University of Alberta

Library Release Form

Name of Author: Denis Richard Papp

Title of Thesis: Dealing with Imperfect Information in Poker

Degree: Master of Science

Year this Degree Granted: 1998

. . . . . .

Denis Richard Papp
2036 - 104A st
Edmonton, Alberta
Canada, T6J-5K4

Date: . . . . . .

University of Alberta

Dealing with Imperfect Information in Poker

by

Denis Richard Papp

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

Department of Computing Science

Edmonton, Alberta
Fall 1998

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Dealing with Imperfect Information in Poker** submitted by Denis Richard Papp in partial fulfillment of the requirements for the degree of **Master of Science**.

. . . . . .
Jonathan Schaeffer

. . . . . .
Duane Szafron

. . . . . .
Oliver Schulte

Date: . . . . . .

Abstract

Poker provides an excellent testbed for studying decision-making under conditions of uncertainty. There are many benefits to be gained from designing and experimenting with poker programs. It is a game of imperfect knowledge, where multiple competing agents must understand estimation, prediction, risk management, deception, counter-deception, and agent modeling. New evaluation techniques for estimating the strength and potential of a poker hand are presented. This thesis describes the implementation of a program that successfully handles all aspects of the game, and uses adaptive opponent modeling to improve performance.

Acknowledgements

The author would like to acknowledge the following:

- A big thanks to Jonathan Schaeffer, for being an excellent supervisor.
- Darse Billings, for all his time, ideas and insightful input.
- Duane Szafron, for contributing to the poker research group.
- The National Sciences and Engineering Research Council, for providing financial support.

As well as the following people who helped make things easier (with or without their knowledge):

- The authors of the fast poker hand evaluation library: Clifford T. Matthews, Roy T. Hashimoto, Keith Miyake and Mat Hostetter. Without this library the implementation would certainly have been slower. ftp://ftp.csua.berkeley.edu/pub/rec.gambling/poker/poker.tar.gz.
- Todd Mummert, the author of the dealer programs that make the IRC poker server possible. http://www.cs.cmu.edu/People/mummert/public/ircbot.html.
- Michael Maurer, the author of the PERL player program that was the basis of *Loki*'s interface to the IRC dealer program. http://nova.stanford.edu/~maurer/r.g/.
- All the other original authors who helped pioneer interest in IRC poker-playing programs and provided us with some additional data: Greg Wohletz, Stephen How, Greg Reynolds and others. An extra thanks to Greg Reynolds for the very handy Windows poker client (GPkr), which made watching *Loki* play a lot more user-friendly. http://webusers.anet-stl.com/~gregr/.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Game playing is an ideal environment for examining complex topics in machine intelligence because games generally have well-defined rules and goals. Additionally, performance, and therefore progress, is easily measured. However, the field of computer game playing has traditionally concentrated on studying chess, and other two-player deterministic zero-sum games with perfect information, such as checkers and Othello. In these games, players always have complete knowledge of the entire game state since it is visible to both participants. High performance has been achieved with brute-force search of the game trees, although there are some exceptions, such as the game of Go where the game tree is far too large. Although many advances in computer science (especially in searching) have resulted, little has been learned about decision-making under conditions of uncertainty. To tackle this problem, one must understand estimation, prediction, risk management, the implications of multiple opponents, deception, counter-deception, and the deduction of decision-making models of other players.

Such knowledge can be gained by studying imperfect information games, such as bridge and poker, where the other players' cards are not known and search alone is insufficient to play these games well. Poker in particular is a popular and fascinating game. It is a multi-player zero-sum game with imperfect information. The rules are simple but the game is strategically complex. It emphasizes long-term money management (over a session of several contiguous interdependent games), as well as the ability to recognize the potential of one specific game and to either maximize gain or minimize loss. Most games are analogical to some aspect of real life, and poker can be compared to "policy decisions in commercial enterprises and in political campaigns" [8].

Poker has a number of attributes that make it an interesting domain for research. These include multiple competing agents (more than two players), imperfect knowledge (your opponents hold hidden cards), risk management (betting strategies and their consequences), agent modeling (detecting and exploiting patterns or errors in the play of other players), deception (bluffing and varying your style of play), and dealing with unreliable information (your opponents also make deceptive plays). All of these are challenging dimensions to a difficult problem.

Certain aspects of poker have been extensively studied by mathematicians and economists. There are two main approaches to poker research. One approach is to use simplified variants that are easier to analyze [10] [11] [12]. For example, one could use only two players or constrain the betting rules. However, one must be careful that the simplification does not remove the challenging components of the problem. The other approach is to pick a real variant, but to combine mathematical analysis, simulation and *ad hoc* expert experience. Expert players, often with mathematical skills, are usually involved in this approach [13] [14] [15].

However, little work has been done by computing scientists. Nicolas Findler worked on and off for 20 years on a poker-playing program for Five-Card Draw [6] [7] [8], however he focused on simulating the thought processes of human players and never achieved a program capable of defeating a strong player. Koller and Pfeffer [10] have investigated poker from a theoretical point of view. They implemented the first practical algorithm for finding optimal randomized strategies in two-player imperfect information competitive games. However, such a system will likely not win much more from a set of bad players than a set of perfect players, failing to exploit the property that human players make many mistakes (*i.e.* it presumes the opponent always plays the best strategy).

One of the interesting aspects of poker research is that opponent modeling can be examined. It has been attempted in two-player games, as a generalization of minimax, but with limited success [5] [9]. Part of the reason for this is that in games such as chess, opponent modeling is not critical for computers to achieve high performance. In poker, it is essential for the best results. Working under the assumption that our opponents will make mistakes and exhibit predictability, opponent modeling should be accounted for and built into the program framework.

Although our long-term goal is to produce a high-performance poker program that is capable of beating the best human players, for our first step we are interested in constructing a framework with useful computer-oriented techniques. It should minimize human expert information and easily allow the introduction of an opponent modeling system, and still make a strong computer poker program. If we are successful, then the insights we gain should have wide applicability to other applications that require similar activities.

We will present new enumeration techniques for determining the strength and potential of a player's hand, will demonstrate a working program that successfully plays 'real' poker, and demonstrate that using opponent modeling can result in a significant improvement in performance. Chapter 2 will introduce terminology (there is a glossary in Appendix C) and will give the rules of poker and of Texas Hold'em (the poker variant we have chosen to study). Chapter 3 describes how humans play poker. Chapter 4 discusses various ways to approach the problem using a computer, and details the architecture we have selected. Chapter 5 describes the enumeration techniques we use for hand evaluation. Chapter 6 describes our betting strategy. Chapter 7 details the opponent modeling system. Chapter 8 discusses the experimental system and some results. Chapter 9 discusses the conclusions and future work.

Parts of Chapters 5 and 6 have been published in *Advances in Artificial Intelligence*

[4], and parts of Chapter 7 have been published in *AAAI-98* [3]. Our poker-playing program is called *Loki* and has been demonstrated at *AAAI-98*. It is written in C and C++ and runs with real-time constraints (in typical play, an action should not take more than a few seconds). The primary mechanism for performance testing is self-play, however we also play against human opponents through an Internet poker server. The interface between the program and the server is written in PERL.

Chapter 2

Poker

Poker is a set of multi-player card games (standard deck of 52 cards) that is typically played as a session consisting of a sequential series of multiple *games* (sometimes called *deals* or *hands*). Each player begins the session with a certain amount of *chips* (equated to money). Poker is a zero-sum game (one player's gain is another's loss) where the long-term goal is to net a positive amount of chips. This is accomplished by maximizing winnings in each individual game within the session.

There are numerous variants of poker. This chapter covers the basic structure that defines the majority of these variants, and describes the specific variant of *Texas Hold'em*.

A note on symbols:

- We use a standard deck of 52 cards (4 suits and 13 ranks per suit).
- For card ranks, we use the symbols 2 (Deuce), 3 (Trey), 4 (Four), 5 (Five), 6 (Six), 7 (Seven), 8 (Eight), 9 (Nine), T (Ten), J (Jack), Q (Queen), K (King), and A (Ace).
- For card suits, we use the symbols ♢ (Diamonds), ♣ (Clubs), ♡ (Hearts), and ♠ (Spades).
- A single card is represented by a pair of symbols, *e.g.* 2♢ (Deuce of Diamonds) and T♣ (Ten of Clubs).
- A set of cards is represented by a list separated by dashes, *e.g.* 4♣-5♣-6♣ (Four, Five and Six of Clubs).

2.1 Playing a Game

Each game is composed of several rounds, which in turn involve dealing a number of cards followed by betting. This continues until there is either only one *active* player left in the game or when all the betting rounds have been completed. In the latter case, the game then enters the *showdown* to determine the winner(s).

Betting involves each player contributing an equal amount of money to the *pot*. This amount grows as the game proceeds and a player may *fold* at any time, which means they lose the money they have already contributed and are no longer eligible to win the pot.

When all players fold but one, the remaining player wins the pot. Otherwise, if the game proceeds to a showdown, the highest ranked set of cards held by a player (the highest *hand*) wins the pot (ties are possible, in which case the pot is split). Note that poker is full of ambiguous terminology; for example, the word *hand* refers both to one game of poker and to a player's cards.

2.1.1 Ante

Before the initial deal, participating players are required to blindly contribute a fixed amount of money (*ante*) to the pot. In some variants an alternative system is used where some of the players immediately following the rotating dealer (called the *button*) are forced to put in fixed size bets (called the *blinds*). Without these forced bets, risk can be minimized by only playing the best hand, and the game becomes uninteresting.

2.1.2 The Deal

Each round begins by randomly dealing a number of cards (the *non-deterministic* element of poker). In some variants these are *community cards* which are shared by all players. Each player receives the same number of cards – each of which is either *face-down* (known only to this player) or *face-up* (known to all players). There are other possible dealing steps such as *drawing* (discarding and replacing face-down cards) and *rolling* (revealing some face-down cards). The face-down cards are the *imperfect information* of poker. Each player knows their own cards but not those of their opponents.

A variant can be defined by a *script* which specifies the number of rounds and what dealing actions are to be taken at each round. This script has one entry for each round in the variant (recall there is also a series of betting that occurs at the end of each round). Here are the scripts for some well-known poker variants:

Five-Card Draw (2 betting rounds):

- deal 5 cards face-down to each player
- each player discards 0-3 cards and receives the same number of new face-down cards.

Seven-Card Stud (5 betting rounds):

- 2 cards face-down and 1 face-up to each player
- 1 face-up to each player
- 1 face-up to each player

- 1 face-up to each player
- 1 face-down to each player

2.1.3 Betting

The betting portion of poker is a multi-round sequence of player actions until some termination condition is satisfied. Without it your probability of winning the game depends solely on nature (the deal of the cards). Betting increases the pot and indirectly gives information about players and their hands. When playing against good opponents who pay attention to the actions of the other players, betting can also be used to give misinformation (the element of deception in poker).

Betting Order

The players are in a fixed seating order around the *table* (even in a virtual environment the set of players is referred to as the *table*). The dealer button rotates around in a clockwise fashion, as do betting actions. Betting always begins with the *first to act*, which in most games is the first active player following the button (in stud games, which have face-up cards, it is usually the player with the highest ranked cards showing). Betting proceeds around the table, involving all active players, but does not end at the last active player.

Termination Condition

Betting continues sequentially around the table until all active players have contributed an equal amount to the pot (or until there is only one active player remaining). The game then proceeds to the next round (as defined by the script). In the final round the remaining players enter the showdown.

Note that all players must be given at least one opportunity to act before betting is terminated (this allows for the case where all players have equally contributed \$0). Being forced to put a blind bet in the pot does not count as having had an opportunity to act. Also, there often is a limit on the number of raises (increments to the contribution amount) which artificially forces an end to the betting.

Betting Actions

In most situations, each player has 3 different actions to choose from. Each action directly affects the number of active players, the size of the pot, and the required contribution to remain active in the game. Here are the 3 action categories and 5 different actions that fit into those categories:

- **Fold**: Drop from the game (become inactive). A player who folds is no longer eligible to win the pot. The player is now out of the current game and loses the money that has been contributed to the pot.

- **Call:** Match the current per-player contribution (*e.g.* if player **A** has contributed \$12 (the most) and player **B** \$8, then **B** must place an additional \$4, the *amount to call*, in the pot). A **check** is a special case of **calling** when the current amount to call is \$0 (which is usually the case with the first active player). It means you forego opening the betting for the round.

- **Raise:** Increase the current per-player contribution (*e.g.* if player **A** has contributed \$12 and player **B** \$8, then **B** can put \$8 into the pot (a raise of \$4) to make the required contribution \$16). A **bet** is a special case of **raising** when the current amount to call is \$0. It means you open the betting for the round.

At any point in the game a player will have three actions available: fold/check/bet or fold/call/raise. An exception occurs in games with blinds where a player was forced to blind bet and everyone else calls or folds. Since the player was not originally given a choice, they are given an option to raise when the action returns to them (the amount to call is \$0). The available actions are fold/check/raise; check because it is \$0 to call and raise because there has already been a bet (the blind). Another exception can occur because there is normally a limit of 4 raises (including the initial bet or blind). If it is a player's turn and the betting has already been *capped* (no more raises allowed) the available actions are fold/call.

Betting Amounts

There are many different ways to restrict the betting amounts and the various systems can produce very different games (requiring different strategies).

- **No-limit poker**: this is the format used for the World Series of Poker championship main event. The amount a player is allowed to bet/raise is limited only by the amount of money that they have.

- **Pot-limit poker**: this format normally has a minimum amount and the maximum raise is whatever is currently in the pot (*e.g.* if the pot is at \$50 and the amount to call is \$20, a player can at most raise \$70 by placing \$90 in the pot).

- **Spread limit**: a format commonly used in friendly games. There is both a fixed minimum and maximum in each round (*e.g.* 1-5 Stud is a game where the raise or bet size can be between \$1 and \$5 in any round).

- **Fixed limit**: a common format used in casinos. There is a fixed bet size in each round (same as spread limit with the minimum equal to the maximum). Usually the bet size is larger in the later rounds. Games that feature the same bet size in all rounds are called **flat limit**.

2.1.4 Showdown

When there is only one active player remaining in the game, that player wins the pot without having to reveal their cards. Otherwise, when the final round terminates

normally, the game enters the *showdown* where all active players reveal their cards and the player with the strongest 5-card hand wins the pot. In the case of a tie, the pot is split evenly.

Note that individual cards are ranked from Deuce – the lowest – to Ace. However, in most games, Ace can optionally be used as a low card (comes before Deuce instead of after King). The suit is not used in ranking (but is sometimes used for other purposes, such as awarding an odd chip when splitting the pot in ties). Here are all the 5-card hands ranked from strongest to weakest:

- **Straight Flush**: (*e.g.* 9♡-8♡-7♡-6♡-5♡) The strongest hand in regular poker – 5 cards that form both a straight and a flush (see below). Straight flushes are ranked by the top card in the straight (note that if Ace is used as low in an Ace to Five straight flush then the Five is the top card). An Ace-high straight flush (the highest possible hand) is called a Royal Flush.

- **Four of a Kind**: (*e.g.* K♠-K♡-K♢-K♣-3♠) 4 cards of identical rank and one unmatched *kicker*. Compare four of a kinds by the rank of the 4 matched cards. The kicker is used to break ties (note that in games with *community cards*, like Texas Hold'em, it is possible for multiple players to hold the same four of a kind).

- **Full House**: (*e.g.* 4♠-4♡-4♢-J♡-J♣) 3 cards of one rank and 2 cards paired but of another rank. Compare full houses first by the triple and then the pair in the event of a tie.

- **Flush**: (*e.g.* A♢-K♢-8♢-7♢-6♢) 5 cards of identical suit. Rank multiple flushes first by comparing the top card, and then each subsequent card (*e.g.* A♣-K♣-9♣-5♣-2♣ is better than A♢-K♢-8♢-7♢-6♢).

- **Straight**: (*e.g.* J♣-T♡-9♢-8♢-7♠) 5 cards in sequence. Straights are ranked by the highest card.

- **Three of a Kind**: (*e.g.* 5♡-5♢-5♣-T♠-7♣) 3 cards of one rank with 2 kickers of unmatched rank. First compare the rank of the triple, and then examine each kicker (the higher one first).

- **Two Pair**: (*e.g.* A♠-A♣-8♠-8♣-9♢) 2 cards of one rank, 2 card of another, and one kicker of a third rank. Always compare by the highest pair first, then the second pair, and finally use the kicker.

- **One Pair**: (*e.g.* Q♢-Q♣-K♢-8♢-2♡) 2 cards of one rank with 3 kickers of unmatched rank (compare by the rank of the pair and then examine each kicker in order from the highest to the lowest).

- **High Card**: (*e.g.* K♠-J♡-T♢-9♣-3♢) 5 unmatched cards. Compare by the highest to lowest cards, like a flush.

Some variants of poker recognize other special hand types (*e.g.* 4-card flush) and allow *wild cards* (cards that can represent any other card), but these are not common in casino games.

2.2 Texas Hold'em

The specific variant under consideration in this thesis is Texas Hold'em, the most popular variant played in casinos. It is used in the main event of the annual World Series of Poker championship to determine the World Champion. It is considered to be one of the most strategically complex poker variants and has "the smallest ratio of luck to skill" [2]. The script for Texas Hold'em is as follows (each of the four rounds is followed by betting):

- *Pre-flop*: each player is dealt two face-down cards (*hole cards*).
- *Flop*: 3 cards dealt face-up to the *board* (community cards).
- *Turn*: 1 card dealt face-up to the *board*.
- *River*: 1 card dealt face-up to the *board*.

After the betting on the river, the best 5-card hand formed from the two hole cards and five board cards wins the pot.

Specifically, we examine the game of Limit Texas Hold'em with a structured betting system of 2-2-4-4 units, and blinds of 1 and 2 units. This means that the bet size is fixed at 2 (the *small bet*) for the pre-flop and flop, and 4 (the *big bet*) for the turn and river. Before the pre-flop, the first player after the button is the *small blind* and is forced to put 1 in the pot, and the subsequent player is the *big blind* and forced to bet 2 (meaning the amount to call is 2 for all subsequent callers, 1 for the small blind, and if there has been no raise, the big blind has the *option* to fold, check or raise to 4 units). Limit Hold'em is typically played with 8 to 10 players, although the minimum is 2 and possible maximum is 23.

In later chapters, we use a special convention for representing Texas Hold'em hands. The designation 8♢-J♡/4♣-5♣-6♣ represents hole cards of 8♢-J♡ with a board of 4♣-5♣-6♣.

Chapter 3

How Humans Play Poker

There have been many books written on how to play poker. However, these are intended for the development of human players and must be reinterpreted to be applicable to computer play. The author typically presents a small number of rules for human players to follow. These rules are frequently based on experience and sometimes also have a mathematical foundation. For example, in his book, Norman Zadeh uses mathematical analysis to deduce a series of generalized rules for several poker variants [15]. His rules all basically follow the form of giving the reader a threshold hand type to take a certain action in a certain situation.

Two of the more useful books for the purposes of this thesis are [14] and [13]. The first book, *Hold'em Poker for Advanced Players* by David Sklansky and Mason Malmuth, presents a high-level strategy guide for the game of Texas Hold'em (which only recently has become the focus of poker literature), with a special treatise on playing the pre-flop. It presents a strong rule-based approach with an emphasis that knowledge of your opponent should always be taken into account. The second book, *The Theory of Poker* by David Sklansky, is described by Darse Billings as "the first book to correctly identify many of the underlying strategic principles of poker" [2] and uses illustrated examples from several variants including Texas Hold'em. In this chapter, some of the more important concepts and strategies are described.

3.1 Hand Strength and Potential

A human player should be able to estimate the probability that a certain set of cards will win. This is implicit in the rule-based systems which give threshold hands for betting decisions.

There are two different measures for the goodness of a hand: the *potential* and the *strength*. Potential is the probability that the player's hand will become the likely winning hand (a 4 card flush counts for nothing but is very strong if a fifth suited card is dealt). It can easily be roughly estimated by humans – good players are usually able to estimate a hand's potential accurately, in terms of *outs*: "the number of cards left in the deck that should produce the best hand" [14]. In contrast, strength is the probability of currently being in the lead (would win if no further cards were dealt).

This is often based on experience or knowledge of the statistical distribution of hand types, although knowledge of one's opponents is used by expert players to get a much more accurate estimate.

Knowing where one stands with respect to these measures is used to determine an appropriate strategy, such as raising to reduce the number of opponents, trying to scare one's opponents into folding by betting aggressively, and so on.

3.2 Opponent Modeling

The better players are at understanding how their opponents think, the more successful they will be. Experts are very good at characterizing their opponents and exploiting weaknesses in their play, and knowing when they do or do not have the best hand. They often try to *put an opponent on* a certain range of hands (guess the cards they hold) based on observed actions. It is important to note here that, if a player wins a game uncontested (no showdown), they do not have to reveal their cards. The showdown (exposure of an opponent's hidden cards) gives away information that can be used with the betting history to infer the decision-making process.

To take a less specific approach, one can estimate probabilities for a generic (or "reasonable") opponent. However, observing an opponent's play may give you useful information that allows you to bias the probabilities, allowing for more informed (and more profitable) decisions. For example, an observant player is less likely to take a bet seriously from someone who bets aggressively every game. Good opponent modeling is vital to having a good estimate of hand strength and potential.

3.3 Position

Another variable expert players take into account for a betting decision is their position at the table with respect to the dealer (how many players have acted before you and how many act after you). In [14] the authors emphasize that a later position is better because you have more information available before you must make a decision. Their pre-flop strategy is dependent on a player's expected position in the later betting rounds. For example, they discuss a tactical raise, called "buying the button", which is used in late position in the pre-flop to hopefully scare away the players behind you to become the last player to act in future betting rounds.

3.4 Odds

This is a fundamental concept introduced in [13] and includes pot odds, effective odds, implied odds and reverse implied odds. Odds gives you a way to compare your cost versus the potential winnings, and determine how good of a hand, in terms of potential or strength, you require to call a bet (or what the expected return is for each of your possible actions).

3.4.1 Pot Odds

Also called *immediate odds*, pot odds are the ratio of money in the pot against the cost to call. For example, if there are \$12 in the pot and it costs \$4 to call then you are getting 3-to-1 odds (*winnings*-to-*"cost to stay in"*). This can be translated to a percentage, representing the size of your contribution in the new pot, by using the following formula:

$$\text{"}winnings - to - cost\text{"} \sim \frac{cost}{(pot\_size + cost)}. \tag{3.1}$$

This percentage is the required probability of winning. If you are on the final round of betting then these are the odds you should have of winning the hand.

Continuing the example, the required probability is $4/(12+4) = 0.25$. Hence, you need at least a 25% chance of winning to warrant a call. For example, if your hand was 4♢-8♢ and the board was 7♢-A♢-6♣-K♡ you would have a four-card diamond flush on the turn. You would estimate having 9 *outs* of the remaining 46 cards to make a winning diamond flush. This translates to a hand potential of $9/46 = 0.196$ so it would be incorrect to call. On the other hand, you also have an *inside straight draw* (any 5 would give you a straight) and this is an additional 3 outs (the 5♢ has already been counted). Now your potential is $12/46 = 0.261$ so it is correct to call.

However, there are several caveats. Simply making the call does not necessarily end the round in a multi-player scenario; if there is a player behind you who has yet to see the bet, they may raise. In the above example, if you were expecting the player behind you to raise another \$4 and the original bettor to call, then your pot odds are now 5-to-2 (pay \$8 to win \$20), elevating the threshold for staying in the hand to $8/(20 + 8) = .286$. Also, knowledge of your opponents is not only required for an accurate estimate of hand strength or potential, but also to determine if you can expect to have to pay more. When considering potential this also assumes that the cards you are hoping for will make your hand the winner and not the second best. Further complications arise when there is more than one card left to be dealt.

3.4.2 Implied Odds and Reverse Implied Odds

Implied odds (and reverse implied odds) are based on the possibility of winning (or losing) more money later in the hand. They consider the situation after the next cards have been dealt and explain situations where things are better (or worse) than pot odds make them seem. Put another way, implied odds is the ratio between the amount you expect to win when you make your hand (more than what is in the pot) versus the amount it will cost to continue playing. In contrast, reverse implied odds is the ratio between the amount in the pot (what you win if your opponent does not make their hand) versus what it will cost you to play until the end of the hand. One of the major factors behind considering implied odds is how hidden your hand is (how uncertain your opponent is of your hand); another is the size of future bets. For the latter reason, implied odds become more important in no-limit and pot-limit games than in fixed-limit games.

As an example of implied odds, consider that at the turn there is \$12 in the pot, it is \$4 to call (pot odds 3-to-1), hitting your hand means you very likely will win, and additionally your opponent is likely play to the showdown. If you miss you will simply fold (costing \$4). If you hit you can expect to make an extra bet of \$4 from your opponent, winning \$16 total so your implied pot odds are 4-to-1.

For reverse implied odds, consider that you have a strong hand but little chance of improving and your opponent has a chance of improving to a hand stronger than yours, or possibly already has a hand stronger than yours (they have been betting and you are not sure if they are bluffing) – essentially a situation where you are not certain that you have the best hand. Say it is the turn and there is \$12 in the pot and it is \$4 to call (pot odds 3-to-1). If your opponent has a weak hand or misses their card they may stop betting in which case you would only win \$12 (it costs \$4 to find out you are winning). Otherwise, you have committed to playing to the end of the hand in which case it would cost you \$8 to find out you are losing (pot odds 3-to-2). There are many variations to this scenario. The essential idea is that reverse implied odds should be considered when you are not certain you have the best hand; it will cost more in future betting rounds to discover this.

3.4.3 Effective Odds

When you are considering the odds of making your hand with two cards remaining, it is difficult to estimate the expected cost to play those two rounds. For example, if there is \$6 in the pot after the flop and your single opponent has just bet \$2, then your pot odds are 3-to-1. However, you have two cards to make your hand so you must try to estimate the cost of the next round. Against a single opponent the worst case is that your opponent will bet next round and you will simply call; you would be paying \$6 to win \$10 (5-to-3) which increases the requirement for playing.

However, since you have two chances to make your hand your potential will improve as well. If you held 4♢-8♢ and the board was 7♢-A♢-6♣, your estimated chance of hitting the flush or the inside straight (12 outs) after two cards is now $12/47 + 35/47 * 12/46 = .45$, making it correct to call (or possibly raise) a bet on the flop.

3.5 Playing Style

There are several different ways to categorize the playing style of a particular player. When considering the ratio of raises to calls a player may be classified as aggressive, moderate or passive. Aggressive means the player frequently bets or raises rather than checking or calling (more than the norm), while passive means the opposite. Another simple set of categories is loose, moderate and tight. A tight player will play fewer hands than the norm, and tend to fold in marginal situations, while a loose player is the opposite. Players may be classified differently for pre-flop and post-flop play.

3.6 Deception and Unpredictability

A predictable player in poker is generally a bad player. Consider a player who never bluffs (when that player bets, they always have a strong hand). Observant opponents are more likely to fold (correctly) when this player bets, reducing the winning pot size. Of course, against weak, unobservant opponents, never bluffing may be a correct strategy. However, in general, deception and unpredictability are important. Although the cost and benefit of such actions must be considered, unpredictability can be achieved by randomly mixing actions. For example, do not raise every time you hold a high pair before the flop, otherwise an observant opponent can assume you are not holding a high pair when you simply call in the pre-flop. Deception is more complex and can be achieved through numerous different high-level strategies. Following are some of these strategies.

- **Changing Styles**: is a simple form of deception to deliberately create false impressions. For example, early in the session you might play a tight conservative style and show a lot of winning hands at the showdown. Later you switch to a looser style, and observant players are likely to continue to treat you as a tight player and take your bets very seriously.

- **Slowplaying**: " ... is playing a hand weakly on one round of betting in order to suck people in for later bets" [13]. Checking or calling in an earlier round of betting shows weakness, and this hopefully leads to your opponents being willing to put money in the pot later in the hand (particularly in those variants of Hold'em where the bet size doubles in later rounds). However, since you will often be up against many opponents, you need a very strong hand for this kind of play.

- **Check-raising**: is another way to play a strong hand weakly. Sklansky calls it a way to "trap your opponents and win more money from them" [13]. Essentially you believe that had you opened betting in the round you would either drive out players or only get one bet (no one would raise). But if you believe that one of your opponents will open the betting you begin the round by checking. Assuming the opening bet is then made, you follow by raising. Hopefully, players who have already put in a bet are willing to put in a second. However, even if players fold you still have their money from the opening bet.

- **Bluffing**: is an essential strategy in poker. It has been mathematically proven that you need to over-play or under-play (bluff or slowplay) in some way for optimal play in simplified poker [11]. Bluffing allows you to make a profit from weak hands, but it also creates a false impression which will increase the profitability of future hands (a lot of money can be won when betting a very strong hand and your opponent suspects you may be bluffing). In practice, you need to be able to predict the probability that your opponent will call in order to identify profitable opportunities. A game-theoretic explanation of the optimum bluffing frequency is presented in [13].

- **Semi-bluffing**: is a bet with a hand which is not likely to be the best hand at the moment but has a good chance of outdrawing calling hands (*e.g.* a four-card-flush). On occasion this play will also win outright when your opponents fold. The combined chances of winning immediately, or improving when called, makes it a profitable play.

Note that sometimes deception can be used to play an action which does not necessarily have the largest expected value, but rather creates a false impression which may indirectly lead to returns in the future. While undoubtedly important, it is difficult to measure the effectiveness of this type of deception.

3.7 Summary

The short term goal in poker (the goal in a specific deal) is either to maximize your gain if you think you can win (either with a strong hand or by bluffing with a weak hand) or to minimize your loss if you think you will lose. However, the outcomes of individual games are not independent. You can afford to make some 'bad moves' (the expected value for the chosen action in the current game is not the highest) provided they contribute to greater gains in later games.

An expert player is one who can usually recognize when they have or do not have the winning hand, and can maximize the money they win appropriately. They also occasionally invest money in misinformation (such as bluffing) and have the ability to identify good hands and understand their opponents (how they will react to certain actions or what hand they likely hold based on their actions). Knowledge of *tells* (physical mannerisms) and psychological plays are sometimes used in the human side of opponent modeling. Overall, the expert player has a good understanding of playing strategies, hand strength and potential, pot odds, and good opponent modeling skills. These factors are used as the basis for every decision made.

Chapter 4

How Computers Play Poker

Very little work has been done on poker by computing scientists, although there are numerous commercial and hobbyist approaches. The various computer-based approaches to poker can be classified into three high-level architecture descriptions (or a mixture thereof): expert system, game-theoretic optimal play, and simulation / enumeration-based systems. Each of these will be discussed in the following sections.

This chapter will also discuss several case studies of programs by computing scientists and hobbyists. Included in the former group is the historical work of Nicolas Findler along with the more recent ideas of Daphne Koller and Avi Pfeffer. Findler worked on a poker-playing program for 5-card draw poker [6]. Koller and Pfeffer implemented the first practical algorithm for finding optimal randomized strategies in two-player imperfect information games [10]. Among the hobbyist approaches examined are several that play poker on an online poker server over IRC (Internet Relay Chat). Three of these programs are *r00lbot*, *replicat* and *xbot* (although variations of these sometimes run under different names). Additionally there are two public domain programs: *Smoke'em Poker* for Microsoft Windows, as well as Seven-Card Stud and Texas Hold'em implementations by Johann Ruegg for the UNIX *curses* package. There are numerous approaches by commercial companies, although only a few have a target audience of professional players. The best of these is *Turbo Texas Hold'em* by Wilson Software (**http://www.wilsonsoftware.com**). It is an extremely rule-based system.

The final section discusses the architecture selected for our poker player and the reasons behind the selection.

4.1 Expert Systems

An expert system is essentially a set of specific rules to cover various game situations. Given the correct knowledge, this is perhaps the simplest approach to a reasonably strong program. However, since it is difficult to make an expert knowledge-based system learn (opponent modeling), it can easily be defeated by a strong player. Figure 4.1 contains a rudimentary example piece of such a system: when it is two or more bets to you on the flop and you do not have top pair (you have not paired the top

```
PlayFlop(CARDS myhand, STATE state)
{
  if (state.bets_put_in==0)
      && state.bets_to_call > 1
      && myhand < TOP_PAIR
      && myhand!=FOUR_FLUSH
      && myhand!=OPEN_STRAIGHT)
    return SelectAction(RAISE,10,CALL,10,FOLD 80)
  else ...

}
```

Figure 4.1: Example of an Expert Knowledge-Based System

card on the board and do not have a hole pair bigger than that card), nor do you have a four card flush or an open-ended straight, then raise 10% of the time, call 10% of the time, and fold 80% of the time.

There are many problems with this type of approach. Clearly, covering enough of the situations that will arise in practice would be very laborious. Also such a system is difficult to make flexible. If the system were made specific enough to be quite strong, conflicting rules could possibly be constructed and there would need to be a way to handle exceptions. Missing rules covering certain situations or making the rules too general would make the program weak and/or predictable. Additionally, you need an expert who can define these rules. This knowledge-acquisition bottleneck may prove to be a serious problem.

4.2 Game-Theoretic Optimal Strategies

Kuhn [11] along with Nash and Shapley [12] have demonstrated that "optimal strategies" using randomization exist for simplified poker. An optimal strategy always takes the best worst-case move, and this means two things: "the player cannot do better than this strategy if playing against a good opponent, and furthermore the player does not do worse even if his strategy is revealed to his opponent" [10]. For example, consider the two-player game of Roshambo (Rock, Paper, Scissors). The optimal strategy is to select a move uniformly at random (*i.e.* $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$) irrespective of the game history.

Finding an optimal approach is not so easy in a complex game like poker; there is a major stumbling block. Due to the enormous branching factor (see Figure 4.2), both the calculation and storage of the game-theoretic optimal strategy would be extremely expensive. Additionally the branching factor numbers are only for the two player environment – the multi-player environment is even more complex due to the addition of more imperfect information and many more possible betting interactions. As demonstrated by the attention devoted to multi-player situations in the poker literature, such considerations are quite important.

Additionally, the game-theoretic optimal approach is not necessarily the best.

Clearly, as the game-theoretic optimal strategy is fixed, it cannot take advantage of observed weaknesses in the opponent. Doing so would risk falling into a trap and losing. Consider Roshambo for an example. After witnessing your opponent playing Rock 100 times in a row, deciding to play Paper risks your opponent anticipating your action (the situation may be intended as a trap or your opponent may know your strategy). The existence of the risk, no matter how small, would violate the optimality of the strategy (the second guarantee, that the player cannot do worse).

Because of this, even against bad players an optimal strategy is likely to only break even. In contrast, a maximal strategy using opponent modeling (which does not assume perfect play from its opponents) would identify weaknesses and exploit them for profit (significantly more than an optimal strategy). There is some risk, because deviation from the optimal opens the door for your opponent to exploit it. But, if your knowledge of the opponent is good, the potential gains outweigh the risk. A game-theoretic optimal strategy would, however, make an excellent default or baseline to complement such an "adaptive" strategy.

4.3 Simulation and Enumeration

Simulation involves playing the hand out several times, where opponent hands and upcoming community cards are dealt randomly each time. The number of simulations can be fixed, variable, or dependent on some real-time constraint. If the sampling method is good, this will give a rough estimate of the strength and potential of your hand.

In contrast, enumeration involves evaluating each possible situation to get exact probabilities of your winning chances. This is not feasible for playing the hand out (given the branching factor and wide variety of possible opponent hands, see Figure 4.2) but is easily calculated for measures such as immediate hand strength: on the flop there are only $\binom{47}{2} = 1,081$ possible cases for opponent cards. Also note that given enough storage space some of the more complex enumerations could be pre-calculated.

These approaches can easily be mixed with an expert system (*e.g.* bet if you have a 50% chance of winning), or with game theory (*e.g.* bluff, or call a possible bluff, some game-theoretic optimal function of the time). In particular, opponent modeling can easily be combined with simulation or enumeration to generate reasonably accurate probabilities of outcomes. A useful opponent model would contain information specifying the probability of an opponent holding each possible pair of cards. In any particular simulation, this probability array could be used to appropriately skew the selection of cards held by an opponent. In an enumeration context, these probabilities would be used as weights for each subcase considered. Additionally, a faster but coarser estimate could be generated by only enumerating over the most likely subcases.

Finally, there is another advantage to being able to combine simulation or enumeration with opponent modeling. If the model contains information such as the calling frequency of a given opponent in a given situation, you would be able to take advan-

Assuming only two players:

- 2 possibilities for who acts first
- $\binom{52}{2} = 1,326$ different hole cards
- $\binom{50}{3} * 47 * 46 = 42,375,200$ significantly different ways the board can be dealt ($\binom{50}{3}$ different flops, 47 different turn cards, 46 different river cards)
- 15 different ways the betting can proceed in the pre-flop (only 7 do not end in one side winning uncontested)
- 19 different ways the betting can proceed in any round after the pre-flop (only 9 do not end in one side winning uncontested)

Therefore:

- $2 * \binom{52}{2} * 7 * \binom{50}{3} = 363.9 * 10^6$ different states at the beginning of the flop
- $363.9 * 10^6 * 9 * 47 = 153.9 * 10^9$ different states at the beginning of the turn
- $153.9 * 10^9 * 9 * 46 = 63.7 * 10^{12}$ different states at the beginning of the river
- Also note there are up to $\binom{n}{2}$ possible opponent hands at each stage in the game ($n = 50$ for the pre-flop, 47 for the flop, 46 for the turn, and 45 for the river). This hidden information was not included in the above products (which are the number of possible variations of known information).

This is still a large tree even though there is some redundancy in the way the cards are dealt (suits are isomorphic):

- 169 significantly different classes of hole cards rather than 1,326 (see Appendix A). This reduces the number of states at the beginning of the flop to $46.4 * 10^6$, and $8.1 * 10^{12}$ at the beginning of the river (approximately a 7.8-fold reduction).
- An additional complex reduction based on the isomorphism of suits can reduce the original number of possible flop states and starting hands from $1326 * \binom{50}{3} = 26.0 * 10^6$ ($3.3 * 10^6$ with the elimination of redundant starting hands) to $1.3 * 10^6$ (approximately an additional 2.5-fold reduction – still leaves a large number at the river). The details of the reduction are not presented here. It is based on enumerating each possible combination of suits in the starting hand and on the flop. For example, there are only $\binom{13}{2} = 78$ significantly different starting hands where both cards are of the same suit (map this suit to, say, ♠), and there are only $11 * \binom{13}{2} = 858$ significantly different flops where one card is of the original suit (mapped to ♠) and the other two cards are of a second suit (mapped to, say, ♡).

The addition of multi-player considerations exponentially complicates the tree:

- three players in any round after the pre-flop: 138 sequences of betting actions end with two players remaining and 46 end with three players remaining (93 end in an uncontested win). For two players there were only 9 different sequences that did not terminate the game instead of 184.
- four players on any round after the pre-flop: 1504 sequences of betting actions end with two players remaining, 874 end with three players remaining, and 161 end with all four players remaining (792 end in an uncontested win).

Figure 4.2: Branching Factor for Structured Betting Texas Hold'em With a Maximum of 4 Bets/Round

tage of a more realistic consideration of some outcomes. For example, consider the situation where you are at the river against one opponent, the pot contains $20, and your options are to check or bet $4. Given the probability array of your opponent's model you calculate by enumeration that you have only a 10% chance of having the stronger hand. The model tells you that, when faced by a check in this situation, your opponent will check 100% of the time, and faced by a bet your opponent will fold 30% and call 70%. You can therefore calculate the expected value (EV) of your two options:

- **check:** win $20 10% of the time, lose $0 90% of the time.

  $EV = 20 * 0.10 - 0 * 0.90 = 2.00.$

- **bet:** since your opponent will likely fold the weakest 30% of hands, and you could only beat 10% of all hands (or the worst third of the hands they fold) then there is no chance that you win if they call.

  - **opponent folds 30%:** win $20 100% of the time.
  - **opponent calls 70%:** win $24 0% of the time, lose $4 100% of the time.

  $EV = 0.30 * (20 * 1.00) + 0.70 * (24 * 0.00 - 4 * 1.00) = 3.20$. Therefore, it is more profitable if you bet (due to the reasonable possibility of scaring your opponent into folding).

This is a simple contrived example but it demonstrates how well an accurate opponent model complements a simulation or enumeration system.

4.4 Findler's Work

Nicolas Findler worked on and off for 20 years on a cognition-based poker-playing program for Five-Card Draw [6] [7] [8]. He recognized the benefits of research into poker as a model for decision-making with partial information. However, much of the applicability of his work to ours is lost due to differing goals; rather than being concerned about producing a high performance poker program, he focused on simulating the thought processes of human players. Hence, to achieve this, instead of relying heavily on mathematical analysis, his approach was largely based on modeling human cognitive processes. He did not produce a strong poker player.

4.5 The *Gala* System

A more theoretical approach by computing scientists was taken by Koller and Pfeffer [10]. They implemented the first practical algorithm for finding optimal randomized strategies in two-player imperfect information competitive games. This is done in their *Gala* system, a tool for specifying and solving problems of imperfect information. Their system builds trees to find the game-theoretic optimal (but not maximal)

strategy. However, even when considering only the two-player environment, only vastly simplified versions of poker can presently be solved, due to the large size of trees being built. The authors state that "... 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."

4.6 Hobbyists

Several programs by hobbyists were examined to explore the architecture and approach used. The most common approach is expert-based, however simulation-based approaches tend to be stronger (although more computationally expensive).

- **xbot** by Greg Reynolds uses an expert system which is manually patched when weakness is observed.

  http://webusers.anet-stl.com/~gregr/

- **replicat** by Stephen How also uses an expert system in combination with observing a large number of possible features about the hand and board (*e.g.* three-straight).

- **r00lbot** by Greg Wohletz is perhaps the strongest of the three IRC programs. For the pre-flop it uses Sklansky and Malmuth's recommendations [14], and for the post-flop it conducts a series of simulations (playing out the hand to the showdown, typically 3,000 times) against N random hands (where N is the number of opponents, and is artificially adjusted for bets and raises). The actual action is dependent on what percentage of simulations resulted in a win.

- **Smoke'em Poker** is a Five-Card Draw program by Dave O'Brien. It uses an expert system and has a set of rules for each opponent type (*e.g.* tight, loose).

  http://www.cgl.uwaterloo.ca/~gmgrimsh/poker.html

- There are two poker games by Johann Ruegg, Sozobon Ltd. Both the Seven-Card Stud and Texas Hold'em games use a simulation-based approach where the program plays the hand to the showdown several times against random opponents. The resulting winning percentage is artificially adjusted depending on the game state and compared against several hard-coded action thresholds.

  ftp://ftp.csua.berkeley.edu/pub/rec.gambling/poker/spoker.tar.Z

4.7 Architecture

Consideration of these other programs and the various advantages of the different approaches led us to select a primarily enumeration-based approach for the purposes of this thesis. There were several reasons:

- The expert system is too limited and context sensitive (the game is far too complex to cover all possible contexts). It also is inflexible, and will inherit any error in the designer's approach to the problem.
- The game-theoretic optimal strategy is very complex to compute and, if such a system could be built, presumes optimal play on the part of the opponents. We are working under the assumption that our opponents will make errors and therefore maximal play is preferable.
- An enumeration-based approach is easy to combine with an opponent modeling system based on the probability distribution of possible opponent holdings.
- Most of the desired values are computationally feasible in real-time. Where this is not so, there are many ways to calculate good approximations of the measures (*e.g.* random simulation, pre-computation, heuristics).
- Values calculated by enumeration (as opposed to simulation) are more accurate since random sampling introduces variance, and rule-based systems are subject to systemic error.

Loki (Figure 4.3) is a complete poker-playing program (able to play a full game of Texas Hold'em unaided). There are three main co-dependent components which control the play of the program. These components are discussed in the following chapters. They are **hand evaluation** (using the opponent models and game state, it generates values which roughly correspond to the probability of holding the strongest hand), **betting strategy** (it uses the values generated by hand evaluation, the opponent models, and the game state to determine the best action), and **opponent modeling** (it translates the betting history of the opponent into information about betting behavior and possible hands held).

4.8 Summary

Three main approaches to program design were summarized in this chapter: the expert system (hard-coded rules based on the knowledge of an expert), game-theoretic optimal strategies, and simulation/enumeration-based. The first two approaches have some obvious limitations. However, the different approaches can be combined to various extents.

While there are many poker-playing programs, none are very strong, and few make source code or a description of the inner workings available. Also, with the exception of Findler and Koller/Pfeffer there are few resources in the computing science literature. There is also little on building a high-performance poker program, except for some ideas presented in [2].

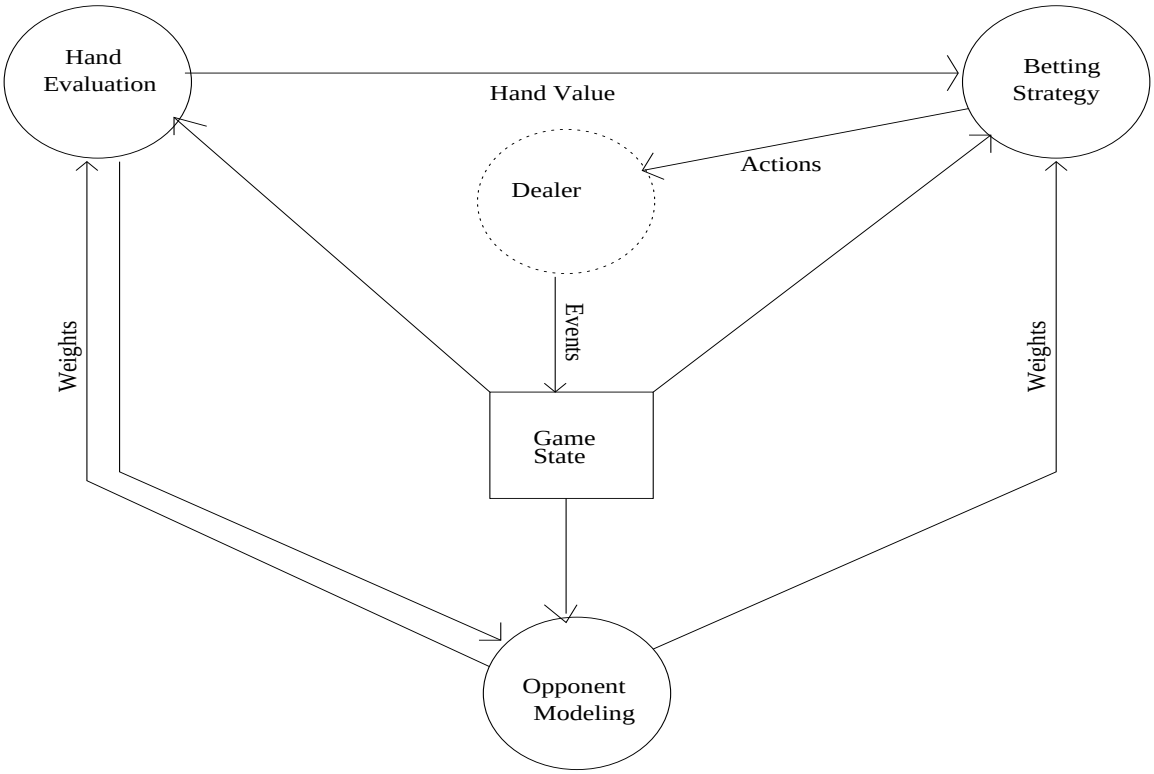

Figure 4.3: *Loki*'s Architecture

Chapter 5

Hand Evaluation

Accurate assessment of your winning chances is necessary when considering the cost of playing versus the payoff with pot odds. Hand evaluation uses the opponent models and game state to calculate estimates of the winning chances of your hand. However, since there are more cards to come on the flop and turn, the present strength of a hand is insufficient information. For this reason, post-flop hand evaluation is broken into two parts: strength and potential. Strength is the probability of a hand *currently* being the strongest and potential is the probability of the hand *becoming* the strongest (or of losing that status to another hand) after future cards have been dealt. Due to the computational complexity of potential for the pre-flop (the first two cards), evaluation in this stage of the game is given special treatment.

5.1 Pre-Flop Evaluation

Hand strength for pre-flop play has been extensively studied in the poker literature. For example, [14] attempts to explain strong play in human understandable terms, by classifying all the initial two-card pre-flop combinations into nine betting categories. For each hand category, a suggested betting strategy is given, based on the strength of the hand, the number of players in the game, the position at the table (relative to the dealer), and the type of opponents. For a poker program, these ideas could be implemented as an expert system, but a more general approach would be preferable.

For the initial two cards, there are $\binom{52}{2} = 1,326$ possible combinations, but only 169 distinct hand types (13 paired hands, $\binom{13}{2} = 78$ suited hands and $13 * \binom{4}{2} = 78$ unsuited hands). For each one of the 169 possible hand types, a simulation of 1,000,000 games was done against each of one, three and six random opponents (to cover the 2, 3-4 and 5 or more player scenarios[1]). Each opponent was simple and always called to the end of the hand. This produced a statistical measure of the approximate income rate (IR) for each starting hand; income rate measures the return on investment.

$$IR = \frac{net\_bankroll}{hands\_played} \tag{5.1}$$

[1] We consider these the most important groupings. See Appendix B.

The computed values are presented in Appendix A. These numbers must always be viewed in the current context. They were obtained using a simplifying assumption, where the players always call to the end. However, this experiment gives a good first approximation of how strong a hand is. For example, in the 7-player simulation the best hand is a pair of aces and the worst hand is a 2 and 7 of different suits. While the absolute IR value may not be useful, the relative order of the hands is. As we discuss in Appendix A, there is a strong correlation between these simulation results and the pre-flop card ordering given in [14].

5.2 Hand Strength

Hand strength assesses how strong your hand is in relation to what other players may hold. Critical to the program's performance, it is computed on the flop, turn and river by a weighted enumeration which provides an accurate estimate of the probability of currently holding the strongest hand. This calculation is feasible in real-time: on the flop there are 47 remaining unknown cards so $\binom{47}{2} = 1,081$ possible hands an opponent might hold. Similarly there are $\binom{46}{2} = 1,035$ on the turn and $\binom{45}{2} = 990$ on the river.

Figure 5.1 contains the algorithm for computing hand strength. The bulk of the work is in the call to the hand identification function *Rank* which, when given a hand containing at least 5 cards, determines the strongest 5-card hand and maps it to a unique value such that stronger poker hands are given larger values and hands of equal strength are given the same value. *Rank* must be called $\binom{n}{2} + 1$ times where n is the number of unknown cards.

The parameter w is an array of weights, indexed by two card combinations, so the function determines a weighted sum. It is the *weight array* for the opponent under consideration (each possible two-card holding is assigned a weight). When the array is normalized so the sum is 1, the weights are conditional probabilities meaning "for each possible two-card holding what is the probability that it is the hand held by this opponent" (given the observed betting). Without normalization, the values in the weight table are conditional probabilities meaning "what is the probability that this opponent would have played in the observed manner" (given they held this hand). Without opponent modeling, it can simply be filled with a default set of values, either a uniform or 'typical' distribution. Under uniform weighting each entry in the array is equal (an appropriate representation if the opponent has a random hand). A more typical distribution would be a set of values based on the IR tables. This is the only model information used directly by the hand strength enumeration.

Suppose our starting hand is A♢-Q♣ and the flop is 3♡-4♣-J♡ (1,081 possible opponent hands). To estimate hand strength using uniform weighting, the enumeration technique gives a percentile ranking of our hand (our *hand rank*). We simply count the number of possible hands that are better than ours (any pair, two pair, A-K, or three of a kind: 444 hands), how many hands are equal to ours (9 possible remaining A-Q combinations), and how many hands are worse than ours (628). Counting ties as

```
HandStrength(CARDS ourcards, CARDS boardcards, FLOAT w[])
{
  RANK  ourrank, opprank
  CARDS oppcards
  FLOAT ahead, tied, behind, handstrength

  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 += w[oppcards]
    else if (ourrank==opprank) tied += w[oppcards]
    else /* < */               behind += w[oppcards]
  }
  handstrength = (ahead+tied/2) / (ahead+tied+behind)
  return (handstrength)
}
```

Figure 5.1: HandStrength Calculation

half, this corresponds to a hand rank (HR) of 0.585. In other words there is a 58.5% chance that our hand is better than a random hand (against non-uniform weights we call it hand strength, or HS).

This measure is with respect to one opponent, but when all opponents have the same weight array it can be roughly extrapolated to multiple opponents by raising it to the power of the number of active opponents (HR_n is the hand rank against n opponents, $HR \equiv HR_1$).

$$HR_n = (HR_1)^n. \tag{5.2}$$

It is not an exact value because it does not take into account interdependencies arising from the fact that two players cannot hold the same card. However, this is a secondary consideration.

Continuing the example, against five opponents with random hands the adjusted hand rank is $HS_5 = .585^5 = .069$. Hence, the presence of additional opponents has reduced the likelihood of our having the best hand to only 6.9%.

This example uses a uniform weighting: it assumes that all opponent hands are equally likely. In reality this is not the case. Many weak hands like 4♡-J♣ ($IR < 0$) would have been folded before the flop. However, with the example flop of 3♡-4♣-J♡, these hidden cards make a strong hand that skews the hand evaluations. Specifically, accuracy of the estimates depend strongly on models of our opponents (the array of weights w). Therefore, we compute weighted sums to obtain hand strength (HS). As with HR, HS_n is the hand strength against n opponents and $HS \equiv HS_1$.

$$HS_n = (HS_1)^n. \tag{5.3}$$

5.2.1 Multi-Player Considerations

The above description of the weighted enumeration system for hand strength is intended for one opponent. When the system was first put to use, the weight array was common to all opponents (either uniform or some fixed 'typical' distribution) so HS_n was easily extrapolated by using Equation 5.3. However, our opponent modeling system will be computing a different set of weights for each specific opponent.

The correct approach would be to treat each possible case independently. For example, one possible case is that player 1 holds A♡-Q♢ and player 2 holds Q♡-J♣. To handle this distinction, the function would need an extra iteration layer for each opponent (and would still be dependent on the order of the iteration). For each possible case it would then use a weight $w[x] = w_1[x_1] * w_2[x_2] * ... * w_n[x_n]$ (where x is the complex subcase, x_i is the subcase for the cards held by player i and $w_i[x_i]$ is the weight of that subcase for player i). The weight of the complex subcase given in the example is w[A♡-Q♢-Q♡-J♣] = w_1[A♡-Q♢] $* w_2$[Q♡-J♣]. The increase in computational complexity is substantial (approximately a factor of 1,000 for each additional player) and becomes infeasible with only 3 opponents.

There are two simpler methods to approach this problem and obtain good estimates of HS_n. The first calculates HS_{p_i} for all opponents p_i (such that $i = 1..n$) given each respective weight array. It then uses the equation

$$HS_n = HS_{p_1} * HS_{p_2} * ... * HS_{p_n}. \tag{5.4}$$

The second method calculates HS_1 using a special weight array, called the *field array*, computed by combining the weight arrays of all active players. HS_n is then calculated with Equation 5.3. The use of the field array as an estimate is a compromise between computational cost and accuracy. It represents the entire table by giving the average weights of the opponents. The process of obtaining hand weights and generating this array is described in Chapter 7.

Both of these methods are only estimates because they ignore the interdependencies arising from the fact that two players cannot hold the same card. Several situations were examined from data gathered during play against human opponents. For both methods, the absolute error was measured with respect to the correct full enumeration but only against two or three active opponents (four opponents was too expensive to compute). For the first method (Equation 5.4), this testing revealed the error never exceeded 2.19%. The average error was 0.307% with two opponents and 0.502% with three opponents. For the second method (using the field array and Equation 5.3), the error never exceeded 5.79% for two opponents and 4.15% for three opponents. In fact, for two opponents only 59 out of 888 cases had an error larger than 2% (and 20 out of 390 three opponent cases). The average errors were 0.671% and 0.751%. The estimated values were usually slight overestimates.

This isolated test scenario suggests that the first method is better but the difference is small. The error also appears to get slightly worse with additional opponents. *Loki* uses the second method due to the faster computation and ease of introduction into the present framework (particularly with respect to multi-player considerations

for hand potential, as will be discussed later). We have not invested the time to further explore the error arising from these interdependencies, however we believe it is minor in comparison to the extra work that would be required for a very accurate measure of HS_n. Also, this amount of error is considered to be negligible given that the error introduced by other components of the system tends to be greater in magnitude.

5.3 Hand Potential

In practice, hand strength alone is insufficient to assess the quality of a hand. Hand potential assesses the probability of the hand improving (or being overtaken) as additional community cards appear. Consider the hand 8♢-7♢ with a flop of 9♢-6♣-2♢. The probability of having the strongest hand is very low, even against one random opponent (11.5%). On the other hand, there is tremendous potential for improvement. With two cards yet to come, any ♢, 10, or 5 will give us a flush or a straight. Hence there is a high probability that this hand will improve substantially in strength, so the hand has a lot of value. We need to be aware of how the potential affects hand strength.

This example describes positive potential ($PPOT$): the probability of pulling ahead when we are behind. We can also compute the negative potential ($NPOT$): the probability of falling behind given we are ahead. Both of these can be computed by enumeration in real-time. We have 1,081 possible subcases (opposing hands for which we have weights) on the flop and 990 on the turn. For each subcase we can either do a two card look-ahead (consider the 990 combinations of the next two cards on the flop) or a one card look-ahead (45 cards on the flop and 44 on the turn). For each subcase we count how many combinations of upcoming cards result in us being ahead, behind or tied. The total number of cases to be considered is:

- $PPOT_2$ and $NPOT_2$ (two card look-ahead on the flop): 1,070,190
- $PPOT_1$ and $NPOT_1$ (one card look-ahead): 48,645 on the flop and 43,560 on the turn

The potential for A♢-Q♣/3♡-4♣-J♡ with uniform weighting is shown in Table 5.1. The table shows what the result would be after seven cards, for cases where we are ahead, tied or behind after five cards. For example, if we did not have the best hand after five cards, then there are 91,981 combinations of cards (pre-flop and two cards to come) for the opponents that will give us the best hand. Of the remaining hands, 1,036 will leave us tied with the best hand, and 346,543 will leave us behind. In other words, if we are behind we have roughly a $PPOT_2 = 21\%$ chance of winning against one opponent in a showdown. Additionally, if we are currently ahead and that opponent plays to the showdown, we have roughly a $NPOT_2 = 27\%$ chance of losing.

If $T_{row,col}$ refers to the values in the table (for brevity we use B, T, A, and S for Behind, Tied, Ahead, and Sum) then $PPOT_2$ and $NPOT_2$ are calculated by:

5 cards	7 cards					
	Ahead	Tied	Behind	Sum		
Ahead	449,005	3,211	169,504	621,720	=	628x990
Tied	0	8,370	540	8,910	=	9x990
Behind	91,981	1,036	346,543	439,560	=	444x990
Sum	540,986	12,617	516,587	1,070,190	=	1,081x990

Table 5.1: Unweighted potential of A♢-Q♣/3♡-4♣-J♡

$$PPOT_2 = \frac{T_{B,A} + \frac{T_{B,T}}{2} + \frac{T_{T,A}}{2}}{T_{B,S} + \frac{T_{T,S}}{2}} \tag{5.5}$$

$$NPOT_2 = \frac{T_{A,B} + \frac{T_{A,T}}{2} + \frac{T_{T,B}}{2}}{T_{A,S} + \frac{T_{T,S}}{2}}. \tag{5.6}$$

Figure 5.2 describes the algorithm for two card look-ahead from the flop. The parameter w is, as for Figure 5.1, for the weight array of the opponent (opponent modeling is discussed later), and can simply be a uniform set of weights. The **HandStrength** calculation is easily embedded within this function, and the one card look-ahead function **HandPotential1** is essentially the same as **HandPotential2**. In this function, the inner loop is executed $\binom{47}{2} * \binom{45}{2} = 1,070,190$ times and so the *Rank* function is called

$$1 + \binom{47}{2} + 2 * \binom{47}{2} * \binom{45}{2} = 2,141,371$$

times. However, there are many redundant calculations. There are only $\binom{47}{2} = 1,081$ possible unique calls in the inner loop to *Rank* for *ourcards* and only $\binom{47}{4} = 178,365$ for *oppcards* (this redundancy exists because there is no order constraint to the evaluation of poker hands). Therefore, with pre-calculation, **HandPotential2** need only make

$$1 + \binom{47}{2} + \binom{47}{2} + \binom{47}{4} = 180,528$$

calls to *Rank* (although the number of times the inner loop is executed is not reduced from 1,070,190). Similarly, **HandPotential1** originally needs

$$1 + \binom{47}{2} + 2 * \binom{47}{2} * 45 = 98,372$$

calls to *Rank* on the flop (92,116 on the turn) but with pre-calculation only

$$1 + \binom{47}{2} + \binom{47}{2} + \binom{47}{3} = 18,378$$

```
HandPotential2(CARDS ourcards, CARDS boardcards, FLOAT w[])
{
  /* Each index represents ahead, tied and behind.  */
  FLOAT   HP[3][3]   /* initialize to 0 */
  FLOAT   HPtotal[3] /* initialize to 0 */
  FLOAT   ppot2,npot2
  RANK    ourrank5,ourrank7,opprank
  CARDS   additionalboard
  INTEGER index

  ourrank5 = Rank(ourcards, boardcards)
  /* Consider all remaining two card combinations for the opponent */
  for each case(oppcards)
  {
    /* after 5 cards */
    opprank = Rank(oppcards, boardcards)
    if (ourrank5>opprank)       index = ahead
    else if (ourrank5==opprank) index = tied
    else /* < */                index = behind
    HPtotal[index] += w[oppcards]

    /* Consider all possible two card board combinations to come */
    for each case(additionalboard)
    {
      board = boardcards + additionalboard
      ourrank7 = Rank(ourcards,board)
      opprank = Rank(oppcards,board)
      if (ourrank7>opprank)       HP[index][ahead] += w[oppcards]
      else if (ourrank7==opprank) HP[index][tied] += w[oppcards]
      else /* < */                HP[index][behind] += w[oppcards]
    }
  }
  /* ppot2:  we were behind but moved ahead (Equation 5.5) */
  ppot2 = (HP[behind][ahead] + HP[behind][tied]/2 + HP[tied][ahead]/2)
    / (HPtotal[behind] + HPtotal[tied]/2)
  /* npot2:  we were ahead but fell behind (Equation 5.6) */
  npot2 = (HP[ahead][behind] + HP[ahead][tied]/2 + HP[tied][behind]/2)
    / (HPtotal[ahead] + HPtotal[tied]/2)
  return(ppot2,npot2)
}
```

Figure 5.2: HandPotential2 Calculation

on the flop (17,251 on the turn).

In Table 5.1 we compute the potential based on two additional cards and it produces

$$PPOT_2 = \frac{91,981 + \frac{1,036}{2} + \frac{0}{2}}{439,560 + \frac{8,910}{2}} = 0.208, \tag{5.7}$$

$$NPOT_2 = \frac{169,504 + \frac{3,211}{2} + \frac{540}{2}}{621,720 + \frac{8,910}{2}} = 0.274. \tag{5.8}$$

The calculation for one card lookahead is exactly the same as the above calculation, except there are only 44 or 45 possible outcomes instead of 990. With only one card to come on the turn, we find $PPOT_1 = 0.108$ and $NPOT_1 = 0.145$. When combined with an array of weights from opponent modeling (each subcase is weighted according to the two card combination defining the subcase) the calculations provide accurate probabilities that take into account every possible scenario. Hence, the calculation gives smooth and robust results, regardless of the particular situation. $PPOT_1$ is used in practice due to the greater complexity of using $PPOT_2$ (calculating the effective odds, or how much it might cost to see a second card) as well as the calculation time. On a 150 MHz SGI Challenge, using pre-calculation but otherwise unoptimized, computing $PPOT_1$ from the flop typically takes 130 ms of CPU time and $PPOT_2$ takes 3200 ms on average.

5.3.1 Multi-Player Considerations

As described in Section 5.2.1, *Loki* makes use of a *field array* provided by the opponent modeling module which is representative of the entire set of active opponents. Unlike hand strength, against multiple (two or more) opponents the $PPOT$ and $NPOT$ values calculated with the *field array* are used without adjustment. The correct calculation would be similar to the one described in Section 5.2.1 (accounting for each player adds an extra iteration layer of approximately 1,000 subcases), but, for potential, the value against one player is believed to be a simple but reasonable estimate (high usefulness and low computational complexity).

Calculating potential in the context of multiple opponents is complex due to the many interactions; there is no easy reduction to an approximate value similar to HS_n. However, it is likely that the present estimates are typically optimistic and the correct value would worsen with additional opponents. For example, consider the situation where you have a straight draw (4♢-5♣/6♡-3♢-T♡) but you are unaware of any tendencies of your opponents (*i.e.* uniform weight arrays). Each additional opponent increases the chances that someone has a flush draw in hearts (in fact, it increases the chances of any particular hand occurring). This means that 'intersection cards' like 2♡ and 7♡ are worth less to you since the likelihood that they give you the winning hand decreases, as does $PPOT$. For another example, consider that you hold top pair (K♣-T♢/6♡-3♢-T♡). Each additional opponent increases the chance that you are up against a flush or straight draw so cards like a 2, a 7 or a heart are more likely to give you a losing hand. Therefore, $NPOT$ increases with additional opponents.

But over-optimism is not always the case; multiple opponents can improve your chances. For example, if you have a straight draw with small cards (4♢-5♣/6♣-3♢-K♡) and your opponents are likely holding high cards, then each additional opponent increases the chance that the upcoming card is not high. This will overall both decrease $NPOT$ and increase $PPOT$, although additional players also increases the size of the pot, which increases the value of our draw.

These are only rough examples intended to demonstrate the complex interactions involved in multi-player considerations for hand potential. The actual effect of each additional opponent is dependent on the probability distribution of possible hands and its relationship to the probability distributions of the other players, as well as its relationship to your needed cards.

5.4 Summary

This chapter describes the methods used for evaluating our hand in any particular situation. The algorithms presented are intended to provide good estimates of your winning chances.

For the pre-flop, we used an unsophisticated simulation technique for the calculation of pre-flop income rates. We feel that these values are sufficient (will not be the limiting factor to the strength of the program), although there is room for improvement.

This chapter also describes the enumeration techniques used for evaluating hands (both strength and potential) after the flop. The algorithms, particularly in multi-player scenarios, abandon the treatment of some complexities in favor of simplicity and computation time. We have not invested the time to fully explore the magnitude of this error, however we believe it is minor in comparison to the savings in time and complexity.

Chapter 6

Betting Strategy

Using the game state, opponent models and hand evaluation derived from that data, the betting strategy determines whether to fold, call/check, or bet/raise. But exactly what information is useful and how should it be used? The answer to these questions are not trivial and this is one of the reasons that poker is a good testbed for artificial intelligence. One approach would be to use the game theoretic optimal betting strategy, but, despite the fact that it is very complex to calculate, human opponents do not play optimally so this may not be the best decision in practice (leading to the most profit).

A minimal system could simply be based on hand strength (*i.e.* ignore hand potential and simply bet/call based on the immediate strength of our hand). Refinements to the betting strategy would involve the addition of high-level strategy concepts (like slowplaying or bluffing). For each decision to be made, one of several variables (like $PPOT$) is compared to some threshold (which may be based on another variable, like *pot_odds*). This structure uses expert knowledge but is easy to implement and the overhead is insignificant. All of these refinements are intended to be quick ways to select the play which (hopefully) has the largest expected value (EV), since computing the exact EV is not feasible. There is a real-time constraint, in that a single game of poker only takes a few minutes to play, and from Figure 4.2 we see that the game tree can be very large.

This chapter first describes the pre-flop betting strategy (which is treated as a special case). This is followed by an explanation of a simple post-flop (flop, turn and river) betting strategy (using only hand strength) which serves as a template, and then the measures and strategies that *Loki* uses for the post-flop rounds. Finally, unimplemented strategies are discussed.

Note that this chapter is included for completeness. The betting strategy is (so far, necessarily) *ad hoc* and therefore left undeveloped (it is a target for future improvement with computer-oriented techniques such as Monte Carlo simulation).

6.1 Pre-Flop Betting Strategy

The implemented pre-flop betting strategy is preliminary and makes use of expert information. It is sophisticated enough to not hamper overall performance so that the focus can be put on post-flop play (the more interesting portion of the game). *Loki* examines several variables, uses the expert knowledge to determine thresholds for various betting actions, and chooses a strategy:

- **Make0**: fold if it costs more than zero to continue playing, otherwise check.
- **Call1**: fold if it costs two or more bets to continue (and we have not already voluntarily put money in the pot this round), otherwise check/call.
- **Make1**: like **Call1** except bet if there has not been a bet this round (with the big blind this cannot happen in the pre-flop).
- **Call2**: always call/check (despite what the name of this strategy suggests, even more than 2 bets).
- **Make2**: bet/raise if less than two bet/raises have been made this round, otherwise call.
- **Make4**: bet/raise until the betting is capped, otherwise call.

Except for the small blind, which is given special treatment, **Call1** and **Call2** are effectively not used, so there are really only 4 different strategies. Once a strategy is selected it is fixed for all subsequent actions in the pre-flop. The small blind is a special case due to only having to call one half of a bet (so **Call1** is really **Call0.5** and **Call2** is really **Call1.5**), and has fixed thresholds for these two strategies.

Call1 (and hence **Make1**) has a special case folding requirement, that "we have not already voluntarily put money in the pot this round." This is a feature added after testing with human opponents on IRC. Many players were very aggressive and would raise frequently. This meant that often when *Loki* called the blind with a decent hand (but not **Call2** or better), two or more players would then raise, causing *Loki* to fold. Due to this commonly exploited weakness, the kludge is necessary until some amount of opponent of modeling is implemented into the pre-flop.

The thresholds for selecting a betting strategy are determined from a set of linear formulas of the form

$$threshold(strategy) = base + increment * position \tag{6.1}$$

where *strategy* is the betting strategy (*e.g.* **Make1**), *base* and *increment* are defined by a human expert (see Appendix B), and *position* is the number of players to act before it is the small blind's turn again (so, if num_inpot is the number of players still in the pot, the small blind is position $num\_inpot - 1$, the big blind is position $num\_inpot - 2$, the first player to act is position $num\_inpot - 3$ and so on, until the button (dealer) who is position 0).

For example, if *Loki* is playing *tight* against 3-4 players, the [*base, increment*] values used for **Make2** are [200,50] (based on Table B.1) so the formula used is

$$threshold(\textbf{Make2}) = 200 + 50 * position$$

based on Equation 6.1. That is, the coefficients used for the linear formula depend on both the variable (*group*) and the parameter (*tightness*).

The variable *group* is based on the expected number of players (*E_num_players*), which is

$$\begin{aligned} E\_num\_players &= num\_guaranteed + \\ &\quad probability\_play * (num\_inpot - num\_guaranteed). \end{aligned} \tag{6.2}$$

There are three cases of interest for the expected number of players. We round *E_num_players* to the nearest integer and determine the appropriate group based on what range that value falls in: "2 players", "3-4 players" and "5 or more players" (see Appendix B). The state variable *num_guaranteed* is the number of players who have already put money in the pot (and have presumably committed to playing). This includes the blinds as well as ourselves (we assume we will play). *Probability_play* is an expert-defined value for the average playing percentage of players (by default 0.60). Appropriate opponent modeling would provide much better estimates for this value, based on the observation of the current session .

The parameter *tightness* is a setting which affects the percentage of hands that *Loki* will play (indirectly, by selecting a different set of thresholds). The three settings are *tight*, *moderate*, and *loose* (the default). With ten players these roughly translate into playing 18%, 21% and 24% of all hands (the distinction is not large so the terms are a misnomer – all levels are relatively conservative).

There is one [*base*, *increment*] pair per set of *group*, *strategy* and *tightness* values (Table B.1) so there are 27 pairs total (since **Make0** is the default strategy only three thresholds are needed for determining the strategy). Once the thresholds are determined, the actual strategy selected is dependent on the pre-calculated income rate (IR) of the hole cards. Figure 6.1 describes the algorithm for selecting a strategy.

For example, consider a six player game. We hold A♢-T♢ and are playing *tight*. The first two players put in blinds, the next two fold, and it is *Loki*'s turn. There is still one player, the button, who has not yet acted. So $position = 1$, $num\_inpot = 4$ and $num\_guaranteed = 3$ (including *Loki*). We then calculate

$$E\_num\_players = 3 + 0.60 * (4 - 3) = 3.6$$

and use this to determine *group* = "3-4 players". Then, using Table B.1 we calculate

$$\begin{aligned} threshold(\textbf{Make1}) = &\quad 50 + 50 * 1 &&= 100, \\ threshold(\textbf{Make2}) = &\quad 200 + 50 * 1 &&= 250, \text{ and} \\ threshold(\textbf{Make4}) = &\quad 580 + 0 * 1 &&= 580. \end{aligned}$$

Finally, we find the value of our hand is $IR = 491$ and select the appropriate strategy. Since $250 \leq 491 < 580$, we select the **Make2** strategy, and raise.

```
/* Called the first time we are asked for an action in the pre-flop,
 * the selected strategy dictates the current and subsequent actions */
GetStrategyPreflop(CARDS myhand, STATE state, PARAMETERS param)
{
  FLOAT   E_num_players   /* expected number of players */
  INTEGER group           /* TWO, THREEORFOUR, or FIVEPLUS */
  FLOAT   IR              /* income rate */
  FLOAT   thresh[]        /* indexed by strategies */

  /* determine the group */
  E_num_players = state.num_guaranteed +
    param.probability_play * (state.num_inpot - state.num_guaranteed)
  if (state.num_inpot < 2.5)
    group = TWO
  else if (E_num_players >= 4.5)
    group = FIVEPLUS
  else
    group = THREEORFOUR

  /* calculate IR (see Appendix A) and thresholds.  */
  IR = IR_table[group,myhand]
  /* only the small blind has different thresholds for CALL1 and CALL2 */
  thresh = SetThresholds(group,param.tightness,state.position)

  /* now use IR to select the appropriate strategy */
  if (IR>=thresh[MAKE4])
    strategy = MAKE4
  else if (IR>=thresh[MAKE2])
    strategy = MAKE2
  else if (IR>=thresh[CALL2])
    strategy = CALL2
  else if (IR>=thresh[MAKE1])
    strategy = MAKE1
  else if (IR>=thresh[CALL1])
    strategy = CALL1
  else
    strategy = MAKE0
  return strategy
}
```

Figure 6.1: Pre-Flop Betting Strategy

6.2 Basic Post-Flop Betting Strategy

A minimal system for betting or calling could be based solely on the immediate strength of our hand. In fact, a more useful betting strategy could still use this basic system as a template and simply proceed to a hierarchy of alternative branches whenever the current decision is rejected. For example, if the initial betting decision based on hand strength decides to fold we could fall back on another consideration such as semi-bluffing. If we then do not decide to semi-bluff, we fall back on to another decision like pot odds. Each time the proposed action is rejected the next decision point can be checked against some defined order of priority. Priority is dependent on how aggressive the action is. Pot odds and showdown odds (described below) are low priority because they are passive calling/checking decisions (if we always used the pot odds decision first, we would never bet). However, a strategy like check-raising is only used with a very strong hand so it would be given the highest priority (*i.e.* considered first).

A simple betting strategy, based on hand strength, is described in Figure 6.2. It uses a function called **Make** which is used to show a level of strength appropriate for the hand held. The **Make** function is used in the pre-flop betting strategy and to some extent in the actual post-flop betting strategy.

6.3 Effective Hand Strength

The majority of betting decisions are made based on a variable which represents the strength of *Loki*'s hand in the current situation. Basic hand strength (HS_n) is the probability that it presently has the strongest hand. This alone is not fully adequate when there are more cards to come which can easily change the situation. For this reason, we compute the potentials $PPOT$ and $NPOT$, which tell us *Loki*'s probability of winning/losing given that it is currently behind/ahead. Using these values we can compute an estimate of *Loki*'s chances of winning at the showdown (or of being the strongest after the next card, if it is the flop and we are using $PPOT_1$). We define effective hand strength as

$$EHS = HS_n + (1 - HS_n) * PPOT - HS_n * NPOT. \tag{6.3}$$

Observe that on the river $EHS = HS_n$, since there are no more cards to be dealt.

However, there are some problems with including $NPOT$ in the calculation. First, when we bet we do not know if our opponent will play. Second, in many situations where we compute a high $NPOT$, it is often a better strategy to bet/raise to force the opponent out of the hand. So when effective hand strength is used as a betting decision (as opposed to a calling decision) it is preferable to use a more optimistic version, EHS':

$$EHS' = HS_n + (1 - HS_n) * PPOT. \tag{6.4}$$

This is an estimate which means "the probability that we are currently leading, or that we will improve to the best hand by the showdown."

```
/* BET/RAISE: when bets_to_make < num_bets
 * otherwise CHECK/CALL/FOLD appropriately */
Make(INT bets_to_make, STATE state)
{
  /* bets_to_make is the level of strength we want to show
   * num_bets is the number of bets or raises made by all players this round
   * bets_put_in is the number we have voluntarily put in this round
   * bets_to_call is the number of bets we have yet to call */
  if (state.num_bets < bets_to_make)
    return BET/RAISE
  /* We will call anything if
   * - we have voluntarily put money in the pot this round (see Section 6.1)
   * - bets_to_make >= 2 */
  else if (state.bets_put_in > 0 || bets_to_make >= 2
          || state.bets_to_call<=bets_to_make)
    return CHECK/CALL
  else
    return FOLD
}

GetAction(FLOAT HSn, STATE state, PARAMETERS param)
{
  if (HSn >= param.make2)
    return Make(2,state)
  else if (HSn >= param.make1)
    return Make(1,state)
  else
    return Make(0,state)
}
```

Figure 6.2: Simple Betting Strategy

The basic betting decision used in *Loki* is similar to Figure 6.2, with the exception that EHS' is used instead of HS_n. A **Make2** hand is defined as a hand with $EHS' \geq 0.85$. We raise when less than 2 bets have been made this round, otherwise we call. A **Make1** hand is defined as a hand with $EHS' \geq 0.50$. We bet if no one else has, and call otherwise, except when it is 2 bets to call and we have not already called a bet this round. Finally, with $EHS' < 0.50$ we consider the strategies of semi-bluffing, pot odds and showdown odds (instead of resorting to **Make0** as in Figure 6.2). These thresholds are an *ad hoc* but reasonable way to decide when to bet based on strength.

The lack of the ability to raise beyond two bets after the flop is a historical artifact, although not likely to be very limiting. Of course it must be addressed eventually, but it is a low priority function and hopefully will be superseded by a more general betting strategy.

6.4 Semi-Bluffing

The decision to semi-bluff has the next highest priority (meaning if we do not bet or call based on EHS' we consider semi-bluffing). If we are faced with 0 bets and have a high enough potential to call both a bet and a raise, we will open the betting ourselves. If none of the other players bet or raise we will continue to bet in subsequent rounds even without sufficient potential (continuing to represent a strong hand while there is a reasonable chance of winning the pot immediately). Semi-bluffing is used when $PPOT >= Pot\_Odds2$, where

$$Pot\_Odds2 = \frac{2 * bet\_size}{(pot\_size + 4 * bet\_size) + 2 * bet\_size}. \tag{6.5}$$

The term '$2 * bet\_size$' represents the bet and raise we are saying we can call. The term '$4 * bet\_size$' represents the money the bettor and raiser will be putting in to the pot to match ours (2 bets each, including the assumption that the initial bettor would call the raise). Since it is not possible to know how much is going to go into the pot, we do this as an approximation of the pot odds we would be getting.

Bluffing in this manner has a positive side effect by contributing to deception. If *Loki* never bluffed the human opposition would recognize this after a few showdowns and would quickly adapt, folding when faced with a bet or raise (lowering winnings). In fact, since *Loki* bluffs infrequently, the more experienced players on IRC often detect and exploit this predictable pattern.

6.5 Calling With Pot Odds

Pot odds is an important concept that differentiates poker from many other games, and contributes to its usefulness as a testbed for concepts in the real world. Pot odds is the comparison of your winning chances to the expected return from the pot. For example, if there is only a 20% chance that *Loki* has the best hand on the river, should we fold, call or bet? Assume the pot contains \$20 after the only opponent bets \$4. Calling in this situation will lose 4 times out of 5, at a cost of \$4 each time. However, we win 1 time out of 5 for a profit of \$20. Therefore, under these assumptions, a call is better than a fold, resulting in an average profit of \$0.80 per hand. However, if the pot only contained \$12, we should fold, since calling would yield an average loss of \$0.80 per hand.

On the flop and turn the calling decision is based on a slightly different concept. If $PPOT = 0.20$ then there is a 20% chance that the next card will give *Loki* a very strong hand. It does not necessarily win the hand, but for the sake of pot odds, we consider this to be the chance that *Loki* will clinch the hand with the next card (the current pot is our winnings). This is a basic decision and does not take into account other nuances such as the fact that the other 80% of the time *Loki* may still have a reasonable chance of winning.

To make this calling decision we verify that the pot odds justify paying to receive one more card (or to play the showdown when we are on the river). We call when

$PPOT \geq Pot\_Odds$ (or $HS_n \geq Pot\_Odds$ on the river), where

$$Pot\_Odds = \frac{to\_call}{pot\_size + to\_call}. \tag{6.6}$$

6.6 Calling With Showdown Odds

Calling with pot odds is based on the immediate situation (the cost to see one more card) and is based purely on potential for improvement (for the flop and turn). It does not cover situations where *Loki* has both a moderate $PPOT$ and a mediocre HS_n but neither is good enough to justify a call. For this reason, we introduce calling when $EHS >= Showdown\_Odds$ (since this is not a decision to bet, we must include $NPOT$, so EHS' is not used). Showdown odds is a defensive measure that ensures the opponent cannot win simply by betting at every opportunity. By calling whenever our hand is strong enough to show a profit in the long run, *Loki* discourages frequent bluffing by the opponent.

On the turn,

$$Showdown\_Odds(turn) = \frac{to\_call + bet\_size}{pot\_size + to\_call + 2 * bet\_size}. \tag{6.7}$$

Expecting to face one more bet on the river, we add one bet to the cost and one to the pot for the bet we expect to be made. On the flop,

$$Showdown\_Odds(flop) = \frac{to\_call + 4 * bet\_size}{pot\_size + to\_call + 8 * bet\_size}. \tag{6.8}$$

This case is more complex. The bet size doubles going to the turn and we expect to face (on average) one more bet on both the turn and river. For this reason, four bets of the current size are added to the cost and to the pot. This is a first approximation and ignores much of the information available, such as the number of players. Additionally on the flop it would be more appropriate to re-evaluate EHS with $PPOT_2$ and $NPOT_2$. However, it is sufficient to capture the essence of the strategy.

6.7 Other Strategies

The betting strategy in *Loki* is a simple approach to enable the use of the more sophisticated hand evaluation system. A preferable approach would be to redesign the system and attempt to make decisions based on computation of expected values (perhaps by simulations playing out the game many times). However, in our expert-strategy dependent architecture there are several more simple refinements that could be made to improve performance. Two categories of refinements are unpredictability and deception.

Unpredictability is a simple addition that makes it harder for the opponent to build a model of the program's play. Consider adding unpredictability to the betting

based on an EHS' decision: we could use a linear scale so that we bet 50% of the time with a 0.50 hand, 0% of the time with a 0.40 hand and 100% of the time with 0.60 hand.

Deception includes strategies such as pure bluffing, slowplaying and check-raising. It causes the opponent to make wrong assumptions about the current state of the game. Deception can also be used to make a play that does not necessarily lead to the highest expected value for the current game but rather is intended as "false advertising" to indirectly lead to increased profits in future hands. To be complete, all possible actions that one can witness from *Loki* should have a dual interpretation so no conclusions can be made with certainty. For example, in the present system (without check-raising), if *Loki* checks, a knowledgeable observer can infer that *Loki* has a hand with $EHS' < 0.5$.

We have included some of these deceptive strategies successfully in later versions: check-raising, pure bluffing (betting with a weak hand on the river) and balancing raises (sometimes raising instead of calling). They are currently used unpredictably (randomly). We also occasionally check-raise with a mediocre hand, so our opponents cannot infer that we always hold a strong hand when we check-raise.

6.8 Summary

The full betting strategy algorithm is presented in Figure 6.3. It is an incomplete expert system using the objective hand evaluation techniques for the betting decisions, used for its easy implementation and low cost. This version has not been adjusted to use the opponent modeling system, which was added afterwards. A more computer-oriented approach would be preferable (for example, computing expected values by playing out several simulations) but the present system is sufficient to not significantly hamper the performance, and to allow for the easy introduction and testing of other high-level betting strategies (like semi-bluffing).

```
/* Uses the Make() function defined in Figure 6.2.  */
GetAction(FLOAT EHS, FLOAT EHS', FLOAT PPOT_1, STATE state, PARAMETERS param)
{
  FLOAT pot_odds,pot_odds2,showdown_odds,showdown_cost

  /* reset the semi-bluff flag on the flop or if someone has bet or raised */
  if (state.round == FLOP || state.bets_to_call > 0)
    state.semi_bluff_flag = FALSE /* may have been set in a previous round */

  /* bet based on strength */
  if (EHS' >= param.make2)
    return Make(2,state)
  if (EHS' >= param.make1)
    return Make(1,state)

  /* decide to semi-bluff, check otherwise (no further betting decisions) */
  if (state.bets_to_call == 0)
  {
    pot_odds2 = 2*state.bet_size / (state.pot_size + 6*state.bet_size)
    if (state.semi_bluff_flag || (state.round != RIVER && PPOT_1 >= pot_odds2))
    {
      state.semi_bluff_flag = TRUE
      return BET
    }
    return CHECK /* all following decisions are CALL/FOLD */
  }

  /* check pot odds */
  pot_odds = state.bets_to_call / (state.pot_size + state.bets_to_call)
  if (state.round == RIVER && EHS >= pot_odds)
    return CALL
  if (state.round != RIVER && PPOT_1 >= pot_odds)
    return CALL

  /* check showdown odds - on flop and turn only */
  if (state.round == RIVER)
    return FOLD
  if (state.round == FLOP)
    showdown_cost = state.bet_size * 4
  else
    showdown_cost = state.bet_size
  showdown_odds = (state.bets_to_call + showdown_cost) /
    (state.pot_size + state.bets_to_call + 2*showdown_cost)
  if (EHS >= showdown_odds)
    return CALL

  /* give up */
  return FOLD
}
```

Figure 6.3: Post-Flop Betting Strategy

Chapter 7

Opponent Modeling

In strategic games like chess, it is acceptable to assume that the opponent is perfect because that assumption does not result in a significant loss in performance for the algorithm against weaker players. Opponent modeling has been attempted in two-player games as a generalization of minimax but with limited success [5] [9]. In contrast, not only does opponent modeling have tremendous value in poker, it can be the primary distinguishing feature between players at different skill levels. If a set of players all have a comparable knowledge of poker fundamentals, the ability to alter decisions based on an accurate model of the opponent may have a greater impact on success than any other strategic principle.

Having argued that some form of opponent modeling is indispensable, the actual method of gathering information and using it for betting decisions is a complex and interesting problem. Not only is it difficult to make appropriate inferences from certain observations and then apply them in practice, it is not even clear how statistics should be collected or categorized.

This chapter describes opponent modeling in *Loki*. Using the betting history of the opponents, it determines a likely probability distribution for their hidden cards which is used by the hand evaluation system. A minimal system might use a single fixed model for all opponents in a given hand, based on the cards played to the flop by typical players. The system in place in *Loki* generates a model for each opponent and maintains information between games. It is a simplistic first approximation that takes specific observed information and transforms it to a more useful form. However, this is sufficient to demonstrate a significant performance improvement using only a simple analysis of context.

The first section discusses the representation of the model: the action frequencies for select decision categories and the weight array for all the possible two card holdings. The second section explains the learning system (how the weight array is determined and how the action frequencies are used for this). The final section discusses how the weight array is used in hand evaluation, in particular how the field array is calculated.

7.1 Representation

There are two structures that comprise an opponent model, the weight array and the action frequencies, and both are updated by the learning portion of the opponent modeling system. The weight array is actually a model of the opponent's hand; it is reset at the beginning of each hand and contains information pertaining to what cards the opponent may be holding. It is used by the hand evaluation system to give more accurate estimates of hand strength. The action frequencies are inter-game statistics on how many times the opponent has taken each action in a given situation. They are used by the learning system to appropriately update the weight array, and they could be used by the betting strategy system.

7.1.1 Weight Array

The weight array is a model of the opponent's hand. Each opponent p has an array of weights $w_p[h]$ where h represents each possible two card hand. There is a weight associated with each h (reset to 1 each new hand) which approximately represents the conditional probability that player p would have played in the observed manner (given that they held hand h). We call these values weights because they act as multipliers in the enumeration computations. If we want a probability distribution of possible hands held by p, we normalize the array by dividing each entry by the total sum of all the entries (the number of valid entries depends on which hands are still possible). The normalized entries $w'_p[h]$ represent the conditional probability that player p holds hand h (given the betting history).

This is a very detailed representation, although there is some error since interdependencies are not considered. In the multi-player scenario we do not address the fact that different opponents cannot hold the same card. For example, if we assign a high relative probability to one specific opponent holding an Ace, we do not reduce the relative probabilities of the other opponents holding an Ace. This also means that when a player folds, the information regarding which hands that player may have held is discarded. The weight array is not adjusted with respect to the weight arrays of the other opponents. However, we feel this error is minor in comparison to the savings in complexity and computation time.

7.1.2 Action Frequencies

When an opponent's action is used to adjust their weight array, it is important to take into account what type of opponent it is. As a first approximation we can assume all opponents play the same (*e.g.* raise with the top 15% of hands, and so on). In this case, all that matters when adjusting the weight array is the action taken, and not the type of player. This is called *generic* opponent modeling because the reweighting system is identical for each player. However, initial tests against human opponents revealed that this was an incorrect assumption. A loose and aggressive opponent might bet with almost anything and therefore a bet should not be taken

too seriously. On the other hand, a tight and passive opponent will usually only bet with a very good hand.

To account for this we introduce *specific* opponent modeling. Statistics for each opponent are tabulated and used to calculate betting frequencies, and these frequencies adjust how the re-weighting system works. For example, if we observe that a certain opponent bets 80% of the time when acting first on the flop, then we will take this into account when adjusting the weight array for a bet observed by that opponent in that same situation.

This introduces a new problem. Statistics can be retained for a large number of categories. Consider, for example, that we index each frequency by the number of active opponents (1, 2, 3+; 3+ is the '3 or more' category), total raises this round (0, 1, 2+), bets to call (0, 1, 2+), and game round (pre-flop, flop, turn or river). The '$n+$' classification is useful for putting together similar cases. In this example, each situation fits into one of $3 * 3 * 3 * 4 = 108$ categories. It would take a large number of observed games before we had sufficient data for some categories. In fact, this categorization could easily be made more complex by also taking into account other important variables such as table position, remaining callers, and previous action taken (to catch advanced strategies like check-raising).

The finer the granularity of the data collected, the harder it is to get meaningful information (more observations are required). There are two ways to make it more manageable. The first way is to distinguish many different categories for the data collection phase (a fine level of granularity) and then combine classes that have similar contexts in the data usage phase. The other, less sophisticated, approach is to simply have few different categories for the data collection phase. This allows the frequency calculations of the data usage phase to be quick and simple. This is what we have done.

Data is categorized into 12 categories, indexed by (0, 1, 2+) bets to call and the 4 different game rounds. This is a simple first approximation but is sufficient to determine if this level of opponent modeling can be beneficial. It also has the advantage of enabling the program to learn 'quickly', although less accurately. It is easy to adjust the definition of the context for the purpose of gathering statistics. The learning system (the re-weighting system that uses this information) is only interested in the frequency of an observed action and not how that figure was calculated.

For each opponent p, we determine the category c of the observed action (the *context* – amount to call and game round) and record the action a taken (fold, check/call, bet/raise) by incrementing $T_p[c][a]$. When we need to know the frequency that a player has taken a certain action in a certain situation, $f_p[c][a]$, we simply compute the ratio of $T_p[c][a]$ versus $S_p[c]$ (the total number of actions recorded for that context):

$$S_p[c] = \sum_{i=0}^{2} T_p[c][i], \text{ and} \tag{7.1}$$

$$f_p[c][a] = \frac{T_p[c][a]}{S_p[c]}. \tag{7.2}$$

Default Frequencies

Until enough data has been accumulated for a category (20 points, this value was selected arbitrarily), we weight the frequencies with hard-coded defaults[1] $d[c][a]$, and mix the observed frequencies with the defaults, using linear interpolation:

$$f'_p[c][a] = \begin{cases} f_p[c][a] * \frac{S_p[c]}{20} + d[c][a] * \frac{20 - S_p[c]}{20} & \text{when } S_p[c] < 20; \\ f_p[c][a] & \text{otherwise.} \end{cases} \tag{7.3}$$

However, each player will play the earlier rounds more than the later rounds (*i.e.* more pre-flops than flops, and so on). Assuming that a player's style does not change drastically between rounds, we can take advantage of data gathered in prior rounds and use only one set of defaults and a recursive definition of frequency. To this end, we must first adjust the notation to consider the round. Given that the definition for the category c is composed of a round r and number of bets to call b, we can also define c as (r, b) where $r = 0$ for the pre-flop. Now we can recursively define the new frequency function f'' based on the reduced set of defaults $d'[b][a]$:

$$f''_p[r,b][a] = \begin{cases} d'[b][a] & \text{when } r < 0; \\ f''_p[r,b][a] * \frac{S_p[r,b]}{20} + f''_p[r-1,b][a] * \frac{20 - S_p[r,b]}{20} & \text{when } S_p[r,b] < 20; \\ f_p[r,b][a] & \text{otherwise.} \end{cases} \tag{7.4}$$

An alternative approach would have been to use *Loki*'s own frequencies as the default behavior for opponents, however the majority of human players on IRC were observed to play much looser than *Loki*. The default frequencies were therefore based on a typical IRC player (and do not involve any additional computation).

7.2 Learning

Each time an opponent makes a betting action, the weights for that opponent are modified to account for the action. For example, a raise increases the weights for the stronger hands likely to be held by the opponent given the flop cards, and decreases the weights for the weaker hands.

With this re-weighting system we consider two distinct levels of modeling, as discussed in Section 7.1.2. First, an opponent's betting actions are used to adjust the weights. The actual transformation function used for the re-weighting is independent of the player in question. A different weight array is still maintained for each opponent, but a raise observed in a certain category is treated the same for all players. Second, we maintain data between games (the action frequencies) and

[1] See Appendix B for the values used.

these frequencies are used to adjust the transformation function itself. This technique is called *specific* opponent modeling, because the re-weighting depends on the opponent's model. In fact, the only difference between the two levels is that without specific opponent modeling, the re-weighting function always uses the generic default frequencies (*i.e.* $f''_p[r,b][a] = d[b][a]$).

The remainder of this section discusses the general idea of the re-weighting system, and then presents the specific details with respect to the pre-flop and post-flop rounds.

7.2.1 Re-Weighting System

The weight adjustment for any particular subcase is based on the threshold hand value needed for the observed action. The threshold is in turn based on the observed frequency of folding, calling and raising of the player in question (or the default frequencies when we are using generic opponent modeling or have insufficient data). From these frequencies we deduce the average (μ, representing the mean hand) and spread (σ, to account for uncertainty) of the threshold needed for the observed action.

The values μ and σ define an expected distribution of hands to be played, realizing a transformation function for re-weighting. We take some ranked distribution of all possible hands held (IR for the pre-flop and EHS' for the post-flop rounds, meaning we presume our opponents rank hands like we do) and each hand h is compared to μ and σ and re-weighted accordingly (see Figures 7.1 and 7.2). When the value for h is equal to μ, the re-weighting value is 0.5; when it is more than σ below μ, it is 0.01; when it is more than σ above μ, it is 1 (the weight is unchanged); and when it is within σ of μ it is linearly interpolated between 0.01 and 1 (we use a simple linear interpolation because it is easy and we do not know what the distribution should look like). Since we do not want to completely rule out any legal subcase we do not allow the weight to go below 0.01.

The value μ is based on the threshold needed for the observed action, which is in turn based on the frequency. For example, consider that player p has been observed 100 times in the pre-flop with 2 bets to call: 20 times it was raised, 70 times called and 10 times folded. If we wanted to re-weight based on a raise we compute the frequency of raising: $F = f''_p[0,2][raise] = \frac{20}{100} = 0.20$, meaning the threshold is $\mu = 1 - F = 0.80$ (player p raises with the top 20% of hands). However, if we wanted to re-weight based on a call (frequency 0.70) we must note that the threshold for a call is not $\mu = 1 - 0.70 = 0.30$ but is rather 0.70 below the raise threshold: $\mu = 0.80 - 0.70 = 0.10$ (player p calls with the middle 70% of hands). There are other intricacies dependent on the actual round of play, such as determining σ and using IR values (which are non-percentile), which will be discussed in the following sections. One immediately noticeable source of error is that this system presumes a proper distribution over the hand rankings (*i.e.* the hands above $\mu = 0.80$ represent 20% of all hands). This is not true for the post-flop rounds, because EHS' is optimistically increased with $PPOT$. However, the relative ranking of hands is still correct.

As a final note, to prevent the weight array from being distorted by automatic or false actions, we only perform one re-weighting per model per round. We store a

```
constant  low_wt 0.01
constant high_wt 1.00

/* weight[] is the weight array from the beginning of the round */
Reweight(FLOAT μ, FLOAT σ, FLOAT weight[])
{
  FLOAT reweight
  FLOAT handvalue /* depending on the round this is either IR or EHS' */
  CARDS hand
  for each subcase(hand)
  {
    handvalue = GetHandValue(hand)
    /* interpolate in the range μ ± σ */
    reweight = (handvalue - μ + σ) / (2*σ)
    if (reweight<low_wt) reweight = low_wt
    if (reweight>high_wt) reweight = high_wt
    weight[hand] = weight[hand] * reweight
    /* don't let the weight go below 0.01 */
    if (weight[hand]<low_wt) weight[hand] = low_wt
  }
}
```

Figure 7.1: Re-weighting Function Code

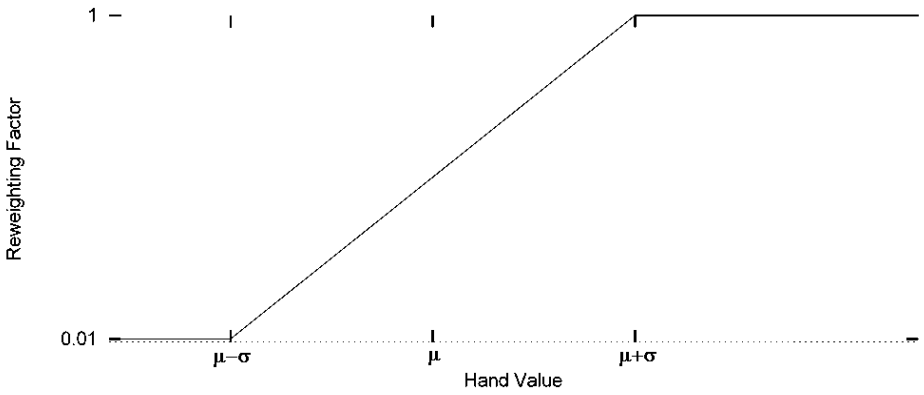

Figure 7.2: Re-weighting Function

copy of the weight array at the beginning of the round and each time a new action is witnessed requiring a higher threshold, the saved weight array is used in the re-weighting. For example, if we witness opponent p calling a bet, we may re-weight using some μ, say 0.5. If, later in the betting round, we see that opponent raise, re-weighting will be done with the higher value of μ, and is based on the original weight array.

7.2.2 Pre-Flop Re-Weighting

In the pre-flop we do not have the convenience of a percentile hand valuing system. So μ needs to be converted from a percentile value to a value in the IR scale. To achieve this we use μ to index into a sorted array (sample the nearest point) of the 1326 IR_7 values from Appendix A (for simplicity IR_7 is always used). For example, suppose an opponent raises 30% of all hands, this translates to hand with $IR = +118$ (roughly corresponding to an average of 0.118 bets won per hand played) so we now use $\mu = 118$.

We do not observe the consistency of an opponent adhering to the estimated threshold, or of any other specific tendencies. While two separate opponents may call on average with a 118 hand, they both may have a very different standard deviation to the distribution of hands they call with (one may rarely call with a hand below 0 while another may sometimes play a hand as low as -200). For the present implementation, we have selected $\sigma = 330$ in an *ad hoc* manner: 68.26% of the hands (or two standard deviations in a normal distribution) lie in the income rate range -323 to +336. However, we only use a linear interpolation for re-weighting values within the range $(\mu - \sigma, \mu + \sigma)$ rather than an S-curve based on a normal distribution.

However, there is one clear source of error when μ is very low. Consider a player who has been observed to take a certain action in a certain situation 100% of the time. We re-weight with $\mu = -495$ (the lowest value), but this means that even the hands with the lowest IR will only be re-weighted with a factor of 0.5 (which should be 1). For this reason, we do not re-weight when μ is below the 5th percentile (-433 in IR_7). A more accurate fix is possible, but is unlikely to be worth the added complexity.

7.2.3 Post-Flop Re-Weighting

For the three betting rounds after the flop, we infer a mean and variance (μ and σ) of the threshold for the opponent's observed action and rank all hands according to the effective hand strength (EHS') calculation from Equation 6.4. Although the relative ranking of all hands is not affected, it is an optimistic view, because hands with an EHS' above $\mu = .80$ represent more than 20% of all hands (due to the inclusion of $PPOT$ in the equation). Additionally, we use HS_1 instead of HS_n so the number of opponents can instead be addressed as part of the context of the action frequencies (although presently it is ignored in the interest of a simplified definition of context).

To calculate EHS' we must compute both hand strength (HS) and positive potential ($PPOT$). Since $PPOT$ is a computationally expensive operation and we

must obtain $PPOT$ values for about 1,000 hands every call to the re-weighting function, we use a crude but fast estimating function ($PPOT_c$). In terms of cost, $PPOT_c << PPOT_1 << PPOT_2$. It is *crude* because it ignores much of the available information (such as weight arrays) and looks for a few features about the hand, using heuristics to associate a value to each. For example, if the board has only 2 diamonds of our 4-card diamond flush draw, then we have 9 outs (1 out per remaining card), but if the board has 3 diamonds, each of the remaining cards is worth 0.5 to 1 out, depending on the rank of your suited hole card (a Two is worth 0.5 and an Ace is worth 1). This is because we are more likely to be up against other diamond flush draws. $PPOT_c$ approximates very roughly our winning chances given each possible card to be dealt). This function produces values within 5% of $PPOT_1$, 95% of the time (this is from a random sampling of five card unweighted cases). Since these values are amortized over about 1,000 possible hands, the overall effect of this approximation is small.

However, the opponent is also likely to play some hands based on the pot odds required for the action (*i.e.* the decision is not always based on EHS'). For this reason we have introduced an adjustment to the re-weighting algorithm for the post-flop. When $PPOT_c$ is sufficient to warrant calling the pot odds, which is

$$\frac{amount\_put\_in}{pot\_size + amount\_put\_in},$$

the weight for that subcase is not reduced. This is a simplification of the betting decision the opponent could be making based on the pot size, however we feel it is sufficient to capture enough cases to prevent hands with low EHS' but high potential from being severely under-estimated. The new re-weighting algorithm can be found in Figure 7.3.

What value do we use for σ? We chose to use a typical value of $\sigma = .2$ at $\mu = .5$ (interpolating over the range .3 to .7) and to increase σ with smaller μ while decreasing it for larger μ. This reflects the tendency for *loose* players (with low μ) to exhibit more uncertainty and *tight* players (with high μ) to adhere more consistently to the threshold. Hence, we use $\sigma = 0.4 * (1 - \mu)$.

However, as was the case with pre-flop re-weighting, there is a clear source of error when μ is very low. Observe that the area of the re-weighting function is a rectangle of height 1 and width $1 - (\mu + \sigma)$, with a triangle between $\mu - \sigma$ and $\mu + \sigma$ (Figure 7.2). That is,

$$A = (1 - (\mu + \sigma)) + \frac{2 * \sigma}{2} = 1 - \mu \tag{7.5}$$

(if we were to ignore the area due to the minimum re-weighting factor of 0.01). But, since the function domain is bounded by 0, when $\sigma > \mu$ ($\mu < 0.2857$) we have $A < 1 - \mu$. For the weaker hands, the re-weighting factor is too low. When we are in this special case, we compensate by changing the re-weighting function so that at $EHS' = 0$ the re-weighting factor is r instead of 0.01 (and the function interpolates linearly between r and 1 when the hand value is between 0 and $\mu + \sigma$).

The value r is calculated to give $A = 1 - \mu$. The function now looks like a rectangle to the right of $\mu + \sigma$, a triangle of height $1 - r$ (width $\mu + \sigma$) and a second rectangle

```
constant  low_wt 0.01
constant high_wt 1.00

/* weight[] is the weight array from the beginning of the round
 * state.pot_odds is the Pot Odds required for the observed action */
PostflopReweight(FLOAT μ, FLOAT σ, FLOAT weight[], STATE state)
{
  FLOAT reweight
  HANDVALUE value /* member elements are PPOT_c and EHS' */
  CARDS hand
  for each subcase(hand)
  {
    /* Compute crude potential PPOT_c and hand value EHS' */
    value = GetHandValue(hand)
    /* Do not change weight if potential is sufficient for pot_odds */
    if (state.round != RIVER && value.PPOT_c >= state.pot_odds)
      next subcase
    /* interpolate in the range μ ± σ */
    reweight = (value.EHS' - μ + σ) / (2*σ)
    if (reweight<low_wt) reweight = low_wt
    if (reweight>high_wt) reweight = high_wt
    weight[hand] = weight[hand] * reweight
    /* don't let the weight go below 0.01 */
    if (weight[hand]<low_wt) weight[hand] = low_wt
  }
}
```

Figure 7.3: Post-Flop Re-weighting Function

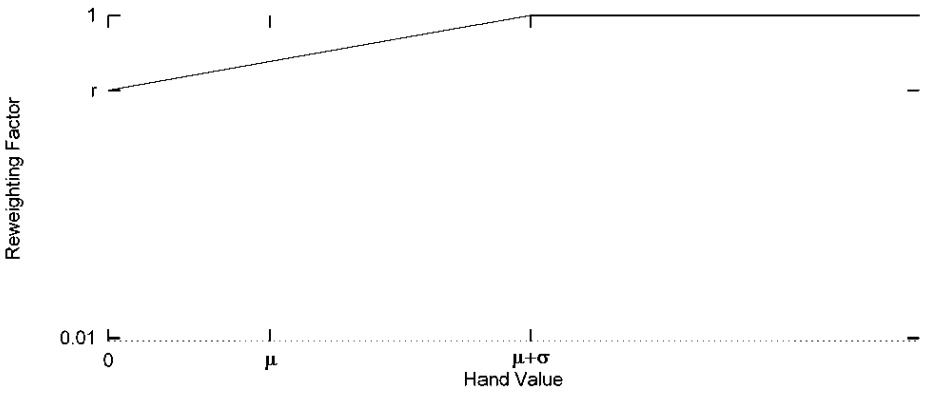

Figure 7.4: Re-weighting Function With $\mu < \sigma$

of height r below it (Figure 7.4). The total area of the function is now the area of the triangle subtracted from 1:

$$A = 1 - \frac{(1-r)(\mu+\sigma)}{2}. \tag{7.6}$$

Since we want $A = 1 - \mu$ we then see that

$$1 - \mu = 1 - \frac{(1-r)(\mu+\sigma)}{2} \tag{7.7}$$

which leads to the conclusion that

$$r = 1 - \frac{2 * \mu}{\mu+\sigma}. \tag{7.8}$$

7.2.4 Modeling Abstraction

In the above discussion, it is suggested that when we maintain weight arrays for our opponents we only maintain all known possible cases (for the flop 1,081 cases of the original 1,326 since we hold two known cards). However, when we compute EHS' for an opponent in the re-weighting function, what do we use for the weight array? Our opponent does not know the two cards we hold, meaning if we hold two aces it is not correct to assume our opponent is aware there are two less aces in the deck. Additionally we may want to consider that sometimes an opponent's estimation of EHS' is inflated by in turn observing actions denoting weakness from other opponents (and ourselves).

To address these problems we have added a level of abstraction to the opponent modeling in the original design. All the opponent models, including a model of ourselves (in effect our opponents' model of us), are gathered and adjusted using the *public* information available (our hole cards, *private* information, are not considered public information). In the re-weighting step for a particular player, when EHS' is calculated, a *field array* (Section 7.3) is used which is composed from the weight

arrays of all the opponents of that player. As a result, the second re-weighting in a round is performed within the context of the first.

Of course, when we compute hand values for our betting strategy, we use all available information and do not include all the subcases which use one or both of our hole cards. Since the weight arrays have been normalized with these subcases included, some additional minor error is introduced.

7.3 Using the Model

At present, the action frequencies are only used in the re-weighting module (although this type of information could be useful in the betting strategy module). The only opponent information used externally is the weight array, which is used in the hand evaluation module (Chapter 5). In that context, since each subcase has a specific weight, the array is used to calculate weighted sums and to infer better estimates of hand strength (which can be expressed as the total weight of all subcases that are weaker than you, added to half the total weight of all tied subcases, divided by the total weight of all subcases). Note that when used in this fashion the weight array does not need to be normalized.

7.3.1 The Field Array

In Chapter 5 we were presented with a problem: how to use the information available in multiple weight arrays to get a general value for strength and potential. The solution was the *field array*, an intuitive approach that fits into the existing framework. We average (or add, since in the relative sense they are the same operation) the weight arrays of all the opponents to give a new array representing the entire table. However, since the weights represent relative probabilities, we must normalize the individual weight arrays (to the same scale) before they can be combined. Specifically, when we need to calculate hand values for player p, we calculate a field array that is the average of the normalized weight arrays of all players except p.

This approach does introduce a small amount of error by abandoning some second-order considerations with respect to intersection cases, but again this error is minor when compared to the reduction in complexity. For example, if we want to compute the probability, given the appropriate normalized weight arrays, that either statement A (player 1 holds A♠-A♡) or statement B (player 2 holds A♠-A♡) is true we need to compute

$$P(A \; or \; B) = P(A) + P(B) - P(A \; and \; B).$$

In fact, it is significantly more complex than this. For each subcase we would need to also rule out all the other intersecting subcases (*e.g.* the probability that player 1 holds A♠-A♡ precludes player 2 holding either the A♠ or the A♡).

7.4 Summary

The opponent modeling module includes a data accumulation system to infer action frequencies, and a model of the player's hand represented by an array of weights. The action data gathered is based on a coarse definition of context, however it is only intended as a simple framework to examine the feasibility of the approach. It is easy to adjust the definition of context because the learning system is only interested in the resulting frequency of an observed action, and not how that value was calculated. A lot of the potential data that could be accumulated is currently ignored. For example, variance is not measured so all opponents are presently assumed to have the same level of uncertainty or consistency in their actions. We also do not consider recency (applying more weight to recent data points) although this is a more complex change to the present system.

There are two different levels of opponent modeling that can be examined. In generic opponent modeling the action frequencies are always assumed to be the same (predetermined defaults) and the only opponent modeling that is done is by inferring the weight array (the re-weighting system). In specific opponent modeling, the observed action frequencies (specific to each opponent) are used to adjust the re-weighting system itself. The difference between the two systems can be examined in the function at the heart of the opponent modeling (Figure 7.5).

The re-weighting system is the complex portion of the opponent modeling system. It involves learning the distribution of probable hands held based on observed actions and storing this information in an array of weights (which are the relative probabilities that the opponent would have played that hand given the observed actions that game). The re-weighting system is given a certain μ and σ representing the threshold required for making the observed action, under whatever hand ranking measure is most appropriate: IR for the pre-flop and EHS' for the post-flop rounds. A linear interpolation transformation function (based on μ and σ) is applied to the weight array to give a new weight array. There are some problems with the system. For example, σ is a fixed expert value when it should be based on observations. However, more importantly we never do inverse re-weightings (*i.e.* a call or check should suggest an upper threshold of hands the opponent likely does not have).

Of course, the opponent's decisions may in reality be based on a different metric than IR or EHS', resulting in an imperfect model. There are other problems with the re-weighting system, such as the presumption that the hand rankings are properly distributed (EHS' is an optimistic view). However, forming a perfect model of the opponent is in general unboundedly difficult, since their exact actions are unknowable. New techniques can improve the results, but the current method does capture much of the information conveyed by the opponent's actions.

In competitive poker, opponent modeling is more complex than portrayed here. One also wants to fool the opponent into constructing a poor model. For example, a strong poker player may try to create the impression of being very conservative early in a session, only to exploit that image later when the opponents are using incorrect assumptions. In two-player games, the M* algorithm allows for recursive definitions

```
/* Update the opponent model given an observed action */
HandleAction(PLAYER p, ACTION action, PARAMETERS param)
{
  FLOAT μ, σ
  /* The model for player p is composed of:
   * - frequency count T_p
   * - frequency f''_p (see Equation 7.4)
   * - weight array w_p */

  /* BLIND is a forced action so gives no information */
  if (action == BLIND)
    return

  /* this is the only difference between SPECIFIC and GENERIC;
   * GENERIC always uses the defaults */
  if (parameter.modeling == SPECIFIC)
    T_p[state.round,state.bets_to_call][action] += 1

  /* we do not re-weight on a FOLD or CHECK */
  if (action == FOLD || action == CHECK)
    return

  /* This function is described in Section 7.2 and Equation 7.4 */
  μ = GetThreshold(T_p,state.round,state.bets_to_call,action)

  /* In both cases, the second parameter is an ad hoc value for σ (see Section 7.2) */
  if (state.round == PREFLOP)
    Reweight(μ, 330, w_p)
  else
    PostFlopReweight(μ, 0.4 * (1 − μ), w_p, state)
}
```

Figure 7.5: Central Opponent Modeling Function

of opponent models, but it has not been demonstrated to improve performance in practice [5].

We have maintained a certain level of abstraction in our modeling system. For example, we maintain an opponent model of ourselves (our opponent's model of us) and all opponent models are maintained using *public* information (meaning when we re-weight the board cards are known, but we do not presume that our own hole cards are). However, we do not attempt to manipulate this information. It merely makes the re-weighting system more accurate since EHS' is then calculated with respect to information our opponent has available.

Chapter 8

Experiments

Experimentation provides a simple way to determine the effectiveness of changes, or to identify potential problems and shortcomings in the system. For *Loki* we have used a variety of different experimental methods. The primary method is self-play simulation between different versions of the program. The other methods, much more difficult to interpret but presented as anecdotal evidence, involve play against other computer programs or human opponents through an on-line poker server running on IRC (Internet Relay Chat).

8.1 Self-Play Simulations

Self-play simulations offer a convenient method for the comparison of two or more versions of the program. In addition to verifying that a certain enhancement has a beneficial effect, it is possible to quantify the contribution made by each new component to the system. Since all participants in the simulated game are versions of the program, play can proceed at a rapid pace, and results can be based on large (statistically significant) sample sizes.

The self-play simulations use the duplicate tournament system described in [2], based on the same principle as duplicate bridge. Since each hand can be played with no memory of the cards dealt in preceding hands, it is possible to replay the same deal, but with the participants holding a different set of hole cards each time. This system simulates a ten-player game. Each hand is replayed ten times (ten *trials*), shuffling the seating arrangement so that every participant has the opportunity to play each set of hole cards once, and no two players are seated in the same relative position more than once (so, for instance, each player will play directly behind each other player exactly once). The hole cards are always in the same betting order so those belonging to the small blind are identical in each trial. The seating permutations are listed in Table 8.1 (T = 10).

This arrangement greatly reduces the "luck element" of the game, since each player will have the same number of good and bad hands. The differences in the performance of players will therefore be based more strongly on the quality of the decisions made in each situation. This large reduction in natural variance means that

Round	Seat Number for Each Player									
	1	2	3	4	5	6	7	8	9	T
1	1	2	3	4	5	6	7	8	9	T
2	2	4	6	8	T	1	3	5	7	9
3	3	6	9	1	4	7	T	2	5	8
4	4	8	1	5	9	2	6	T	3	7
5	5	T	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	T	5
7	7	3	T	6	2	9	5	1	8	4
8	8	5	2	T	7	4	1	9	6	3
9	9	7	5	3	1	T	8	6	4	2
T	T	9	8	7	6	5	4	3	2	1

Table 8.1: Seating assignments for tournament play (reproduced from [2])

meaningful results can be obtained with a much smaller number of trials than in a typical game setting.

There are numerous different ways to use self-play simulation to test different versions of the program. One simple application would be to play five copies of a new version against five copies of an older version, differing only in the addition of one new feature. If the new component has improved the program (against itself), then the newer version will win against the older version. The average margin of victory, in terms of expected number of small bets per hand, can also give a preliminary indication of the relative value of the new enhancement.

However, one must be careful when drawing conclusions from self-play experiments. It is important to not over-interpret the results of one simulation [1]. With the above format, there are limitations to how much can be concluded from a single experiment, since it is representative of only one particular type of game and style of opponent. It is quite possible that the same feature will perform much worse (or much better) in a game against human opposition, for example. A wider variety of testing is necessary to get an accurate assessment of the new feature, such as changing the context of the simulated game.

However, most versions of *Loki* are very similar and have fairly conservative styles. It is quite possible that the consequences of each change would be different against a field of opponents who employ different playing styles. For example, against several human players, the effect of the weighting function may be much bigger than that of hand potential. Inter-dependencies between the involved players can also affect results. The second-best player may perform first overall if it can exploit a particular bad player more than the best player can.

8.2 Other Experiments

Loki has been tested for extended periods of time in more realistic settings against human opposition (and an occasional computer player). For this purpose, the program participates in an on-line poker game, running on the Internet Relay Chat

(**irc.poker.net**). Human players connect to IRC and participate in games conducted by dedicated server programs. Bankroll statistics on each player are maintained, but no real money is at stake, and this may contribute to results being a little optimistic. There are three different games available for limit Texas Hold'em. For the first level, you begin with $1000 and the betting scale is 10-20. Once you have $2000 (*i.e.* won a net of $1000) you are allowed to play in a second game where the betting scale is 20-40. At $5000 you are allowed to play in a third game where the betting scale is 50-100. The competition becomes much stronger at each level, but it is more difficult to find opponents, so games at the higher levels are less common. Early versions of *Loki* participated in games with 2 to 12 players. Later the server was changed to allow only 2 to 10 players.

Playing short-handed (2-4 players) emphasizes the need for strong opponent modeling. Typically, with many players (5 or more) the computer can win in the long run by playing the odds, usually because there are some bad players. With only a few opponents, we face many one on one situations where the non-mathematical elements of the game become more important. When those few players are strong (or colluding), *Loki* loses a large amount of money. It does not "give up" in a bad situation, so it continues to lose. Since games go by significantly faster, there is too much data in this limited context. We ignore results for games with 2 to 4 players because the overall results are distorted. All the reported results are for games with 5 to 12 players (or 5 to 10 in the later sessions).

As this is not a closed environment, the natural variance in these games is very high, and the results depend strongly on which players happen to be playing. Consequently, not enough information has been gathered to make any safe conclusions.

Very early versions of *Loki* had mixed results on the IRC server, but played too few games to be conclusive. However, it appeared to play at about the same level as the average human participant in the open games, roughly breaking even over the course of about 12,000 hands. Opponent modeling appeared to be much stronger: in one session of 8,894 games (5 to 12 players), a version using generic opponent modeling (GOM) achieved a winning rate of 0.07 small bets per hand (this places *Loki* comfortably in the top 10% of players who play the 10-20 games on the server). In a later session of 29,301 games (5 to 10 players), another version that used specific opponent modeling (SOM) achieved a winning rate of 0.08 small bets per hand. While the difference between the two modeling versions may not be significant, they both win consistently and perform much better than the previous versions.

Recognizing that many human opponents were easily identifying when *Loki* had a strong or weak hand (occasional semi-bluffing did not add enough deception), we added some new deceptive strategies: *pure bluffs* (betting with the weakest hands on the river), *balancing raises* (occasionally raising instead of calling), and *check-raising* (following a check with a raise in the same round). Check-raising is normally used with the strongest hands, but to ensure that no information can reliably be gained from any particular action, we also use "fake" check-raises with mediocre hands. However, these are simply more expert rules (*e.g.* check-raise 60% of the time with three callers behind us when $EHS' >= 0.92$). We are interested in machine-

dependent approaches to computer poker where *Loki* can discover for itself what the best strategy is in a situation (which makes it easier to introduce opponent modeling into such decisions). So we are not interested in the performance contribution of any particular strategy. However, the introduction of these advanced tactics probably explains why the following results are better. The two opponent modeling versions that use these features are GOM' and SOM'.

We used these stronger features to see if we could find a noticeable performance difference between GOM' and SOM'. In 35,607 games, SOM' maintained a winning rate of 0.12 small bets per hand. In 36,299 games, GOM' maintained a winning rate of 0.10 small bets per hand. This is stronger evidence that specific opponent modeling is better. In fact, we believe that it may not be worth as much against the weaker class of human players (in the 10-20 game) and may lead to a stronger disparity in the higher level games.

In the stronger IRC game (20-40), earlier versions of the program without opponent modeling lost, averaging about -0.08 small bets per hand in 2,354 games. This is too small a sample size for conclusive results, but strongly suggests it was a losing player overall in these games. Opponent modeling demonstrated a noticeable difference at this level; SOM averaged about +0.05 small bets per hand in 34,799 games. This was probably influenced by good results early on (before the human players had adjusted to the new style of *Loki*) so it is probably closer to a break-even player. However, this is noticeably better than the earlier version without opponent modeling. We have have not tested GOM, GOM' or SOM' at this level.

A third form of competition was introduced strictly against other computer programs on the IRC server, called **BotWarz**. In **BotWarz I**, four programs participated, using three copies of each in a 12-player game. Two programs, R00lbot and *Loki*, were clearly dominant over the other two, Xbot and Replicat, with the more established R00lbot winning overall. In 39,786 hands, *Loki* averaged about +0.03 small bets per hand. It should be noted, however, that this competition is representative of only one type of game, where all the players are quite conservative. Replicat in particular performed much better in the open games against human opposition than in this closed experiment, in which it lost the most money. Also, despite the closed environment, the variance was still quite high.

There were also noticeable interdependencies between the different players. Late in the competition, Replicat dropped out and it became apparent that *Loki* may have been taking advantaging of this player more than the other two. In a final session of 23,773 hands, *Loki* lost 0.03 small bets per hand.

After some changes, like the introduction of the new pre-flop system (but prior to opponent modeling) *Loki* participated in **BotWarz II**. Again, there were 3 copies each of 4 different programs; this time the opponents were Prop, R00lbot, and Xbot. Prop only played roughly the first 9,000 hands and Xbot only played roughly the first 18,000 hands. In 10,103 games with all 12 players, *Loki* averaged a winning rate 0.03 small bets per hand. Against only Xbot and R00lbot it lost at a rate of approximately 0.02.

Finally, **BotWarz III** was played after the introduction of the 10-player limit.

This time there were five programs, with two copies of each. The four opponents were USAbot, Xbot, R00lbot and Fishbot (USAbot and Fishbot are most similar in design to Xbot). The results had a much higher variance because, in addition to only having two copies, numerous players kept dropping out and coming back in. This time, Xbot easily won the most money overall while USAbot and Fishbot lost the most. In the first 19,000 hands of the tournament, both GOM and SOM approximately broke even.

For the latter part of the tournament (significantly longer) GOM and SOM were replaced by GOM' and SOM' (recall they used additional expert rules for bluffing and check-raising). GOM' played 64,037 hands, but half of these were with 8 players. One quarter involved 6 players and the remaining quarter involved only 4 players. Over all the games, it broke even, but with all the players involved (3,840 hands) it achieved a winning rate of 0.05 small bets per hand. SOM' won about 0.01 small bets per hand over all the games, and 0.06 over the 3,840 games with all 10 players. The results of these tournaments suggest that a simple program with a decent betting strategy (expert rules) can play better than a program with many other strengths, but a weak link in its betting strategy.

In addition to programs by hobbyists playing over IRC, there are numerous commercial programs available. However, we have not tested *Loki* against them because we have not found any with a programmable interface. Hence, it is not known if they are better or worse.

One final important method of evaluation we have not mentioned is the critique of expert human players. Experts can review the play of the computer and determine if certain decisions are "reasonable" under the circumstances, or are indicative of a serious weakness or misconception. Based on this opinion, it appears to be feasible to write a program that is much stronger than the average human player in a casino game, although *Loki* has not yet achieved that level.

8.3 Betting Strategy Experiments

As a sample self-play experiment, we have tested five different versions of *Loki* together, using different components of the betting strategy, for a 10,000 hand tournament (100,000 trials). We use the average bankroll of the two copies of each version as a metric for performance. The results can be seen in Figure 8.1 (note this is with a 2-4 betting structure). Player A used the entire betting strategy, and B, C, and D each lacked a particular feature (B did not use showdown odds, C did not use pot odds, and D did not use semi-bluffing). Finally, player E used the simple betting strategy from Figure 6.2 (only HS_n is used). All five versions used a *moderate tightness* level, and generic opponent modeling to ensure reasonable weights.

This experiment reveals the danger of over-interpreting self-play simulations. It suggests that every feature except showdown odds is a gain, especially semi-bluffing. Player B won the tournament by a large margin suggesting that *Loki* is better off without the showdown odds feature. However, in practice this is often not the case.

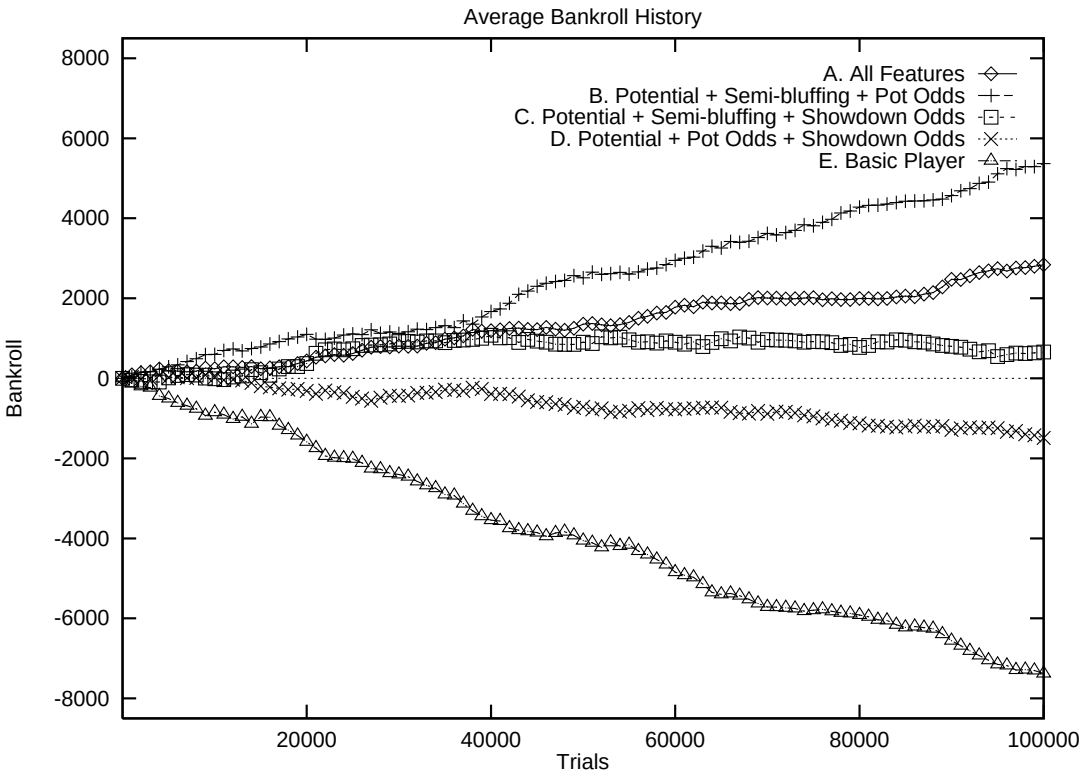

Figure 8.1: Betting Strategy Experiment

For example, in one long IRC session (4857 games) against a variety of human opponents, showdown odds were considered 883 times and used to call and continue playing 123 times. In these games, *Loki* later folded 30 times for a total cost of \$108 (scaled to a betting structure of 2-4), won 6 games without a showdown, and won 37 of the 87 remaining hands that went to a showdown. The total winnings were \$1221 and the total cost (of the initial call and later betting actions) was \$818. Thus, the EV of showdown odds was a net gain of \$3.28 per decision, or 1.64 small bets (the EV of a folding decision is \$0 since there is no cost and there are no winnings). Although this was a good session, the EV for this feature was consistently positive in other sessions. In a longer set of 12,192 games, it was \$0.72 per decision (or 0.36 small bets). Although the performance after the decision point is dependent on the performance of the betting strategy overall, *Loki* would have netted \$0 in these situations, instead of a consistent gain, without showdown odds.

Showdown odds was originally added because *Loki* often over-estimated what its IRC opponents were holding. Bets were taken too seriously and EHS' would be just under the betting threshold. With showdown odds we will typically decide to play a hand which has a $PPOT$ that is just under the calling threshold and an EHS' that is just under the betting threshold. When *Loki* over-estimates its opponents, showdown odds is usually a gain. However, in self-play, because it plays a very tight game (the other extreme), a bet is not taken seriously enough (EHS' is too high). So when we decide to play the showdown odds it is often a mistake. However, it will be profitable

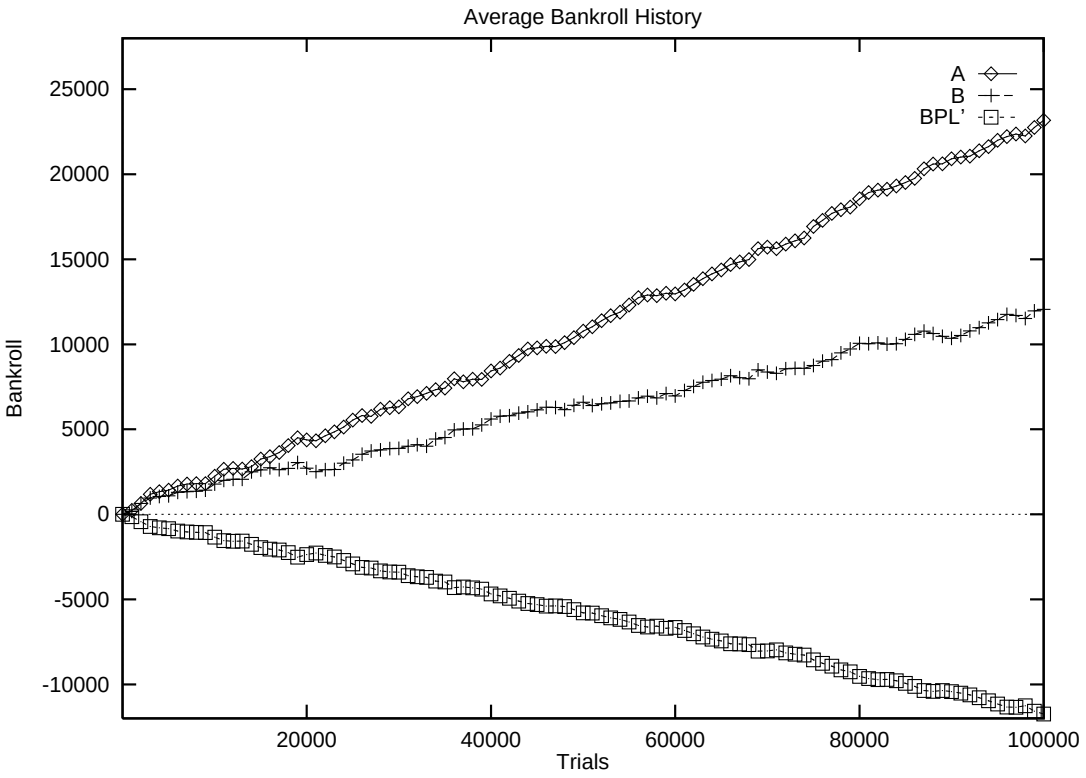

Figure 8.2: Showdown Odds Experiment

in an environment where opponents are frequently bluffing or otherwise betting too much.

To see the reverse of this effect (showdown odds as a winner in self-play), Figure 8.2 shows the results from an experiment between two copies of A (all features), two copies of B (no showdown odds), and six copies of BPL'. BPL ("Best Player Loose") is a *loose* non-modeling player, who uses all features of the betting strategy but with reduced thresholds for looser play. This player also uses a fixed weight array for all opponents regardless of their actions (since it performs no re-weighting, a "reasonable" set of weights is much more realistic than uniform weights). BPL' is the same as BPL, except it ignores the number of opponents (uses HS_1 instead of HS_n in EHS calculations) for much more aggressive play. In this experiment, it is clear that using showdown odds resulted in a significant performance gain, although both A and B had a large advantage over BPL'.

Because we recognize that the present betting strategy is a potential limitation and is in need of re-designing, we are not particularly interested in the value of each particular feature. Since showdown odds is theoretically a gain (positive expected value) when the weights are accurate, this experiment shows the limitations of the present system and reinforces the need for good opponent modeling. The main focus of our experimentation is to examine the benefits and problems of opponent modeling, including the difference between generic and specific modeling.

8.4 Opponent Modeling Experiments

Humans can be very good at opponent modeling because they can make inferences (extrapolation) based on limited data to identify general trends or errors in an opponent's play. This ability relies on an opponent being predictable, but this is often the case against all but the best players. For example, you may observe an opponent showing a failed flush draw in the showdown twice in situations where they made calls with very poor pot odds. You then infer that this opponent over-values flush draws until evidence contrary to this conclusion is presented.

For a computer, it is difficult to identify what parts of the context of an action are important, or how to make accurate inferences without large quantities of data. It is also difficult to make a computer identify trends outside of its mathematically understood value system (probabilistic measures of potential and strength), such as a particular opponent's over-optimistic evaluation of flush draws. Such a feature would require the computer to somehow learn how each opponent values the various features of particular hands. This is a complex problem, unless possible tendencies are anticipated so the computer can look for them.

There are numerous statistics that a computer could gather in a poker game, and conclusions that could be inferred with sufficient data. Every time an opponent reveals cards in the showdown, a retroactive analysis of betting actions in that game could be used for data. However, the majority of observations are not so informed. Without showdowns, the complete betting history of a player could be recorded, and data could be classified by a variety of contextual information: betting round, bets to call, bets put in that round, number of active opponents, and the previous action by this player.

As a first cut, *Loki* does not use the extra information presented in showdowns, and makes inferences using only information for which it has sufficient data points. Until 20 data points have been acquired for a particular context, the opponent modeling action frequencies are based on a weighted average between some pre-defined defaults and the observed data. The context definition used is coarse in granularity, considering only betting round and bets to call. This is a simple approach to learn more about the requirements for opponent modeling. A better approach might be to try to identify which aspects of the context are most valuable for consideration.

8.4.1 Generic Opponent Modeling (GOM)

Generic modeling is our first attempt at opponent modeling. It assumes that our opponents use a value scale similar to our own. The observed actions of each opponent are used to adjust their weight array. No statistics are gathered, so the re-weighting system treats all actions equally, dependent only on the context and regardless of which player it is.

We pitted together four different versions of *Loki* for 100,000 trials. The focus of the experiment was four copies of *GOM* (a player using all betting strategy features, generic opponent modeling, and a *tightness* setting of *loose*). The competition was

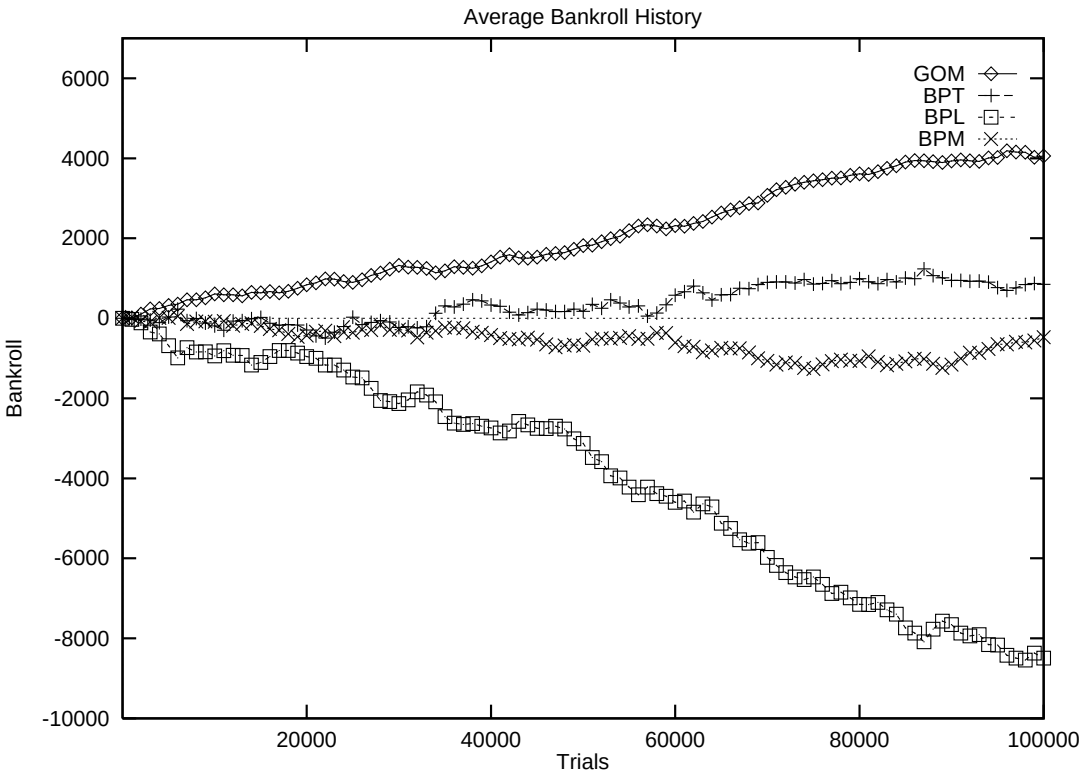

Figure 8.3: Generic Opponent Modeling Experiment

three different non-modeling players (two copies of each, also using all betting strategy features):

- BPL ("Best Player Loose") is a *loose* player, who uses (0.4, 0.8) for ($make1$, $make2$), and subtracts 100 from all pre-flop IR thresholds for looser play.
- BPM ("Best Player Moderate") is a *moderate* player who otherwise uses all the defaults.
- BPT ("Best Player Tight") is a *tight* player, who uses (0.6, 0.9) for ($make1$, $make2$), and adds 100 to all pre-flop IR thresholds for tighter play.

Figure 8.3 shows the results of the experiment. GOM quickly demonstrated clear dominance while the three non-modeling players were ranked based on the tightness of their play. As expected, GOM is able to exploit the basic players because its model of how they play is fairly accurate, and is used to make better decisions. GOM might not perform as well against players with very different styles of play, because its model would be less accurate, but it would be better than using no modeling at all.

8.4.2 Specific Opponent Modeling (SOM)

Specific modeling is our first attempt at using the observed betting history to distinguish different types of players. It is clearly what human experts use, although

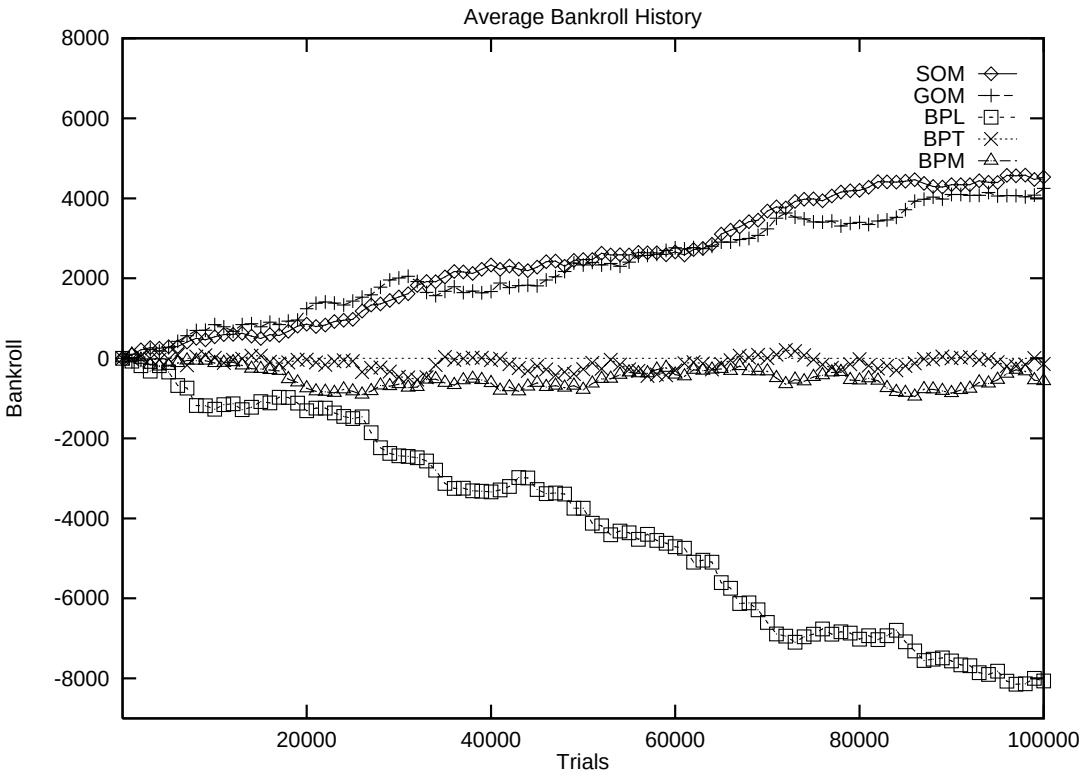

Figure 8.4: Specific Opponent Modeling Experiment

our approach is crude and only captures the essence. We measure action frequencies based on a rough description of context and use these frequencies to appropriately adjust the weight array on future actions.

The experiment's focus was on two copies of SOM (specific opponent modeling) with respect to two copies of GOM. Like the previous experiment, the other six players were two copies each of BPL, BPT and BPM. The results can be seen in Figure 8.4.

Very quickly, the two opponent modeling programs asserted their superiority over the non-modeling versions. SOM is able to attain a comparable degree of success to GOM based on observed frequencies rather than a good default model. While we cannot conclude on the basis of this one experiment that there is a statistically significant difference between the two modeling versions, anecdotal evidence is somewhat promising: on IRC GOM achieved a winning rate of 0.07 small bets per hand and SOM achieved a winning rate of 0.08 small bets per hand.

The advantage of good opponent modeling is clear. *Loki* with opponent modeling is a noticeably stronger program than without it. However, our implementation of specific modeling does not appear to produce a significant advantage over generic modeling. We recognize that the granularity of the context for gathering action frequencies may be so coarse that the error undermines the gain. For example, *Loki* does not recognize that, for some players, calling a bet is automatic given they have already called a bet previously in that round – the second action contains no in-

formation. Some other variables which may make the identification of the context stronger include the previous action, number of opponents and number of callers remaining. However, the number of cases becomes very large so it would seem necessary to somehow combine the action frequencies of "similar" scenarios.

Additionally the re-weighting system is not adjusted based on the specific opponent (which only provides simple frequencies). It would be better if we could predict how each opponent would handle each specific subcase rather than using our value system for assigning strength/potential with a simple linear function. For example, some opponents may over-value flush draws, or may find it more imperative to bet a mediocre hand with low potential. The present re-weighting system itself neglects handling certain information. When a check or call is observed we could infer that the opponent does not have a hand above a certain threshold and can apply an inverse re-weighting function to the weight array. This would prevent *Loki* from being too pessimistic and would allow it to bet more often after several checks (currently, since checks are ignored and *Loki* uses HS_n, a checking opponent still poses the same threat as one who has not yet acted).

8.5 Summary

There are several different ways to evaluate *Loki*'s performance, however since self-play is the most controlled, results are easily measured. But, we must always be careful in interpreting the results (such as taking into account the interdependencies between players). With only self-play, it is difficult to measure *Loki*'s strength. To help determine this, we use evidence gathered from play on IRC which suggests that it is better than the average human amateur (at least in multi-player scenarios). In particular, all of the evidence indicates that the addition of opponent modeling results in a significant increase in strength.

The performance gain from generic modeling is conclusive, however the further superiority of specific modeling is not so clear. This could be due to the granularity of the data gathering, or how the action frequencies are used in the re-weighting, or because the information is not used in the betting strategy. In fact, we strongly believe that the present *ad hoc* betting strategy, which was originally designed to allow us to quickly test other components, may be a bottleneck preventing further significant progress. This is witnessed in **BotWarz** with the relatively good performance of less sophisticated computer programs, which put more effort into designing a (rule-based) betting strategy. It is also evidenced by the ease with which strong human players can take advantage of *Loki* in short-handed play (too many head to head situations against better players).

Chapter 9

Conclusions and Future Work

Loki successfully demonstrates beneficial opponent modeling in a high-performance game-playing program. In closed self-play experiments it was clearly beneficial to use modeling, and the results from IRC play are also promising. However, it does not necessarily follow that it will be equally successful in games against strong human players. Humans can be very good at opponent modeling, and less predictable than the players in these experiments.

In our self-play experiments, we have not yet investigated modeling opponents who vary their strategy over time. There are also many other interesting questions to be addressed. Our approach was a first approximation using an intuitive approach, and the major benefits came from the introduction of the weight array (and the re-weighting). The enumeration algorithms for hand evaluation are well suited to this expression of opponent modeling, allowing it to be a very useful asset to the accounting system.

The overall performance was hampered by the *ad hoc* betting strategy. In fact, many aspects of *Loki* were a tradeoff between usefulness and correctness – in many places we selected the simple (and cost-effective) approach for its reasonable approximations. Provided the error is not one-sided we should see an amortizing effect. We have not actually examined what the error is in our many approximations, but it is not worth the effort until it is a limiting aspect of play. Since we plan on replacing the betting strategy with something less dependent on expert rules, it is not worth examining the benefits of particular features. Similarly, we feel that **Bot Warz** or general IRC play could have had better results had we put more time into the betting strategy. However, this would have amounted to tweaking artificial parameters without general applicability.

The betting strategy should also use the opponent modeling information. A good approach might be to run simulations to the end of the game using the weight array to randomly select "reasonable" opponent hands (and to weight the results as in our enumeration techniques). The specific opponent information could then be used to predict opponent actions in the simulations, resulting in estimates of the expected value for each of our options. Presumably, strategies such as check-raising or bluffing would emerge naturally. For example, bluffing may turn out to be the best action in

a situation where we recognize that our opponent is likely to fold.

The specific opponent modeling program (SOM) was hampered by the crude method used for collecting and applying observed statistics. Much of the relevant context was ignored for simplicity, such as the previous action taken by a player. A more sophisticated method for observing and utilizing opponent behavior would allow for a more flexible and accurate opponent model.

The re-weighting system could be adjusted, such as inverse re-weightings for passive actions like checking/calling. Presently, every witnessed action leads to the opponent's average hand getting "stronger". If we considered upper thresholds on actions implying some weakness, like checking and calling, we could appropriately re-weight their weight array. Specific modeling could also observe variance, or the consistency the opponent exhibits in their behavior. This information could be used in the re-weighting function instead of the simple linear function we use (with a fixed σ).

Poker is a complex game. Strong play requires the player to handle all aspects of the game adequately, and to excel in several. Developing *Loki* seems to be a cumulative process. We improve one component of the program until it becomes apparent that another aspect is the performance bottleneck. That problem is then tackled until it is no longer the limiting factor, and a new weakness in the program's play is revealed. We have made an initial foray into opponent modeling and are pleased with the results, although it is far from a completed subject.

Wherever possible, the project should be driven to remove whatever human expert information is used. Betting strategy is clearly a major component that needs to be addressed. However, there are other candidates such as more sophisticated opponent modeling. Eventually, more sophisticated simulations for learning good pre-flop play could be based on *Loki*'s post-flop playing ability.

Concepts such as hand strength and potential are appropriate for any poker variant. While parts of our implementation, such as the weight array, may be specific to Texas Hold'em, our ideas are easily mappable to other variants.

Is it possible to build a program which is the best poker player in the world? Certainly we can construct a program which has a very strong mathematical basis and runs within the real-time constraints. It is also clear that some form of opponent modeling (in addition to other advanced features) are necessary to beat the better players. However, it is not clear how difficult it will be to build and maintain opponent models that are sufficiently detailed and context sensitive. While we are probably close to a program which can win money in most typical low-limit casino games, we are far from the lofty goal of being the best in the world.

Bibliography

[1] H. J. Berliner, G. Goetsch and M. S. Campbell. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43(1):7–20, April 1990.

[2] D. Billings. Computer poker. Master's thesis, University of Alberta, Dept. of Computing Science, 1995. http://www.cs.ualberta.ca/~games/poker.

[3] D. Billings, D. Papp, J. Schaeffer, D. Szafron. Opponent modeling in poker. In *AAAI-98 Proceedings*, pages 493–499. The MIT Press, 1998. http://www.cs.ualberta.ca/~games/poker.

[4] D. Billings, D. Papp, J. Schaeffer, D. Szafron. Poker as a testbed for AI research. In Robert E. Mercer and Eric Neufeld, editors, *Advances in Artificial Intelligence*, pages 228–238. Springer-Verlag, 1998. http://www.cs.ualberta.ca/~games/poker.

[5] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. *AAAI*, pages 120–125, 1996.

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

[7] N. Findler. Computer model of gambling and bluffing. *IRE Transactions on Electronic Computers*, EC-10(1):5–6, March 1961.

[8] N. Findler, H. Klein, W. Gould, A. Kowal, and J. Menig. Studies on decision making using the game of poker. *Proceedings of IFIP Congress 1971*, pages 1448–1459, 1972.

[9] H. Iida, I. Kotani, J.W.H.M. Uiterwijk, and H.J. van den Herik. Gains and risks of OM search. In H.J. van den Herik and J.W.H.M. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 153–165, Maastricht, The Netherlands, 1997. Department of Computer Science, Universiteit Maastricht.

[10] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, July 1997. http://robotics.Stanford.EDU/~koller/papers/galapaper.html.

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

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

[13] D. Sklansky. *The Theory of Poker.* Two Plus Two Publishing, Las Vegas, NV, 1994 3rd ed.

[14] D. Sklansky and M. Malmuth. *Hold'em Poker for Advanced Players.* Two Plus Two Publishing, Las Vegas, NV, 1994 2nd ed.

[15] N. Zadeh. *Winning Poker Systems.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1974.

Appendix A

Pre-Flop Income Rates

These are the computed income rates (* 1000) used for all 169 distinct hand types (13 paired, $\binom{13}{2} = 78$ suited and $13 * \binom{4}{2} = 78$ unsuited). Each table is labeled IR_x where x is the number of players (hands dealt) in the simulation (so there are $x - 1$ opponents). Each entry is indexed $IR_x[row][col]$ and the cards are suited when $row > col$. This means that for 2 players the income rate for a 3 and 2 of the same suit is $IR_2[3][2] = -279$, and for a 3 and 2 of different suits is $IR_2[2][3] = -351$.

In all simulations a pair of aces had the highest income rate (a gain of 2.043 with 7 players, meaning an investment of \$1 would return a *profit* of \$2.043, on average). In the 7-player simulation, a 2 and 7 of different suits had the lowest income rate (a loss of \$0.495 for every \$1 invested) and 88 of the 169 different hand types returned non-negative income.

There is a strong correlation between these rankings and the pre-flop hand rankings given in Sklansky and Malmuth [14]. They break the pre-flop hands into 9 groups, ranked by their strength (call this ranking of hands SM). If we take IR_7 (most reflective of the full game they are assuming) and break it into the same number of groups with the same number of hands per group (5 for group 1, 5 for group 2, 6 for group 3, and so on) we note that all but 16 of the 169 hand types are either in the same group or are only one away. While most of the hands in the middle groups are shifted by one class, the top three groups are virtually identical. Details of the comparison by groups can be found in Table A.4.

The most interesting similarity is that the top two groups contain the same hands but there is only one different hand in the third group. In IR_7, **KTs** (King and Ten of the same suit) replaces **JTs** (Jack and Ten of the same suit). In fact, there appears to be a trend favoring big cards in IR_7. However, any minor discrepancy could be due to the simple-minded approach of the simulations.

	2	3	4	5	6	7	8	9	T	J	Q	K	A
2	7	-351	-334	-314	-318	-308	-264	-217	-166	-113	-53	10	98
3	-279	74	-296	-274	-277	-267	-251	-201	-148	-93	-35	27	116
4	-263	-225	142	-236	-240	-231	-209	-185	-130	-75	-17	46	134
5	-244	-206	-169	207	-201	-189	-169	-148	-114	-55	2	68	153
6	-247	-208	-171	-138	264	-153	-134	-108	-78	-43	19	85	154
7	-236	-200	-162	-125	-91	324	-99	-72	-43	-6	37	104	176
8	-192	-182	-143	-108	-75	-43	384	-39	-4	29	72	120	197
9	-152	-134	-122	-84	-50	-17	16	440	28	65	106	155	215
T	-104	-86	-69	-56	-19	12	47	81	499	102	146	195	254
J	-52	-35	-19	0	11	46	79	113	149	549	161	212	271
Q	2	21	34	55	72	86	121	153	188	204	598	228	289
K	63	79	98	116	132	151	168	200	235	249	268	647	305
A	146	164	180	198	198	220	240	257	291	305	323	339	704

Table A.1: IR_2: income rates for 1 opponent

	2	3	4	5	6	7	8	9	T	J	Q	K	A
2	-121	-440	-409	-382	-411	-432	-394	-357	-301	-259	-194	-116	16
3	-271	-42	-345	-312	-340	-358	-371	-328	-277	-231	-165	-87	54
4	-245	-183	52	-246	-269	-287	-300	-308	-252	-204	-135	-55	84
5	-219	-151	-91	152	-200	-211	-227	-236	-227	-169	-104	-24	118
6	-247	-177	-113	-52	256	-145	-152	-158	-152	-145	-74	9	99
7	-261	-201	-129	-65	3	376	-76	-79	-68	-66	-44	48	148
8	-226	-204	-140	-73	-2	66	503	0	15	24	45	84	194
9	-191	-166	-147	-79	-5	68	138	647	104	113	136	177	241
T	-141	-116	-91	-69	-4	75	150	235	806	226	255	295	354
J	-89	-67	-41	-12	7	82	163	248	349	965	301	348	410
Q	-29	-3	22	51	80	108	185	274	379	423	1141	403	473
K	47	76	101	128	161	199	230	318	425	473	529	1325	541
A	175	211	237	266	249	295	338	381	491	539	594	655	1554

Table A.2: IR_4: income rates for 3 opponents

	2	3	4	5	6	7	8	9	T	J	Q	K	A
2	-6	-462	-422	-397	-459	-495	-469	-433	-383	-336	-274	-188	-39
3	-180	21	-347	-304	-365	-418	-447	-414	-356	-308	-248	-163	-1
4	-148	-69	67	-227	-273	-323	-362	-391	-334	-287	-223	-133	32
5	-121	-38	31	122	-198	-230	-270	-303	-309	-259	-200	-103	64
6	-174	-95	-10	64	206	-151	-175	-204	-217	-235	-164	-72	23
7	-208	-135	-47	35	108	298	-87	-106	-112	-128	-124	-26	72
8	-184	-164	-83	2	93	168	420	-5	6	-10	-10	22	126
9	-146	-128	-111	-26	64	153	245	565	134	118	118	151	189
T	-88	-68	-46	-29	59	155	268	383	765	299	305	336	373
J	-38	-15	1	30	51	147	256	377	536	996	380	420	462
Q	35	49	72	99	127	162	268	384	553	628	1279	529	574
K	117	141	167	190	223	261	304	423	591	669	764	1621	712
A	269	304	333	363	313	365	416	475	644	720	815	934	2043

Table A.3: IR_7: income rates for 6 opponents

Group	Size	Matches in IR_7 grouping	Group	Size	Matches in IR_7 grouping
1	5	5	6	10	2
2	5	5	7	17	6
3	6	5	8	16	1
4	8	4	9	85	69
5	17	11			

Table A.4: Comparison between SM and IR_7

Appendix B

Expert-Defined Values

Loki uses several hard-coded constants where the optimal value is too difficult to determine (or we have not invested the effort to determine a better value). They have been selected by a poker expert, Darse Billings, and have not been experimentally validated. They are used as placeholders, and we foresee eventually upgrading the system so that the computer can determine proper behavior on its own.

For the post-flop betting strategy, there are only two expert-defined values: *make1* and *make2* (these are absolute constants, 0.50 and 0.85 respectively, not to be confused with **Make1** and **Make2**). These are the betting and raising thresholds for EHS', and represent the likely hand strength required for these actions to be profitable.

For the pre-flop, when the function *SetThresholds* is called, it is given three parameters: *group*, *tightness* and *position*. The first two parameters are used to retrieve a set of values ([*base, increment*], one per *strategy*). Then the function generates a threshold for each strategy using Equation 6.1, which in turn uses *position*.

- *probability_play*: This value is hard-coded at 0.6, meaning when estimating the number of players (for *group*) we expect 60% of all players, who have not yet acted, to play to the flop.

- *group*: For the pre-flop, we have reduced the number of classifications, based on the number of players, to three groups: 2, 3-4 and 5 or more players. We consider these the most important groupings and the IR rankings for more than 4 players are nearly identical in any case.

- *tightness*: There are three settings (*tight*, *moderate* and *loose*). By default *Loki* plays with the *loose* setting. Note that all three sets of parameters are relatively tight styles of play. Even the *loose* set of parameters is considerably more conservative than a typical loose human player.

- *strategy*: There are several different pre-flop strategies that we may use: **Make0**, **Call1**, **Make1**, **Call2**, **Make2** and **Make4**. Each strategy has a different threshold value based on the context, except **Make0** which is the folding strategy. The values for **Call1** and **Make1** are the same, as are the values for **Call2**

2 players			
	tight	*moderate*	*loose*
Make1	(-50,50)	(-50,50)	(-50,50)
Make2	(150,50)	(50,50)	(0,0)
Make4	(300,0)	(300,0)	(300,0)
3-4 players			
	tight	*moderate*	*loose*
Make1	(50,50)	(50,25)	(50,10)
Make2	(200,50)	(200,25)	(200,10)
Make4	(580,0)	(580,0)	(580,0)
5 or more players			
	tight	*moderate*	*loose*
Make1	(0,70)	(0,50)	(0,30)
Make2	(450,50)	(450,25)	(450,10)
Make4	(900,0)	(900,0)	(900,0)

Table B.1: Values for [*base, increment*]

	action a		
bets to call x	Fold	Check/Call	Bet/Raise
0	0	0.5	0.5
1	0.5	0.3	0.2
2+	0.7	0.2	0.1

Table B.2: Default frequencies $d'[x][a]$

and **Make2**, except for the case of the small blind, which has fixed values for these two strategies depending only on *group*.

- *base, increment*: There is one pair of [*base, increment*] values defined per *group*, *strategy* and *tightness* (presented in Table B.1). For the small blind, the **Call1** threshold is fixed at 0 for 5 or more players, -75 for 3-4 players, and equal to the **Make1** threshold for 2 players. The **Call2** threshold is 450 for 5 or more players, 200 for 3-4 players, and equal to the **Make2** threshold for 2 players.

For opponent modeling, we use expert values in the re-weighting function.

- *Default Frequencies*: In the absence of sufficient data, the action frequencies are weighted by hard-coded default frequencies (Equation 7.4). The set of values for $d'[x][a]$ are presented in Table B.2.

- *Re-weighting Function* σ: As described in Section 7.2, for the pre-flop we use a σ of 330. For the post-flop we use $\sigma = 0.4 * (1 - \mu)$, to reflect the fact that *tighter* players adhere to a narrow range of threshold hands.

Appendix C

Glossary

This is a glossary of both technical terms (indicated with *tt.*) and poker jargon used in the thesis. Many of the poker term definitions are based on selected excerpts from the *rec.gambling Glossary of Poker terms*, used by permission of the main author, John C. Hallyburton, Jr.. http://www.conjelco.com/faq/rgpglossary.html.

- **Action Frequency:** *n. tt.* The frequency a certain opponent makes a certain action in a certain context – based on inter-game statistics.
- **Active Player:** *n.* A player who is still in the pot.
- **Ante:** *n.* A small bet all players are required to make before a hand is dealt. Not all games have an ante. Related terms: **blind**.
- **Bankroll:** *n.* Current total gambling funds available. To be distinguished from the current money you happen to have on the table. See also: **stake**.
- **Bet:** *v.* To put money into the pot, pursuant to the rules of the game, thus maintaining a chance of winning the pot.
- **Bet For Value:** *v.* Betting a hand that, in the long run, is expected to win more than it loses. Antonym: **bluff**.
- **Bettor:** *n.* In a given round, the first person to put money in the pot.
- **Big Blind:** *n.* A blind bet, usually a raise of an earlier blind which would be called the **small blind**. In limit poker, the **big blind** is usually the size of the minimum bet on the first round of betting.
- **Blind:** *n.* A mandatory bet made by certain player(s) usually sitting left of the **button** before each new hand is dealt. Used in place of, or in conjunction with, antes. See also: **ante**, **big blind**, **small blind**.
- **Bluff:** *n.* A bet or raise made with a poor hand, hoping that the remaining active player(s) will fold. Can also be used as a verb.

- **Board:** *n*. The exposed cards in Hold'em. Also called **board cards**. See also: **community cards**.
- **Button:** *n*. A distinctive token held by the player sitting in the theoretical dealer's position (when the dealer is not a player in the game). The button rotates around the table so that every player has an opportunity to be the last to act. "**The button**" can refer to the player who currently has the button.
- **Call:** *v*. To put in to the pot the minimum amount of money necessary to continue playing.
- **Caller:** *n*. One who calls. Sometimes used collectively, as in "3 callers."
- **Cap:** *v*. To cap the betting is to make the last permitted raise in a round. In this case, the betting is said to have been **capped**.
- **Check:** *v*. To bet zero, when it is legal to do so. Frequently a sign of only a fair hand.
- **Check-Raise:** *v*. To check initially, then raise a bet made later on in the same betting round. Frequently a sign of strength, but may be a bluff. See also: **sandbag**.
- **Chip:** *n*. A gaming token used in place of cash for convenience in handling and counting. The standard form of currency in most casinos.
- **Community Cards:** *n*. Cards that are available for every player to use in making a hand. Usually dealt face-up somewhere in the middle of the table. See also: **board**.
- **Deuce:** *n*. A two.
- **Draw:** *n*. [1] A class of poker games characterized by players being dealt 5 cards face-down and later having the opportunity to replace some of the original 5. "Draw poker" and "Five-card draw" are examples of usage.
- **Draw:** *n*. [2] In Hold'em games, the set of cards that will be dealt later can be collectively called "the draw."
- **Draw:** *v*. To discard some number of cards and have dealt an equal number of replacements.
- **Early Position:** *n*. Being one of the first players to act in a betting round. See also: **middle position**, **late position**.
- **Effective Odds:** *n*. A refinement to **pot odds** which includes estimated extra winnings and cost to see more than one card. The ratio of the expected winnings, when you make your hand, to the cost to play. Used when considering a hand with a high potential given more than one card to come.

- **EHS:** *n. tt.* Acronym for *effective hand strength*: the probability of being the strongest hand in the future (*i.e.* using $PPOT$ and $NPOT$). EHS' is an optimistic version which ignores $NPOT$ (used when EHS is considered for a betting decision instead of for a calling decision).

- **Face Card** *n.* A jack, queen or king (a card with a face on it, not joker).

- **Face-Down:** *adj.* Specifies a card that is known only to the owning player (*e.g.* your hole cards in Hold'em).

- **Face-Up:** *adj.* Specifies a card is that is known to all players (*e.g.* **community cards** or **board**).

- **Field Array:** *n. tt.* The average of the normalized weight arrays of all opponents.

- **First to Act:** *n.* First active player following the **button**.

- **Fixed Limit:** *n.* A betting structure where the amount of each bet is a specific fixed quantity. Usually specified as A-B, where A is the amount to bet in the first few betting rounds and B (larger than A) is the amount bet in the later rounds. Related terms: **flat limit**, **no limit**, **pot limit**, **spread limit**.

- **Flat Limit:** *n.* A variant of **fixed limit** where all bets are the same amount.

- **Flop:** *n.* In Hold'em, the first three community cards, dealt simultaneously.

- **Flush:** *n.* A poker hand consisting of five cards all of the same suit.

- **Flush Draw:** *n.* Four cards of the same suit (*i.e.* missing one card to make a flush).

- **Fold:** *v.* To decline to call a bet, thus dropping out of a hand.

- **Four of a Kind:** *n.* A hand containing all four cards of the same rank.

- **Free Card:** *n.* A card dealt after all players checked in a betting round.

- **Full House:** *n.* A hand consisting of **three of a kind** and a (different) pair.

- **GOM:** *n. tt.* Acronym for *generic opponent modeler*: a version of *Loki* that uses default **action frequencies** for all opponents (the **re-weighting** system is the same for all opponents). GOM' is a later version that uses additional betting strategies like check-raising. Compare: **SOM**.

- **Hand:** *n.* [1] A player's hand is the set of cards that only they may use at the showdown. In Texas Hold'em, a player's hand is their set of **hole cards**.

- **Hand:** *n.* [2] One full game of poker (*i.e.* from the blinds or ante until the pot is awarded). See also: **trial**.

- **High Card:** *n.* The weakest poker hand, 5 unmatched cards.
- **Hit:** *v.* To **make** a hand or catch a card or cards that improves one's hand (*e.g.* you hold 2-3-4-6 and a 5 is dealt, giving you a straight). Antonym: **miss**.
- **Hold'em:** *n.* [1] Generic name for a class of poker games where the players receive a certain number (2 to 4) of hole cards and 5 community cards. Usually there are betting rounds after dealing the hole cards, then after dealing 3 face-up cards (**flop**), after dealing a 4th face-up card (**turn**) and finally after dealing a 5th face-up card (**river**).
- **Hold'em:** *n.* [2] When used in the specific sense (*e.g.* "We're playing Hold'em") the term usually refers to the game of Texas Hold'em.
- **Hole Cards:** *n.* In certain poker variants, such as Hold'em, the face-down cards dealt to each player. Sometimes called the **hole**.
- **HR:** *n. tt.* Acronym for *hand rank*: the probability of being the strongest hand in the present state, against one random hand. HR_n is the hand rank against n hands.
- **HS:** *n. tt.* Acronym for *hand strength*: the probability of being the strongest hand in the present state. HS_n is the hand strength against n opponents.
- **Immediate Odds:** See **pot odds**.
- **Implied Odds:** *n.* A refinement to **pot odds** which includes money not yet in the pot. Considers the ratio of estimated extra winnings, when a player forms a good hand, to the present cost to call.
- **In:** *adj.* Still eligible to win the pot. "I'm in" is often spoken as one calls.
- **Inside Straight:** *n.* Four cards to a straight, where only one rank will complete the hand. For example, 4-5-6-8 is an inside straight since only a 7 will complete the hand. Compare: **open-ended straight**.
- **Kicker:** *n.* In hands containing **pairs** and **three of a kind**, the highest card not matched. In draw games, sometimes a card kept for deception purposes.
- **Late Position:** *n.* For a particular betting round, a player who does not have to act until most of the other players have acted. See also: **early position**, **middle position**.
- **Limit Poker:** *n.* A poker game wherein the amount to be bet is fixed, or at most variable within a prescribed minimum and maximum. Antonym: **no-limit poker**. See also: **fixed limit**.
- **Loose:** *adj.* Playing more hands than the norm. Antonym: **tight**.

- **Middle Position:** *n*. Betting positions approximately halfway around the table from the first player to act. See also: **early position**, **late position**.
- **Make:** *v*. To **make** a hand is to receive cards that improve one's hand. See also: **hit**.
- **Miss:** *v*. To receive a card that does not improve one's hand. Antonym: **hit**.
- **No-Limit Poker:** *n*. A game where there is no maximum bet; a player can wager any amount (perhaps above some minimum) up to whatever money is on the table in front of him. Antonym: **limit poker**. See also: **fixed limit**.
- **NPOT:** *n. tt*. Acronym for *negative potential*: the probability of falling behind given that we are ahead (approximately, the percentage of upcoming cards that help our opponents). $NPOT_1$ is the one card potential and $NPOT_2$ is the two card potential. Compare: **PPOT**.
- **Off-Suit:** *adj*. Not of the same suit. "I held A-Q off-suit" or "The flop was 10-6-2 off-suit." When speaking of 5 or more cards, then not all of the same suit (*i.e.* no flush).
- **One Pair:** *n*. The second weakest poker hand. It contains two cards of the same rank and 3 unmatched cards (**kickers**).
- **Open:** *v*. Make the first bet in a hand, especially in draw poker.
- **Open-Ended Straight:** *n*. Four cards to a straight which can be completed by drawing a card at either end. *E.g.* 6-7-8-9 is an open-ended straight. Compare: **inside straight**.
- **Out:** *n*. A card that will improve your hand, often substantially. A hand with many **outs** is preferable to a hand with only a few. *E.g.* with a flush draw in diamonds, any diamond is an out.
- **Out:** *adj*. Folded, ineligible to bet or win this hand. "I'm out" is often a synonym for "I fold."
- **Overpair:** *n*. In Hold'em, a pair in the hole that is larger than any community card on the board.
- **Pair:** *n*. Two cards of the same rank.
- **Position:** *n*. One's location in the betting sequence, relative to the players still in the hand. First position is first to act.
- **Post-Flop:** *n*. In Texas Hold'em, the rounds following the turning of the flop cards (*i.e.* the **flop**, **turn** and **river**). Compare: **pre-flop**.
- **Pot:** *n*. The total amount of money bet so far in a hand.

- **Pot Limit:** *n.* A game where the maximum bet is determined by the size of the pot at the time. Note that a player wanting to raise first calls the bet, then totals the pot to determine the maximum amount he can raise. See also: **fixed limit**.

- **Pot Odds:** *n.* The ratio of the money in the pot to the amount of money to call. Often used with respect to the potential of your hand to determine if a pot offers enough reward to pay to see the next card.

- **PPOT:** *n. tt.* Acronym for *positive potential*: the probability of pulling ahead given we are behind (approximately the percentage of upcoming cards that make us a sure winner). $PPOT_1$ is the one card potential and $PPOT_2$ is the two card potential. In some cases where we need a very quick estimate, we use $PPOT_c$ (for *crude*). It uses some fixed rules to quickly determine an estimate of $PPOT_1$. Compare: **NPOT**.

- **Pre-Flop:** *n.* In Texas Hold'em, the round preceding the turning of the flop cards, where there are no community cards. Compare: **post-flop**.

- **Pure Bluff:** *n.* A **bluff** made with a minimal chance of winning, usually on the final round with no further cards to come.

- **Raise:** *v.* To wager more than the minimum required to call, forcing other players to put in more money as well.

- **Raiser:** *n.* One who raises.

- **Represent:** *v.* Implying, by one's betting style, that one has a particular hand.

- **Reraise:** *v.* To raise after an opponent has raised.

- **Reverse Implied Odds:** *n.* A refinement to **pot odds** which includes extra cost to play the hand. Considers the ratio of the present pot to the estimated extra cost to play to the end of the hand.

- **Re-weight:** *v. tt.* The process of taking an observed action (and its **action frequency**) and applying a transformation function to adjust the **weight array** to represent a more likely distribution of hands held by the opponent.

- **River:** *n.* The last card dealt in a hand of Hold'em.

- **Roll:** *v.* In some variants, instead of dealing extra cards the rules may call for players to **roll** over some face-down cards (turn them face-up).

- **Round:** *n.* A poker variant is composed of several rounds. In each round, players take some action (such as being dealt a new card) and then proceed to a betting series. *E.g.* Texas Hold'em has four rounds (**pre-flop**, **flop**, **turn** and **river**).

- **Royal Flush:** *n.* An ace-high straight flush, the best possible hand in regular poker.
- **Sandbag:** *v.* Playing a strong hand weakly. See also **slowplay**, **check-raise**.
- **Script:** *n. tt.* The definition for a poker variant presented as a sequence of events (*e.g.* each player receives x cards face-down, followed by a round of betting, *etc.*).
- **Semi-Bluff:** *n.* A bet with a weak hand that has good drawing potential and is likely to win if it hits (*e.g.* a flush draw with no pair is often a good semi-bluffing hand). Can be used as a verb.
- **Session:** *n.* A contiguous series of **hands** (games of poker).
- **Short-Handed:** *adv.* Playing against only a few other opponents (usually 2-4 players total). Opponent modeling is much more important.
- **Showdown:** *n.* The point at the end of the hand where all active players reveal their cards and the pot is awarded to the winner(s).
- **Showdown Odds:** *n.* A refinement to **pot odds** which considers the ratio of the estimated winnings to the estimated cost to play to the end of the hand.
- **Slowplay:** *v.* To play a strong hand weakly, by checking instead of betting or by calling instead of raising. Usually done to win extra bets by keeping more players around for future rounds of betting. See also **sandbag**.
- **Small Blind:** *n.* In games with two blinds the first blind is the **small blind** because it is usually one-half (or less) the second or **big blind**.
- **SOM:** *n. tt.* Acronym for *specific opponent modeler*: a version of *Loki* that accumulates statistics for each opponents to calculate **action frequencies** which are used to appropriately **re-weight** based on observed actions. *SOM′* is a later version that uses additional betting strategies like check-raising. Compare: **GOM**.
- **Spread Limit:** *n.* A variation on **fixed limit** wherein the minimum and maximum bets are different. A 1-4-8 game allows bets from 1 to 4 in the early rounds and 1-8 in the last round. A 1-4-8-16 game allows bets from 1 to 4 in the early rounds, 1 to 8 in the next-to-last round, and 1 to 16 in the last round.
- **Stake:** *n.* The amount of money a player is willing or able to play with in a given session. Compare: **bankroll**.
- **Steal:** *v.* To win the pot by bluffing.
- **Straight:** *n.* A hand consisting of 5 cards in sequence but not in suit.

- **Straight Draw:** *n.* A four card straight. See also: **inside straight**, **open-ended straight**.
- **Straight Flush:** *n.* A hand consisting of 5 cards in sequence and the same suit.
- **Suited:** *n.* Two or more cards all the same suit. Antonym: **off-suit**.
- **Table:** *n.* The set of players playing together.
- **Tell:** *n.* Any personal mannerisms that reveal the quality of one's hand. *E.g.* constantly looking at one's hole cards is often a tell of a poor hand. (Some players, knowing this, will at times check their hole cards when they have a great hand and don't need to look).
- **Texas Hold'em:** *n.* A Hold'em game where players receive two hole cards and may use zero or more of them, together with 5 board cards, to make their hands. See **Hold'em**.
- **Three of a Kind:** *n.* Three cards of the same rank.
- **Tight:** *adj.* Playing fewer hands than average. Antonym: **loose**.
- **To Call:** *adj.* The amount that the current player must call to continue playing; it is a factor of the previous betting action this round. *E.g.* if the current player has called the first bet of $10 but there has since been a raise of $10 then it is "$10 to call" for that player. Compare: **to go**.
- **To Go:** *adj.* The current betting level, as in "$20 to go" meaning every player must contribute $20 (total) or drop. A $10 raise would then make the pot "$30 to go."
- **Top Pair:** *n.* In flop games, having a hole card that matches the highest card on the board.
- **Trey:** *n.* A three.
- **Trial:** *n. tt.* In the self-play tournaments that *Loki* uses, a trial is one particular instance of a hand. For each particular distribution of cards, the hand is re-played ten times (trials), shuffling the seating arrangement each time to reduce the luck element.
- **Turn:** *n.* The fourth community card in Hold'em.
- **Two Pair:** *n.* A poker hand which contains two pairs of different ranks and one **kicker**.

- **Weight Array:** *n. tt.* The component of an opponent model that is used to obtain a weighted sum in the hand evaluation algorithms. For each opponent there is a weight for each possible combination of hole cards. This weight approximately represents the conditional probability that they would have played in the observed manner (given that hand). See also: **re-weight**.

- **Wild Card:** *n.* A joker or standard card that, by player agreement and/or dealer's choice, can be used to represent any card desired.

- **World Series of Poker:** *n.* A series of several different poker games with relatively large entry fees, culminating in a $10,000 entry-fee no-limit Hold'em tournament, the winner of which is crowned the World Poker Champion. Sponsored by Binion's Horseshoe Club in Las Vegas.