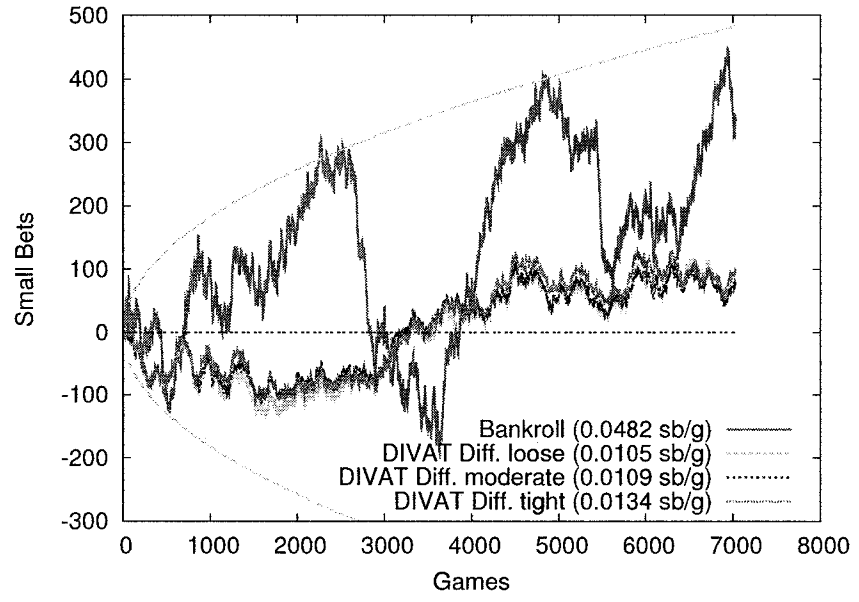
Figure 5.13(a) shows three runs of DIVAT, holding all parameters constant except for the fold policy on the turn. These are varied to reflect a loose style (calling with rather weak hands that may or may not have any chance of winning), a tight style (surrendering in all borderline cases), and a normal well-balanced style between the two extremes. Simulations of 100,000 complete roll-out turn scenarios indicate that each increment increases the fold frequency by approximately 6.9% absolute (*e.g.*, for a pot-size of 4 sb: from 36.3% folds for tight, to 29.3% folds for normal, to 22.5% folds for loose).

Figure 5.13(b) shows three runs of DIVAT, holding all parameters constant except for the betting and raising thresholds on the turn. These are varied to reflect an aggressive style (having rather low standards for betting and raising), a conservative style (requiring a very solid holding), and a normal well-balanced style between the two extremes. Simulations of 100,000 complete roll-out turn scenarios indicate that each increment decreases the bet frequency by approximately 7.2% absolute (*e.g.*, from 45.0% betting for aggressive, to 37.2% betting for normal, to 30.5% betting for conservative).

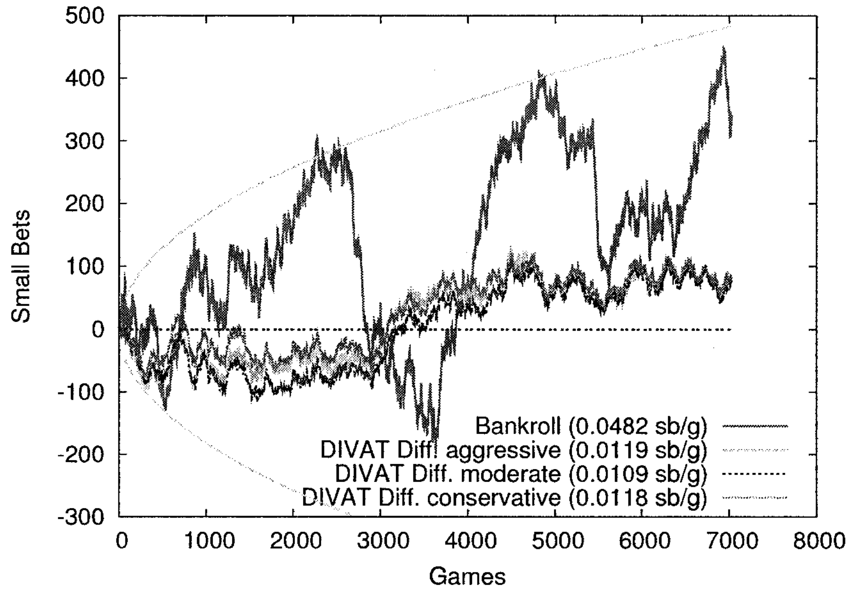
The lines are all close together, so clearly these settings did not have a significant impact on the overall DIVAT Difference in this match. This is not too surprising, in view of how the metric is being used. Even if the absolute measurements are somewhat inaccurate in certain cases, the same objective standards are always being applied equally to both players. To put it more colloquially, even a "crooked measuring stick" can be used to compare the relative lengths of two objects (and using a hockey stick or a driftwood walking stick could work equally well for that particular purpose).

If certain kinds of inaccuracies are common, they will tend to cancel each other out fairly quickly. Relatively infrequent opportunities might not balance out as quickly, but those sources of inaccuracy might not occur at all over the short term, or might not have much impact on the overall score in any case.

Similar robustness results were observed for broad ranges of all parameters, over a large variety of match conditions. Even under extreme settings, the over-



(a) Varying DIVAT Turn Fold Offsets
(+0.05, +0.10, +0.15)



(b) Varying DIVAT Turn Bet Thresholds
(Make1 0.52, 0.58, 0.64)

Figure 5.13: Varying DIVAT parameters.

all assessment appears to degrade gracefully. For example, there is relatively little change in the DIVAT Difference line until using fold policies that are obviously much too liberal, or much too restrictive (well outside the normal behavior of strong players). In some cases, the settings might be inconsistent with the actual styles of the players involved, such as very aggressive betting parameters for a match between very conservative players, but this does not appear to lead to serious consequences or suspicious evaluations.

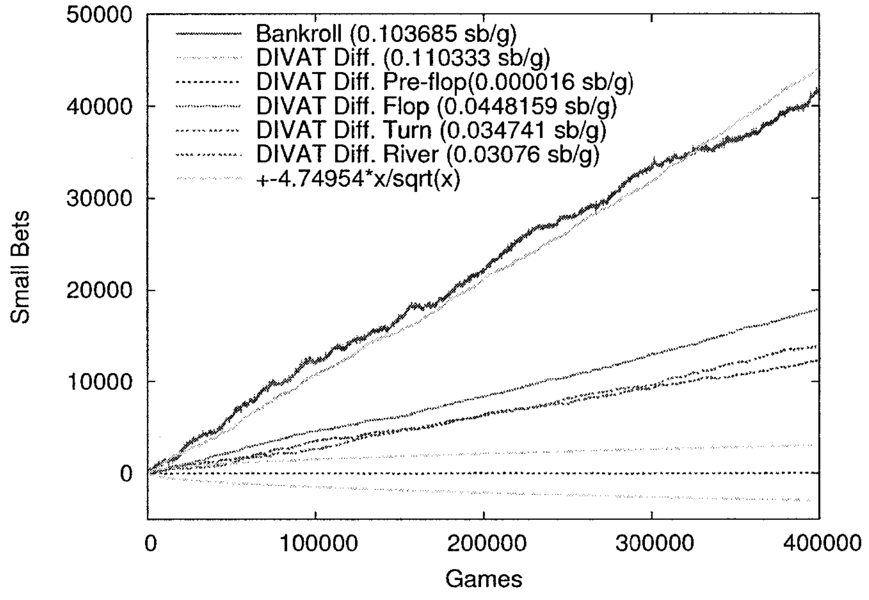
As previously mentioned, it has been shown that roll-out equity DIVAT is a statistically unbiased estimator of long-term expected value. Although this is certainly a desirable theoretical property, it is not absolutely essential for the assessment tool to have practical value. We have seen that stochastic noise can have disastrous consequences on evaluation, occluding the true averages even for very long matches. In practice, an assessment technique that has a lot of variance reduction power can be highly valuable, even if it cannot be formally proven to be unbiased. By design, the DIVAT Difference is largely insensitive to the fluctuations incurred from stochastic outcomes, imbuing it with considerable power for variance reduction, regardless of the precise formulation.

5.4.4 Round-By-Round DIVAT Analysis

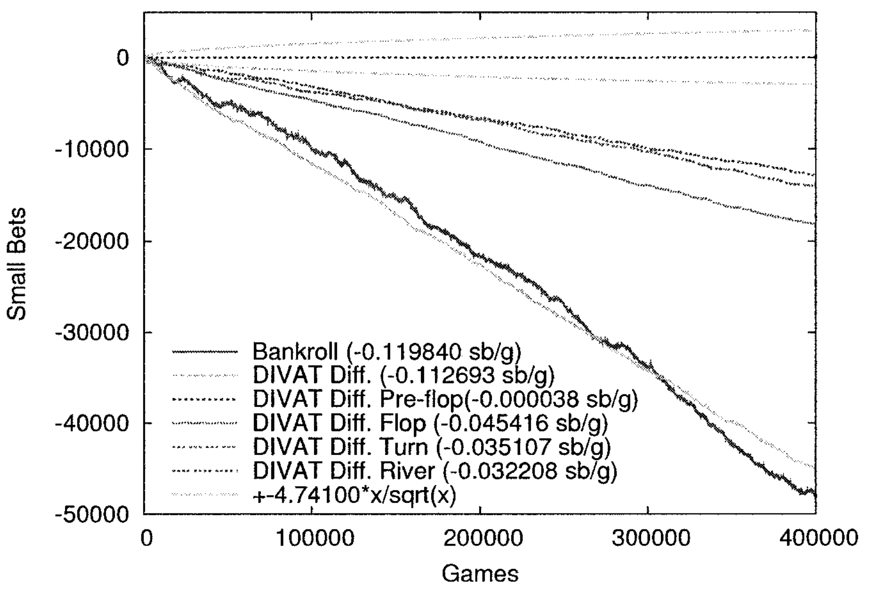
The DIVAT system is designed for round-by-round analysis of each game, with the sum of all rounds characterizing the game as a whole. However, viewing the cumulative results for each round independently can be very enlightening.

Figure 5.14 shows the breakdown of DIVAT Differences for the pre-flop, flop, turn, and river in the PsOpti4 *vs* PsOpti6 duplicate match. The difference in pre-flop skill is shown to be minuscule, which is correct because both programs use the same (randomized) pre-flop expert system. The remaining rounds all contribute roughly equal portions to the total DIVAT Difference, meaning that PsOpti4 consistently outplayed PsOpti6 in every phase of the game.

The EV magnitude of decisions on the flop is generally much smaller than decisions in the last two rounds, because (1) the bet size is half as much (so the ***pot odds*** are roughly double), (2) pot equities are usually closer to 50%, meaning less EV is



(a) PsOpti4 (North) *vs* PsOpti6 (South)



(b) PsOpti6 (North) *vs* PsOpti4 (South)

Figure 5.14: Round-by-round analysis of PsOpti4 *vs* PsOpti6 duplicate match.

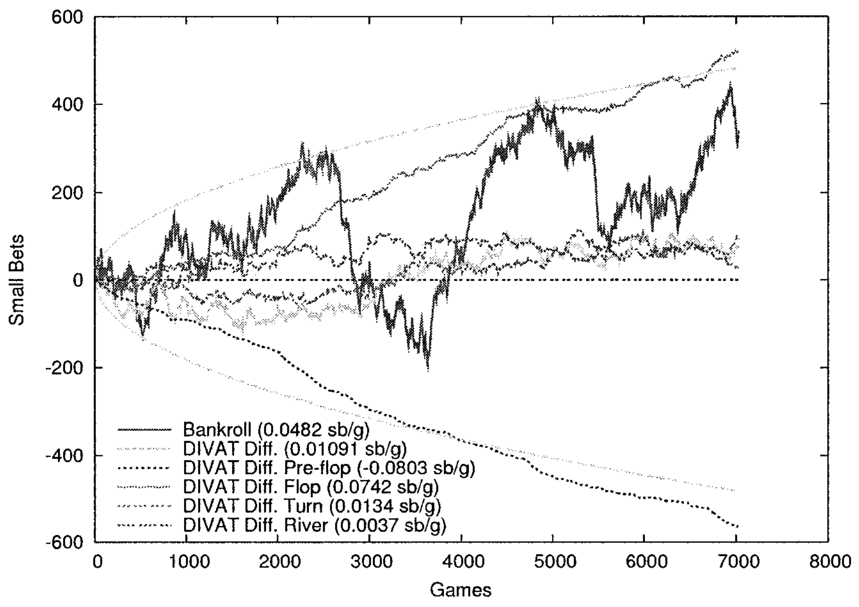


Figure 5.15: Round-by-round analysis of "thecount" *vs* PsOpti1.

at stake, and (3) large implied odds further equalize the true equities. On the other hand, the flop occurs more frequently than the later rounds.[33] Similarly, decisions on the river usually count for a full big bet in equity, which is somewhat larger than decisions on the turn, but they are also somewhat less frequent.

Figure 5.15 shows the round-by-round breakdown for the match between "thecount" and PsOpti1, with some fascinating revelations. Of immediate note is the pre-flop DIVAT Difference line, which indicates that "thecount" was being badly outplayed before the flop. The loss rate of -0.08 sb/g is substantial, exceeding the standard per game win rate for most professional players. This is quite surprising, because the EV magnitude of pre-flop decisions is normally very small,[34] making it the least important round in two-player Limit Hold'em.

In reviewing the match log, the reason for the difference became apparent: "thecount" was folding too frequently before the flop (about 16% of the time). By refusing to fight with weak hands, he was sacrificing the 0.5 sb small blind (or 1.0 sb

[33] The graphs show the total (absolute) effect of each round, rather than the average (relative) magnitude of the differences.

[34] Most of the time one player is no more than a 60-40 favourite, and the small fraction of a bet in EV advantage is further diminished by the effects of implied odds.

big blind) too often, incurring a large net loss. With this kind of objective analysis, even good players can identify weaknesses or "leaks" in their strategy, and make appropriate adjustments.

On the flop betting round, the situation is reversed, with "thecount" holding a large edge in play. Although it has not been verified, it is possible that PsOpti1 was surrendering excessive amounts of equity by folding too easily after the flop. It is also possible that "thecount" was exhibiting a superior understanding of certain strategic aspects, such as the effects of implied odds. There could also be some compensation for his pre-flop selectivity, in that he is more likely to have the better hand when both players connect with the flop. Whatever the reasons, the DIVAT analysis reveals a huge disparity in play, thereby identifying an area of weakness for researchers to investigate.

On the turn and river rounds, "thecount" maintained a small DIVAT advantage. Given the styles of the two players (folding many hands on the pre-flop and on the flop), the turn and river rounds occurred less often than usual, thus having a smaller effect on the final outcome. Again of interest is the fact that "thecount" appeared to improve his results on the turn as the match progressed, as he learned more about the opponent's predictable style and weaknesses. In comparison, he did not appear to find additional ways to exploit the opponent on the river round. The full-game DIVAT summary simply indicates a slight playing advantage for "thecount"; but the round-by-round analysis goes well beyond that, providing many additional insights into the reasons for the differences in equity.

Figure 5.14 shows the round-by-round breakdown for the match between VexBot and PsOpti4, again with some surprising insights. VexBot held a negligible advantage from play on the turn, and was actually losing equity on the pre-flop and (especially) on the flop. However, those losses were more than offset by huge equity gains on the river.

To an expert observing the match, the meaning of this in terms of poker strategy is clear. VexBot was employing a hyper-aggressive "fast" style on the flop, putting in many raises and re-raises. This creates a false impression of strength, at a moderate expense in terms of EV, because the relative differences in pot equity on

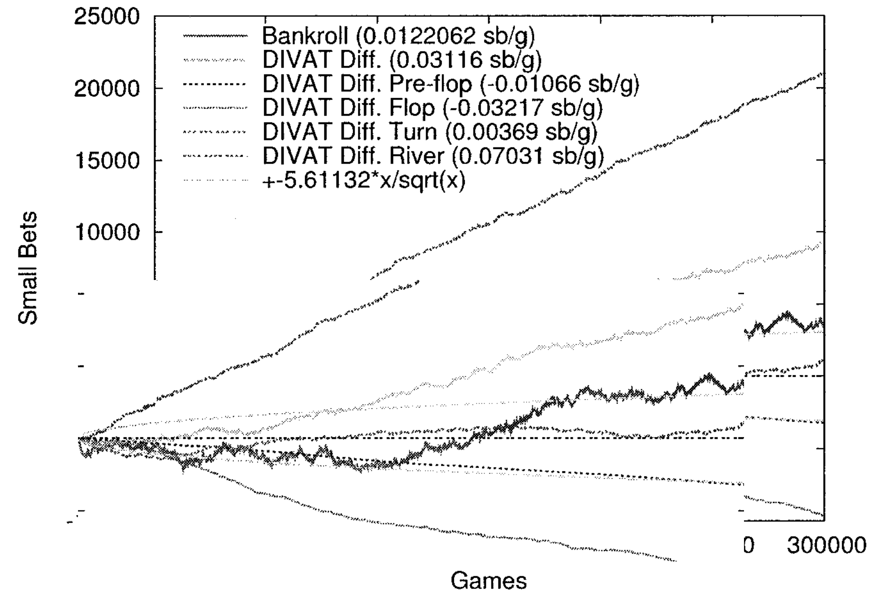


Figure 5.16: Round-by-round analysis of VEXBOT *vs* PSOPTI4.

the flop are small. That false impression was then exploited with follow-up bluffs on the river, when PSOPTI4 folded too often based on its implicit beliefs about the strength of the opponent's hand. A casual observer might conclude that VEXBOT is a "maniac" based on its play on the flop, but the DIVAT analysis clearly shows that there is a method to its madness.

Moreover, it is evident that VEXBOT discovered this imbalance in the strategy of PSOPTI4 very early in the match, since the river DIVAT score maintains the same slope from the beginning. The change at 80,000 games was in fact due to a shift in style on the flop, where VEXBOT eventually learned that it did not need to put in as many extra raises to set-up the same exploitative plays on the river.

5.5 Conclusions

Stochasticity is a major impediment to the accurate assessment of player skill in poker. The simple money outcome of a short-term match is highly unreliable – many thousands of games are required to obtain statistically significant results. Some variance-reduction methods are available, such as duplicate tournaments, but they do not eliminate the problem of assessment in normal live play.

The Ignorant Value Assessment Tool provides an accurate unbiased estimate of the long-term expected value between two players. DIVAT is a practical system that provides a significant reduction in variance, thus extracting signal from the noise. DIVAT uses hindsight analysis to quantify the difference in value between the players' actual actions and a standard benchmark betting sequence. The comparison sequence is based on game-theoretic invariant frequencies and quasi-equilibrium policies, reflecting an appropriate amount to be invested by each player in the given situation. Although much of the relevant context is largely ignored (the previous rounds of betting in particular), the most important aspects are estimated adequately enough for the system to be highly effective in practice.

DIVAT is versatile in its uses, and promises to be an important tool for all researchers working in the domain of poker. The results from a match can be broken down in many ways and analyzed with DIVAT. For example, the player position can be isolated (separating the games played as the first player from those as the second player). We have also used a variation of the tool, called *runtime DIVAT*, to provide more meaningful feedback during matches (where special consideration must be given for the biases caused by the partial observability in live games). There are numerous other uses for the DIVAT analysis technique that have not been addressed here, in the interest of brevity.

The first extension to DIVAT will likely be for multi-player games of Limit Hold'em. In principle, this generalization should be much easier than other two-player to multi-player generalizations (such as game-theoretic Nash equilibria, or imperfect information game-tree search approaches). However, there will invariably be some theoretical and practical issues that will need to be resolved.

Developing DIVAT systems for other betting structures (*e.g.*, Pot Limit, No Limit), and other poker variants (*e.g.*, Omaha, 7-card Stud) should be fairly straightforward. Imperfect information games with a known equilibrium strategy can employ similar methods, using a weighted mixture of baselines corresponding to the relative weights of actions in each mixed strategy.

Applying the general methods to other imperfect information domains is feasible in principle, but might encounter new obstacles due to fundamental differences in the structure of the game trees. For example, games in which the chance nodes and decision nodes are intertwined and not easily separable could be problematic.

Acknowledgments

This research was supported by the Izaak Walton Killam Memorial Scholarship Foundation, the Alberta Heritage Scholarship Fund, the Alberta Informatics Circle of Research Excellence (iCORE), the Alberta Ingenuity Center for Machine Learning (AICML), and the Natural Sciences and Engineering Research Council of Canada (NSERC).

Bibliography

[1] D. Billings. Computer Poker. Master's thesis, Department of Computing Science, University of Alberta, 1995.

[2] D. Billings. Vexbot wins poker tournament. *International Computer Games Association Journal*, 26(4):281, 2003.

[3] D. Billings, N. Burch, A. Davidson, T. Schauenberg, R. Holte, J. Schaeffer, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *The Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 661–668, 2003.

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

[5] 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 H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04*, LNCS 3846, pages 21–34. Springer-Verlag GmbH, 2004.

[6] D. Billings and M. Kan. Development of a tool for the direct assessment of poker decisions. Technical Report TR06-07, University of Alberta Department of Computing Science, April 2006.

[7] I. Frank and D. A. Basin. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 252(11):217–256, 2001.

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

[9] M. Kan. Post-game analysis of poker decisions. Master's thesis, Department of Computing Science, University of Alberta, 2006. In preparation.

[10] B. Sheppard. *Toward Perfection of Scrabble Play.* PhD thesis, Computer Science, University of Maastricht, 2002.

[11] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.

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

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

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

[15] D. Wolfe. Distinguishing gamblers from investors at the blackjack table. In J. Schaeffer, M. Müller, and Y. Björnsson, editors, *Computers and Games 2002*, LNCS 2883, pages 1–10. Springer-Verlag, 2002.

[16] M. Zinkevich, M. Bowling, N. Bard, M. Kan, and D. Billings. Optimal unbiased estimators for evaluating agent performance. In *American Association of Artificial Intelligence National Conference, AAAI'06*, pages 573–578, 2006.

Chapter 6

Conclusion

A Retrospective on Computer Poker

6.1 Introduction

In this chapter, we review each of the major approaches we have addressed in this thesis for building a complete poker-playing system:

(1) simple deterministic *rule-based methods*,

(2) probabilistic *formula-based methods*,

(3) Monte Carlo and enhanced *simulation methods*,

(4) Nash equilibrium *game-theoretic methods*, and

(5) *adaptive imperfect information game-tree search.*

While many of the strengths of each system have been addressed in the core chapters, some of the most important short-comings did not become fully apparent until much later. We take this opportunity to re-visit some of the main issues involved with each architecture, identifying some of the key lessons for developing improved systems in the future.

6.2 Rule-based Methods

A natural first step to building a computer program to play poker is to define a set of conditional if-then-else rules, specifying what action should be taken for each of a large set of situations that can arise. This approach is intuitively reasonable, being similar to how human experts would describe how they play, on a case-by-case basis. In domains with a small number of distinct cases to be handled, this type of "expert system" might be reasonably effective.

Unfortunately, this approach is dependent upon having a knowledgeable domain expert, who is also conversant with the principles and limitations of computer programming. Apart from this inherent "knowledge acquisition bottleneck", the approach will normally be found to be inadequate for strategically rich domains, simply due to the vast number of distinct situations and consequent actions that can arise. It could also be argued that such an approach is of limited scientific interest, since there is little thinking or learning involved within the system itself. Nevertheless, the practical limitations of such an approach cannot be fully understood until it has actually been tried and tested.

A *deterministic rule-based approach* is a set of conditional rules that specify a *pure strategy* (*i.e.* exactly one move, or one betting action in poker) for each of the distinguished contexts. This method is strictly more rigid and restrictive than a *probabilistic rule-based approach*, where each identified context is assigned a randomized mixed strategy (*i.e.* a probability distribution over the set of available moves or actions).

It is well-known that deterministic rule-based strategies cannot obtain the same level of performance as probabilistic mixed strategies in general. A simple example would be the game of Rock-Paper-Scissors, where the *game-theoretic equilibrium strategy* is to choose each action randomly one-third of the time. Any pure strategy, such as always going Rock in a particular set of conditions, is subject to detection and exploitation by an intelligent opponent, both in theory and in practice [3, 2].

In poker, deterministic rule-based systems are overly simplistic in theory, and do not perform well in practice. The fundamental limitations of the approach severely hamper performance, regardless of the programmer's diligence. This has been amply demonstrated in the commercial poker program TURBO TEXAS HOLD'EM, written by Bob Wilson [45]. After more than 15 years of conscientious development and refinement, the program handles thousands of specific conditions, accounting for many important variables such as the cards held, the number of active opponents, relative position, and so on. Nevertheless, it is easily defeated by players of average skill level after a short learning phase.

Moreover, *any* deterministic strategy for Texas Hold'em is necessarily wrong in many circumstances that commonly arise. This is simply a consequence of having trillions of specific contexts in the game.[1] Furthermore, it is relatively easy to understand and learn an appropriate counter-strategy against an opponent who adopts only one rigid style of play. When an expert plays against a simplistic rule-based system, it is particularly easy to make correct inferences about the set of possible opponent holdings as the game progresses. This leads to highly profitable opportunities, such as correctly folding a hand that would normally be much too strong to abandon, or bluffing when it will succeed an inordinately high percentage of the time. It is much more difficult to adapt to players who vary their style over time.

It is not surprising that a small finite set of conditionals cannot capture the subtlety and nuance of a strategically complex game like poker. Distinguishing and handling thousands of distinct contexts in Texas Hold'em is not sufficient to produce an adequate approximation of excellent play, because superficially similar cases can have subtle but important distinctions. Two situations that are grouped together might require radically different actions, such as raising in one and folding in the other. Since a perfect partitioning of situations is unattainable, frequent errors are unavoidable, and are a fundamental limiting factor to performance.

6.3 Formula-based Methods

A *formula-based approach* is a more flexible generalization of rule-based approaches, where an arbitrarily complex formula or procedure is used as the criterion for distinguishing cases. For example, an exact weighted enumeration of subcases might be used to determine a hand assessment value in the range zero to one; with a consequent probabilistic action based on predetermined *thresholds* for that value. The thresholds themselves might be determined by means of a simple constant, a linear

[1] Two-player Limit Texas Hold'em has a relatively small search space compared to other real poker variants, but the imperfect information game tree has more than a quintillion states (1,179,000,604,565,715,751 nodes in total).

formula, or a complex formula based on many parameters.

Probabilistic formula-based approaches can consider many more variables pertaining to strategic elements in the game, effectively multiplying the number of distinct situations with each additional variable. The first versions of our formula-based programs, LOKI and POKI, easily out-performed TURBO TEXAS HOLD'EM, due to their more accurate context-sensitive hand assessment, flexible strategies, and gradual adaptation to each specific opponent's biases and tendencies [10, 9, 29, 11, 8, 30, 33].

Nevertheless, formula-based approaches still rely on the principle of abstracting trillions of specific cases onto a much smaller number of general circumstances, controlled by some particular parameterization of situations defined by the architect. As a result, they still run into many of the same liabilities as rule-based systems, albeit to a lesser extent. Even in the best case, there are inherent limitations on how well any such collection of components can perform.

There is also a serious practical limitation to this approach, in that the code becomes disjointed and difficult to maintain. Adding an additional feature to the "patchwork quilt" can produce unpredictable and undesirable side effects, requiring a complete re-tuning of the system. Eventually, the code base becomes cumbersome to embellish and maintain.

POKI has proven to be more skillful than average human poker players, based on empirical testing that includes tens of millions of games played against human opposition. Nevertheless, it became clear early on that this approach would be inadequate to achieve the goal of surpassing all human players [14, 5, 13].

Of particular concern is the lack of a *well-balanced betting strategy*. The formula-based systems are adequate for playing a straightforward style, but are much too predictable, in general. The instances of bluffing (including ***semi-bluffs***) and ***trapping*** (either ***check-raising*** or ***slow-playing***) are generally not frequent enough to adequately mask the strength and nature of the program's holding. This "information leakage" is compounded as the game progresses, allowing an astute opponent to make correct inferences and superior decisions frequently enough to obtain significant expected value gains. It is not sufficient to simply decrease the predictability of the program by bluffing and trapping more often, as that would lose ***equity*** against most typical opponents. We do not know what game conditions we will face in the future, and tuning the program for one particular set of circumstances means that it will not play well in other circumstances. Thus, it is not really a tunable parameter, because it is opponent dependent.

More generally, to reach the highest levels of poker skill, a player must be able to *reason* about the prevailing game conditions, and adjust to any specific circumstances. This ability is essential to mastering the game of poker, whether the player is a human or a program.

6.4 Simulation Methods

Simulation is simply the repetition of many instances in order to obtain a statistical average. In basic Monte Carlo simulation, each *trial* is simple and unbiased, such as sampling uniformly at random from the complete space of possibilities. Each individual outcome might be a rather poor indicator of the average, but that is of little consequence as long as the relative balance of instances is fair. One tremendous advantage of simulation is that it is conceptually simple, and easy to program.

In many computing science domains (not limited to games), Monte Carlo simulation has proven to be a powerful technique in practice. In backgammon, for example, simple roll-out evaluations can perform as well as the more famous neural network trained evaluation functions [41, 23].[2]

An example in poker is the *cold roll-out* of many pre-flop instances, in order to determine the relative values of starting hands. The resulting averages are **not** an accurate reflection of the true expected values, because all of the subsequent betting has been ignored. Nevertheless, it turns out that the resulting *relative ranking* of starting hands is reasonably accurate. These raw scores are the basis of the formula-based expert systems that the author designed for pre-flop play in LOKI, LOKI II, and POKI.

[2] In the *incomplete information* domain of hockey pool drafting, the author used Monte Carlo simulations of a Poisson model of scoring to produce a computer program that apparently outperforms expert humans (unpublished course project).

With *iterated roll-outs*, described in Chapter 2, the results of a simple roll-out are used as the decision criteria for playing each hand in a subsequent generation of simulations. This quickly ameliorates the flawed *always-call assumption* built into each trial, and the results are remarkably good, despite the relative simplicity of the process. When applied to the two-player game of *Pre-flop Texas Hold'em*, the results were in excellent agreement with Selby's direct linear programming solution using a modification of the Simplex algorithm [34]. It is quite likely that much more could be done with this high-leverage methodology in the future.

Unfortunately, with purely random sampling, it can take a long time for the simulation to converge on accurate estimates. To accelerate the process, various methods of *biased sampling* have been introduced. The objective is to emphasize samples that provide the most information gain, without introducing a severe bias (or at least to maintain a measurable degree of bias) in the result.

In poker, we did this with *selective sampling*, where the opponent's hand was selected from the weight table of current likelihoods for each possible holding, as explained in Chapter 2. An *ad hoc* method was used to quickly generate (randomized) future actions of all players, and the trial was played through to the end of the game to obtain a net value. Averaging the outcomes of many such trials produces a refined estimate of the expected value of each available action. In Scrabble, selective simulations have also been used to enhance the basic evaluations [36, 37].

As noted in Chapter 2, this method also resulted in *new strategies*, like ***check-raising***, as an emergent behavior that was not part of any of the individual trials. Thus, the value of simulations is not limited to a refinement of basic evaluation.

A major drawback of this approach in practice is that it can be highly *volatile*, and result in extremely unbalanced play. For example, if the relative probability of the opponent folding to a future bet is slightly too high, the maximum expected value play could be to raise 100% of the time in the current situation. This form of instability was witnessed on many occasions, with the program raising constantly as a result. Similar instability of simulations was seen by Ginsberg in the game of Bridge [21, 22].

A more subtle but theoretically critical limitation is that repeated trials of perfect information instances cannot, in general, produce accurate results for an imperfect information situation [16, 15]. In Bridge, this is nicely illustrated by simulations of *double finesse* situations. The declarer has a choice of two tactical finesses, one through the left-hand opponent, the other through the right-hand opponent. In each perfect information instance, the declarer can win the finesse (one way or the other) and make the contract, thus the simulation returns a 100% chance of success. In the actual imperfect information situation, the declarer must choose which way to take the finesse, and the real chance of success is only 50%. In poker, this means that future actions *must* be determined in some objective manner that does not depend on explicit knowledge of the opponent's cards in each simulated trial.[3]

Since future actions are based on some kind of heuristic speculation, there is also a problem with *compounding errors*. The relative frequencies for future actions have some degree of error, and that error is multiplied with each subsequent action. As a result, the generated sequence of actions leading to the end of the game could be rather unrealistic in a large fraction of the samples.

One work-around to this problem is to generate a sequence of actions leading to the end of the game *first*, according to the presumed probability distributions. The opponent hand is then assigned *after* the entire sequence has been determined, according to some measure of consistency with the given sequence of actions. Aaron Davidson experimented with this technique, and found that it produced much more stable results for the simulation. Of course, it could also contain a huge amount of bias in the relative frequencies of possible opponent holdings, making the results arbitrarily inaccurate.

However, suppose one was to simulate *all* possible future action sequences, with an accurate estimate of the relative frequency for each, and combine that with a probabilistic estimate for the net outcome of each trial. The simulation would now be much better grounded, with only a single error term for each estimate, rather than compounding errors. In fact, we could eliminate the variance from

[3] This fact has not been a serious impediment in Scrabble, because imperfect information does not play a critical role in the strategy of the game. Approximate methods are sufficient to account for the hidden information (*i.e.*, the probability distribution of the opponent tiles), and selective simulation has produced a super-human level of play [35].

running many trials, by running each action sequence exactly once and computing the weighted combination of expected values according to the relative probability for each sequence.

This reformulation of the problem was the basis of the author's imperfect information game-tree search algorithms, *Miximax* and *Miximix*, discussed in detail in Chapter 4. Since those direct expected value calculations essentially subsume the improved simulation method, with zero variance, the selective sampling techniques have largely been abandoned.

6.5 Game-Theoretic Methods

The major limitations of rule-based and formula-based approaches naturally lead to the study of architectures utilizing *game-theoretic equilibrium strategies*, or approximations of equilibrium solutions. Game-theoretic strategies for the full game are inherently well-balanced, and thus the leakage of information is automatically prevented.

In game theory, all two-player zero-sum games have at least one equilibrium strategy, where a randomized mixed strategy for every decision point can ensure the best worst-case outcome, regardless of the opponent's strategy. This is known as the *Minimax Theorem*, and was proven in the foundational work by John von Neumann, who used poker as a model of all two-player adversarial contests [42]. John Nash later proved the necessary existence of equilibrium strategies in multi-player games [27, 28].

It has long been known that bluffing and trapping are an essential part of a sound poker strategy. Even in extremely simple poker games, equilibrium strategies involve a mathematically determined frequency of bluffing and trapping, which accomplishes the task of ***information hiding*** [26]. Moreover, the game-theoretic value of the game cannot be guaranteed without these deceptive plays.

To apply game-theoretic principles to a practical poker-playing system, it greatly simplifies matters to study the two-player game first. Multi-player games are vastly more complex in theory, and should be considered after the pertinent lessons from the two-player game have been learned.

Interestingly, the two-player game of Limit Texas Hold'em has certain characteristics that make it *more* difficult than the full ten-player game in practice. With more than one opponent, deceptive plays are less common, both in theory and in practice. Bluffing is less frequent because the chance of a successful bluff is greatly reduced against several opponents, and slow-playing a strong hand is more dangerous against multiple drawing hands. Thus, the play tends to be considerably more straightforward in the multi-player game, making it more amenable to formula-based systems than the two-player game.

The greater prevalence and necessity of tricky play in the two-player game dramatically exposes the serious imbalances of rule-based and formula-based betting strategies – all programs to date have performed terribly in this variant. The best of this class of programs is POKI, but in the two-player game it is easily defeated by the author at a win rate exceeding +0.8 small bets per game (sb/g), meaning that the program would actually lose less by folding its ***blind*** every game.

In principle, an equilibrium strategy would never lose to any opponent over the long run (assuming a zero-sum game, with no ***rake***), and might possibly win. In theory, an imperfect opponent can make *dominated errors*, losing expected value to an equilibrium strategy, thereby losing in the long run. For example, a highly sub-optimal player could fold very strong hands. In contrast, no dominated strategies exist in the game of Rock-Paper-Scissors. This means that the game-theoretic equilibrium strategy ($Pr(R, P, S) = \{1/3, 1/3, 1/3\}$) can only break even against *any* opponent strategy whatsoever.

Whether or not a perfect equilibrium strategy would win at a significant and sustainable rate against top human poker players is an open question. In all likelihood it would win slowly in a zero-sum game, but might not overcome the overhead cost of the rake in a game played for real money. An equilibrium strategy can be viewed as a primarily *defensive strategy*, designed to *not lose*, as opposed to *trying to win.*

The game of Oshi-Zumo is similar to poker in that equilibrium strategies are complex, and might be a viable approach for practical play due to the existence of dominated strategies. However, an equilibrium strategy does not necessarily perform well in practice. Michael Buro has demonstrated that very simplistic strategies, which are sub-optimal and highly exploitable in general, can nevertheless break even or have a negligible losing rate against an equilibrium solution [12].

Regardless, an equilibrium strategy would still be highly useful in practice as a default strategy to use while observing an unknown opponent, since it cannot be exploited. Provided that a sufficient variety of strategies are naturally explored during this observation phase, the safe baseline could then be abandoned at any time in favour of positive expected value (+EV) exploitive play. Since this switch would be at the discretion of the program, it can be done with a controlled minimal level of risk.

Although game theory has been applied in a wide range of disciplines, it has always been severely limited by the size of problems that can be solved computationally. With the traditional *normal form* representation, the number of combined strategies for two players is doubly exponential in the number of variables, thereby restricting its use to only tiny problems. However, the normal form ignores certain dependencies that may exist in the game.

For certain classes of problems, including poker, there exists an alternative representation called *sequence form*. This formulation was popularized in AI by Daphne Koller [24, 25], although it was apparently known to practitioners as early as the 1960s [31]. The method produces a *linear program* (LP) representation that is only singly exponential (*i.e.*, linear in the size of the poker game tree). Thus, moderate-sized games (*e.g.*, having more than one million nodes in the game tree) can be solved. This is still many orders of magnitude smaller than real poker variants, but it does open the possibility of solving small scale models of real games to be used as an approximation of an equilibrium strategy.

We used several methods of abstraction to define smaller poker variants that retain most of the key underlying probabilities and structural properties of Limit Texas Hold'em. The most powerful of these is the natural grouping of hands by relative strength, which we called *bucketing*. A very similar approach was used by Shi and Littman for the reduced game of *Rhode Island Hold'em* [38].

Linear programs for these abstracted games were generated automatically, and their solutions produced a reasonably well-balanced strategy for the real game. This class of programs, collectively called PsOpti or Sparbot, constituted a breakthrough in playing performance for two-player Hold'em. Despite reductions in the size of the game tree by a factor of more than 10 billion, the programs were able to defeat all previous computer programs, defeated average human players by a significant margin, and did not lose rapidly against very strong opposition [4].

A major difference in the style of Sparbot from most human players is the willingness of the program to make mathematically correct defensive calls with weak hands. Whereas sheer aggression is very successful against most human players, the computer opponent is not easily intimidated. This feature was a large factor in the program's success in its 7030-game match against world-class player Gautam Rao. When Mr. Rao changed his approach to a less aggressive style half-way through the match, the effects were dramatically positive. However, recent analysis indicates that much of that turn-around was due to the luck of the cards – the program was simply destined to lose with reasonable play (as shown in Chapter 5). Overall, the objective quality of the program's play was nearly on par with that of the opponent, thus the statistically inconclusive outcome was appropriate.

However, this approach only gives a crude approximation of a true equilibrium strategy. Over time, a skilled player can eventually discover subtle tendencies and weaknesses in the program's play, and exploit them. In particular, after studying Sparbot's style for many thousands of games, the author is able to defeat it at a win rate more than ten times greater than any other human opponent the program has faced. Although there is unquestionably a lot of value in knowing the intimate details of the program's design and playing habits, this outcome cannot be dismissed as a special case, as it does indicate the level of exploitability that could be achieved by master strength opponents who play against the program continuously over a long period of time. Moreover, an effective counter-strategy could be communicated to another expert player in just a few sentences.

There are three notable liabilities in the play of Sparbot. One is a tendency for weak actions to indicate a weak holding (trapping plays with strong hands are relatively uncommon). In contrast, a strong action does not necessarily indicate a strong holding, as the program's bluffing levels are high enough to create sufficient doubt. As a result of these biases, it is often possible to detect bluffing opportunities that have an inordinately high likelihood of success. Thus, the program could benefit from trapping more often.

Secondly, the program will usually bet in second position after the opponent checks, leaving itself vulnerable to frequent check-raises. The betting sequence check-Bet-raise-Fold (kBrF) implies that either the opponent earned an extra bet with a legitimate hand, or executed a successful check-raise bluff. The program could obtain a more effective balance by checking more often with mediocre hands, rather than only the weakest of hands.

Perhaps the most serious flaw is a disconnect between the betting actions *before the flop*, and subsequent play *after the flop*. This is likely due to the manner in which the abstract models are constructed. A 3-round abstracted model of the game is defined for the first three betting rounds (the ***pre-flop***, ***flop***, and ***turn***), with expected values applied at the truncated leaf nodes to improve the approximation. The solutions to these *pre-flop models* are then used to estimate the relative distribution of hand holdings for each of the pre-flop betting sequences. These distributions are then used as input priors for computing the 3-round *post-flop models* (***flop***, ***turn***, and ***river***). Unfortunately, overlaying these 3-round models is not seamless, and there are detectable inconsistencies in the resulting play. Although the prior distributions are not unreasonable in general, it has been repeatedly observed that applying any one particular pre-flop distribution as a specific model of the opponent can be counter-indicated for the post-flop equilibrium solutions.

In other words, the prior distributions might be appropriate for *some* post-flop equilibrium strategy, but there are infinitely many post-flop equilibrium strategies in general [18, 43]. There is no guarantee that the solution to the pre-flop model will be harmonious with the post-flop solution. In principle, combining two equilibrium strategies in this fashion can produce inconsistencies, and a non-equilibrium strategy overall.[4]

The same limitation is true for other methods of overlaying subtree solutions.

[4] This observation was proven by Neil Burch, using counterexamples from simpler domains.

Recently, Andrew Gilpin and Tuomas Sandholm at Carnegie Mellon University have repeated the PsOpti approach, using a powerful automated abstraction technique called *gameshrink*, which was used to solve Shi and Littman's game of *Rhode Island Hold'em* [19]. To produce a program for the much larger game of Limit Texas Hold'em, they pre-compute solutions for the first two rounds, then generate and solve real-time LP problems for the turn and river. A very similar idea was tried by our group in 2003, but was abandoned because: (a) we had no satisfactory method for obtaining appropriate prior distributions for the turn and river LPs, and (b) the real-time LP generation and solving could not maintain the proper pace of Limit Hold'em (roughly one second per decision). The method Gilpin and Sandholm used for obtaining priors was to interpret the previous betting as a specific model of the distribution of opponent holdings [20]. Unfortunately, this can leave the program highly susceptible to acquiring false beliefs when facing an opponent who uses a very different (but equally viable) strategy.[5]

To avoid these problems, it would be highly preferable to use a single 4-round model for the abstract game. Unfortunately, to date the results for 4-round models have been unsatisfactory because the size of the problem is prohibitive (or the abstractions need to be so severe that the solutions suffer serious distortions, making them unsuitable as approximations of the real game).

The lack of harmony between the pre-flop and post-flop play of SPARBOT is quite obvious in practice. For example, when the pre-flop betting round has two or three raises, the program will frequently fold to a single small bet on the flop. Having built up a very large pot (getting 7-to-1 ***pot odds***, or better), it is seldom correct to fold, especially in second position. The probable explanation is that the prior distributions for those pre-flop sequences are too extreme in terms of assigning

[5] A recent poker competition for computer programs was held at the annual conference of the American Association of Artificial Intelligence (AAAI'06) [39]. The University of Alberta team won every match it played in both tournaments (using a simple combination of PsOpti4 and PsOpti6, along with a performance feedback signal based on a runtime version of the DIVAT analysis technique) [1]. The second place finisher in both tournaments was BLUFFBOT, a SPARBOT clone written by Teppo Salonen [32]. A slower-paced competition was held to accommodate the entry from Carnegie Mellon, GS2, which required extra time for decisions on the turn and river (for solving real-time LPs). GS2 used roughly 100 times as much time for the turn and river, but lost duplicate matches by a statistically significant margin to the University of Alberta program (-0.18 sb/g), and to BLUFFBOT (-0.12 sb/g).

a strong hand to the opponent, and the bucketing function largely disregards the residual draw value of a hand (unless it belongs to the special bucket for high-potential hands).

Although the overall average strength of the opponent's hand might be high in these cases, the specific context of the actual hand is ignored. In particular, if the flop ***board cards*** contain two or three low cards, there is a high likelihood that the flop failed to help either player, and the program either still has the best hand, or has sufficient chance of improvement to easily warrant a call.

The failure of SPARBOT to account for this type of *context-sensitive information* is actually the major liability of the method as a whole. To put it simply, it fails to grasp the specific meaning of certain chance outcomes, and the implications of certain ***board textures***.

In the process of abstraction, most subtle forms of information have been sacrificed. First, knowledge of the cards and any specific features of Texas Hold'em have been abstracted away, leaving only generic bucket classifications that are correlated with general hand strength (or a special case of a weak hand with high potential). While this might cause an acceptable amount of error (and is for the most part unavoidable), the abstraction over chance outcomes is far more limiting.

Between each betting round, a transition function is used to model the effect of dealing cards to the board. The function maps the probability of each pair of hand classes onto the set of subsequent class pairs. For example, with seven buckets for each player, each of the 49 pairs $(P1, P2) = (i, j)$, $0 \le i, j \le 6$, on the flop goes to 49 pairs $(P1, P2) = (x, y)$, $0 \le x, y \le 6$, on the turn, with edge weights summing to the parent.[6]

These transition networks are computed by averaging the changes of hand bucket classes over a large sample of representative examples, either by simulation or by exhaustive enumeration of all cases. This can provide an accurate estimate of the overall effect for "the turn happens", but does nothing to account for the *actual* turn

[6] Typically, changes to the matching bucket or next lower bucket have high probabilities; and jumps up to high buckets are more likely than jumps down to much weaker hands. The transition network from the pre-flop to the flop has a much flatter natural distribution, with all edges having a higher baseline probability.

card that lands. The system cannot understand the card itself, since cards do not exist in the abstraction. More importantly, it has no means of distinguishing outcomes that are generally favourable, unfavourable, or neutral. For example, if the program holds a medium pair, there is a huge difference between the turn card being a Deuce (good) or an Ace (bad). Apart from the small shift in hand value (which may or may not change the bucket classification), there is nothing to indicate that the Ace outcome is generally much more dangerous than the Deuce.

Using the grand aggregate average for a single transition function fails to account for the specific context of the game, resulting in relatively uninformed decisions. Certain extreme boards, such as four cards in the same suit or four cards to a straight, are handled poorly by the program, because the general guidelines do not apply to those special circumstances.[7]

A significant improvement could be obtained by using a family of transition functions. For example, averages could be computed for turn cards that land in each of the four ranges around existing board cards, or that pair a board card (for instance, a flop of Q-8-5 would partition turn outcomes into sets {A,K}, {J,T,9}, {7,6}, {4,3,2}), and {Q,8,5}. The actual turn outcome would then be interpreted and used accordingly. With more computing power, each rank could have a distinct transition function; or even every legal turn card for a designated flop. Unfortunately, the size of the LP problems generated is already as large as can be conveniently computed, so it is not an easy task to improve the existing method.

As it stands, SPARBOT is a difficult opponent to learn against, due to its well-balanced mixture of bluffs, traps, and straightforward play. In comparison, the large majority of human players are much easier to categorize and exploit quickly. Nevertheless, SPARBOT has the major liability of being a *stationary player*. Although it uses a randomized mixed strategy, it is a static strategy (does not change over time), and the program is completely oblivious to the opponent. Knowing this fact makes it much easier to play against the program in practice. A human player can

[7] As an extreme example, if the board contained A-K-Q-J-T with four suits, then all hands would be equivalent – a tie is guaranteed. SPARBOT does not understand this, and could conceivably raise and then *fold* with its "average hand". This type of logically dominated behavior did actually occur, prompting the use of a wrapper program that could veto obviously bad decisions.

systematically experiment, probing for weaknesses and statistical biases, without any fear of being punished for playing in a highly predictable fashion. Although it might take a fairly long time to discover the flaws in the program's strategy, once they are learned they can be exploited forever after that time. Thus, the program is destined to be less and less successful as time goes by, and will be soundly defeated in the long run.

As pointed out by Michael Bowling, this is not actually a failing of game theory itself, but is due to implicit simplifications in our application of game theory. We are striving to find an approximation of the Nash equilibrium for the *single-shot game*. In reality, the previous history cannot be ignored, and poker should be treated as a *repeated game*. In addition, there are numerous sub-classes of equilibria that might be appropriate to look at, such as *subgame perfect equilibria*, *perfect Bayesian equilibria*, and *trembling hand perfect equilibria* [18, 17, 44]. Unfortunately, there is no obvious way of accounting for the additional context, even if the already enormous scope of the problem somehow permitted the inclusion of thousands of previous games. So, it seems that a crude estimate of a simple stationary equilibrium strategy is the best we can do in practice.

Much more could be said on the practical limitations of the game-theoretic approach, but even in the strongest case, there are clear motivations to investigate architectures that produce rapidly learning players. In particular, a computer opponent that is constantly observing and adapting to the opponent does not allow such a relaxed approach to learning, because regular systematic play can and will be punished. Thus, a program that detects and exploits betting patterns should put up a much tougher fight against strong opposition in practice. In the final analysis, the surest way to win against the best human players is to actually *try to win*.

6.6 Adaptive Imperfect Information Game-Tree Search

A major concern of all previous poker programs is that they do not handle the important task of *opponent modeling* particularly well. General and specific statistical models are maintained within the formula-based architectures (LOKI and POKI), but the manner in which they are maintained and updated is a limitation. Since those procedures are hard-coded, the process is inherently rigid, and presupposes many assumptions about the opponent's approach to the game. To be completely flexible, the modeling system should not presume anything about the opponent's understanding of the game, and should handle any arbitrary style fluidly.

There are many dimensions to the opponent modeling problem that have only been touched on to date. We have discussed the general necessity for deceptive plays (bluffing and trapping) for information hiding purposes. However, the specific opportunities for trick plays are not assessed very carefully, in regard to the opponent's past behavior and tendencies. Normally these plays occur as part of an overall randomized mixed strategy, such as bluffing 10% of the time when we would otherwise check. A much more exploitive approach would try to *deduce* when a bluff will be relatively likely (or unlikely) to succeed, based on the prevailing conditions of the game and opponent.

For these and other reasons, the most promising program architecture performs a detailed analysis of the expected value for each possible action, using an adaptive imperfect information game-tree search. Two algorithms have been developed for this purpose: *Miximax* and *Miximix* [13, 6]. In essence, these algorithms "do the math" of poker. For every combination of future chance outcomes, the net loss or gain of every future betting sequence is considered, each weighted by its relative likelihood. This yields an overall expected value for each initial action available.

For example, suppose we face a bet on the turn. We can elect to fold, call, or raise. If we call, there are 46 possible river cards that could be dealt, and 19 possible betting sequences that can occur afterward. In cases where one player folds, the outcome is known and exact. In cases that lead to a ***showdown***, we will estimate our probability of winning given the particular sequence of actions to determine our equity. The linear combination of relative weights for each of those 874 cases produces an overall expected value for calling. The expected value for raising is computed similarly, but has more cases due to remaining possibilities for bet sequences on the turn. Having computed the expected values for fold, call, and raise, we can decide how to proceed (such as simply choosing the action with maximum expected value; or perhaps mixing actions when two alternatives are close in value).

This process mimics the type of calculations done by human poker players, albeit at a much finer level of granularity. The computation can be performed in real time, even for decisions after the flop with more than a million subcases. Thus, it can be very *precise*, but that does not guarantee that it is *accurate*. The accuracy is limited by the correctness of the relative weights assigned to each future action of the opponent, and to our equity estimates for each showdown outcome. These two properties form the heart of the adaptive modeling system. The data structures and update policies used to estimate these values will govern how successful this approach can be in practice.

This architecture has been implemented in the program VEXBOT. Although it is not yet fully refined, the results have been noteworthy. The program learns to exploit simplistic rule-based players, such as ALWAYS_CALL and ALWAYS_RAISE, at close to the maximum win rate possible. Against POKI (which was never specifically designed to play the two-player game), it wins at a rate comparable to this author, and occasionally better (approximately 1.0 sb/g). Against SPARBOT, it eventually learns to exploit the gaps in the equilibrium approximation, surpassing the win rate attained by any human player other than this author. In play against strong human players, it has proven to be a dangerous opponent, possessing many positive attributes not seen in previous programs [6]. However, despite these encouraging results there are numerous causes for concern and issues to be resolved.

In principle, we are computing a *best response*, based on our current beliefs about the opponent, which are subject to change over time. Although the best response is a powerful counter-strategy for maximizing our exploitation of perceived weaknesses, it is not necessarily the best policy to follow in all cases, for a number of reasons.

First, as a poker-playing policy, pure best response without modifications represents a highly predictable reaction to the given circumstances. Good poker players can easily understand the resulting straightforward style and use it to their advantage. In fact, *any* form of predictability is potentially vulnerable to exploitation (which is why mixed strategies are a good idea, in general).

With a best response player, bluffing and trapping plays are only executed if the calculation determines that they have the highest expected value. This tends to be exhibited as an "all or nothing" response, depending on the particular beliefs currently held about the opponent. Specifically, if the belief state suggests that the opponent folds slightly too often, there can be a sudden switch in behavior toward betting and raising very frequently. There is no moderation in its strategy adjustments. Similarly, if the system believes that the opponent bluffs slightly below the optimal frequency, the reaction may be to fold *all* marginal situations. This latter case is particularly dangerous, since it prevents future observations that could revise that belief if it is mistaken (a situation we refer to as a "folding trap"). Against a player who is constantly changing styles, the over-reactive best response opponent may swing like a pendulum between belief states. Since the program's beliefs are rather transparent and obvious, a master opponent can stay one step ahead, defeating it easily.

Even if those patterns were not an issue, it is not clear that maximum exploitation would be desirable. A strong poker player will often consider *not* exploiting a known opponent weakness too aggressively; because punishing an error too severely could make it easy for the opponent to identify the flaw in their strategy, forcing them to change behavior. When possible, it could be preferable to exploit the error at a slower *sustainable* rate, so that the known weakness will persist indefinitely.

Another reason to deviate from a simple best response is for the purpose of *exploration*. Regardless of how well the current belief state may be performing, there is always the possibility that other untested lines might be more profitable. The "exploration / exploitation trade-off" is a well-known problem in machine learning, and is no less important in poker, with no ideal solution in general.

Although this architecture has numerous advantages over other approaches, it also has many practical limitations. More technical details will need to be addressed before the system can perform at its highest potential.

First, there is a desperate need for good default data. The frequentist model allows for essentially *any* belief to be held, regardless of how irrational it might be. If no default seeding is provided, the system can infer that a small number of repeated observations are indicative of what will continue to happen in the future. For example, if the opponent should happen to win with an Ace-high flush, the system may implicitly believe that the opponent will always produce an Ace-high flush in those circumstances, since it has no built-in knowledge of how often that outcome "should" happen.

The particular case of having no defaults whatsoever can lead to even more bizarre beliefs. Suppose there is no betting whatsoever through to the showdown, and the program wins, beating the opponent's 32o. All else being equal, the program will develop a bias in favour of checking, because of the chance of leading to the same favourable outcome. In a sense, the system believes that it can increase the likelihood of the opponent having 32o through the act of checking!

Although it is not proper to be ascribing causality relations, it should be clear that unconstrained belief systems can admit highly irrational beliefs. It could be argued that this is a good thing – that the program should be free to hold any belief whatsoever. For instance, the game could be rigged against us, such that the opponent really *will* produce the Ace-high flush every game. A purist would argue that there should be no impediment to forming any possible belief, since a seemingly strange belief might be entirely appropriate in response to some hypothetical opponent. However, in practice, this uncontrolled range of beliefs is a major encumbrance, and often leads to embarrassing imbalances in play. In fact, the play of VEXBOT can be compared to that of an overly emotional player, where the program's "mood" can quickly swing from elated (hyper-aggressive and over-optimistic) to dejected (very passive and pessimistic).

By initializing the system with good default data, we implicitly provide a wealth of basic knowledge about how the game is played, what kind of outcomes are possible, and the relative likelihood of each. By mixing actual observations with a larger proportion of "normal" data, we temper the interpretation of noisy input and dampen the over-reactions, without eliminating the ability to learn and adapt. Indeed, without good default seeding, it is impossible to determine just how well the program could potentially play.

Unfortunately, generating good defaults is a difficult task, and the current default data leaves much to be desired. The author's current research is investigating methods for automatically generating good default data, and much more, by producing a probabilistic map of the entire strategy space. Another interesting research topic would be to use data mining to determine good defaults empirically.

Another major issue with the current design is the problem of *data sparsity*. The structure of the imperfect information game tree provides a natural separation of contexts, providing a fine level of granularity that is both a strength and a liability. While it permits a more than adequate level of precision, it also partitions actual observations into thousands of subcases. To effect reasonable learning rates and to detect patterns with statistical significance, these distinct contexts must be combined according to greatest similarity, which is a difficult research problem in its own right. In fact, some aspects of opponent modeling are essentially impossible, in principle. On the other hand, a solution does not have to be perfect to be highly effective in practice. Indeed, it might be the case that a relatively simple system will end up out-performing all human players, possibly by a large margin.

There are many obstacles still to be overcome with this approach, some of which may be quite challenging. Nevertheless, this method for building adaptive poker players has already proven to be successful, with much greater promise for improved implementations.

6.7 Conclusions and Future Work

The many years of study in poker AI has raised as many questions as it has answered, which is a sure indication of its vitality. The domain offers many unique challenges of theoretical and practical importance to computer game-playing, and to computer science in general.

The evolution of architectures for poker programs has followed a natural progression over the past 14 years, with each new generation addressing some of the fundamental limitations encountered by the previous generation. The following summary highlights some of the main strengths and major limitations of each approach.

1. **Deterministic rule-based methods**

   Example: Wilson Software TURBO TEXAS HOLD'EM.

   Advantages: simple, intuitive, minimal CPU and memory requirements.

   Limitations: requires extensive poker knowledge, creates unavoidable gaps, predictable play, easily exploited.

2. **Probabilistic formula-based methods**

   Examples: LOKI, POKI, POKER ACADEMY.

   Advantages: intuitive, flexible, modest CPU and memory requirements.

   Limitations: requires extensive knowledge, complicated multi-component systems, difficult to build on, produces a single playing style.

3. **Monte Carlo and enhanced simulation methods**

   Examples: LOKI II, POKI, POKER ACADEMY.

   Advantages: conceptually simple, emergent strategies, less dependent on poker knowledge.

   Limitations: perfect information instances are inadequate, volatile play, compounding errors over future actions.

4. **Nash equilibrium game-theoretic methods**

   Examples: PSOPTI0 - PSOPTI7 (a.k.a. SPARBOT), BLUFFBOT, GS2

   Advantages: well-balanced overall strategy with deception, requires minimal poker knowledge.

   Limitations: crude approximation, loss of relevant context, non-exploitive (oblivious), vulnerable to systematic probing, generalization to multiplayer will be difficult.

5. **Adaptive imperfect information game-tree search**

   Example: VEXBOT.

   Advantages: theoretically correct EV calculation, adaptive, independent strategies, creative play.

   Limitations: data sparsity, requires good abstractions, volatile learning, over-reactive style, deducible belief states, gets stuck in local minima.

Not surprisingly, deterministic rule-based systems have failed to produce strong poker-playing programs. The more flexible but still relatively simple formula-based approach produces much better practical results. Although still below the skill level of a human expert, the most recent version of POKI (in the commercial software POKER ACADEMY) is the strongest known computer program for multi-player Limit Texas Hold'em.

Simulation can be used to further refine an existing formula-based system, or to make independent betting decisions. However, the results have been somewhat mixed, and the resulting play can be quite volatile. More could be done with Monte Carlo or full-information selective simulations, but the method is largely subsumed by the EV calculations in adaptive imperfect information game-tree search.

Looking at the two-player game, crude approximations of game-theoretic equilibrium strategies have produced programs like the PSOPTI family, with a vast overall improvement in well-balanced strategies. These simple stationary players are able to defeat average players, and can hold their own against strong opposition, at least until their subtle flaws are discovered.

The *Miximax* and *Miximix* algorithms for adaptive imperfect information game-tree search address the critical principles of opponent modeling and strategy adjustments. The program VEXBOT, based on this architecture, is currently the strongest computer program for two-player Limit Texas Hold'em. Although still in the relatively primitive stages of development, it eventually learns to defeat all other computer programs, and can provide a dangerous opponent for elite human players.

Many lessons have been learned through the course of this research, and future architectures will build upon these lessons. For instance, well-balanced *quasi-*

equilibrium strategies can be obtained by alternate means, such as the author's *pdf-cutting algorithm*, avoiding the crude abstractions that necessarily sacrifice valuable context.

Learning systems will need to be much faster and smoother. One method currently being investigated is to pre-define parameterized categories of player styles, and use Bayesian inference to classify opponents based on the limited observations [40]. By jumping to a best guess about the opponent's predilections, we hope to immediately find an appropriate (although not maximal) counter-strategy, and increase the exploitation rate as the classification improves over time. As long as the early strategies are not seriously counter-indicated, this should be much more effective than trying to build a model of the opponent gradually from scratch, bridging over thousands of specific contexts.

It is also becoming apparent that there will always be a need for better abstraction techniques, partitioning distinct contexts into classes for collective treatment. Ideally, these would be based on abstract distance metrics that can be applied to automated learning, without requiring a great deal of poker-specific knowledge.

The problem of assessment will always be a concern, because of the high variance inherent in the game. The DIVAT analysis technique, discussed at length in Chapter 5, does a commendable job of separating the signal from the noise, but many improvements are possible. Tools of this kind will likely become indispensable, in view of the ever-increasing number of games required to statistically discern players of nearly equal skill. Moreover, DIVAT analysis can be used directly in competition, providing better feedback on our performance than the simple money outcome. However, since we do not gain knowledge of the opponent's cards when a game is won uncontested, some care is needed to avoid introducing a significant bias.

There is a long way to go before poker programs surpass all human players. Current lines of research might be successful, or entirely new paths and new architectures might be necessary. It is clear that future systems must be able to adapt rapidly to maximize success. They must be able to cope with talented opponents who quickly and repeatedly change their style, adapting to us at the same time we are adapting to them. While these are very complex problems in principle, there is hope for pragmatic success, if only because humans are far from perfect themselves.

Many popular algorithms from other areas of AI and machine learning could be applied to poker. In fact, the game can provide a challenging test for those algorithms, because poker may not have certain "convenient" properties that make other domains considerably easier in practice. It is a worthwhile field of battle, where the best ideas can prove their worthiness in direct combat.

Regardless of the methods employed, poker will continue to provide a vibrant and demanding domain for Artificial Intelligence research for many years to come.

Bibliography

[1] D. Billings. The University of Alberta Computer Poker Research Group webage. World Wide Web, 1997.
`http://www.cs.ualberta.ca/~games/poker/.`

[2] D. Billings. The First International RoShamBo Programming Competition. *The International Computer Games Association Journal*, 23(1):42–50, 2000.

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

[4] D. Billings, N. Burch, A. Davidson, T. Schauenberg, R. Holte, J. Schaeffer, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *The Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 661–668, 2003.

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

[6] 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 H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04*, LNCS 3846, pages 21–34. Springer-Verlag GmbH, 2004.

[7] D. Billings and M. Kan. A tool for the direct assessment of poker decisions. *The International Computer Games Association Journal*, 2006. To appear.

[8] D. Billings, D. Papp, L. Peña, J. Schaeffer, and D. Szafron. Using selective-sampling simulations in poker. In *American Association of Artificial Intelligence Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, pages 13–18. American Association of Artificial Intelligence, 1999.

[9] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 493–499, 1998.

[10] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Poker as a testbed for machine intelligence research. In R. Mercer and E. Neufeld, editors, *Advances in Artificial Intelligence, AI'98*, pages 228–238. Springer-Verlag, 1998.

[11] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *American Association of Artificial Intelligence National Conference, AAAI'99*, pages 697–703, 1999.

[12] M. Buro. Solving the Oshi-Zumo game. In H. J. van den Herik, H. Iida, and E. A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges*, pages 361–366. Kluwer Academic, 2004.

[13] A. Davidson. Opponent modeling in poker. Master's thesis, Department of Computing Science, University of Alberta, 2002.

[14] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI'00*, pages 1467–1473, 2000.

[15] I. Frank and D. A. Basin. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 252(11):217–256, 2001.

[16] I. Frank, D. A. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 500–507, 1998.

[17] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games.* MIT Press, May 1998.

[18] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, August 1991.

[19] A. Gilpin and T. Sandholm. Optimal Rhode Island Hold'em poker. In *American Association of Artificial Intelligence 20th National Conference, AAAI'05*, pages 1684–1685, 2005. Intelligent Systems Demonstration.

[20] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *American Association of Artificial Intelligence National Conference, AAAI'06*, pages 1007–1013, July 2006.

[21] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *The International Joint Conference on Artificial Intelligence, IJCAI'99*, pages 584–589, 1999.

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

[23] T. Hauk, M. Buro, and J. Schaeffer. *-minimax performance in backgammon. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag GmbH, 2004.

[24] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, 1992.

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

[26] 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.

[27] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

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

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

[30] L. Peña. Probabilities and simulations in poker. Master's thesis, Department of Computing Science, University of Alberta, 1999.

[31] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[32] T. Salonen. The BLUFFBOT webage. World Wide Web, 2006. `http://www.bluffbot.com/`.

[33] J. Schaeffer, D. Billings, L. Peña, and D. Szafron. Learning to play strong poker. In *The International Conference on Machine Learning Workshop on Game Playing*. J. Stefan Institute, 1999. Invited paper.

[34] A. Selby. Optimal heads-up pre-flop hold'em. WWW, 1999. `http://www.archduke.demon.co.uk/simplex/`.

[35] B. Sheppard. *Toward Perfection of Scrabble Play*. PhD thesis, Computer Science, University of Maastricht, 2002.

[36] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.

[37] B. Sheppard. Efficient control of selective simulations. In H. J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG'04, Ramat-Gan, Israel, July 5-7, 2004. Revised Papers*, volume 3846 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag GmbH, 2004.

[38] J. Shi and M. Littman. Abstraction methods for game theoretic poker. In T. A. Marsland and I. Frank, editors, *Computers and Games 2000*, LNCS 2063, pages 333–345. Springer-Verlag, 2002.

[39] C. Smith and M. Zinkevich. The AAAI'06 poker competition webage. World Wide Web, 2006. `http://www.cs.ualberta.ca/~pokert/`.

[40] 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'05)*, pages 550–558, July 2005.

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

[42] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[43] Wikipedia. Game theory. Wikipedia: The Free Online Encyclopedia. `http://en.wikipedia.org/wiki/Game_theory`.

[44] Wikipedia. Nash equilibrium. Wikipedia: The Free Online Encyclopedia. `http://en.wikipedia.org/wiki/Nash_equilibrium`.

[45] B. Wilson. Wilson software. WWW, 1993. `wilsonsoftware.com`.

[46] M. Zinkevich, M. Bowling, N. Bard, M. Kan, and D. Billings. Optimal unbiased estimators for evaluating agent performance. In *American Association of Artificial Intelligence National Conference, AAAI'06*, pages 573–578, 2006.

Appendix A: Glossary of Poker Terms

This appendix contains informal definitions of common poker terms used in this dissertation. More extensive and precise poker glossaries are available on the world wide web, such as `http://www.seriouspoker.com/dictionary.html`, `http://conjelco.com/pokglossary.html`, and the online Wikipedia, `http://en.wikipedia.org/wiki/Poker_jargon`.

- ***All-in***. To have one's entire stake committed to the current pot. Action continues toward a ***side pot***, with the all-in player being eligible to win only the main pot.
- ***All-in Equity***. The ***expected value*** income of a hand assuming the game will proceed to the ***showdown*** with no further betting (*i.e.*, a fraction of the current pot, based on all possible future outcomes).
- ***Bad Beat***. An unlucky loss. In particular, losing a game where the opponent probably should have folded, but instead got extremely lucky to win.
- ***Bet***. To make the first wager of a betting round (compare ***raise***).
- ***Bet for Value***. To ***bet*** with the expectation of winning if ***called*** (compare ***bluff***).
- ***Big Bet***. The largest bet size in Limit poker (*e.g.*, \$20 in \$10-\$20 Hold'em).
- ***Big Blind*** (sometimes called the ***Large Blind***). A forced ***bet*** made before the deal of the cards (*e.g.*, \$10 in \$10-\$20 Hold'em, posted by the second player to the left of the ***button***).
- ***Blind***. A forced ***bet*** made before the deal of the cards (see ***small blind*** and ***big blind***).
- ***Bluff***. To play a weak hand as though it were strong, with the expectation of losing if ***called*** (see also ***semi-bluff*** and ***pure bluff***, compare ***bet for value***).
- ***Board*** (or ***Board Cards***). The ***community cards*** shared by all players.
- ***Board Texture***. Classification of the type of ***board***, such as having lots of high cards, or not having many draws (see ***dry***).
- ***Button***. The last player to act in each betting round in Texas Hold'em. Also called the ***dealer button***, representing the person who would be the dealer in a home game.

- ***Call***. To match the current level of betting. If the current level of betting is zero, the term ***check*** is preferred.

- ***Cap***. (a) The maximum number of raises permitted in any single round of betting (typically four in Limit Hold'em, but occasionally unlimited). (b) *(vt)* To make the last permitted raise in the current betting round (*e.g.*, after a ***bet***, ***raise***, and ***re-raise***, a player *caps* the betting).

- ***Check***. To decline to make the first wager of a betting round (compare ***call***).

- ***Check-Raise***. To ***check*** on the first action, with the intention of ***raising*** in the same betting round after an opponent ***bets***.

- ***Community Cards***. The public cards shared by all players.

- ***Connectors***. Two cards differing by one in rank, such as 7-6. More likely to make a straight than other combinations.

- ***Dominated***. A Hold'em hand that has a greatly reduced chance of winning against another because one or both cards cannot make a useful pair (*e.g.*, KQ is dominated by AK, AQ, AA, KK, and QQ, but not by AJ or JJ).

- ***Draw***. A holding with high potential to make a strong hand, such as a *straight draw* or a *flush draw* (compare ***made hand***).

- ***Draw Potential***. The relative likelihood of a ***hand*** improving to be the best if it is currently behind.

- ***Drawing Dead***. Playing a ***draw*** to a hand that will only lose, such as drawing to a flush when the opponent already holds a full house.

- ***Drawing Hand***. A ***hand*** that has a good ***draw*** (compare ***made hand***).

- ***Dry***. Lacking possible draws or betting action, as in a *dry board* or a *dry game*.

- ***Equity*** (or ***Pot Equity***). An estimate of the ***expected value*** income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (*e.g.*, ***all-in equity***).

- ***Expected Value*** (EV) (also called ***mathematical expectation***). The average amount one expects to win in a given game situation, based on the payoffs for each possible random outcome.

- ***Flop***. The first three ***community cards*** dealt in Hold'em, followed by the second betting round (compare ***board***).

- ***Fold***. To discard a hand instead of matching the outstanding ***bet***, thereby losing any chance of winning the pot.

- ***Fold Equity***. The ***equity*** gained by a player when an opponent folds. In particular, the positive equity gained despite the fact that the opponent's fold was entirely correct.

- ***Forward Blinds***. The logical extension of ***blinds*** for ***heads-up*** (two-player) games, where the first player posts the ***small blind*** and the second player (***button***) posts the ***big blind*** (compare ***reverse blinds***). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).

- ***Free-Card Danger***. The risk associated with allowing an opponent to improve and win the ***pot*** without having to ***call*** a ***bet*** (in particular, when they would have ***folded***).

- ***Free-Card Raise***. To ***raise*** on the ***flop*** intending to ***check*** on the ***turn***.

- ***Game***. (a) A competitive activity in which players contend with each other according to a set of rules (in poker, a contest with two or more players). (b) A single instance of such an activity (in poker, from the initial dealing of the cards to the ***showdown***, or until one player wins uncontested).

- ***Game Theory***. Among serious poker players, game theory normally pertains to the optimal calling frequency (in response to a possible bluff), or the optimal bluffing frequency. Both depend only on the size of the ***bet*** in relation to the size of the ***pot***.

- ***Hand***. (a) A player's private cards (*e.g.*, two ***hole cards*** in Hold'em). (b) One complete ***game*** of poker (see ***game*** (b)).

- ***Heads-up***. A two-player (head-to-head) poker game.

- ***Hole Card***. A private card in poker (Texas Hold'em, Omaha, 7-Stud, etc.).

- ***Implied Odds***. (a) The ***pot odds*** based on the probable future size of the pot instead of the current size of the pot (positive or negative adjustments). (b) The extra money a strong hand stands to win in future betting rounds (compare ***reverse implied odds***).

- ***Kicker***. A side card, often deciding the winner when two hands are otherwise tied (*e.g.*, a player holding Q-J when the ***board*** is Q-7-4 has ***top pair*** with a Jack *kicker*).

- ***Large Blind*** (usually called the ***Big Blind***). A forced ***bet*** made before the deal of the cards (*e.g.*, $10 in $10-$20 Hold'em, posted by the second player to the left of the ***button***).

- ***Loose Game***. A game having several ***loose players***.

- ***Loose Player***. A player who does not fold often (*e.g.*, one who plays most ***hands*** at least to the ***flop*** in Hold'em).

- ***Made Hand***. A ***hand*** with a good chance of currently being the best, such as ***top pair*** on the ***flop*** in Hold'em (compare ***draw***).

- ***Mixed Strategy***. Handling a particular type of situation in more than one way, such as to sometimes ***call***, and sometimes ***raise***.

- ***Offsuit***. Two cards of different suits (also called ***unsuited***, compare ***suited***).

- ***Open-Ended Draw***. A ***draw*** to a straight with eight cards to make the straight, such as 6-5 with a ***board*** of Q-7-4 in Hold'em.

- ***Outs***. Cards that will improve a hand to a probable winner (compare ***draw***).
- ***Pocket Pair***. Two cards of the same rank, such as 6-6. More likely to make three of a kind than other combinations (see ***set***).
- ***Post-flop***. The actions after the ***flop*** in Texas Hold'em, including the ***turn*** and ***river*** cards interleaved with the three betting rounds, and ending with the ***showdown***.
- ***Pot***. The common pool of all collected wagers during a ***game***.
- ***Pot Equity*** (or simply ***Equity***). An estimate of the ***expected value*** income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (*e.g.*, ***all-in equity***).
- ***Pot Odds***. The ratio of the size of the pot to the size of the outstanding bet, used to determine if a ***draw*** will have a positive ***expected value***.
- ***Pre-flop***. The first round of betting in Texas Hold'em before the ***flop***, beginning with the posting of the ***blinds*** and the dealing of the private ***hole cards***.
- ***Pure bluff***. A ***bluff*** with a hand that can only win if the opponent ***folds*** (compare ***semi-bluff***).
- ***Pure Drawing Hand***. A weak ***hand*** that can only win by completing a ***draw***, or by a successful ***bluff***.
- ***Raise***. To increase the current level of betting. If the current level of betting is zero, the term ***bet*** is preferred.
- ***Raising for a Free-card***. To ***raise*** on the ***flop*** intending to ***check*** on the ***turn***.
- ***Rake***. A portion of the ***pot*** withheld by the casino or host of a poker game, typically a percentage of the pot up to some maximum, such as 5% up to $3.
- ***Re-raise***. To increase to the third level of betting after a ***bet*** and a ***raise***.
- ***Reverse Blinds***. A special rule sometimes used for ***heads-up*** (two-player) games, where the second player (***button***) posts the ***small blind*** and the first player posts the ***big blind*** (compare ***forward blinds***). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).
- ***Reverse Implied Odds***. The unaccounted (negative) money a mediocre hand stands to lose in future betting rounds (compare ***implied odds*** (b)).
- ***River***. The fifth ***community card*** dealt in Hold'em, followed by the fourth (and final) betting round.
- ***Semi-bluff***. A ***bluff*** when there are still cards to be dealt, with a hand that might be the best, or that has a reasonable chance of improving to the best if it is ***called*** (compare ***pure bluff***).
- ***Second pair***. Matching the second highest ***community card*** in Hold'em, such as having 7-6 with a ***board*** of Q-7-4.
- ***Session***. A series of ***games***, typically lasting several hours in length.

- ***Set***. Three of a kind, formed with a ***pocket pair*** and one card of matching rank on the ***board***. A very powerful and well-disguised hand (compare ***trips***).
- ***Short-handed Game***. A game with less than the full complement of players, such as a Texas Hold'em game with five or fewer players.
- ***Showdown***. The revealing of cards at the end of a ***game*** to determine the winner.
- ***Side pot***. A second pot for the remaining active players after another player is ***all-in***.
- ***Slow-play***. To ***check*** or ***call*** a strong hand as though it were weak, with the intention of ***raising*** in a later betting round (compare ***smooth-call*** and ***check-raise***).
- ***Small Bet***. The smallest bet size in Limit poker (*e.g.*, $10 in $10-$20 Hold'em).
- ***Small Blind***. A forced ***bet*** made before the deal of the cards (*e.g.*, $5 in $10-$20 Hold'em, posted by the first player to the left of the ***button***).
- ***Smooth-call***. To only ***call*** a ***bet*** instead of ***raising*** with a strong hand, for purposes of deception (as in a ***slow-play***).
- ***Suited***. Two cards of the same suit, such as both Hearts. More likely to make a flush than other combinations (compare ***offsuit*** or ***unsuited***).
- ***Table Image***. The general perception other players have of one's play.
- ***Table Stakes***. A poker rule allowing a player who cannot match the outstanding bet to go ***all-in*** with his remaining money, and proceed to the ***showdown*** (also see ***side pot***).
- ***Texture of the Board***. Classification of the type of ***board***, such as having lots of high cards, or not having many draws (see ***dry***).
- ***Tight Player***. A player who usually folds unless the situation is clearly profitable (*e.g.*, one who folds most ***hands*** before the ***flop*** in Hold'em).
- ***Time Charge***. A fee charged to the players in a poker game by a casino or other host of the game, typically collected once every 30 minutes.
- ***Top Pair***. Matching the highest ***community card*** in Hold'em, such as having Q-J with a ***board*** of Q-7-4.
- ***Trap***. To play a strong hand as though it were weak, hoping to lure a weaker hand into ***betting***. Usually a ***check-raise***, or a ***slow-play***.
- ***Trips***. Three of a kind, formed with one ***hole card*** and two cards of matching rank on the ***board***. A strong hand, but not well-disguised (compare ***set***).
- ***Turn***. The fourth ***community card*** dealt in Hold'em, followed by the third betting round.
- ***Unsuited***. Two cards of different suits (also called ***offsuit***, compare ***suited***).
- ***Value Bet***. To ***bet*** with the expectation of winning if ***called*** (compare ***bluff***).
- ***Wild Game***. A game with a lot of raising and re-raising. Also called an ***action game***.