University of Alberta

Library Release Form

Name of Author: Maria de Lourdes Peña Castillo

Title of Thesis: Probabilities and Simulations in Poker

Degree: Master of Science

Year this Degree Granted: 1999

. . . .

Maria de Lourdes Peña Castillo
Avenida 2 - 131
Fraccionamiento San Nicolas
Cordoba, Veracruz
Mexico, 94540

Date: . . . .

“..know when to hold’em,
know when to fold’em
know when to walk away and
know when to run...”
The Gambler by Kenny Rogers

University of Alberta

Probabilities and Simulations in Poker

by

Maria de Lourdes Peña Castillo

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 1999

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 **Probabilities and Simulations in Poker** submitted by Maria de Lourdes Peña Castillo in partial fulfillment of the requirements for the degree of **Master of Science**.

. . . .
Jonathan Schaeffer

. . . .
Duane Szafron

. . . .
Paul Lu

. . . .
Steve McQuarrie

Date: . . . .

"Gracias a la vida que me ha dado tanto..." Violeta Parra
A mis padres... por todo.
(To my parents... for everything)

Abstract

Poker is an imperfect information game that requires decision-making under conditions of uncertainty, much like many real-world applications. Strong poker players have to skillfully deal with multiple opponents, risk management, opponent modeling, deception and unreliable information. These features make poker an interesting area for Artificial Intelligence research. This thesis describes work done on improving the knowledge representation, betting strategy, and opponent modeling of *Loki*, a poker-playing program at the University of Alberta. First, a randomized betting strategy that returns a *probability triple* is introduced. A probability triple is a probabilistic representation of betting decisions that indicates the likelihood of each betting action occurring in a given situation. Second, real-time simulations are used to compute the expected values of betting decisions. These simulations use *selective sampling* to maximize the information obtained with each simulation trial. Experimental results show that each of these enhancements represents a major advance in the strength of Loki.

Acknowledgements

Thanks a lot to:

- Jonathan for being an excellent supervisor and introducing me to scientific research.
- Duane for creating positive discussion in the poker meetings and your co-supervision.
- Darse for all your insight about poker, ideas, and thoughtful scanning of the log files =-).
- Denis for writing Loki-1, Loki-2 wouldn't be possible without it.
- Dima and Michael for proof-reading my thesis and your friendship.
- Lara and Oscar for being present in my life all this time.
- CONACYT (Consejo Nacional de Ciencia y Tecnologia) for providing financial support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Games are one of the oldest areas of research in the Artificial Intelligence (AI) community. In fact, in 1950, at the dawn of the computer revolution, Alan Turing and Claude Shannon pioneered the work in chess programs. None of them wrote a program which actually ran on a computer. Turing simulated his program by hand [28] and Shannon described the underlying principles of modern computer game-playing programs in one article [23]. Since then, there has been a wealth of AI research in games.

1.1 Why games?

As in some other computer applications, games raise the question of whether computers can make good decisions based on the evaluation of present and possible future situations. They also provide a suitable environment to support experimentation in different areas of computer science such as algorithms, data structures, machine learning, knowledge engineering, tree search, and reasoning.

If computers cannot solve decision-making problems in "simple" domains like games, then how can we be sure that they can make good decisions in other complex domains where rules are ill-defined, or there are high levels of uncertainty? Four characteristics make games suitable for computer representation:

1. the state of the world is easy to represent,

2. there is a fairly small number of well-defined rules and a clearly specified goal,

3. the relative success obtained by playing a game can be measured with quantifiable results, and

4. the basic infrastructure for a game-playing program is easy to build.

Games are an abstraction of worlds in which hostile agents act to diminish each other's well-being. Thus, they can be used to design and analyze situations with multiple interacting agents having competing goals. Since real life contains many situations of this kind, a method to solve a game may be applied to problems in other areas. For example, in *Theory of Games and Economic Behavior*, Von Neumann and Morgenstern state that a study of "games of strategy" is required in order to develop a theory for the foundations of economics and for the main mechanisms of social organization, because games are analogous to a variety of behaviors and situations that occur in these two areas [29]. In fact, games are already used to model certain economic problems.

In addition, the development of a program to play a strategic game often involves the application of theoretical concepts to practical situations. Programs that implement different theories can be played against each other to provide a comparison of the effectiveness of these theories in a practical domain. Therefore, games can be used as an experimental environment to obtain supporting or refuting evidence for new ideas, and to stimulate discussion on different approaches to solve a particular problem.

1.2 Why poker?

So far, the primary focus of games researchers has been placed on algorithms to solve games with perfect information. As a result, high-performance systems have been developed for games such as chess, Othello, and checkers. In many of these games, high performance can be achieved by brute-force search. Recently, attention has been given to games with imperfect information, such as bridge and poker, where searching seems not to be the key to success. Since these games offer different algorithmic and conceptual challenges, the successful development of a program capable of playing them well may provide solutions to open problems in computer science.

Poker has several features that make it attractive for AI research. These include imperfect information, multiple competing agents, risk management, opponent modeling, deception, and dealing with unreliable information. These characteristics are also present in many real-world applications that require rational behavior.

1. **Imperfect information** implies that a choice must be made from a set of actions without complete knowledge. The relative desirability of each action depends on the state of the world, but the agent does not know exactly which state prevails. In poker, a player does not know the opponents' cards. Without knowing the complete state of the world, how can the player find which actions are, "optimal", in some sense?

2. Having **multiple competing agents** exponentially increases the complexity of the computations required to play poker by enlarging the game tree.

3. **Risk management** requires making a decision to gain a profit while considering how much one can afford to lose. Making a good decision based on the evidence available and "cost-benefit" considerations is a skill required in many real-world activities. For instance, investing in the stock market has the same adrenaline-releasing characteristic. Every time a player makes a betting decision in a poker game, there is the risk of losing money. However, there is always a chance to win. In the long run, a player's objective is to end up with a positive balance.

4. **Opponent modeling** involves identifying patterns in the opponents' play and exploiting any weaknesses in their strategy. For example, opponent modeling is extensively applied in political campaigns. In poker, it can be done by observing the opponents' betting habits, and determining likely probability distributions for their cards. If a player can predict the opponents' actions, then this player will be capable of making much better decisions.

5. **Deception** and **the ability to deal with unreliable information** are traits of a strong poker player. In fact, these activities are also necessary in real-world situations. For example, assume one wants to acquire a used car. How much

shall one believe from all the wonders the salesman says about the car? How can one get a reduction on the price of the car? Good poker players have to be unpredictable by bluffing and varying their playing style, and must also be able to deal with their opponents' deceptive plays. For example, if a player is known to raise only with a strong hand (a predictable player), the opponents are likely to fold in such cases. Therefore, this player is missing opportunities to earn more money on the best hands. By occasionally raising on a weak hand, this player will either profit from a successful bluff, or will implant doubt that will result in greater profits for strong hands. Hence, it is necessary to mislead the opponents by letting them know that an occasional raise or high bet is possible with a weak hand.

1.3 Thesis contributions

Loki is a poker-playing program developed at the University of Alberta starting in 1997. The name refers to the Norse god of mischief and discord. At the beginning of the work presented in this thesis, Loki (henceforth referred to as Loki-1) was already a intermediate level poker player (see [4], [5], [19]). It had the infrastructure to incorporate advanced features for the next step towards the goal of creating a high-performance poker program capable of defeating the best human players. However, its rigid, deterministic, hand-tuned betting strategy was becoming a limiting factor for its future development. This thesis presents the work done to improve the knowledge representation and the betting strategy of Loki-1.

The first improvement is a probabilistic representation of poker betting decisions. A betting strategy attempts to determine which betting action is most profitable in a given situation, based on the evaluation function of the program. Loki-1's evaluation function was a deterministic strategy since it always returned a single value: the "best" betting action. A deterministic strategy is vulnerable to being predictable, which gives a skilled opponent the opportunity to find a counter-strategy that takes advantage of this fact. The new version of Loki (henceforth referred to as Loki-2) returns three probabilities, one for each betting action (fold, call/check, and raise/bet). Loki-2 can then randomly select the betting decision based on this probability distribution. This new evaluation function is a mixed (randomized) strategy that adds unpredictability to Loki-2's play without sacrificing much in immediate expectation. This routine also merges all the expert knowledge components used in Loki-1, since the probability triple representation can be used throughout the program.

The second improvement is the use of simulation (search) to compute the expected value of betting alternatives. The Loki-2's betting strategy uses a simulation-based approach: selective sampling simulation. This approach consists of simulating the outcome of a hand many times. In every simulation trial, a likely instance of the hidden information (opponents' hands) is generated, and the hand is played out once for each betting alternative as the first action of Loki-2 in the trial. The results of all the trials are averaged and the betting action with the highest expectation is returned. To select the most likely (selective sampling) opponents' hands and actions from the sample space during a simulation, Loki-2 uses all the information available about the game and the opponents. The simulation refines the quality of the evaluation function and the selective sampling increases the information gained with each trial.

Experimental results will be presented to demonstrate that both enhancements represent a notable improvement in Loki's playing ability. In all the self-simulation experiments performed, Loki-2 outperformed Loki-1. Loki-2 has also consistently increased its bankroll playing against human opponents on an Internet poker server; at a rate that appears to be significantly higher than Loki-1's.

This thesis is organized as follows. Chapter 2 introduces poker terminology, describes the game of Texas Hold'em (the poker variation played by Loki) and discusses other work done in computer poker. Chapter 3 describes Loki-1 in detail. Chapter 4 discusses the probabilistic representation of betting actions (probability triples) used to improve Loki-2's betting strategy and opponent modeling. It also discusses the new design of the program. Chapter 5 discusses the selective sampling simulations in Loki-2. Chapter 6 is an overview of related work in selective sampling simulations. Conclusions and future work are presented in Chapter 7.

Chapter 2

Poker

Poker is a *multi-player non-deterministic zero-sum game with imperfect information*. In game theory, a game is considered *strictly competitive* if the players do not co-operate. A strictly competitive game is a *zero-sum game* if the sum of the utility (outcome) obtained for each of the players is zero, independent of the strategy followed by each player. In poker, the profit of one player is the loss of other players. The long-term goal of all the players is to leave the table with more money than they had at the beginning.

A poker session is played in a sequential series of games[1] with a standard deck of 52 cards. Each card is identified by its suit and rank. There are four suits: ♣ Clubs, ♢ Diamonds, ♡ Hearts and ♠ Spades. The thirteen card ranks are (in increasing order of importance): Deuce (2), Three (3), Four (4), Five (5), Six (6), Seven (7), Eight (8), Nine (9), Ten (T), Jack (J), Queen (Q), King (K) and Ace (A). In this thesis cards are represented by two characters, one for the rank and one for the suit, *e.g.* A♢ (Ace of Diamonds) and Q♡ (Queen of Hearts). A set of cards is represented by the cards separated by dashes, *e.g.* K♠-Q♠. A set of cards held by a player can also be called a *hand*.

2.1 A poker game

A poker game is composed of several rounds. A round consists of a number of cards being randomly dealt followed by betting. Every active player is given the chance to act at least once in a round. Every time it is a player's turn to act, there are three

[1] Also called *deals* or *hands*.

alternative actions:

- *fold* – become inactive for the game, losing all investment done in the current game,
- *call* – match the current per player contribution, or
- *raise* – increase the amount to call.

If there has not been any previous bet in the round, a call is referred to as a *check*, and a raise is said to be a *bet*. A round ends when each player has either folded or contributed the same amount of money as all the other active players. A poker game has two termination conditions:

1. all the players have folded except one, who wins all the money wagered (*the pot*), or
2. all the betting rounds have been completed.

In the latter case, the game proceeds to a *showdown* where all the active players reveal their cards and the winner is determined. The winner is the player holding the highest poker hand (see Table 2.1 for hand ranking). In the case of a tie, the pot is split evenly.

The word *poker* refers to a collection of card games that share common features such as betting rounds and ranking of hands. The card games classified as poker are divided into flop games (some cards of each player are shared), stud games, and draw games (no cards are exposed, some are discarded and replaced with cards from the deck). Loki plays the limit version of *Texas Hold'em.* Texas Hold'em is a flop game which is played as the main event of the annual World Series of Poker to determine the world champion. In the limit version of this game there is a fixed bet size for each round. Texas Hold'em was chosen as the variant of poker played by Loki, because it is the most strategically complex poker form that is widely played, and has the smallest ratio of luck to skill [2].

Sample hand	Name and description
T♣-9♣-8♣-7♣-6♣	Straight Flush (includes Royal Flush) 5 cards of the same suit in sequence
Q♠-Q♣-Q♡-Q♢-2♠	Four of a Kind 4 cards of the same rank
J♠-J♣-J♡-5♢-5♠	Full House 3 cards of identical rank and 2 cards of another rank
A♠-9♠-7♠-6♠-2♠	Flush 5 cards of the same suit
A♢-K♣-Q♡-J♢-T♠	Straight 5 cards of different suit in sequence
A♠-A♣-A♡-8♢-4♠	Three of a Kind 3 cards of the same rank
K♠-K♣-T♡-T♢-9♡	Two Pair 2 cards of one rank and 2 cards of another rank
Q♡-Q♣-T♠-8♢-3♣	Pair 2 cards of the same rank
T♠-8♣-5♡-3♢-2♢	High Card 5 cards of different suit and rank

Table 2.1: 5-card hands ranked from strongest to weakest

2.2 Texas Hold'em

A game of Texas Hold'em has four betting rounds called the *preflop*, *flop*, *turn* and *river*. In the preflop every player receives two cards face-down (known only to the player) called *hole* or *pocket* cards, and a betting round ensues. During the flop three *community* or *board* cards are dealt face-up (known by all the players) and the second betting round follows. At the turn a fourth face-up community card is dealt, followed by a betting round. Finally, at the river, a fifth face-up community card is dealt and the last betting round occurs, followed by a showdown. In the showdown, the pot is awarded to the best five-card hand that an active player can make combining the hole cards and the five community cards. There is normally a maximum of three raises allowed per betting round. In Texas Hold'em, the number of players can vary from 2 to 23; but it is usually played with 8 to 10 players. Figure 2.1 shows a hand of Texas Hold'em (on the turn) from Loki's point of view against two opponents. The cards denoted by a single question mark represent the imperfect information of the game. The card denoted by two question marks represents the non-deterministic outcome

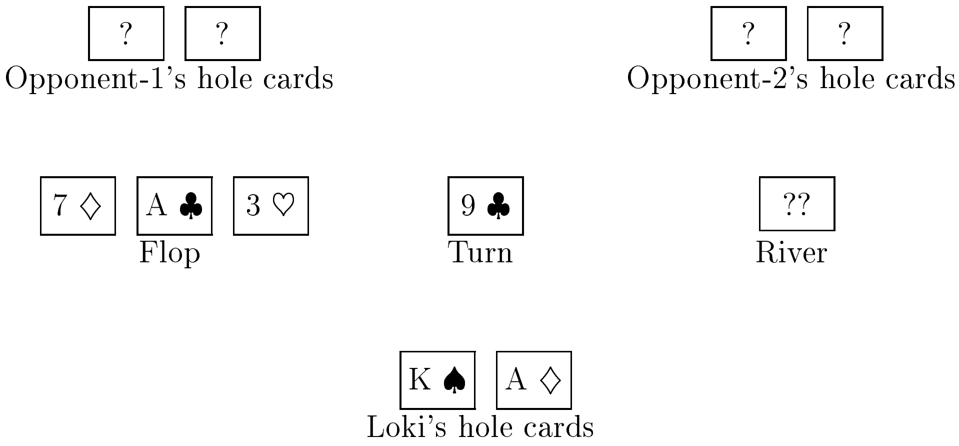

Figure 2.1: A Texas Hold'em hand

in the game (the card to come).

The players are in a fixed seating order at the table. The *dealer-button* rotates clockwise around the table to indicate the (theoretical) dealer of each hand. The player to the immediate left of the button (the *small blind*) is first to receive a card. Figure 2.2 shows a table with the button's and blinds' positions indicated. Betting on the preflop starts with the player on the left of the *big blind*. On subsequent rounds, the first active player left of the button acts first (the small blind if not folded).

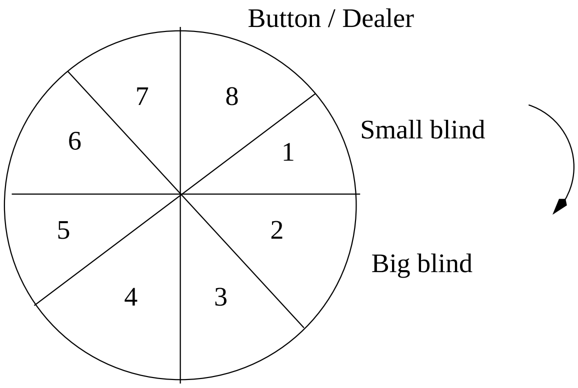

Figure 2.2: Seat position and betting order

The betting structure of 2-4 Limit Texas Hold'em starts with two forced bets (*blinds*) on the preflop: a *small blind* of one unit and a *big blind* of two units. Blinds are a "forced-bet" alternative to the more familiar *ante* in other games, where each player is required to put a fixed amount into the pot before the game begins. In a 2-4 Texas Hold'em game, all bets and raises are a fixed size of two units during the preflop and flop. This doubles to four units for all bets and raises on the turn and river. Other amounts can also be used. For example, in a \$10-\$20 game, the unit size is \$5.00. The small blind posts a \$5 blind, the big one blinds \$10. The size of the bet during the first two rounds (preflop and flop) is \$10.00 and during the last two rounds (turn and river) is \$20.00.

2.3 Other work in computer poker

Since the founding of game theory by John Von Neumann, poker has been the subject of mathematical and economics analyses. However, these studies have used oversimplified versions of poker, making most of the work done in these areas not applicable to the development of strong poker-playing programs. In computer science, poker has been used as a testbed in different areas such as cognitive science, machine learning, search and Bayesian networks.

2.3.1 Findler's work

One of the first studies of computer poker was done by Nicolas Findler [9] [10]. The variation of poker used in his research was a simplified version of five-card draw poker. During the years Findler's project was carried out, various poker-playing programs were created, each different in its structure and approach to decision-making.

Most of the computer players developed in Findler's research were based on simulating human cognitive processes involved in decision-making under uncertainty and risk. His approaches were based on psychological precepts of human thought rather than mathematically-oriented analysis. He considered that:

> "In order to program a computer to play poker well it is necessary to understand the cognitive processes employed when human beings play poker. (The mathematical theory of games can only treat simplified versions of the game)." [10]

Findler's goal was not to create a world-class poker-playing program and, indeed, none of his programs appears to have been a strong player.

2.3.2 Machine learning

Waterman [30] and Smith [25] used poker as a testbed for automatic learning techniques. Both of them worked on the problem of acquiring problem-solving heuristics through experience.

Waterman worked in two areas: 1) the representation of heuristics as production rules to facilitate their dynamic manipulation, and 2) the automatic modification and creation of these heuristics by a learning program on the basis of information obtained during training. During his research, five computer players were created, differing in the number and source of the heuristics initially provided to the program. The performance of his best program was evaluated to be the same degree of skill as a "nonprofesional but experienced human player". His programs played a two-player standard version of five-card draw poker. He stated that by choosing poker, the representation and generalization techniques he developed were shown to be an effective approach to implementing decision making and learning in an imperfect information environment.

Smith proposed an alternative method for dynamically learning heuristics by using adaptive search (genetic algorithms). Poker was used as a testbed for this technique to provide a basis for comparison with Waterman's work.

2.3.3 Koller and Pfeffer's work

Recently, Koller and Pfeffer [15] [16] have developed *Gala*, a system for automating game-theoretic analysis for two-player competitive games with imperfect information. The system takes a description of a game, analyzes it, and outputs strategies for the different players which are game-theoretically optimal for the situation described. The implementation is composed of two main interacting pieces: a special-purpose game specification language, and an automatic game-theoretic analyzer for games in *extensive form*. The extensive form represents the games as a tree with the *information sets* (players' knowledge states) indicated.

For games with imperfect information, the system finds an optimal randomized strategy. The system can now solve simplified versions of two-player poker (*e.g.* 3-card deck, 1 card dealt to each player, 3 rounds; or 11-card deck, 5 cards each player, 3 rounds). However, the authors state that:

> "While we can now solve games with tens of thousands of nodes, 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. A game tree for five-card draw poker, for example, where players are allowed to exchange cards, has over 10^{25} different nodes. The situation (for zero-sum games) is now quite similar to that of perfect-information games: We have algorithms that are fairly efficient *in the size of the game tree*; unfortunately, the game tree is often extremely large." [16]

Optimal versus maximal player

Koller and Pfeffer's goal is to create an optimal poker player. By definition, the optimal player does the best that can be done against a rational (perfect) opponent, and it does not do worse even if its strategy is revealed to its opponent. However, the optimal strategy does not take advantages of mistakes when they become apparent, and human players invariably make mistakes. A *maximal* player will deviate from the optimal strategy to exploit the observed weak points in the opponent's play. In theory, the maximal player must take the risk of being sub-optimal to exploit a sub-optimal opponent. In practice, the risk is small and well rewarded.

In contrast to Koller and Pfeffer's aim, Loki is not an optimal player. Our goal is to create a maximal player, which uses opponent modeling to exploit patterns in its opponents' play, with the intention of winning the most money it can in every situation. Furthermore, since it does not seem feasible to determine an optimal player for real multi-player poker, a program to play real-world poker in the near future most likely will not be a game-theoretic optimal player.

Nevertheless, Koller and Pfeffer have suggested that an alternative approach to deal with less-than-perfect players is to learn the type of mistake that a player is prone to make. This approach can be used when there is a long-term interaction with the same player. The authors point out that the ability of the Gala language to capture regularities in the game may be particularly useful in this context, since the high-level description of a game state can provide features for the learning algorithm.

One can see this learning algorithm as a potential opponent modeling component for a program based on the Gala system.

2.3.4 Bayesian poker

Kevin B. Korb, Ann E. Nicholson and Nathalie Jitnah [17] at Monash University are working in the Bayesian Poker Program (BPP). BPP plays two-player five-card stud poker using a Bayesian network structure to represent the relationships between current hand type, final hand type (after the five cards have been dealt) and the behaviour of the opponent. Given evidence for BPP's current hand type and the observed cards and actions of the opponent, BPP obtains its posterior probability of winning the game. BPP uses this estimated probability of winning the game to randomly select its action based on probabilistic curves for each betting action.

BPP performs opponent modeling. It uses the relative frequencies of the opponent's betting actions to update the conditional probabilities per round of passing or calling versus betting or raising given the opponent's current hand type.

BPP is work in progress as pointed out by Korb *et al.* The authors state that poker appears to be an ideal domain for investigating the application of Bayesian networks, and report positive results of BPP playing against a simple probabilistic program, a rule-based program and non-expert amateur human players.

2.4 Summary

Although poker has been used as a testbed in different areas of computer science, mathematics and economics, full-scale poker has been largely overlooked as a topic of AI research. However, computer poker research, besides being interesting and challenging, has the potential to provide results with real-world implications.

Chapter 3

Loki-1

This chapter describes the architecture, betting strategy, and opponent modeling of the previous version of Loki (Loki-1), as it was at the beginning of the work done in this thesis (1998). Also, the limitations detected in Loki-1 and the changes implemented to overcome these limitations are outlined. This chapter provides a summary of the material necessary to place the research described in this thesis in context. See [4], [5] and [19] for more details on Loki-1.

3.1 Architecture

Figure 3.1 shows Loki-1's architecture and the interactions between the main system components. In the diagram, rectangles are major components, rounded rectangles are major data structures, and ovals are actions. The data follows the arrows between components. An annotated arrow indicates how many times data moves between the components for each of Loki-1's betting decisions (on the flop, in this case).

To make a betting decision, the Bettor calls the Hand Evaluator to obtain an assessment of the strength of Loki-1's hole cards. The Bettor uses the hand evaluation, the public information about the state of the game, and expert-defined betting rules to generate an action (fold, check/call or bet/raise). The probability distribution of the opponents' hands after the flop is not uniform. For example, hole cards of Ace-Ace are more likely held by the opponents than hole cards of 7-2, since most players will fold 7-2 in the preflop. The Opponent Modeler maintains an array for each opponent with the probabilities (weights) of each possible hand being held by each opponent. The Hand Evaluator uses these weights to estimate the strength of

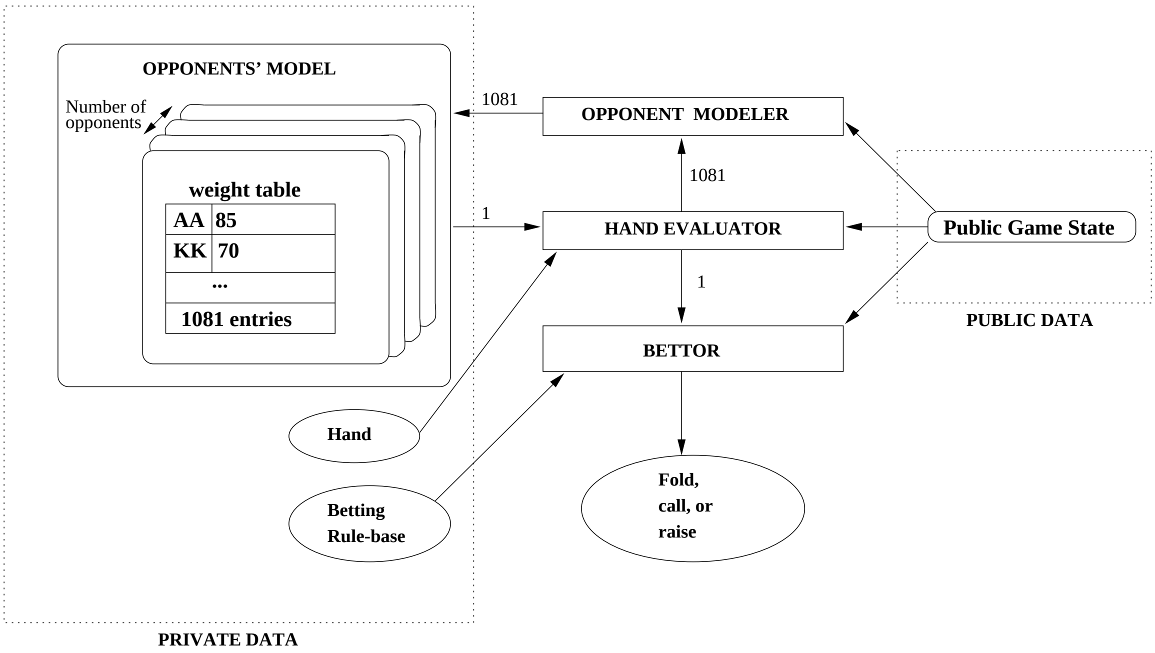

Figure 3.1: Loki-1's architecture

a hand. Thus, the assessment of the strength of a hand is sensitive to the actions of the opponents. Loki-1's hand evaluation decreases if the opponents have shown signs of strength (by raising) during the game, and increases if all the opponents have only checked or called.

The Opponent Modeler modifies an opponent's weight table after it observes an action of this opponent taking into account the entire game context (community cards). Updating the probabilities for all hands is a process called reweighting. After each opponent action, the Opponent Modeler calls the Hand Evaluator once for each possible hand and modifies the weight for that case to be consistent with the new information.

3.2 Betting strategy

3.2.1 Preflop expert system

When it is Loki's (either Loki-1 or Loki-2) first chance to act in the preflop, Loki uses a rule-based expert system to select one of four defined preflop strategies (or six if Loki is the small blind). These strategies determine the number of bets Loki will call and under which conditions it will bet/raise. The selection of the preflop betting strategy is based on the average return on investment (income rate) of Loki's hole cards, and thresholds defined by linear formulas using expert values. The income rate of all the two-card hands was determined with off-line simulations. The linear formulas take into account the expected number of players (players who will play the hand), Loki's position on the table, and the *tightness* of Loki. The tightness is a parameter that specifies the percentage of hands that Loki will play. There are three settings for this parameter: *tight*, *moderate* and *loose*. The most aggressive strategy whose threshold is less than or equal to the income rate of Loki's hand is selected.

For example, assume that Loki's is in the dealer's position (last player to act) and its hole cards are A♡-8♡. There are four players still active in the game and Loki's tightness has been defined as moderate. The income rate value of Loki's hand obtained by a table lookup is 338. By using the linear formulas, we calculate that the thresholds for the four strategies from the most aggressive to the most passive one are $[M4 = 580, M2 = 200, M1 = 50, M0 = -\infty]$. Thus, Loki's preflop strategy in this case is $M2$, since $200 \leq 338 < 580$. With the $M2$ preflop strategy, Loki will raise if there have been less than two bet/raises in the round, otherwise it will call.

3.2.2 Postflop

Loki-1's postflop betting strategy consists of expert-defined rules that uses the hand evaluation and the public information about the state of the game to decide on a betting action.

Hand Evaluation

To assess the quality of a hand after the flop, the Hand Evaluator combines together the strength and the potential of the hand in a value called *effective hand strength* (EHS). The EHS is an estimate which gives the probability that the given hand is currently the strongest one, or that it will become the strongest one by the showdown with the next community cards (potential).

$$EHS = hand\_strength + (1 - hand\_strength) \times hand\_potential$$

To calculate a hand's strength (HS) against a single opponent, the Hand Evaluator enumerates all the possible opponent hands and sums the weights of the hands that would win, lose or tie the given hand. Recall that the weight of a hand is the probability that an opponent would still be active with that particular hand.

$$HS = \frac{ahead + tied/2}{ahead + tied + behind = total\_number\_of\_hands}$$

For instance, assume that Loki's hole cards are A♡-8♡ (the same as in the above preflop example), the community cards on the flop are 9♡-8♣-2♣, and all weights are equal to 1 (uniform distribution). From the $\binom{47}{2}$= 1081 possible opponent's two-card hands on the flop: 903 hands lose against Loki's hand, six hands tie and 172 hands defeat Loki's[1]. Therefore, Loki's HS is

$$HS = \frac{903 + 6/2}{903 + 6 + 172 = 1081} = 0.84$$

To extrapolate the hand strength value to multiple opponents, the Hand Evaluator raises it to the power of the number of opponents still active in the game (HS_n). For the above example, if there are four players active in the game (including Loki), the Hand Evaluator calculates $\mathrm{HS}_n = (0.84)^3 = 0.59$.

The potential of a hand can be either positive or negative. Positive potential (PPOT) is the probability of a hand becoming the strongest one when it is behind. Negative potential (NPOT) is the probability of a hand falling behind when it is ahead. Both potentials are calculated by enumerating all possible opponents' hands and community cards to come in the next rounds. Potential calculations on the flop can be done by looking ahead either one round (considering the 45 possible cards on the turn) or two rounds (considering the $\binom{45}{2}$= 990 possible two-card combinations on the river). PPOT is calculated by adding the weights of the cases where Loki's hand improves.

$$PPOT = \frac{be\_behind\_end\_ahead + be\_behind\_end\_tied/2 + be\_tied\_end\_ahead/2}{total\_be\_behind + total\_be\_tied/2}$$

NPOT is given by:

$$NPOT = \frac{be\_ahead\_end\_behind + be\_ahead\_end\_tied/2 + be\_tied\_end\_behind/2}{total\_be\_ahead + total\_be\_tied/2}$$

[1]144 hands with one or two 9s and 28 hands with a pair of either A, K, Q, J, T, 8 or 2.

Flop's situation	River's situation	Number of cases
Ahead	Ahead	722,463
	Behind	170,249
	Tied	1,258
Behind	Ahead	37,659
	Behind	132,570
	Tied	51
Tied	Ahead	270
	Behind	90
	Tied	5,580

Table 3.1: Number of cases where Loki's hand situation changes after two community cards are dealt

Consider the same above example. Table 3.1 shows the number of cases where Loki's hand situation at the flop changes (or remains the same) by the time the other two community cards are dealt. Assuming uniform weights, this table shows the sum of the weights of all the cases. $Total\_be\_ahead$ is equal to the number of cases where Loki's hand on the flop is the strongest one multiplied by the number of possible next two-card combinations ($903 * 990 = 893,970$). $Total\_be\_tied$ is 5,940 ($6 * 990$) and $Total\_be\_behind$ is 170,280 ($172 * 990$). The total number of cases enumerated in the potential calculations is $1081 * 990 = 1,070,190$. Thus, PPOT for Loki's hand (A♡-8♡) is

$$PPOT = \frac{37,659 + 51/2 + 270/2}{170,280 + 5,940/2} = 0.22$$

and NPOT is :

$$NPOT = \frac{170,249 + 1,258/2 + 90/2}{893,970 + 5,940/2} = 0.19$$

Strategy

The basic postflop strategy is based on the EHS. Two thresholds determine the postflop betting actions: a postflop-raise threshold and a postflop-call threshold. If Loki-1's EHS is greater than or equal to the postflop-raise threshold then it will raise when less than two bets have been made this round and call otherwise. When its EHS is greater than or equal to the postflop-call threshold (but not greater than the postflop-raise threshold), Loki-1 will bet if nobody else has done so and call otherwise, except when it is two or more bets to call and Loki-1 has not already called a bet this round. In the cases where Loki-1's EHS is less than the postflop-call threshold or its decision is to fold, the options of semi-bluffing, calling with pot odds, and calling with showdown odds are also considered.

- Semi-bluffing consists of betting if nobody has done so in the current round, and Loki-1's hand has a high enough PPOT to call both a bet and a raise. In the subsequent rounds, Loki-1 will continue to bet (even without sufficient PPOT) if no other player bets. With semi-bluffing Loki-1 pretends to have a strong hand while there is a reasonable chance of winning the pot immediately (to scare the opponents out from the game).

- Pot odds is the ratio of the amount of money in the pot to the amount required to call the current bet. By using pot odds, Loki-1 will stay in the hand if its winning chances (PPOT before the river and HS on the river) surpass the expected return from the pot. For example, assume the pot is \$20 and the amount to call is \$4. The pot odds are $\frac{4}{24} = 0.16$ and Loki-1's winning chances are 0.25 (25%). In this situation, three times out of four that Loki-1 calls, its hand will lose at a cost of \$4 each. However, it wins \$20 one time out of four resulting in an average profit of \$2 per hand. A call is better than a fold when Loki-1's winning chances are greater than or equal to the pot odds.

- Showdown odds is the ratio of the amount of money expected to be in the pot by the showdown, to the amount it will cost Loki-1 to stay in the hand to actually see the showdown. Loki-1 calls when its EHS is greater than showdown odds. This strategy was introduced to discourage frequent bluffing by the opponent.

Also, Loki-1's betting strategy contains the knowledge of some advanced strategies such as check-raising. This knowledge was introduced in Loki-1 as deceptive strategies to add unpredictability to its play.

3.3 Opponent modeling

Although opponent modeling has been studied in perfect information games (for example [7]), the performance loss by ignoring it and assuming a perfect opponent is small, and hence it is usually ignored. In contrast, opponent modeling in poker can be the 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.

Deciding how to gather information about the opponents and how to use it to improve the quality of betting decisions is a complex and interesting problem. Loki-1's Opponent Modeler was a first attempt at making appropriate inferences from observing the opponents' actions and then applying them by changing betting decisions to exploit any identified pattern or weakness in the opponents' play. The Opponent Modeler uses the betting history of the opponents to determine a likely probability distribution for their hole cards which is used by the Hand Evaluator. Opponent modeling was experimentally shown to significantly improve Loki-1's performance [4].

3.3.1 Representation

The Opponent Modeler assigns an array of weights (weight table) to each opponent indexed by the two-card starting hands. Since Loki knows its two hole cards and the three flop cards, there are $\binom{47}{2} = 1081$ used entries in the weight table after the flop. The probabilities for each of the 1,081 subcases are called weights, since they act as multipliers in the enumeration computations. Each time an opponent makes a betting action, the weights for that opponent are modified to account for the action and the community cards revealed. A weight for a hand reflects the relative probability that a specific opponent has that particular hand.

3.3.2 Reweighting process

When an opponent action is observed, the Opponent Modeler obtains the threshold hand value needed for the observed action and bases the weight adjustment on that value. The Opponent Modeler maintains statistics for each opponent between games. These statistics are used to calculate the frequency of folding, calling and raising of each opponent per round and number of bets to call. From these frequencies, the Opponent Modeler deduces the average (μ) and variance (σ) of the threshold needed for the observed action. The threshold can be obtained either from default action frequencies (generic opponent modeling) or from the opponent's observed action frequencies (specific opponent modeling).

During the reweighting process, the reweight factors (rwt) are assigned based on the distance between the hand value (income rate for the preflop and EHS for the postflop rounds) and μ. Since the income rates used in the preflop are not a percentile hand valuing system like EHS, the μ obtained needs to be converted from a percentile value to a value on the income rate scale. To achieve this, μ is used to index into a sorted array (sample the nearest point) of the $\binom{52}{2} = 1326$ (all two-card hands) income rate values.

$$\mathsf{rwt} = \begin{cases} 0.01 & \text{if hand\_value} < \mu - \sigma, \\ 0.5 & \text{if hand\_value} = \mu, \\ 1 & \text{if hand\_value} > \mu + \sigma, \\ \frac{hand\_value - \mu + \sigma}{2 \times \sigma} & \text{otherwise.} \end{cases}$$

For example, based on observed frequencies, the Opponent Modeler deduces that an opponent needs a median EHS (μ) of 0.6 to call a bet on the flop, with a lower bound of 0.4 and an upper bound of 0.8 ($\sigma = 0.2$). In this case, all hands with an EHS greater than 0.8 are given reweighting factors of 1.0. Any hand with a value less than 0.4 is assigned a reweighting factor of 0.01, and a linear interpolation is performed for values between 0.4 and 0.8.

To avoid eliminating legal subcases completely, no weight is allowed to go below 0.01. In Loki-1, the Opponent Modeler only performs one reweighting per model per round. A copy of the weight table is stored at the beginning of each round and used in the reweighting process each time a new action is witnessed that requires a higher threshold. For example, assume an opponent calls a bet, and the reweight process uses $\mu = 0.5$ to adjust the weight table. If, later in the betting round, that opponent raises, the reweighting will be done with the higher value of μ over the stored copy of the weight table.

3.4 Modifications to Loki-1

Loki-1's design has several limitations. First, expert knowledge appears in various places in the program (Bettor, Opponent Modeler), making Loki-1 difficult to maintain and improve. Second, the Bettor is deterministic (it always returns the same single action: fold, call, or raise, given identical input). This makes Loki-1's betting actions predictable. Finally, the Opponent Modeler does not distinguish between the different actions that an opponent might take (a call and a raise are treated the same) and does not perform "negative reweighting". The lack of negative reweighting gives a pessimistic vision to the Hand Evaluator, since the weights of the top hands are never decreased when a less aggressive opponent's action is observed. These issues led to a redesign of how knowledge is used in Loki-2.

The new version of Loki, called Loki-2, makes two fundamental changes to Loki-1's architecture. First, it introduces a data object called a *probability triple* that is used throughout the program. Chapter 4 explains probability triples in detail. For now consider a probability triple as three values defining the probability distribution of the betting actions (fold, call, raise) in a given context. In fact, Loki-2's architecture revolves around generating and using probability triples. Second, simulation with selective sampling is used to refine the betting strategy (see Chapter 5).

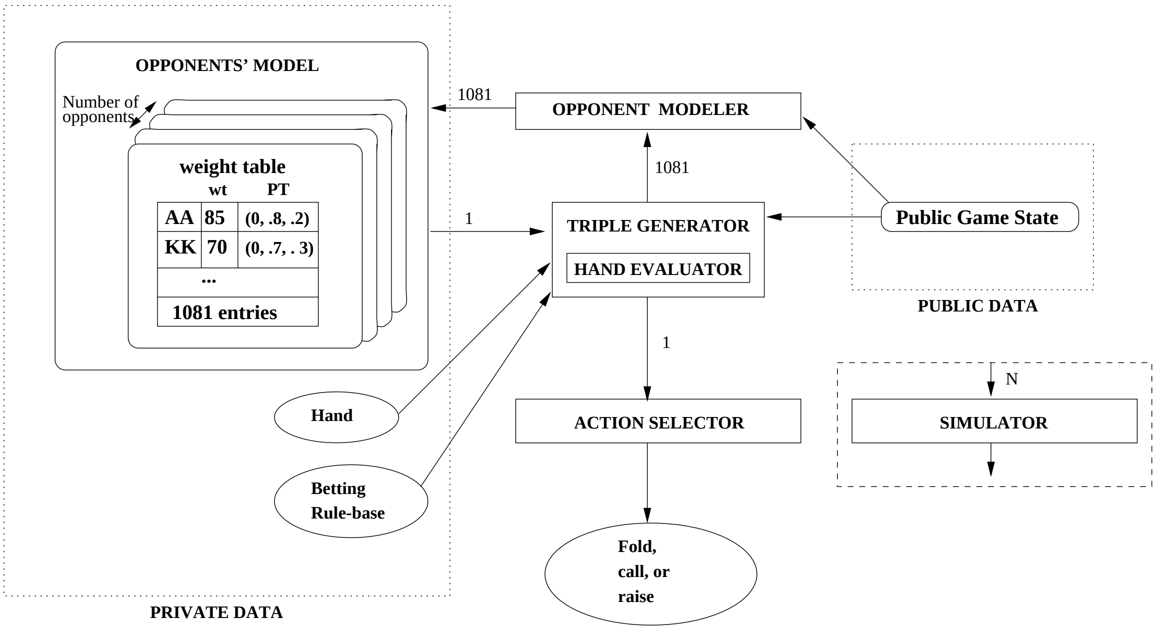

Figure 3.2: Loki-2's architecture

Figure 3.2 shows Loki-2's architecture. The Triple Generator contains the poker knowledge, and is analogous to an evaluation function in two-player games. The Triple Generator uses the hand value provided by the Hand Evaluator, the current game state, and expert-defined betting rules to compute a probability triple. Probability triples are used in three places in Loki-2. The Action Selector uses a single probability triple to decide what action to take (fold, call, raise). The Simulator uses probability triples to choose actions for simulated opponent hands. The Opponent Modeler uses an array of probability triples (PT) to update the weight table of each opponent. Loki-2 can be used with or without selective sampling simulation, as shown in the diagram. With simulation, the Simulator component replaces the simpler Action Selector.

3.5 Summary

Loki-1 was a intermediate level poker player as shown by the experimental results in [19]. Also, Loki-1 demonstrated the benefits of using opponent modeling. However, weaknesses in its betting strategy were hampering the overall performance of the program and the Opponent Modeler required improvement. These issues led to a redesign of Loki's architecture to facilitate the addition of new components to the program: probability triples and selective sampling simulations.

Chapter 4

Probability triples

To make a betting decision Loki-1 uses an evaluation function to determine which of the three actions (fold, check/call, or bet/raise) is more likely to be profitable. In fact, the "best" action can be considered as the action with the highest expected value over all the possible scenarios. However, Loki-1 provides too much information to the opponents if it always takes the best action. Loki-1 is predictable and the opponents exploit that predictability. We use *probability triples*, a set of three probabilities representing the three types of actions, to provide a randomized betting strategy for Loki-2 and to represent the probabilistic nature of poker. This representation isolates the expert knowledge in a single function (probability triple generation routine) and allows a computer oriented (i.e. easy to maintain and modify) design of the evaluation function.

A *probability triple* (PT) is an ordered list of three values, $PT = (f, c, r)$ such that $f + c + r = 1.0$. Each value represents the likelihood that the next action in a given state is a fold (f), a call (c), or a raise (r), respectively. Probability triples are used in three places in Loki-2: 1) as a stand-alone betting strategy, 2) as the reweighting factor in the opponent modeling module, and 3) as the action generation mechanism during the simulations.

4.1 Probability triple generation function

The PT generation function describes how a player should behave with a particular pair of cards in a specific situation. The function returns the probability distribution that, given a specific two-card hand and the public information about the state of the game, the action should be either fold, call, or raise.

$$f = P(\text{action} = \text{fold} \mid \text{pair of cards and game context})$$

$$c = P(\text{action} = \text{call} \mid \text{pair of cards and game context})$$

$$r = P(\text{action} = \text{raise} \mid \text{pair of cards and game context})$$

The function uses the hand evaluation in an expert-defined rule-based betting strategy to compute the three values. The hand evaluation comprises the strength and the potential of the hand; the strength represents the probability of the hand presently being the strongest one and the potential represents the probability of the hand becoming the strongest after future cards have been dealt (see [19]).

The first version of the PT generation function was a completely new betting strategy that was simpler than Loki-1's betting strategy. Although this function sufficed to show experimentally the advantages of having a non-deterministic betting strategy, it was outperformed by the old one. The main advantage of having a non-deterministic betting strategy is that we allow Loki-2 to randomly choose its action based on a set of probabilities rather than follow the single action returned by Loki-1's betting strategy.

The second attempt to create the PT function was to translate the strong, but rigid, betting strategy of Loki-1 into the PT scheme. A literal translation of the previous betting strategy into the PT function produced *pure* or deterministic probability triples. A pure PT has the value of the most likely action equal to one and the other two actions equal to zero. Once the PT function mimicked Loki-1's betting strategy, the boundaries between actions were smoothed by applying linear interpolation to create unpredictability.

With Loki-1's betting strategy in PT form, small modifications to the PT function, such as the one described in the previous paragraph, are less time consuming and the consequences of each change can be evaluated independently. By compartmentalizing the expert knowledge in a single routine, the design was improved by standard software engineering concepts. The benefits of the PT generation function are:

- it hides the poker specific details of the evaluation function from the rest of the system,
- it provides a well-defined interface,
- it confines the impact of changes in Loki-2's knowledge to a single function, and
- it facilitates the verification of changes.

To generate the PT for a hand, the hand value is computed first. The hand value is an estimate of the probability of winning. This value is then used by a set of rules to compute the probabilities of folding, calling and raising. Consider that $S()$ gives the public information about a game, h is a hand, $EHS(h, S())$ gives the hand value, and $\mathcal{PT}_{S,EHS}(f, c, r)$ represents a PT generation function. An abstract view of a simplistic PT generation function is:

$$\mathcal{PT}_{S,EHS} = \begin{cases} (0, .25, .75) & \text{if } EHS(h, S()) > .75, \\ (.20, .80, 0) & \text{if } .75 \geq EHS(h, S()) > .50, \\ (.50, .40, 0) & \text{otherwise} \end{cases}$$

Note that one can add as many rules as needed to the PT generation function. Since all the knowledge is located in one single function, the addition of extra rules is a minor change in the program. In Figure 4.1 the algorithm for a simplified four-rule PT generation function is shown. The threshold values that define the likelihood of each action (*param_postflopRaise* and *param_minToRaise*) and the probabilities of each action for every case are defined by a poker expert and can be modified to vary Loki-2's playing style. Loki-2's PT generation function uses nine rules to produce the PTs used to choose an action in a game and eight rules to generate the PTs used in the opponent modeling module as a reweighting factor. When the PT generation function is called to determine a PT to select an action in the game, it considers rules containing more expert knowledge such as calling based on pot odds (the ratio of the amount of money in the pot to the amount of money it will cost us to call) and based on showdown odds (the ratio of the amount of money it will cost us to stay in the game to see the showdown to the amount of money we will make if we win in the showdown). During the reweighting process a simpler and faster PT generation algorithm is used. In addition, this PT generation function does not generate zero probabilities for an action, because we do not want to rule out any opponent's hand.

```
#define MAXPROBABILITY 0.99
#define MINPROBABILITY 0.01

generate_probabilityTriple(PT[], bets_to_call, two_card_hand) {
   PT[fold] = PT[call] = PT[raise] = 0.0;
   /* The evaluation of the hand includes the hand strength and
      the hand potential */
   hand_evaluation = evaluate(two_card_hand,game_state);
   /* 4 expert-defined rules to calculate a probability triple */
   if (hand_evaluation > param_postflopRaise[bets_to_call]) {
        PT[raise] = MAXPROBABILITY;
        PT[call]  = MINPROBABILITY;
   } else if (hand_evaluation > param_minToRaise[bets_to_call]) {
        /* Linear interpolation between
           param_postflopRaise[bets_to_call] and
           param_minToRaise[bets_to_call] */
        r = (1 / (param_postflopRaise[bets_to_call] -
            param_minToRaise[bets_to_call])) *
            (hand_evaluation - param_minToRaise[bets_to_call]);
        PT[raise] = r;
        PT[call]  = 1 - r;
    } else if (hand_evaluation > param_postflopCall[bets_to_call]) {
        PT[raise] = MINPROBABILITY;
        PT[call]  = 1 - 2 * MINPROBABILITY;
        PT[fold]  = MINPROBABILITY;
    } else {
        if (hand_evaluation > param_minToCall) {
            c = (1 / (param_postflopCall[bets_to_call] -
                param_minToCall[bets_to_call])) *
                (hand_evaluation - param_minToCall[bets_to_call]);
        } else c = 0;
        d = calculate_PotOdds();
        /* Join probability */
        c = d + c - cd;
        PT[raise] = MINPROBABILITY;
        PT[call]  = c * MAXPROBABILITY;
        PT[fold]  = (1 - c) * MAXPROBABILITY;
    }
    return (PT);
}
```

Figure 4.1: Pseudocode for a simplified PT generation function

4.2 Using probability triples

4.2.1 As a betting strategy

Loki-2 can decide what action to take either using PTs or selective sampling simulations. This section discusses the use of the PT generation function as the betting strategy of the program. Selective sampling simulations are discussed in Chapter 5.

Every time Loki-2 has to act in a game, it calls the PT generation function and selects its action based on the PT returned. The choice is made by generating a random number in the range $0.0 - 1.0$. For example, assume our hand and the current information about the state of the game is given to the PT function and it returns the triple $[0.1, 0.65, 0.25]$; if the random number is less than 0.1 Loki-2 folds, if it is less than 0.75 Loki-2 calls, otherwise Loki-2 raises. A single random number is generated at the beginning of each hand and used every time it is necessary to select an action in the game. The random number is kept constant, because it defines Loki-2's level of aggressiveness in that game. If the random number is high then Loki-2's probability of betting or raising in the game increases and thus Loki-2's playing style is more aggressive. Loki-2's aggressiveness should be consistent throughout a hand, because it is not a good idea to bet strongly early in the game only to give up later. A good player does not normally invest a lot in a hand and then fold easily in the next round. Using a single random number keeps Loki-2's style fixed in a game. However, varying the random number from game to game makes it more difficult for the opponents to create an accurate model of Loki-2 over a session.

The use of the PT generation function as Loki-2's betting strategy adds unpredictability to Loki's play. Unpredictability is a requirement to play strong poker, because if the opponents recognize a playing pattern then they are able to make better informed decisions. For example, if an opponent realizes that Loki will bet just with a very good hand ($EHS \geq 0.8$) then the opponent will fold when faced with a bet by Loki. The result is that Loki's winnings will be smaller. Using PTs to randomly select a betting action allows the program to vary its play over time, even in identical situations, making it difficult for opponents to predict the behavior of Loki-2 and to exploit its weaknesses.

```
reweight_weightTable(observed_action, opponent_wtTable) {
  remove_from_deck(community_cards);
  remove_from_deck(our_cards);
  hand_list = enumerate_all_possible_2card_hands(deck);
  for hand = hand_list[first] to hand_list[last]  {
    PT = probability_triple(hand, game_state);
    opponent_wtTable[hand] =
              opponent_wtTable[hand] * PT[observed_action];
  }
}
```

Figure 4.2: Pseudocode for the reweighting algorithm using PTs

4.2.2 As a reweighting factor

The opponent modeling module maintains an array for each opponent with weights for all the hands that opponent can hold. What does the weight for a specific hand represent? For instance assume that the weight for Q♣-T♢ in Loki's weight table for a particular opponent is 0.60. This weight indicates that *if* Q♣-T♢ has been dealt to this opponent then Loki believes there is a 60% chance that this opponent would have played in the observed manner so far in the game. In other words, the weight for a hand is the probability of an opponent's past behavior in a game given a specific pair of cards.

$$wt = P(\text{observed actions} \mid \text{pair of cards})$$

When an opponent action is observed, the weight table for that opponent is modified to reflect the latest action. In Loki-2 probability triples are used during the postflop rounds (after the three community cards have been dealt) as a reweighting factor to update the weight table of each opponent. Loki-2 computes the PT for each hand the opponent can hold (the community cards and Loki-2's cards are removed from the deck) and multiplies the weight of each hand by the entry in the probability triple that corresponds to the observed opponent's action (see Figure 4.2).

By using PTs to update the weight tables, the opponent modeling module was simplified. It was also improved since it makes better use of the information provided by an opponent's action by differentiating between a call and a raise, and by not ignoring an opponent's check. For example, assume that the entry in the weight table for the hand A♣-A♢ is 0.90, and the opponent calls. In the previous reweighting system the weight for A♣-A♢ would still be high, because the program only distinguished between fold and play. Now, if the PT for A♣-A♢ in the current context is $[0, 0.20, 0.80]$ then the updated weight for this hand would be $0.90 \times 0.20 = 0.18$ (i.e. A♣-A♢'s weight times the probability of the observed action). The relative likelihood of the opponent holding A♣-A♢ has decreased from 0.90 to 0.18 since no raise was made.

However, an opponent might try to deceive Loki by calling with a strong hand instead of raising. The call value of 0.20 in the above example reflects the uncertainty in Loki's beliefs about the actions of this particular opponent. Probability triple values allow Loki-2 to deal with the unreliable information during opponent modeling. This feature was not supported in Loki-1.

Specific opponent modeling

Loki-2 performs *generic opponent modeling* (GOM) in the sense that it uses the same PT generation function for all the opponents without accounting for each opponent‘s playing style. Obviously, treating all opponents the same is clearly wrong. Each player has a different style, ranging from *loose* (plays most hands beyond the flop) to *tight* (usually plays only those hands that have a very high probability of winning). In addition a player may be passive (calling instead of raising even with a strong hand) to aggressive (raising instead of calling). If the style of an opponent is known, a player can adjust betting decisions based on the opponent's style. For example, a perceived tight player who bets aggressively, probably has a strong hand. A loose player will play many marginal hands or may bluff a lot. This is useful information and may allow a player to fold a strong hand or call with a weak one when it is correct to do so. In general, a bet made by a loose aggressive player should not be taken as seriously as one made by a tight passive player.

Loki-2 can gather information about the opponents to obtain betting frequencies for each opponent and use this data to customize the PT function to account for the playing style of each opponent. This process is called *specific opponent modeling* (SOM). The default call and raise thresholds used in the PT generation function can be adjusted by betting frequency statistics gathered on each opponent from previous hands. Thus, the reweighting factors applied to the entries of each opponent's weight table are adjusted to better fit their playing style. For example, assume a tight opponent raises. Since in the case of a tight player, the call and raise thresholds will increase, few PTs generated will have a high raise value. Hence, after reweighting, this opponent's weight table will indicate fewer hands that are likely to be held.

Loki-1 already collected the betting action frequencies of the opponents each round based on the number of bets to call. These action frequencies are used by Loki-2 in the PT generation function to obtain the EHS thresholds required to perform the observed action. The EHS thresholds are obtained in the same way as was done in Loki-1. For example, assume Loki has observed twenty actions by a specific opponent on the turn with one bet to call. Assume the observations are six raises, eight calls and six folds. The EHS raise threshold used for this opponent by the PT generation function when an action is observed on the turn with one bet to call is $1 - \frac{6}{20} = 0.7$. The EHS call threshold used is $0.7 - \frac{8}{20} = 0.3$. Besides the accumulated (historic) action frequencies collected by Loki-1, Loki-2 keeps track of the last twenty actions of each opponent observed in the current session. EHS thresholds are calculated from both records (historic action frequencies and last twenty observed actions) and averaged to obtain the EHS thresholds used by the PT generation function. Keeping track of the last twenty actions allows Loki-2 to react more quickly to changes in the opponents' playing style.

Superior opponent modeling is much more complex than the current techniques used by Loki. Players can act to mislead their opponents into constructing an erroneous model. For example, early in a session strong poker players may try to create the impression of being very conservative, only to exploit that image later in that session when their opponents are using an incorrect model about them. Players can also vary their style over a session, and then recency of the information gathered about them has to be considered. Therefore, a strong player must continually adapt the model for opponents who may be varying their playing style or trying to deceive.

4.3 Experiments

4.3.1 Design

One goal of this research project was to construct a series of self-play poker tournament experiments to obtain statistically significant results that show each enhancement improved Loki-2's performance under different playing conditions (as is typically seen against human competition). The experimental design to accomplish these goals is described in this section.

Each self-play tournament consists of playing two versions of Loki against each other: eight copies of a control version and two copies of a modified version. To reduce the "luck" factor of the game and consequently the variance, the tournaments follow the pattern of duplicate bridge tournaments described in [2] and [19]. Each deal is played ten times, each time changing the seat order so that 1) every player holds every set of hidden cards once, and 2) every player is seated in a different position relative to all opponents. A tournament consists of 2,500 different deals (i.e. 25,000 games or *trials*).

The playing style of a player is defined by the percentage of hands played (e.g. liberal-loose or conservative-tight) and the frequency of raising when active (e.g. aggressive or passive). Players are classified using a two character notation where the first letter represents the percentage of hands played and varies from tight (T) to loose (L), and the second letter represents the raising frequency and goes from passive (P) to aggressive (A). These characteristics are not exclusive in a player. For example, a conservative/aggressive (T/A) player will play few hands (fold most of the hands in the preflop), but will bet/raise often when active.

To test an enhancement, one particular version of the program is first played against an identical program with the new feature in a homogeneous field (all the players have the same playing style). For example, one can play eight conservative/aggressive base Loki-1 players against two conservative/aggressive Loki-2 players that are augmented with the PT function betting strategy. Second, the enhancement is tested in combination with other changes. Third, the modification is tested against opponents that have different playing styles.

To measure the impact of each new enhancement on the program's performance, we use the average number of small bets won per hand (sb/hand). This is a metric sometimes used by human players. For instance, in a game of \$10-\$20 Holdem (small bets are \$10 and big bets are \$20), a player who has an improvement of +0.20 sb/hand will make an extra \$60 per hour (based on 30 hands per hour); anything above +0.05 sb/hand is considered a large improvement. One must be cautious when interpreting the results of these self-play experiments, since any feature could perform worse (or better) playing against human opposition [1]. The main function of these experiments is to weed out bad ideas. Ultimately, the only performance metric that is important is how Loki plays against humans. Since it is difficult (and expensive) to get this data, most of our experimentation must be done with self-play first.

4.3.2 Results

This section contains the experimental results of using Probability Triples as a stand-alone betting strategy (Section 4.2.1) and as a reweighting factor in opponent modeling (Section 4.2.2). Both the individual effect of each enhancement and their combined result are discussed. In the experiments, B stands for the use of the PT generation function as a betting strategy and R stands for the change of the reweighting system to use PTs.

Figure 4.3 shows the results of playing ten Lokis against themselves in a homogeneous environment (only one type of player) with the B and R enhancements individually and combined (B+R). The players were eight Loki-1 players against two enhanced Lokis. Loki-1's performance is the baseline for the comparison. The B feature represents a 0.041 ± 0.009 sb/hand improvement, the R feature represents a 0.055 ± 0.016 sb/hand and B+R represents a 0.085 ± 0.020 sb/hand improvement. The B+R results show that the effect of these enhancements is nearly additive since these features are almost independent of each other. Note that each enhancement is a win by itself and in combination with the other one.

A second series of experiments was conducted to see how well the new features performed against a mixture of opponents with different styles. For this set of experiments, opponents with different playing styles were used. In each experiment there was a pair of players from each of the four categories: tight/passive (T/P), tight/aggressive (T/A), loose/passive (L/P) and loose/aggressive (L/A). In each pair,

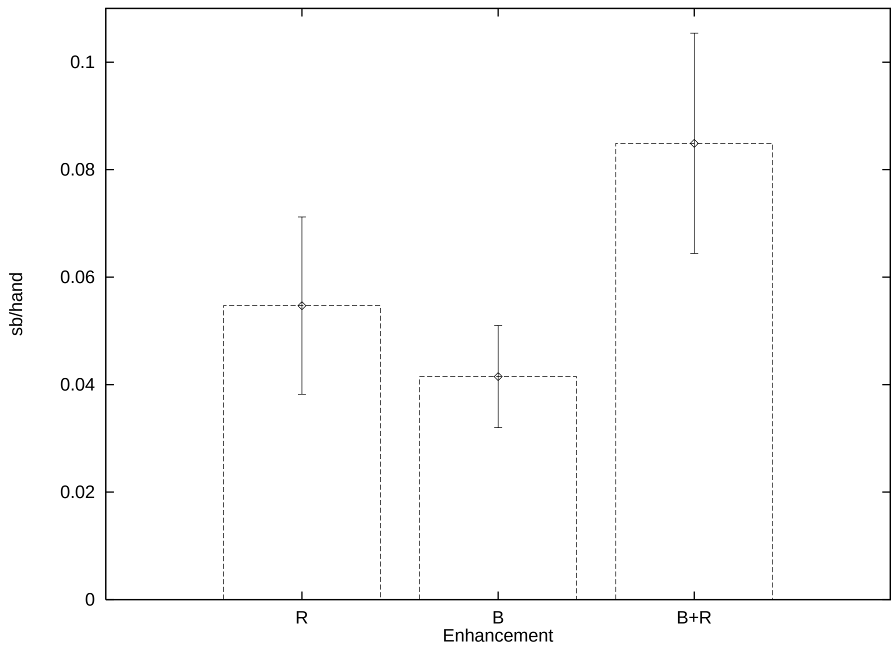

Figure 4.3: Loki-1 versus Loki-2 with PT enhancements

one of the players was a basic Loki-1 player and the other was a Loki-2 player with either new betting strategy (B), new reweighting system (R) or both features (B+R). Figures 4.4, 4.5 and 4.6 show the absolute improvements per type of player obtained by adding B, R or B+R respectively. Since ten players are required to play a tournament, we actually used two pairs of tight/passive players and reported the average over the results of both pairs.

In each mixed experiment, the enhanced player always outperformed the corresponding un-enhanced player. The absolute individual improvement varies greatly from one style of player to another. For example, the L/A player enhanced by R in Figure 4.5 had a 0.11 sb/hand improvement going from -0.033 sb/hand to 0.077 sb/hand, while the T/P player enhanced with R went from -0.031 sb/hand to 0.023 sb/hand showing a 0.054 sb/hand improvement. On average the B enhancement produced an improvement of 0.045 sb/hand, the R enhancement of 0.070 sb/hand and B+R of 0.096 sb/hand. These experiments showed that both enhancements win, regardless of playing style.

Loki-2 with B and R was also tested under more realistic conditions against human opposition. Loki-2 plays in an on-line poker game running on the Internet Relay Chat

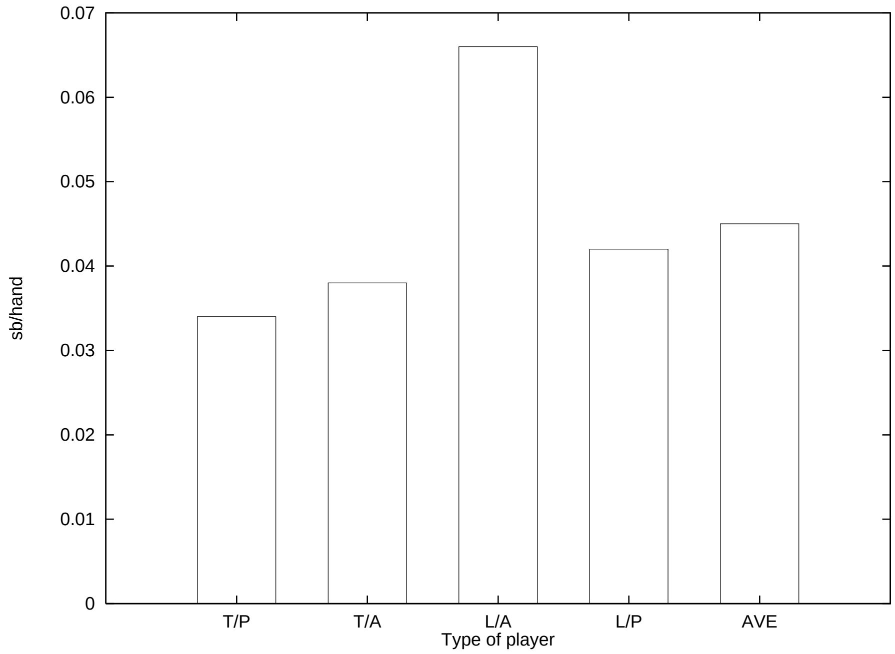

Figure 4.4: PTs as betting strategy (B) in a mixed environment

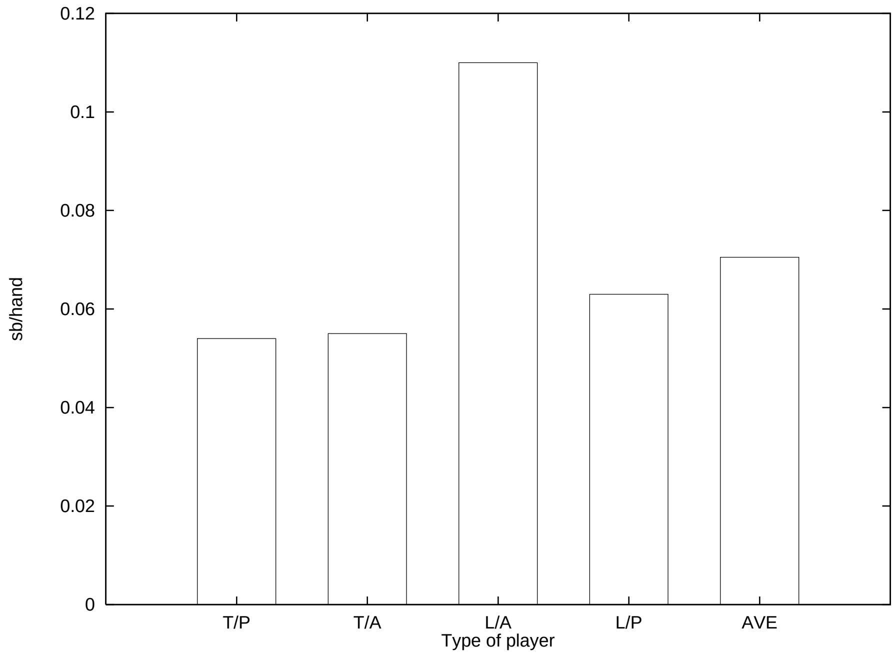

Figure 4.5: PTs as reweighting factor (R) in a mixed environment

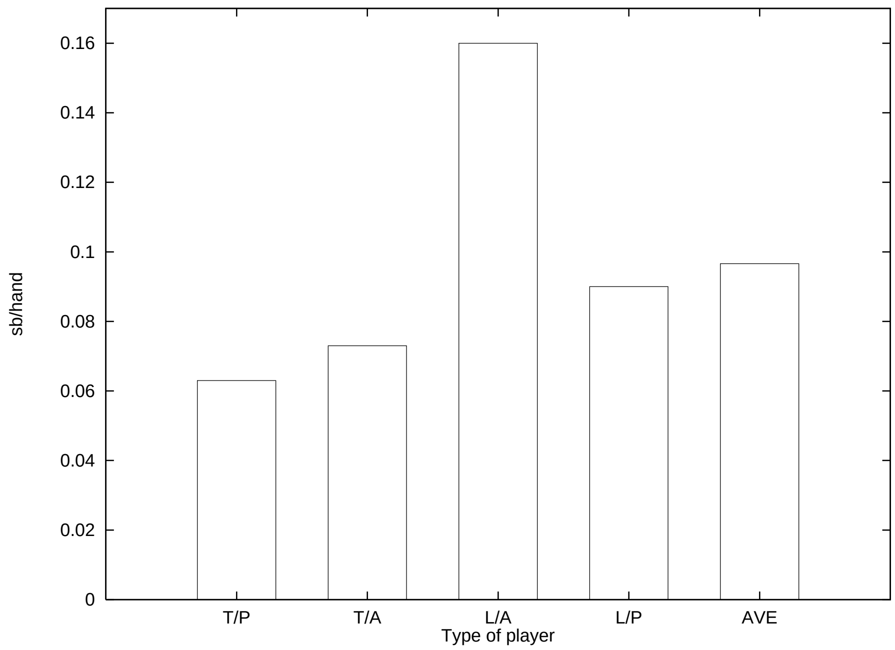

Figure 4.6: PTs as reweighting factor and betting strategy (B+R) in a mixed environment

(IRC) poker server (irc.poker.net). Human players and other poker-playing programs connect to IRC and participate in games conducted by dedicated server programs. No real money is at stake, but bankroll statistics on each player are maintained. Since there is no control over the quality and type of opponents, the performance of the program depends strongly on which players happen to be playing and the variance in these games is very high. However, Loki-2 is a consistent winner increasing its bankroll at a rate of 0.11 sb/hand in 25,703 games (Loki-1's winning rate on IRC was 0.08 sb/hand).

4.4 Summary

Representing poker decisions as a set of three probabilities provides a suitable infrastructure to perform well in a noisy environment, where randomized strategies and misinformation are important aspects of strong play. Besides the performance improvement obtained by the use of PTs as a non-deterministic betting strategy and as the reweighting factor in the opponent modeling module, the PT generation function improves the design of the system by encapsulating all the knowledge-based components of Loki-2 in a single routine.

Although the results obtained are encouraging, there are still opportunities for improvement. Using showdown and bluffing information about the opponents to perform more accurate specific opponent modeling, replacing Loki-1's preflop betting strategy with a PT-based strategy, using PTs in preflop reweighting, and refining the knowledge in the current probability triple function are some of the possible next steps.

Chapter 5

Selective sampling simulation

The general structure of a program for a perfect information game, such as chess or checkers, contains an evaluation function and a search algorithm. Loki's knowledge-based betting strategy is, in fact, analogous to a static evaluation function. If deterministic perfect information games are used as a model then the obvious extension is to add "search" to Loki's evaluation function.

In checkers or chess, the average branching factor is 3 and $30-40$ respectively. One can consider all possible moves as deeply as resources permit. However, in poker the existence of hidden information, uncertainty and multiple players makes the same approach infeasible. There are too many possibilities to consider. In a two-player Texas Hold'em game there are 363.9×10^6 possible states at the beginning of the flop and $\binom{47}{2} = 1,081$ possible opponent's hole cards (see Figure 4.2 in [19]) plus multiple possibilities for each betting round. Therefore, computing the complete game tree for poker is prohibitively expensive in real-time. If exhaustive search is out of the question, how do we add "search" to Loki?

We can examine (*simulate*) a representative sample, as large as resources permit, from all the possible game scenarios. The larger the sample and the more informed the selection process, the higher the chances to deduce meaningful conclusions.

A simulation consists of playing out a hand in many likely scenarios, from the current state of the game through to the end, to determine how much money each decision will win or lose. Every time Loki-2 faces a decision, it performs a simulation to get an estimate of the *expected value* (EV) of each betting action and then chooses the action with the greatest expectation. Inside each simulation, Loki-2 uses probability triples (PTs) to generate actions for all the opponents and itself, as well as, opponent modeling information (weight tables) to bias the selection of opponent's cards.

5.1 How simulation works

When it is Loki-2's turn to act, it invokes the simulation routine to get an estimate of the EV of calling and raising. Folding is considered to have a zero EV, because there is no further profit or loss. The simulation routine plays out Loki-2's hand a specified number of times (*trials*). However, each trial is actually played out twice – once to consider the consequences of a check/call and once to consider a bet/raise. For each case the amount of money won or lost is determined and averaged with the corresponding results of all the trials. At the end of the simulation the averages of the two sets of trials are taken as the EVs of the corresponding actions.

Simulation is analogous to a selective expansion of some branches of a game tree. Since not all the branches of the game tree can be expanded due to time constraints, the information obtained from a simulation needs to be maximized. The "perfect" simulation would examine only the real game state (complete information about the opponent hands, played out over all possible combinations of future community cards). However, the "perfect" simulation is impossible without knowing the opponents' cards, and an accurate estimate may be found without looking at all possible outcomes of future cards. One can try to approximate the EV values obtained by the "perfect" simulation by expanding and evaluating the nodes which are most likely to occur. In poker not all opponent's hands are equally likely. For example, a player who has been raising the stakes is more likely to have a strong hand than a player who has just called every bet. To consider the opponents' hands in proportion to their underlying probability distribution, Loki-2 uses the information gathered by the opponent modeling module. At the beginning of every trial, Loki-2 randomly generates a hand for each opponent based on the weight table of that opponent. A random method is used to generate the opponents' hands, because of the simplicity of its implementation.

Loki-2's first betting action is predetermined to be either call or raise. Every time it is a player's turn to act inside the simulation, an action is chosen from one of three alternatives (fold, check/call, bet/raise). Since the choice is strongly correlated to the quality of the cards that the player holds, Loki-2 can use the PT generation routine to obtain the likelihood that the player will fold, check/call, or bet/raise. Thus, when a player (an opponent, or Loki-2 after its first action) has to act in the simulation, the PT generation function is called with the player's hand and the current state of the simulated game. The player's action is then randomly selected based on the probability distribution defined by the triple returned, and the simulation proceeds.

As more trials are performed, if the EV of one betting action exceeds the alternatives by a statistically significant margin, one can say that this action is an *obvious move* and the simulation can be stopped early, with full knowledge of the statistical validity of this decision. We currently define an obvious move as any action where the separation between the EV of the best action and the EV of the second best action is greater than the sum of the standard deviations of the EVs. This criterion for defining an obvious move is extremely conservative, since the separation between the "best" decision and the next one is usually not more than two small bets, and the average standard deviation of the EVs is six small bets for calling and eight small bets for raising. This situation results in declaring fewer than 5% of actions as obvious moves. Given the real-time nature of the game, more liberal criteria for distinguishing obvious moves need to be tested to produce more frequent cutoffs while retaining same statistical validity.

The interactions between the opponent modeling module (Opponent Modeler), the PT generation routine (PT Generator) and the simulation module (Simulator) are shown in Figure 5.1. In the diagram squares are system components and rounded rectangles are data structures. The data follows the arrows between components. The square corresponding to the Simulator also illustrates the major steps inside the simulation process. The dashed square around the Opponent Modeler and the PT Generator indicates that their interaction occurs before the simulation starts.

1. For every trial, the Simulator generates the opponents' hands based on their weight tables which have been updated by the Opponent Modeler.

2. Each trial is played twice – once with call as the first action and once with raise as the first action. As the hand is played the PT Generator is called to obtain

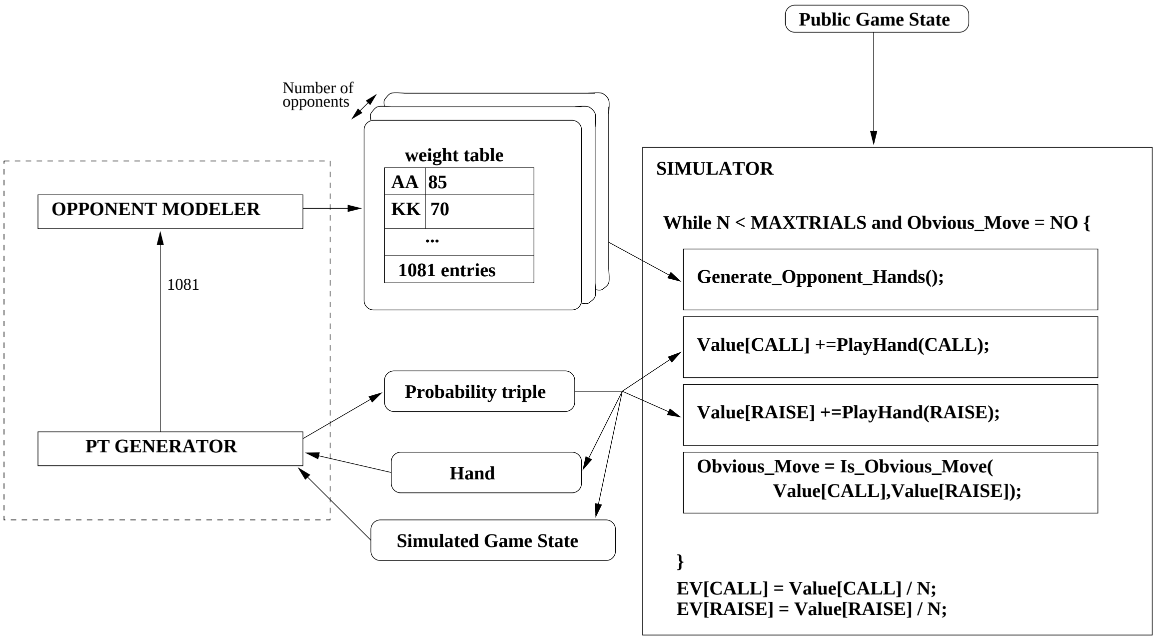

Figure 5.1: Simulation process

the likelihood of the actions of the players (including Loki-2). This means that all players in the simulation use the PT Generator as their betting strategy.

3. The Simulator stops when an obvious move is found or the maximum number of trials is performed.

4. At the end of the simulation the expected value (the average over all the trials) for each action is calculated.

When the simulation returns the EV values for check/call, bet/raise and zero for fold, the current version of Loki-2 simply chooses the action with the greatest expectation. If two actions have the same EV, the program opts for the most aggressive one (call over fold; raise over call). However, against human opposition, a better strategy will be to randomize the selection of betting actions whose EVs are close in value to increase unpredictability.

5.1.1 Dealing cards out

The opponents hands are generated according to the seating order of the players (the small blind gets cards first and the dealer gets cards last). The criterion for assigning the hole cards to an opponent depends on whether the opponent is still active in the game or not. Hole cards are dealt to the folded players because we want to choose the cards to come (turn and river cards) not only from the correct number of cards, but also from cards with the correct distribution of weak/strong cards.

To deal the hole cards to an opponent, the Simulator uses selective sampling. It randomly extracts two cards from the deck and generates a random number in the range $0.0 - 1.0$. In the case of an opponent already folded, the cards extracted are kept as the opponent's hole cards if the preflop hand value (income rate of the hand) is less than the random number; otherwise, the cards are returned to the deck and the generation process is repeated. Thus, preference is given to weak hands (hands that are likely to fold on the preflop). For an opponent who is still active in the game, the cards extracted are kept if the weight for the two cards in the weight table of the opponent is greater than or equal to the random number; otherwise, the cards are re-inserted into the deck, and the Simulator extracts two cards and generates another random number. Since the weight of a pair of cards indicates the likelihood of the opponent holding these cards, preference is given to the most likely holdings. However, all the two-card combinations have some opportunities to be selected.

In analyzing the results of self-play experiments with selective sampling simulation, we noticed that simulations contain high variance and a lot of noise. We need to keep the sampling in the simulation as representative and fair as possible to get the best possible (reliable) results. Thus, different methods to reduce variance have to be tested. For example, in the current version, a random selection of the turn and river cards is made for every trial in a simulation. To reduce statistical anomalies and variance, one can obtain a perfect representation of the one-card potential by dealing all 47 possible turn cards exactly once. Then a certain number of river cards can be chosen, without replacement, for each of these turn cards.

5.2 Experiments

The experimental design used to test Loki-2's simulation-based performance is the same as described in Section 4.3.1. In a tournament, there are eight Loki-1 players playing against two Loki-2 players. A tournament consists of 2,500 different deals played ten times each (i.e. 25,000 games). The number of trials per simulation was chosen to meet real-time constraints and statistical significance. In the experiments, 500 trials per simulation were performed, since the results obtained after 500 trials were quite stable. For example, 4.6% of the betting actions selected with 100 trials changed after more trials were performed, whereas only 0.5% of the decisions were changed after 500 trials.

Selective sampling simulation was tested alone and in combination with the PT enhancements. Figure 5.2 shows the increment in Loki-2's performance in a homogeneous environment obtained by using the following modifications in Loki-1:

- S = Selective sampling simulation,
- S+R = Selective sampling simulation with PT-based reweighting,
- S+B = Selective sampling simulation with PT-based betting strategy as the action generation mechanism inside the simulation, and
- S+B+R = Selective sampling simulation with PT-based reweighting and PT-based betting strategy.

In the graph, Loki-1's performance is the baseline for comparison. Selective sampling simulation (S) represents an improvement of 0.098 ± 0.038 small bets per hand (sb/hand). By adding both PT enhancements (S+B+R), an improvement of 0.11 ± 0.035 is obtained. As can be seen in the graph, the effects of S, B and R are not additive. These enhancements may exploit the same aspect of the opponents' play and their effects overlap. Another reason may be the hyper-aggressive playing style of the simulation-based players. They are very successful against Loki-1 players, and can lead to over-optimistic conclusions about the performance improvement represented by S. Since the B enhancement allows us to simulate less tight opponents, S+B may result in a less aggressive playing style, lowering S+B winnings against Loki-1 opponents.

Also, one has to consider that the larger the winning margin, the smaller the opportunity there is for demonstrating further improvement against the same opposition. There is a limit to how much money one can make from an opponent in a game.

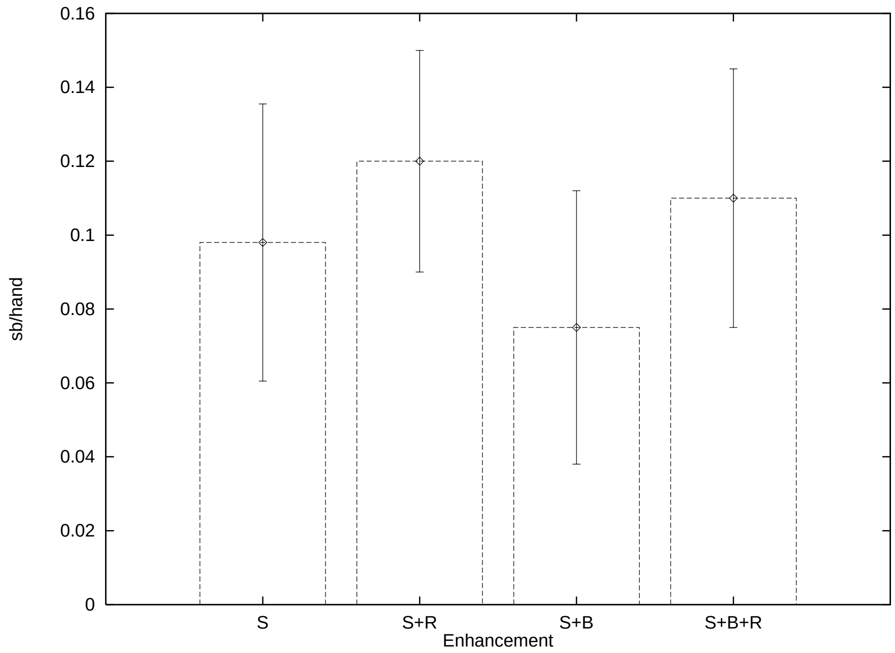

Figure 5.2: Selective sampling simulation experiments

Two other experiments were carried out raising the baseline for comparison. In the first experiment, two Loki-2s with S were matched with a field of Loki-2s with B+R. The S enhancement won 0.023 ± 0.044 sb/hand. In the second experiment, the same field of opponents (B+R) played against two Loki-2s with S+B+R. The S+B+R's winning rate was 0.087 ± 0.038 sb/hand.

A mixed environment experiment was conducted to see how well selective sampling simulation performed against different playing styles. In this experiment, the field of opponents was composed of pairs of players with the styles: tight/aggressive (T/A), loose/passive (L/P) and loose/aggressive (L/A), as well as two pairs of tight/passive (T/P) players. From each pair of players, one player used selective sampling simulation and the other one was a Loki-1 player. Figure 5.3 shows the average performance of the players without simulation (AVE) and the average performance with simulation (AVE+S). The average improvement obtained for all different players by using simulation is 0.036 sb/hand.

Loki-2's playing ability with the three enhancements (S+B+R) was also tested against human opposition in on-line poker games on the Internet Relay Chat (IRC). Loki-2's winning rate is 0.13 sb/hand in 26217 games (Loki-1's winning rate on IRC

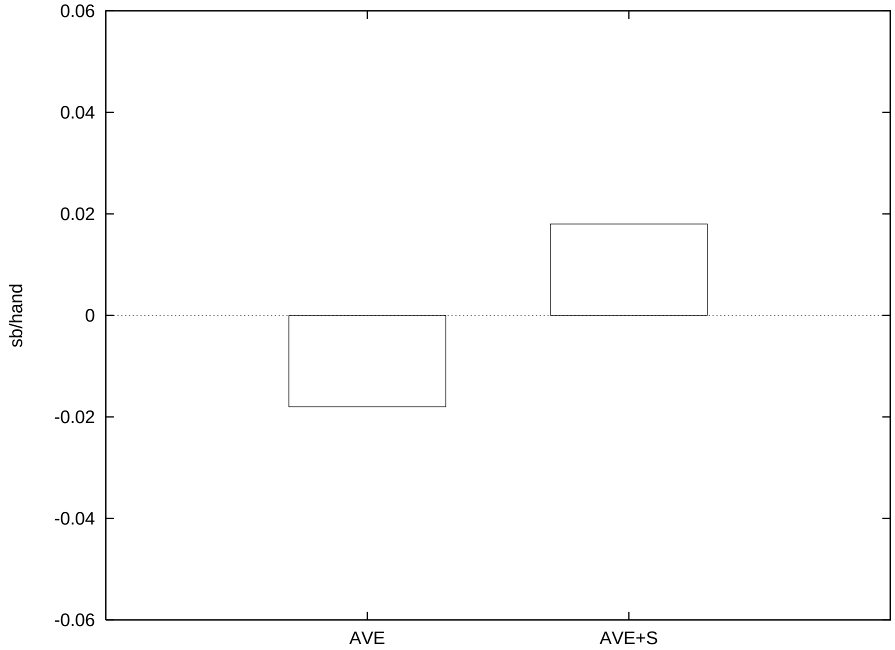

Figure 5.3: Selective sampling simulation - mixed environment

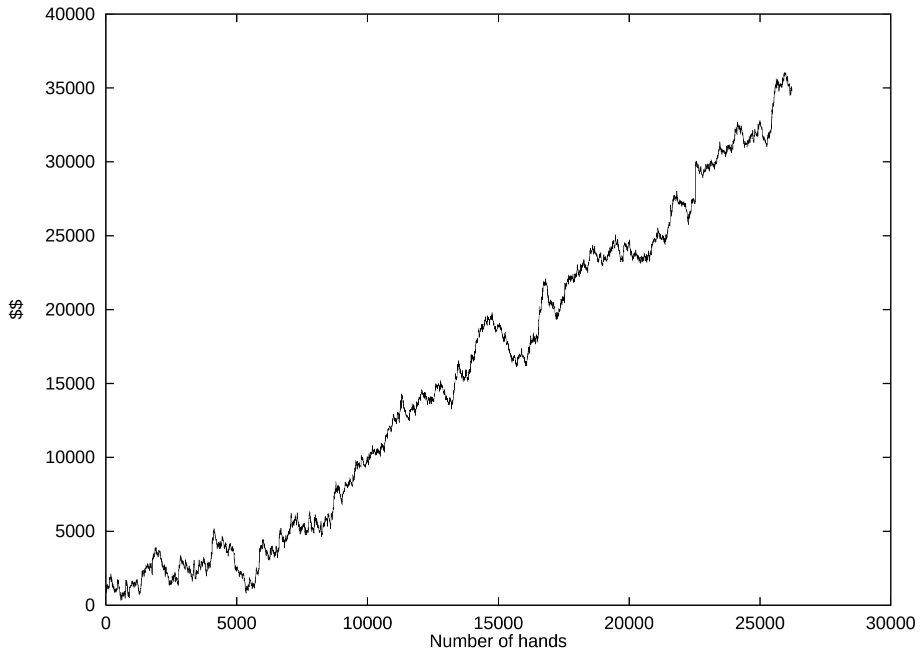

Figure 5.4: Loki-2's behaviour on IRC

was 0.08 sb/hand). Figure 5.4 shows Loki-2's behaviour on the first level of IRC.

5.3 Comments about selective sampling simulation

5.3.1 Advantages

The simulation-based approach used in Loki-2 has experimentally proved to be better than Loki-1's static approach. This should not be at all surprising, since the simulation approach essentially uses a selective search to augment and refine the static evaluation function. Excluding a serious misconception (or bad luck on a limited sample size), playing out relevant scenarios can only be expected to improve the values obtained by a heuristic (i.e. by the static evaluation function), resulting in a more accurate estimate.

Selective sampling simulations discover information that improves the values obtained by the static evaluation function. In both Loki-1's betting strategy and the PT generation function, actions are taken based on the hand evaluation. During a simulation, the accuracy of the hand evaluation is increased. The number of trials where our hand is stronger than the one assigned to the opponents refines the estimate of hand strength. The fraction of trials where our hand becomes the best one, or is overtaken, with the next cards dealt refines the calculation of hand potential. In addition, a simulation yields information about subtler implications difficult to address in a static betting strategy.

By performing simulations Loki-2 is able to find game strategies which are not specified in the knowledge contained in its evaluation function. For example, if a player has a strong hand then the player can pretend weakness by checking in the first turn to act in a betting round. The opponents will likely bet to their hands (thinking that the player's hand is not good), and then a raise will collect more money than betting as the first action. This strategy is known as *check-raising*. Loki-2 check-raises its opponents without having the explicit knowledge to do it. Selective sampling simulations uncover the benefits of complex strategies such as check-raising without providing additional expert knowledge to the program.

The use of available information about the game to bias the sampling of the game tree is the key difference between *selective sampling simulations* and *Monte Carlo simulations*. Selective sampling is context sensitive. In Loki-2 the opponents' weight tables are used to influence the selection of hole cards for each opponent. The sample is taken in accordance with the underlying probability distribution of the opponents' hands rather than assuming uniform or other fixed probability distributions. Although Monte Carlo techniques may eventually converge to the right answer, selective sampling converges faster and with less variance. This is essential in a real-time game like poker.

Finally, another benefit of the simulation-based framework is that the simulation can be terminated early based on the statistically well-defined concept of an obvious move instead of using an *ad hoc* technique as is frequently done in alpha-beta-based programs.

5.3.2 Simulation trade-offs

As encouraging as the simulation results have been, there is still room for improvement. Simulation can reveal information that results in better betting decisions and refine the estimate of the evaluation function. Yet, a simulation contains high variance and it is unclear how the other components of the program impact on its performance.

As was discussed in the previous section, simulation can recover from lack of knowledge in the evaluation function, produce complex game strategies and refine the estimated EV of the actions. This feature raises the question: how much knowledge does simulation require? Every time a new card is dealt in a trial, the hand strength and hand potential for active hands in the trial are calculated. For each betting action to be played, a PT generation function is called to generate an action for the opponents and Loki-2. Thus, during the simulation every trial is an accurate representation of a real game. However, each trial is a time-consuming process. In fact, one can exchange the accuracy of each trial for the number of trials performed in real-time. This tradeoff was explored by replacing the PT generation function betting strategy with an "always call" betting strategy inside the simulation. An "always call" betting strategy is probably the simplest betting strategy that can be provided to the simulation; it always returns call as all of the players' actions and all of Loki's subsequent actions. Therefore, in an "always call" simulation, there is no further betting after Loki's first decision, which is predetermined to be either call or raise, and every trial only consists of dealing all the cards and determining the hand that takes the pot. "Always-call" simulation-based Loki-2 wins against Loki-1 by a healthy margin (0.057 sb/hand) and runs 2.5 times faster than Loki-2 using the PT generation function. However it does not win as much as the full simulation-based Loki-2 does. Even though simple simulation is better than no simulation at all, knowledgeable simulation seems to provide better results.

5.3.3 Comparison with alpha-beta

The alpha-beta algorithm [20] has proven to be an effective tool for the design of two-player, zero-sum, deterministic games with perfect information. Its origins go back to the beginning of the 1960's. Since that time the basic structure has not changed much, although there have been numerous algorithmic enhancements to improve the search efficiency. The selective-sampling simulation technique is becoming an effective tool for the design of zero-sum games with imperfect information or conditions of uncertainty (see Chapter 6). Table 5.1 shows a comparison between the usual characteristics of both approaches.

Criterion	**Alpha-beta**	**Selective sampling simulation**
Search	Full breadth, but limited depth	Full depth, but limited breadth
Iteration	Search depth (iterative deepening)	Number of samples taken
Heuristic Evaluation	At the leaf nodes	At the interior nodes
Interior node alternatives considered	All, except for those logically eliminated	A subset, to reduce the cost of a sample
Statistical support to best move choice?	No	Yes

Table 5.1: Comparison between search frameworks

In poker the heuristic evaluation at the interior nodes of the simulation is done to determine the appropriate opponents' actions and Loki-2's actions. The leaf node evaluations are the amount of money won or lost, since the simulation done for each sample goes to the end of the game. In deterministic games, the heuristic evaluation is done to estimate the expected utility of the game from a given position (i.e. the value of the subtree beyond the maximum search depth of the program). A simple leaf node evaluation for some perfect information games can be the material balance. Figure 5.5 illustrates where the evaluation occurs in the search and simulation approaches and the game space explored by them.

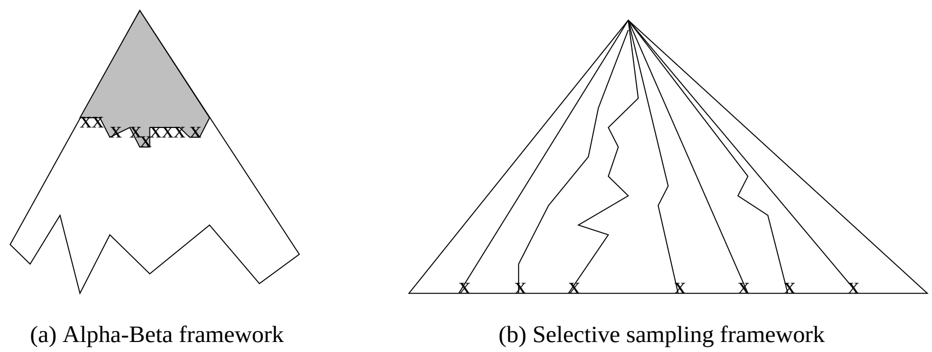

Figure 5.5: Search space explored

The alpha-beta algorithm gathers confidence in its move choice by searching deeper along each line. The deeper the search, the greater the confidence in the move choice, although diminishing returns quickly takes over. The alpha-beta algorithm is designed to identify a “best” move, and not differentiate between other moves. Hence, the selection of the best move may be brittle, in that the misevaluation of a single node can propagate to the root of the search and alter the best move choice. Similarly, selective sampling simulation increases its confidence in the answer as more nodes are evaluated. However, diminishing returns takes over after a statistically significant number of trials have been performed.

Selective sampling simulation can be compared to selective search or forward pruning techniques in alpha-beta algorithms. These techniques discard some branches to reduce the size of tree; however, their major drawback is the possibility that the lookahead process will ignore a key move at a shallow level in the game tree [18]. To be reliable, forward pruning methods need to reason about the tree traversal to deduce which “future branches” can be excluded. On the other side, selective sampling simulation uses available information about the game and the opponents to explore the most likely “current branches” of the game tree.

5.4 Summary

According to the results of self-play experiments, a selective sampling simulation-based betting strategy for Loki-2 significantly outperforms the static-evaluation based alternatives. Similar to what has been seen with brute-force search in games like chess, the effect of the simulation (search) amplifies the quality of the evaluation function, allowing high performance to be achieved without adding additional expert knowledge. Selective sampling uses the data available about the game and the opponents to increase the quality of the information obtained with each simulation run. Yet, the work on selective-sampling simulation in poker is still in its early stages. The knowledge component and selection methods have to be tuned with the algorithmic component of the simulation, and the right balance between the different simulation tradeoffs (cost per trial versus number of trials, random versus systematic approach) has to be found.

Chapter 6

Other examples of selective sampling simulation

The use of stochastic simulation to solve problems where exhaustive methods take too long is not new to the Artificial Intelligence community. Specifically, stochastic methods have been developed for performing inference in belief networks and for decision-making in imperfect information or non-deterministic games.

There are two important advantages of stochastic simulation algorithms. They have an "anytime" property in the sense that they can be stopped at any time and will give the best answer available. Given more time, their estimates will improve. This "anytime" property is especially appropriate for real-time domains like poker. Also, simulation algorithms are trivial to perform in parallel.

In this chapter, simulation algorithms used in belief networks and games will be reviewed. The algorithms discussed perform selective sampling on the search space by using available information about the state of the world to bias the sample selection. The aim of selective sampling methods is to converge faster than approaches using uniform sampling.

6.1 Belief networks

Bayesian belief networks (BBNs) are a graphical representation for reasoning under uncertainty [20]. A BBN is a directed acyclic graph with a conditional probability distribution for each node. BBNs contain nodes representing domain variables, and arcs between nodes representing probabilistic dependencies. The basic task for a BBN is to compute the posterior probability distribution for a set of query variables, given exact values for some evidence variables.

Since exact ("brute-force") algorithms for performing inference on BBNs take exponential time (in the number of nodes) in the worst case, they cannot handle large or highly connected networks. Stochastic simulation algorithms have been developed to give approximate results for a wider variety of belief network topologies. Simulation algorithms differ from the brute-force approach in that they select only a sample of the node states, whereas a brute-force strategy selects every state. One of the simulation algorithms for BBNs, *likelihood weighting* [22] [11], is similar to the idea of selective sampling simulation in poker.

Likelihood weighting selects the node states based on their prior probability of occurrence. By choosing more likely states more often, this algorithm typically is able to converge much more quickly than *equiprobable sampling* which randomly chooses a state for each node in the network [8]. Likelihood weighting chooses a state for a node by generating a random number between 0 and 1 and using this number to select a state according to the conditional probability table of the node. For each simulation trial, the probability of the evidence given the sampled state values is used to increment the count of each event of interest. The estimated probability distribution is obtained by normalizing after all the simulation trials are completed.

By using selective sampling simulation in poker, we estimate the expected values of betting actions instead of estimating the posterior probability distribution for a set of variables. Both simulation methods have the following characteristics:

1. They select the states to sample (node states or opponents' hands) on the basis of their probability of occurrence by using either conditional probability tables or (in our case) weight tables.

2. Both methods can use heuristics or information available about the world to modify the tables for choosing states and bias the state selection to the most likely ones.

6.2 Games

In this section, three game-playing programs that use simulations are described. These programs do not use Monte Carlo sampling to generate instances of the missing information. They use variations of selective sampling; sampling biased towards taking advantage of all the available information. They use information about the game state to skew the underlying probability distribution of the opponents' moves, cards or tiles, rather than assuming uniform or other fixed probability distributions. The general simulation algorithm used by these games to select a move from a set M of candidate moves is:

1. Construct a set I of instances of the missing information consistent with the public information about the state of the game and the program's assumptions (information) about the opponents.

2. For each move $m \in M$ and each instance $i \in I$, evaluate the result of making the move m in the instance i. Denote the score obtained by making this move $s(m, i)$.

3. Return that m for which $\sum_i s(m, i)$ is maximal.

6.2.1 Backgammon

In backgammon the unknown outcome of the dice rolls makes the brute-force approach infeasible by raising the branching factor to several hundreds moves (21 possible dice combinations, each of them having 20 legal moves). The backgammon program TD-Gammon [26] [27] uses *temporal difference* (TD) learning to learn by itself how to play backgammon at a world-championship level. The TD-Gammon neural network is trained by self-play simulations. During training, TD-Gammon considers each of the 21 ways it can play its dice role and the corresponding positions that will result. Then, the move that leads to the position with the highest estimated value is chosen. This learning method is used even at the start of the training when the network's strategy is random. After playing about 300,000 games against itself, TD-Gammon 0.0 with essentially zero backgammon knowledge learned to play approximately as well as the best previous backgammon computer program. Self-play training refined with some initial backgammon knowledge produced a program that played at a world-class level.

Recent versions of the program were augmented with a selective two-ply or three-ply search procedure. A *ply* is an individual playing action (only one of the players makes a move). To select moves, these programs look ahead to consider the opponent's possible dice rolls and moves. Assuming that the opponent always takes the move that appeared immediately best for the opponent, the expected value of each candidate move is computed and the best move is selected. The second ply of search is conducted only for candidate moves that were ranked high after the first ply. This selective search procedure affects only the move selection; the learning process proceeds exactly as before.

Also, simulations are used in backgammon to perform "rollouts" of certain positions. The rollouts are now generally regarded as the best available estimates for the equity of a given position. A simulation consists of generating a series of dice rolls, playing through to the end of the game, and then recording the outcome.

6.2.2 Bridge

In bridge the hidden information consists of the cards that the opponents hold. The current best bridge program GIB [14] performs simulations in two stages of the game: during the auction to make a bid and during the actual game to decide which card to play.

To select a bid, GIB deals cards to the opponents in a way that is consistent with the bidding observed so far. GIB uses a database to project how the auction will continue if a certain bid is made, and then computes the result of playing out the hand. The hands are played in a double dummy variation of bridge (assuming perfect information – knowledge of all four hands). At the end of the simulation the bid with the maximal expected value is returned.

During a game, a simulation consists of dealing cards to the opponents in a manner that is consistent with the bidding and the cards played so far. The score of a move is determined by playing out the hand in a double dummy mode [12]. Repeated deals are played until either enough confidence is gained to decide which card to play, or a maximum number of hands is simulated, or a real-time constraint is met.

Opponents' cards are constrained by the information given by each player about the hand during the bidding. GIB also uses a probability distribution of the possible cards held by an opponent to bias the card dealings towards the most likely ones. This probability distribution is adjusted by identifying mistakes the opponents might make during the game. For example, assume that GIB's analysis says that 75% of the time that a player holds a specific card and does not play it in a particular game situation that an error has occurred. The probability of this opponent holding that card is modified accordingly after GIB observes that the card was not played.

Simulations have allowed GIB to play hands at a world-class level; however, limitations in the simulation-based approach and the high variance have prompted the author of GIB, Matt Ginsberg, to look at other solutions (including building the entire search tree) [13].

6.2.3 Scrabble

A simulation-based approach has been used for a long time in Scrabble programs. Brian Sheppard, whose Scrabble program Maven defeated Grandmaster Adam Logan (a top-ranked player in the world) in the AAAI-98 Hall of Champions, coined the term "simulator" for this type of game-playing program structure.

During the non-endgame stage of a Scrabble game, Maven [24] chooses its moves using simulation to try to determine which move for the computer leads to the maximum number of points. To select a move Maven generates a set of candidate moves, simulates these moves a specific number of trials and chooses the move whose expected value is highest. A simulation trial consists of a two to four ply search of the game tree, except in the pre-endgame where a trial simulates to the end of the game. Since in Scrabble, the opponent's tiles are unknown, they need to be generated for every trial in the simulation and used to play out all the candidates moves. The tile generation is constrained by the tiles in the computer's hand and those that have appeared on the board. Maven does not randomly assign seven of the remaining unknown tiles to the opponent. Instead, it tries to match the distribution actually seen in games. To achieve this, it biases its choice to give the opponent a "nice" hand, since strong players like to have a balanced hand with lots of potential.

Opponent modeling is not performed in Maven, since it does not seem to be a critical component in playing strong Scrabble. However, inferences about the opponent's tiles can be done based on previous opponent's moves.

Chapter 7

Conclusions and future work

Using probability triples as Loki-2's betting strategy and as the reweighting factor in its opponent modeling module represents a significant improvement in Loki-2's play against previous versions of Loki in self-play experiments and against human opponents on IRC. Selective sampling simulations show impressive results in self-play experiments. Against human opponents on IRC, the best results were obtained when all three enhancements were used. In self-play experiments, the playing style of the computer players certainly matches the opponents' actions generated inside the simulations. Thus, the simulation-based betting strategy successfully exploits all the weaknesses in the computer opponents' play. In the more realistic environment on IRC, the less predictable approach of the simulation-based Loki-2 paid dividends by making it more difficult for regular opponents to form a correct model of Loki-2's play.

Developing Loki is an iterative process. The work concentrates on improving an aspect of the program until it becomes apparent that another aspect is the main performance bottleneck. That problem is then addressed until it is no longer the limiting factor, and new weaknesses in the program's play are revealed. Loki-1's deterministic betting strategy was its limiting factor. This bottleneck was overcome in two ways. Probability triples provide as a probabilistic representation of betting decisions to increase unpredictability. Simulations add dynamic functionality to static betting strategies. The PT-generation function also supports better use of the information available to the Opponent Modeler, and is more tolerant of the uncertainty in the opponents' actions. However, the opponent modeling still needs to be refined. In fact, it seems that further performance gains will depend on perfecting the opponent modeling module together with improvements to the simulation-based betting strategy.

This thesis presents the first steps in using a simulation-based betting strategy and improving the reweighting process in the Opponent Modeler. These are the initial steps and there are still many to take. Some avenues to explore in Loki-2's future development are:

1. The opponent modeling information can be used to improve the simulations. Currently, the opponent modeling data is used to select the most likely opponents' hands; however, it can also be used to simulate the most likely opponents' actions.

2. Simulations can also improve the opponent modeling. For example, after doing a simulation, the expected reaction for each opponent can be recorded. If their actions frequently differ from what is predicted, then Loki-2 can adjust its opponent model.

3. Loki-2 can easily collect lots of data about the opponent while playing. The problem is filtering and utilizing this data. If these problems are not solved, Loki-2's opponent modeling will be too slow to react or its betting strategy will base its decisions on irrelevant information.

4. Other metrics that may be better predictors of an opponent's style and future behavior have to be considered. For example, measuring the amount of money that a player invests per game may be a good predictor of loose/tight play.

5. Using showdown information to re-play a hand and obtain clues about how an opponent perceived each decision during the hand may help to adaptively measure important characteristics like aggressiveness, bluffing frequency, predictability, affinity for draws and so forth.

6. The employment of learning algorithms in Loki-2's simulation-based strategy and in its Opponent Modeler may help to make inferences based on limited data.

7. Loki-2's preflop behavior can be improved by using a preflop PT-based betting strategy.

As experimental results point out, Loki-2 wins more money (plays better) than last-year's Loki. However, does the program play world-class level poker? It is not there yet, but many improvements are being made to its performance and there are still lots of ideas to try.

Bibliography

[1] Hans Berliner, Gordon Goetsch, Murray Campbell, and Carl Ebeling. Measuring the performance potential of chess programs. *Artificial Intelligence*, 12(1):23–40, 1990.

[2] Darse Billings. Computer poker. Technical Report TR 95–22, Department of Computing Science, University of Alberta, 1995.

[3] Darse Billings, Denis Papp, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron. Using selective-sampling simulations in poker. In *AAAISS-99 Proceedings on search techniques for problem solving under uncertainty and incomplete information*, 1999.

[4] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI–98*, pages 493–499. The MIT Press, 1998.

[5] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane 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.

[6] Darse Billings, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron. Using probabilistic knowledge and simulation to play poker. In *AAAI–99*, 1999. In press.

[7] David Carmel and Shaul Markovitch. Incorporating opponent model into adversary search. In *AAAI–96*, pages 120–125. The MIT Press, 1996.

[8] Steve B. Cousins, William Chen, and Mark E. Frisse. A tutorial introduction to stochastic simulation algorithms for belief networks. *Artificial Intelligence in Medicine*, 5:315–340, 1993.

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

[10] Nicholas V. Findler. Computer poker. *Scientific American*, 239:144–151, July 1978.

[11] Robert Fung and Kuo-Chu Chang. Weighting and integrating evidence for stochastic simulation in bayesian networks. In M. Henrion, R. Shachter, L. N. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 209–220. Elsevier Science Publishers, 1990.

[12] Mathew L. Ginsberg. Partition search. In *AAAI–96*, pages 228–233. The MIT Press, 1996.

[13] Mathew L. Ginsberg, April 13, 1999. Personal communication.

[14] Mathew L. Ginsberg. Gib: Steps toward an expert-level bridge-playing program. In *IJCAI–99*, 1999. In press.

[15] Daphne Koller and Avi Pfeffer. Generating and solving imperfect information games. In *IJCAI–95*, pages 1185–1192, August 1995.

[16] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1–2):167–215, 1997.

[17] Kevin B. Korb, Ann E. Nicholson, and Nathalie Jitnah. Bayesian poker. In *UAI–99*, 1999. In press.

[18] T. Anthony Marsland. Computer chess and search. Technical Report TR 91–10, Department of Computing Science, University of Alberta, 1991.

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

[20] Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach,*. Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-103805-2, 1995.

[21] Jonathan Schaeffer, Darse Billings, Lourdes Peña, and Duane Szafron. Learning to play strong poker. In *ICML–99 Workshop on Machine Learning in Game Playing*, 1999. In press.

[22] Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R. Shachter, L. N. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 221–231. Elsevier Science Publishers, 1990.

[23] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950.

[24] Brian Sheppard, October 23, 1998. Email.

[25] Stephen F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI–83*, pages 422–425, 1983.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, chapter 11. The MIT Press, Cambridge, MA, 1998. http://www-anw.cs.umass.edu/ rich/book/11/node2.html.

[27] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995. http://www.research.ibm.com/massdist/tdl.html.

[28] Alan M. Turing, Christopher Strachey, M. A. Bates, and Bertram V. Bowden. Digital computers applied to games. In B. V. Bowden, editor, *Faster Than Thought*, pages 286–310. Pitman, London, 1953.

[29] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior.* Princeton University Press, Princeton, New Jersey, first edition, 1944.

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

Appendix A

Table of Abbreviations

AAAI American Association for Artificial Intelligence

AI Artificial Intelligence

BBN Bayesian Belief Network

BPP Bayesian Poker Program

EHS Effective Hand Strength

EV Expected Value

GOM Generic Opponent Modeling

HS Hand Strength

IRC Internet Relay Chat

NPOT Negative Potential of a Hand

PPOT Positive Potential of a Hand

PT Probability Triple

rwt Reweight Factor

sb/hand Small bets won per hand

SOM Specific Opponent Modeling

TD Temporal Difference