Pseudo-Optimal Strategies in No-Limit Poker

Rickard Andersson

May 8, 2006
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Michael Minock
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

Games have always been a strong driving force in artificial intelligence. In the last ten years huge improvements have been made in perfect information games like chess and othello and the strongest computer agents can beat the strongest human players. This is not the case for imperfect information games such as poker and bridge where creating an expert computer player has shown to be much harder. Previous research in poker has either adressed fixed-limit poker or simplified variations of poker games. This paper tries to extend known techniqes successfully used in fixed-limit poker to no-limit. No-limit poker increases the size of the game tree dramatically. To reduce the complexity an abstracted model of the game is created. The abstracted model is transformed to a matrix representation. Finding an optimal strategy for the abstracted model is now a minimization problem using linear programming techniques. The result is a set of pseudo-optimal strategies for no-limit Texas Hold'em that perform well as long as the players' stack sizes are fairly small.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The recent poker boom throughout the world has created an increased interest in poker related research. Just as with chess, checkers, go and many other games, poker has forced researchers to venture into areas of artificial intelligence that have not been examined in depth before. Poker is a game of imperfect information, the opponent's cards are hidden so the complete state of the game is not known. With the exception of the game go there exist today strong programs for most games of perfect information (the exact state of the game is known). The prime example is chess and the best computer programs are regularly able to beat the best humans in the world. This is not the case for imperfect information games such as poker. The technique of brute-force searching the game tree used in perfect information games do not work as we do not know exactly where in the game tree we are.

This paper adresses the task of finding strong strategies for heads-up no-limit poker. From a game theoretic point of view, heads-up no-limit poker can be described as a two player zero-sum game of imperfect information and perfect recall. A zero-sum game is a game where the total amount the players win or loose is zero. It means that what one player wins another player has to loose. The players' intentions are in direct opposition to each other. The notion of imperfect information has already been explained above. Perfect recall means that the players remembers everything that has happened in the game so far. Games of imperfect recall are not common in the real world and will not be discussed much further.

When dealing with strategies in imperfect information games the notion of pure and mixed strategies and also optimal and maximal strategies are very important. A pure strategy is a strategy that do the same action every time it is in the same exact situation. A mixed strategy is a strategy that uses some randomization to determine what to do. A human poker player adapts to his opponent and tries to exploit the opponent's mistakes in order to get greatest possible edge. This is in essence a maximal strategy. A maximal strategy's goal is to win as much as possible against a specific strategy. An optimal strategy's goal on the other hand is to loose the least against any arbitrary strategy. It is a strategy that is impossible to exploit. An expert human player who plays against an

unknown opponent generally starts out by playing something very close to an optimal strategy. When he gets a feel for the strategy of the opponent he adjusts his strategy to exploit any weaknesses and it becomes more of a maximal strategy. Another term for an optimal strategy is a Nash equilibria. If two optimal strategies plays aginst each other it is said that a Nash equilibrium is reached. Finding a Nash equilibrium of a game is the same as solving the game.

1.1 The Rules of No-Limit Texas Hold'em

The game used in this thesis will be the two-player version of No-Limit Texas Hold'em. Each player starts with an amount of **playing chips**. The amount of chips is called the **stacksize** of each player. Texas Hold'em is divided in four stages or **streets**. At each street the players get some new information (more cards) and then proceed with a betting round where they can wager chips that their combination of cards, their **hand**, is better than their opponent's. All the chips wagered so far in a game is called the **pot**. The possible actions in a betting round are:

Fold The opponent has wagered an amount of chips the player does not want to wager so he folds. The opponent wins everything in the pot and the game ends.

Bet When no one has bet anything previously in the betting round the player can wage that his hand is the best. The size of the bet is only limited by the player's stacksize. When the player has bet the opponent can either fold, call, or raise.

Raise When the opponent has already made a bet or a raise the player can raise by first matching the amount the opponent has wagered and then increase it further. The size of the raise is only limited by the player's stacksize. When the player has raised the opponent can either fold, call, or raise.

Call If the opponent has bet or raised the player can call and thereby matches the amount wagered and the betting round ends.

Check If the player is first to act and checks it becomes the opponents turn to act. If the opponent has already checked and the player checks the betting round ends.

At the start of the game the first player called the **small blind** puts a predetermined amount in the pot. The next player, the **big blind**, puts twice the amount of the small blind in the pot. The first stage is called **preflop**. Each of the players is dealt two cards face down and the betting round starts with the small blind first to act. As the small blind has only wagered half of what the big blind has done, he either has to fold, call or raise. If he folds the game ends and if he calls or raises it is the big blind's turn to act.

The second stage is called the **flop**. Three cards are dealt face up that both players can use. The betting round starts with the big blind first to act. He can either fold, check, or bet.

The third stage is called the **turn**. One card is dealt face up and added to the three cards already dealt. The betting round has the same structure as the one on the flop.

The fourth and final stage is called the **river**. A fifth card is dealt face up followed by a betting round exactly as on the turn. When the betting round ends and no one of the players have folded the **showdown** is reached. Both players show their two face down cards. The player with the strongest five card combination of his two private cards and the five cards on the table wins the pot and the game ends. In the next game the players switch places. The player who was the small blind is now the big blind and vice versa.

1.2 Thesis Contribution

Several methods from simple rule based behavior to advanced opponent modeling using neural networks have been suggested to create strong strategies in imperfect information games. This paper will take an approach that was originally done by Billings et al. [1] for fixed-limit Texas Hold'em but modify the techniques to fit no-limit Texas Hold'em. The idea is to transform the original game to an abstracted mathematical model of it. This new model can be transformed to a matrix of linear equations. The matrix can either be in the normal form or the sequence form. The normal form was for a long time the standard way of describing a game but the size grows exponentially with the size of the game tree. The fact that the game tree is exponential in itself, the normal form matrix is exponential in the size of the game tree and the algorithms for solving the final model are polynomial in size of the matrix, makes the extensive form very ineffective and can only be used to solve very small games. The sequence form creates a more compact matrix that is linear in size of the game tree and allows an expontentially larger set of games to be solved. A solution to the abstracted game is found by solving the matrix of linear equations with some linear programming software. The resulting strategy is not truly optimal because it is based on a model that is not exactly like the real game. It is thus called a pseudo-optimal strategy.

Previous research on optimal or pseudo-optimal strategies has either adressed simplified variations of poker games or fixed-limit poker. Fixed-limit means that the amount a player can bet or raise is always fixed. In no-limit the players can choose how much they want to bet and raise. This addition of choices for the players increases the size of the game tree dramatically. This paper tries to extend known techniqes successfully used in fixed-limit poker to no-limit. Three abstraction techniques successfully used in previous research for fixed-limit are being modified to work for no-limit and a new fourth abstraction technique specifically for no-limit is designed.

In the next chapter previous research in the field will be presented. Chapter 3 discusses different ways to represent a game. A game can be described by rules, by a game tree, by a matrix. The logic behind the normal form and the sequence form which are both matrix representations is explained. Finally an example is shown how to transform a game from the game tree representation to the sequence form. Chapter 4 explains the four abstraction techniques used to simplify Texas Hold'em to a manageable complexity. The simplified game is transformed to the sequence form and solved using linear programming software to get a pseudo-optimal strategy. Chapter 5 discusses issues with measuring the strength of the found strategy and also presents the test results. Chapter 6 has a discussion on the results and some observations on the project as a whole. Chapter 7 gives some ideas on how to improve the model built in this thesis

and how to to possibly find optimal strategies using other approaches.

Chapter 2

Previous Research

Game theory provides us with the tools for studying strategic environments where more then one agent act and the outcomes of an agent's actions are dependent on the other agents' actions. The foundations of game theory were built by Neumann and Morgenstern in 1944 with their book Theory of Games and Economic Behavior [9]. They also proved that there always exists an equilibrium in two-person zero-sum games. A few years later John Nash extended this and proved that there exists an equilibrium in every n-person game where the players act in their best self interest [11].

The standard way of finding an equilibrium is to create a matrix in normal form of the game and then solve it using linear programming techniques. The normal form is exponential in size of the game tree and only small games are therefore solvable with this approach. In 1994 the sequence form was introduced by Koller et al. [10] which is only linear in size of the game tree. The sequence form made it possible to solve mush larger models than was previously feasible.

When online poker started to become popular around the year 2000 it became more interesting to create strong computer players for real poker games. Shi and Littman [14] discussed abstraction techniques that was able to drastically reduce the complexity of the games. Using the sequence form and several abstractions the first approximation of optimal strategies for a real poker game was made by Billings et al. in 2003 [1].

Instead of trying to find optimal strategies another approach is to use opponent modeling to exploit the opponent's weaknesses. The poker research group at university of Alberta has described many techniques in this area [2], [6], [5].

In 2001 a poker game called Rhode Island Poker was specifically designed for artificial intelligence research [14]. It was less complex than real poker games which made it easier to analyze, but with more than 3.1 billion nodes in the game tree [8] it was still much larger than any game that had been solved previously. This changed when an algorithm called GameShrink was presented in 2005 [7]. Unlike many other abstraction techniques GameShrink abstracts a game while preserving equilibrium and makes it possible to

solve games much larger than was previously possible. Using the GameShrink algorithm Rhode Island Poker was finally solved.

There has not been much research in no-limit poker. One reason is probably that fixed-limit poker is less complex and still offers many problems that are not solved yet. The few papers that discuss the no-limit structure use a very theoretical approach where for example the players are given a randomized number between 0 and 1 instead of real cards. Ferguson and Ferguson [4] discusses two of these models [9], [3].

Chapter 3

Game Theory

This chapter will discuss different ways of representing a game. A game can be described either by its rules, by a tree, or by a matrix. To calculate game theoretic optimal strategies in games with imperfect information prior research has shown that the matrix form(s) are clearly superior. Proofs and examples of the various representations and algorithms to produce optimal strategies will be given.

3.1 Description by Rules

By giving a set of rules that can be applied for every situation in the game it is possible to describe a game completely. The rule based approach is probably the most intuitive way to describe a game and the way humans would normally use as it often is possible to describe a complex game with a simple set of rules. Unfortunately using rules to describe games is often impractical in computer science. First of all it is complicated to create an environment that can take any rule or even meta-rules as input. The fact that the number of rules does not say anything about the size of the game tree (it could even be infinite!) is also an indication that rules might not be the best way when trying to find optimal strategies.

3.2 The Extensive Form

The extensive form is commonly used in perfect information games such as Chess, Checkers, and Othello. It is a tree structure with a node for each possible state in the game. The root is the start of the game and each leaf node correspond to an ending of the game. The leaf nodes have two numbers associated with them that describes the value that ending has for each player. In a zero-sum game there is only one number at each leaf node describing the outcome for one of the players. The outcome for the other

player is simply the negative of that number. The internal nodes can either be a chance node (such as rolling a dice) or a decision node (one player can choose between a set of options). These nodes are called public when all players can see the outcome from that situation and private when only a subset of all the players can see the result. Games with no private nodes are called perfect information games and means that all players have the exact same information about the state of the game. Chess and Connect4 are both examples of perfect information games.

Strong ai players for many perfect information games can be created by using backward induction to find a good move from a game state. The best chess programs today for example uses among other techniques MiniMax search with AlphaBeta pruning together with some heuristics.

In imperfect information games backward induction is not possible. We cannot calculate the expected value of a move if we do not know exactly what state the game is in. Examples of these kind of games are Scrabble, Bridge and Poker. But as an example we will take maybe the simplest game of them all: Rock-Paper-Scissors (called RPS from this point on). RPS is a game where two players simultaneously chooses to make a figure of a rock, paper, or scissors with their hands. Rock beats scissors, scissors beat paper, and paper beats rock. If both players chooses the same figure it is a tie. Figure 3.1 shows the RPS game in extensive form.

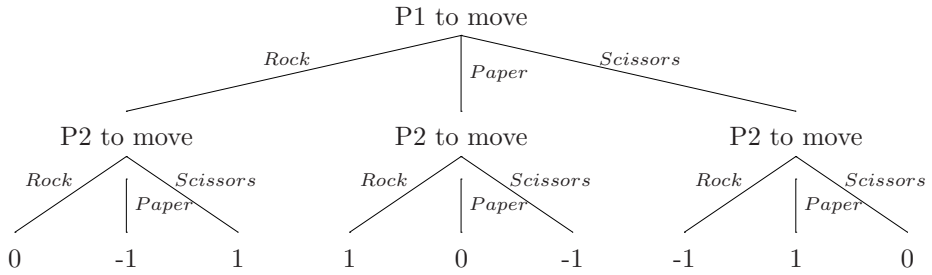

Figure 3.1: The game Rock, Paper, Scissors in extensive form. The outcome of the game at the leaf nodes are from P1's point of view.

In reality the players make their moves simultaneously but here the game is changed so that P1 first makes a move but do not show it to P2. The root node is a private decision node to P1. So when it is P2's turn to act he do not know in which of the three states the game is in. When P2 has made a move both players show their moves and the outcome is decided. The leaf nodes describes the outcome from P1's perspective. The outcome for P2 is the negative of the leaf node's value. In this imperfect information game backward induction techniques such as MiniMax search are clearly not possible, other approaches are needed. We will see that transforming the game tree to normal form will allow us to find a nash equilibrium to the game.

3.3 The Normal Form

To understand the normal form we need a few more definitions. A strategy s is a set of rules that determines which action an agent will take for every possible game state the can arise from s. A pure strategy is a strategy where the agent will act the same every time the exact same situation occurs. A pure strategy is deterministic. In the opposite, a mixed strategy, the agent assign a specified probability to every specific situation. An example of a mixed strategy in RPS would be to play rock 40% of the time, paper 10% and scissors 50% of the time. Playing rock 100% of the time would be a pure strategy.

The normal form is a "payoff matrix" that lists the outcome of a game for each possible combination of pure strategies for the players. From this point forward, we will only consider two-persons zero-sum games with the players called Max and Min. Max want to maximize the outcome of the game and Min wants to minimize it. The payoff matrix P is of size $m \times n$ where m and n is the number of pure strategies that is available to Max and Min. The value $P_{ab}(1 \leq a \leq m, 1 \leq b \leq n)$ is the average outcome of the game when Max plays the pure strategy a and Min plays the pure strategy b. Figure 3.2 shows the normal form for the RPS game.

		Min's strategies		
		rock	scissors	paper
	rock	0	1	−1
Max's strategies	scissors	−1	0	1
	paper	1	−1	0

Figure 3.2: The RPS game in normal form.

Given the payoff matrix for the game we can find an optimal mixed strategy by transforming the payoff matrix to a corresponding system of linear equations. The first thing we need to do is to make sure all entries in the payoff matrix are positive. It is possible to achieve this by adding a fixed constant to every entry in the matrix without altering the structure of the game. The reason for this transformation will be clear later on. Our new payoff matrix looks like this.

$$\begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \tag{3.1}$$

We start by noting that every mixed strategy can be represented by a probability distribution on all pure strategies available. Let the vectors s^{1*} and s^{2*} be optimal strategies for Max and Min. s^{1*}_k is the probability Max will play the pure strategy k. A strategy s is only valid if the probabilities for each pure strategy sums to 1.

$$\sum_k s_k = 1 \tag{3.2}$$

As both s^{1*} and s^{2*} are optimal the *expected value* of the game e is defined by:

$$e = {s^{1*}}^T \times P \times s^{2*} \tag{3.3}$$

The *expected value* e is the average value Max will gain from the game when playing it one time. We will try to find an optimal strategy for $Min : s^{2*} = \begin{pmatrix} \text{u} & \text{v} & \text{w} \end{pmatrix}^T$. An optimal strategy s^{2*} for Min is a strategy that minimizes the maximum of e against any of Max's pure strategies. Max can never archieve a higher e with a mixed strategy than with the best pure strategy against s^{2*}, so we do not need the consider $Max's$ mixed strategies at the moment. The game's outcome against a not necessarily optimal strategy s^1 is less than or equal the *expected value* of the game.

$${s^1}^T \times \begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} \text{u} \\ \text{v} \\ \text{w} \end{pmatrix} \leq e \tag{3.4}$$

For every pure strategy for Max this gives us:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T \times \begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} \text{u} \\ \text{v} \\ \text{w} \end{pmatrix} \leq e \implies 2u + 3v + w \leq e \tag{3.5}$$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}^T \times \begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} \text{u} \\ \text{v} \\ \text{w} \end{pmatrix} \leq e \implies u + 2v + 3w \leq e \tag{3.6}$$

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}^T \times \begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} \text{u} \\ \text{v} \\ \text{w} \end{pmatrix} \leq e \implies 3u + v + 2w \leq e \tag{3.7}$$

Divide by e.

$$\frac{2u}{e} + \frac{3v}{e} + \frac{w}{e} \leq 1 \tag{3.8}$$

$$\frac{u}{e} + \frac{2v}{e} + \frac{3w}{e} \leq 1 \tag{3.9}$$

$$\frac{3u}{e} + \frac{v}{e} + \frac{2w}{e} \leq 1 \tag{3.10}$$

Define i, j, k as

$$i = \frac{u}{e}, j = \frac{v}{e}, k = \frac{w}{e} \tag{3.11}$$

We know that $u \geq 0, v \geq 0, w \geq 0$ and from 3.1 that $e > 0$ (we made every entry in the payoff matrix ≥ 0). That gives us the following constraints.

$$i \geq 0, j \geq 0, k \geq 0 \tag{3.12}$$

Combine 3.8, 3.9, 3.10, and 3.11

$$\begin{array}{rcrcrcl} 2i & + & 3j & + & 1k & \leq & 1 \\ i & + & 2j & + & 3k & \leq & 1 \\ 3i & + & j & + & 2k & \leq & 1 \end{array} \tag{3.13}$$

Combine 3.2 and 3.11

$$\frac{u}{e} + \frac{v}{e} + \frac{w}{e} = \frac{1}{e} \Longrightarrow i + j + k = \frac{1}{e} \tag{3.14}$$

If we would maximize $i+j+k$ that would minimize e which is exactly what Min wants to do to find an optimal strategy. By combining this problem with the constraints from 3.12 and 3.13 we get a linear programming problem.

$$\begin{array}{ll} \text{Maximize} & p = i + j + k \\ & \\ & \begin{array}{rcrcrcl} 2i & + & 3j & + & 1k & \leq & 1 \\ i & + & 2j & + & 3k & \leq & 1 \\ 3i & + & j & + & 2k & \leq & 1 \end{array} \\ \text{subject to} & \\ & i \geq 0, j \geq 0, k \geq 0 \end{array} \tag{3.15}$$

This problem can now be solved with standard linear programming algorithms such as the "Simplex Method" [16]. In a correct solution to 3.15 $\frac{1}{p}$ equals e. This equality can be used to find an optimal strategy for Min:

$$s^{2*} = \begin{pmatrix} \text{u} & \text{v} & \text{w} \end{pmatrix}^T = \begin{pmatrix} \frac{i}{p} & \frac{j}{p} & \frac{k}{p} \end{pmatrix}^T. \tag{3.16}$$

Solving these equations for the RPS game yields the solution:

$$u = \frac{1}{3}, v = \frac{1}{3}, w = \frac{1}{3} \tag{3.17}$$

It should be no surprise that the optimal strategy in RPS is to completely randomize between the three different choices. Using this strategy the expected value of the game is zero and it is not possible for the opponent the exploit the strategy in any way.

3.4 The Sequence Form

The problem with the Normal form is that the payoff matrix generally is exponential in size of the game tree. In the sequence form the payoff matrix is only linear in size of the game tree and is therefore able to solve a much larger group of games [10]. In this section a proof of why the sequence form works will be developed. In the next section a more complex game than RPS will be used as an example when we transform a game from the extensive form to the sequence form and explain exactly the structure of the matrices needed.

To find a way to calculate an optimal strategy from a game in sequence form we start by trying to find a solution to a more simple problem. How do we find the best counter

strategy to an arbitrary mixed strategy? Given $Max's$ strategy s^1 of size $1 \times m$, a payoff matrix P from $Max's$ perspective of size $m \times n$ and a number of linear constraints that needs to be satisfied for s^2 to be a valid strategy, we can define the problem as a linear programming problem where $Min's$ best counterstrategy s^2 of size $1 \times n$ is the solution to:

$$\begin{array}{ll} \underset{s^2}{\text{Maximize}} & (s^{1^T}(-P))s^2 \\ \text{subject to} & \begin{array}{c} Cs^2 = c \\ s^2 \geq 0 \end{array} \end{array} \tag{3.18}$$

$Cs^2 = c$ is the constraint(s) that needs to satisfied for s^2 to be a valid strategy. In the normal form case s^2 needs to be a probability distribution and C is a matrix of size $1 \times n$ filled with ones and c is a matrix of size 1×1 with value one. In the sequence form C and c are different and will be explained in the next section. For now, just consider them as a set of equations that need to be true for s^2 to be a valid strategy.

From optimization theory we know that every linear programming problem has a complementary problem called the dual problem. The dual [16] of problem 3.18 is by definition

$$\begin{array}{ll} \underset{t}{\text{Minimize}} & t^T c \\ \text{subject to} & t^T C \geq s^{1^T}(-P) \end{array} \tag{3.19}$$

The new variable t is matrix with the same number of rows as C and the same number of columns as c. The strong duality theorem [16] states that the optimal objective functions of the primal and the dual problem are equal:

$$(s^{1^T}(-P))s^2 = t^T c \tag{3.20}$$

We now repeat the process of 3.18 3.19 3.20 but this time we try to find $Max's$ best counter strategy s^1 against $Min's$ fixed mixed strategy s^2.

$$\begin{array}{ll} \underset{s^1}{\text{Maximize}} & s^{1^T}((P)s^2) \\ \text{subject to} & \begin{array}{c} s^{1^T} D^T = d^T \\ s^1 \geq 0 \end{array} \end{array} \tag{3.21}$$

We use r as the new matrix in the dual:

$$\begin{array}{ll} \underset{r}{\text{Minimize}} & d^T r \\ \text{subject to} & D^T r \geq P s^2 \end{array} \tag{3.22}$$

And again by the strong duality theorem:

$$s^{1^T}((P)s^2) = d^T r \tag{3.23}$$

We know that the optimal solution to 3.21 is the value Min has to give to Max if Max plays a perfect counter strategy. Min wants to play a strategy s^2 that minimizes this value. We also know that 3.21's optimal solution equals 3.22's optimal solution. So Min

wants to play the s^2 that minimizes 3.22. If we add the constraints we have on s^2 we get the equation:

$$\begin{array}{ll} \underset{s^2, r}{\text{Minimize}} & d^T r \\ & D^T r \geq P s^2 \\ \text{subject to} & C s^2 = c \\ & s^2 \geq 0 \end{array} \tag{3.24}$$

The same reasoning can be used to find an optimal strategy for Max. 3.18 is the value Max has to give to Min by playing s^1. 3.18 equals 3.19 so Max wants to minimize 3.19 by playing an optimal strategy s^1. Adding the constraints we have for s^1 to 3.19 we get:

$$\begin{array}{ll} \underset{s^1, t}{\text{Minimize}} & t^T c \\ & t^T C \geq {s^1}^T (-P) \\ \text{subject to} & {s^1}^T D^T = d^T \\ & s^1 \geq 0 \end{array} \tag{3.25}$$

By solving equation 3.24 and 3.25 with a linear programming problem algorithm such as the "Simplex Method" we obtain a pair of optimal strategies for Min and Max. Observe that this proof is only valid if it is a zero-sum game ($Max's$ payoff matrix P is the negative of $Min's$ payoff matrix). If that is not the case the final equation will not be a linear programming problem but a linear complementary problem, but that is outside the scope of this paper.

3.5 Creating a Game in Sequence Form

To understand the sequence form we need to introduce the notion of information sets. Consider a game of cards where you know what cards you have but you do not know what cards your opponent has. Because you do not know what your opponent has you do not know exactly where in the game tree you are. All the states in the game tree that you might possibly be in belongs to the same information set. A player has no way of distinguishing between two states in the same information set.

In this section we give an example of how to transform a game from the extensive form to the sequence form and solve it. Figure 3.3 shows the game tree of the game we will use as an example. At the start of the game there is a private chance node to P1. Only P1 knows the outcome of that node. P2 is the first player to act and does not know in which of the to states the game is in. After that it is P1's turn to act and then as long as P1 does not play action l or p P2 makes the final action. At the last move P2 still do not know the outcome of the chance node in the beginning and hence the information sets. The leaf nodes show the value of the end position from P1's perspective. P1's goal is to maximize this value and P2's to minimize it.

We saw that a strategy S in the normal form was a probability distribution over all possible pure strategies. A strategy s in the sequence form could be said to denote probability distributions over all the actions (sequences) for each of the game's information sets, that is, at what percentage should a player take a specific action in each

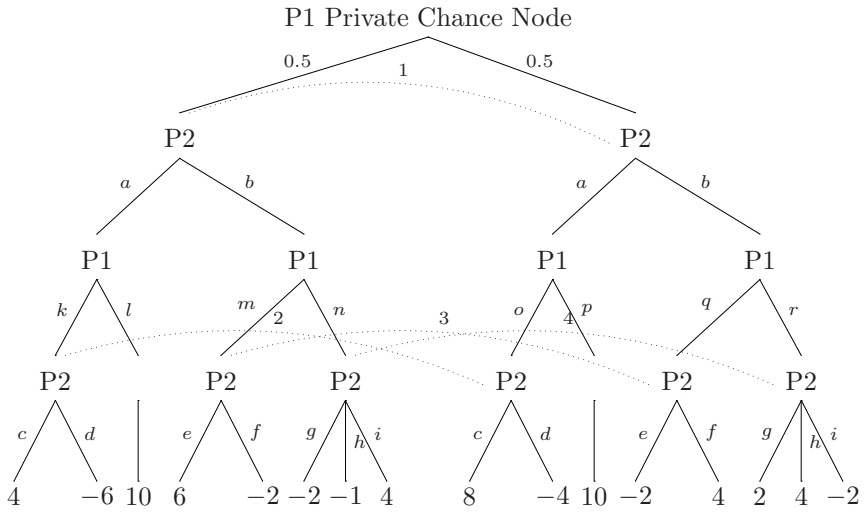

Figure 3.3: A more complex game in the extensive form. The dotted lines shows the information sets for P2.

information set. An entry P_{kb} in the payoff matrix P in the sequence form is the outcome of the game when Max plays sequence k and Min plays sequence b multiplied by the probability that Max and Min get a chance to play k and b. Given this definition we can create the payoff matrix for the game in our example (3.4).

Left to define is the strategy constraint equations ${s^1}^T D^T = d^T$ and $Cs^2 = c$ that s^1 and s^2 need to satisfy to be a valid strategies. Let us start with the strategies for P1 and create matrix D. Each row denotes an information set. The first row is the start of the game. P1 is always forced to play sequence θ (the starting position of the game) 100% of the time so we put a 1 there. The next row is the information set where the the outcome of the private chance node was the left side in figure 3.3 and P2 played sequence a. P1 can either play k or l. For s^1 to be a valid strategy the sum of the probability to play k and l needs to be equal to the probability to play the "parent sequence", namely θ. So on this row we put a -1 under θ and and a 1 under k and l. We do the same steps for the three other of P1's information sets. The result is the the matrix in figure 3.5. d^T will always have the same format independent of the game. The first entry is a one and the rest are zero $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}$. The one forces a valid strategy to always choose the inital state of the game 100% of the time. The zeroes force the sum of the percentages for all the child sequences to be equal to the percentage of the parent sequence (see figure 3.7).

Now we repeat the same procedure for P2. There exist four information sets if we do not count the initial position of the game. The first is after P1's chance node and P2 has two choices here, a or b. At row 1 in the constraint matrix we put a 1 under a and b and a -1 under the parent sequence θ. If P2 chooses the a he might reach information set 2 and have to do another choice between c and d. At row 2 we put a 1 under c and d and a -1 under the parent sequence a. Information set 3 and 4 are also added. Finally we get the matrix in figure 3.6. c is similar to d^T, a leading one and trailing zeroes.

					Sequence($P2$)					
	θ	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
Sequence($P1$)										
θ	0	0	0	0	0	0	0	0	0	0
k	0	0	0	2	−3	0	0	0	0	0
l	0	5	0	0	0	0	0	0	0	0
m	0	0	0	0	0	3	−1	0	0	0
n	0	0	0	0	0	0	0	−1	−0.5	2
o	0	0	0	4	−2	0	0	0	0	0
p	0	5	0	0	0	0	0	0	0	0
q	0	0	0	0	0	−1	2	0	0	0
r	0	0	0	0	0	0	0	1	2	−1

Figure 3.4: Payoff matrix P from P1's perspective. Sequence 0 denotes the start of the game for both players.

	Sequence								
	θ	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**
$InfoSet$									
0	1	0	0	0	0	0	0	0	0
1	−1	1	1	0	0	0	0	0	0
2	−1	0	0	1	1	0	0	0	0
3	−1	0	0	0	0	1	1	0	0
4	−1	0	0	0	0	0	0	1	1

Figure 3.5: P1's strategy constraint matrix D.

	Sequence									
	θ	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
$InfoSet$										
0	1	0	0	0	0	0	0	0	0	0
1	−1	1	1	0	0	0	0	0	0	0
2	0	−1	0	1	1	0	0	0	0	0
3	0	0	−1	0	0	1	1	0	0	0
4	0	0	−1	0	0	0	0	1	1	1

Figure 3.6: P2's strategy constraint matrix C.

We now know the structure of every matrix in equation 3.24. With some reorganization we get:

$$\begin{array}{lll} \text{Minimize} & d^T r & \\ s^2, r & & \\ & -Ps^2 + D^T r \geq 0 & (3.26) \\ \text{subject to} & Cs^2 = c & \\ & s^2 \geq 0 & \end{array}$$

The complete linear problem looks like:

Minimize r_0 with the following constraints:

$$\begin{array}{cccccccccccccccccc}
 & & & & & & & & & +r_0 & -r_1 & -r_2 & -r_3 & -r_4 & \geq & 0 \\
 & & -2s_c^2 & 3s_d^2 & & & & & & & +r_1 & & & & \geq & 0 \\
 & -5s_a^2 & & & & & & & & & +r_1 & & & & \geq & 0 \\
 & & & & -3s_e^2 & +s_f^2 & & & & & & +r_2 & & & \geq & 0 \\
 & & & & & & s_g^2 & 0.5s_h^2 & -2s_i^2 & & & +r_2 & & & \geq & 0 \\
 & & -4s_c^2 & 2s_d^2 & & & & & & & & & +r_3 & & \geq & 0 \\
 & -5s_a^2 & & & & & & & & & & & +r_3 & & \geq & 0 \\
 & & & & s_e^2 & -2s_f^2 & & & & & & & & +r_4 & \geq & 0 \\
 & & & & & & -s_g^2 & -2s_h^2 & s_i^2 & & & & & +r_4 & \geq & 0 \\
+s_0^2 & & & & & & & & & & & & & & = & 1 \\
-s_0^2 & +s_a^2 & +s_b^2 & & & & & & & & & & & & = & 0 \\
 & -s_a^2 & & +s_c^2 & +s_d^2 & & & & & & & & & & = & 0 \\
 & & -s_b^2 & & & +s_e^2 & +s_f^2 & & & & & & & & = & 0 \\
 & & -s_b^2 & & & & +s_g^2 & +s_h^2 & +s_i^2 & & & & & & = & 0 \\
+s_0^2 & & & & & & & & & & & & & & \geq & 0 \\
 & +s_a^2 & & & & & & & & & & & & & \geq & 0 \\
 & & +s_b^2 & & & & & & & & & & & & \geq & 0 \\
 & & & +s_c^2 & & & & & & & & & & & \geq & 0 \\
 & & & & +s_d^2 & & & & & & & & & & \geq & 0 \\
 & & & & & +s_e^2 & & & & & & & & & \geq & 0 \\
 & & & & & & +s_f^2 & & & & & & & & \geq & 0 \\
 & & & & & & & +s_g^2 & & & & & & & \geq & 0 \\
 & & & & & & & & +s_h^2 & & & & & \geq & 0 \\
 & & & & & & & & & +s_i^2 & & & & & \geq & 0
\end{array}$$

Figure 3.7: The complete linear programming problem in sequence form for P2.

Solving 3.7 yields the solution:

$$
\begin{aligned}
s_0^2 &= 1 \\
s_a^2 &= 0 \\
s_b^2 &= 1 \\
s_c^2 &= 0 \\
s_d^2 &= 0 \\
s_e^2 &= 0.25 \\
s_f^2 &= 0.75 \\
s_g^2 &= 0.09375 \\
s_h^2 &= 0.6875 \\
s_i^2 &= 0.21875
\end{aligned}
$$

Figure 3.8: Optimal strategy for P2

According to figure 3.8 P2 should always choose sequence b in information set 1. Because of this information set 2 will never be reached and the probabilities for c and d are therefore 0. In information set 3 P2 should choose sequence e 25% of the time and f 75%. In information set 4 the probability distribution for g, h and i are 9.375%, 68.75% and 21.875%. As long as P2 uses this mixed strategy it is impossible for P1 to exploit P2's strategy.

Chapter 4

Abstractions

The game tree for limit Texas Hold'em is in the size of $O(10^{18})$ [1] and the no-limit version is even more complex with large stacks. Because of this complexity it is doubtful we will ever develop the techniques and hardware to analyze the game completely. To reduce the complexity of the game, researchers have come up with different ways of creating abstractions that makes the game more simple to analyze. The problem is to create abstractions that are strategically as close as possible to the real game.

In this section we will discuss the properties of abstractions such as bucketing, betting round reduction, independent betting rounds, and discreetization of betting amounts. All of them but the last have been used successfully previously in fixed-limit poker. Discretization of betting amounts is a way to handle the extra complexity that no-limit brings.

4.1 Bucketing

Bucketing is a commonly used technique to drastically simplify games ([14] [1] [15]). The idea is to partition the possible hands in buckets (sometimes called groups or bins). This is very close to how a human would reason about a hand. It does not really matter much if a human get K2 offsuit or K3 offsuit, both hands are in the bucket "a king and a low card". When the buckets are created the game is solved on the bucket level instead of the card level. With six buckets on each betting round in limit Hold'em the game would be reduced from $O(10^{18})$ to $O(10^{10})$ [1].

How to assign the hands to the different buckets is not a trivial task. Preferably we want hands with the approximate same strategic properties to end up in the same bucket. A simple approach would be to partition the hands based on how likely they are to win at showdown against a random hand, or how likely they are to be ahead against a random hand at the current state of the game. This is a one dimensional solution to a multidimensional problem. A hand can not be categorized completely by only one value.

Consider the hand a "pair of sevens" and the hand "ace king" for example. Both hands are about equally likely to win at showdown against a random hand, but strategically they play very different. The hand with the sevens becomes most of the time a medium strength hand on the flop. A small percentage of the time the sevens become very str ong when the flop contains another seven. The "ace king" hand becomes either fairly weak or fairly strong depending on if the flop contains a king or an ace. It is not possible to model the exact the properties of a hand so compromises need to be made. [1] created a very strong player by allocating $n-1$ buckets to rollout hand strength and one bucket to hands that are weak but have the potential to become very strong in future betting rounds.

When it has been established how to create the buckets the transitional probabilities needs to be generated. That is the probability to move from each bucket on one betting round to each bucket on the next betting round. To find the probability for every transition either sampling (fast and inaccurate) or enumeration (slow and accurate) could be used. When the transitional probabilities are finished we have enough information to build a model of the game and solve it to get a pseudo optimal strategy for our real game.

Bucketing is a very powerful abstraction in the sense that it is easy to change the size of the game tree depending on the resources available. The drawbacks are the properties of the real game that are lost and will not be taken account for in the resulting strategy. An example would be if all hands with an ace was grouped in one bucket and the optimal strategy found would always raise a certain amount with the hands in this bucket, a smart human player would recognize this and exploit it by betting on flops without and ace and folding on flops with an ace. On the bucket level it is not possible to see if a flop contains an ace so the computer will not be able to defend against this strategy.

4.2 Reduction of Betting Rounds

Reducing a betting round consists of removing possible actions for one or more players in the betting round. To make the abstraction effective, actions that are removed should either almost always be bad (e.g. folding to a third raise preflop in limit hold'em), or almost never occur (e.g. make more then 3 raises in a betting round). Billings et al. [1] experimented with different reductions for limit poker and found that only allowing three bets per round instead of four "does not appear to have a substantial effect on play, or on the expected value (EV) for each player".

4.3 Independent Betting Rounds

By considering each betting round as independent from the others we get a few subproblems which are smaller and faster to solve than the original problem [14]. The level of independence can be varied and the more independent the smaller the model and the more strategic properties are lost.

To make all betting rounds completely independent from each other will create very weak strategies. We will take an approach that is similar to the one Billinges et al.[1] used for fixed limit poker. We create a preflop model that consists of all but the last betting round and solve that as a first step. The next step is to create a number of postflop models that starts at the flop and ends at the river (the last betting round). One postflop model is created for each possible sequence of actions during the preflop stage. Now we can look at the solutions of the preflop stage to approximate how strong each player's hand is depending on how they acted preflop. These approximations is used to define how likely each player is to end up in each bucket on the flop in the postflop model. As our final strategy we use the solution to the preflop model only at the first betting round and, depending on the preflop action, the solution to the corresponding postflop model at the rest of the betting rounds.

One problem is that an optimal solution from the preflop model might never do certain actions at the preflop stage. An example would be if the optimal solution never raises two times the big blind, then we can not approximate an opponents hand strength if he raises two times the big blind. The solution is to solve the preflop model two times. The first time we modify the rules of the game so that each player is forced to do each preflop action a small percentage of the time. Now we can use this solution the for the approximations of the postflop models. We then solve the preflop model one more time without the constraints and use that solution as our actual strategy.

It is hard to determine how accurate this abstraction is. It is somewhat similar to transforming the game to an imperfect recall game in which the players forget what bucket they had during the preflop stage.

4.4 Discretization of Betting Amounts

In no-limit poker the players can choose how much they want to bet[1]. It is not practical to model every possible action for each player as the game tree would grow exponentially in terms of the stacksize of the players. One way to create a simplified abstraction is to create a discretisation of the betting amounts.

In this application the betting choices was discretized to four different categories all relative to the amount that is already in the pot. A small bet is half the pot, a normal bet is the size of the pot and a large bet is twice the size of the pot. The forth action is all-in where the player puts all the chips he has left into the pot. With only four different amounts that can be bet the game tree is dramatically reduced but the complexity still depends on how many chips the players have at the beginning of the game. One drawback with this approach is that it requires some understanding of the game as the programmer has to know which betting amounts are common in the game. The model will not for example be able to find optimal strategies that requires a player to only bet one third of the pot. Another problem is that even if the agent only bets one of the four amounts it is not sure that the opponent does it. We need a way to handle cases

[1] If a bet or raise has already been made in the current betting round and the player wants to raise he has to raise with an amount at least equal to the last bet/raise.

when the opponent bets only one fourth of the pot or when it bets three times the pot. The naive approach of rounding off the bet to the nearest discretised choice is easily exploitable. It will be very profitable for the opponent to bet 30% of the pot as he only invests 30% but the bet will be treated as a 50% bet by the agent. To counter this tactic randomization is used. If the opponent's bet is between two discretised betting amounts the chance the action will be represented as each of the actions is inversely linearly dependent on how close it is to that action. If the pot is 10 and the opponents bets 3 it is a 40% chance the action will be represented as a check and a 60% chance it will be represented as a small bet internally by the agent. Intuitively the opponent cannot exploit this method easily, but there exists a risk for compounding errors where the agent makes several randomizations and end up with an internal representation very different from what has actually happened.

Chapter 5

Simulations

5.1 Measuring Performance

There are several aspects that need to be kept in mind when discussing the performance of a specific strategy in a game like poker. First of all, the performance of strategies are not transitive. Second, there are fundamental differences between maximal strategies and optimal strategies. Finally, it can be very time consuming to get a good approximation of how two strategies do against each other because of the high variance in poker.

5.1.1 Optimal vs. Maximal Strategies

Optimal and maximal strategies are very different in their goals. Optimal strategies try to minimize their losses against an arbitrary strategy. A perfect optimal strategy is impossible to exploit in any way and called a Nash Equilibrium. In poker we do not have the techniques nor the hardware to be able to calculate a perfect optimal strategy. The approach used is to solve a simplified game of poker and get a pseudo-optimal strategy. It is optimal but only to the extent of how similiar the simplified game is to the real game. The weakness of an optimal strategy is that it does not adapt to the opponent's strategy. In the game "Rock, Paper, Scissors" the optimal strategy is to randomize between the three moves so that each move is equally likely. This strategy is impossible to exploit. But what if the other player play rock all the time? The optimal strategy will not loose, but it will not win either! Rock, Paper, Scissors is a very simple game and the problem of optimal strategies only tie against weak strategies becomes smaller as the game becomes more complex.

A related phenomen is that optimal strategies can lead to strange plays that is very different from how a human would play. An extreme example of this was discovered during the development of the algorithms for this paper. A very simplified game of No-

Limit Poker was created with the following rules: Each player got assigned a random number between one and ten and they did not know what number the other player had been assigned. One player posts a blind of 1 unit. There were only one betting round with a maximum of one bet and two raises. After the betting round was finished the players showed their hidden number and the player with the highest number won the money in the pot. In case of a tie they split the pot. The stack size of each player was limited to ten units. The optimal strategy found for the player in the blind was a surprise. If the opponent bet the maximum the player would fold a 10, the highest possible number, about 30% of the time. After a closer investigation it turned out that the strategy was in fact optimal. The average outcome of the game for the player in the blind was calculated to -0.23 if both players played optimal. If the player in the blind played optimal and the opponent always raised the maximum the average outcome would be -0.21 for the blind player even though he sometimes folded a 10. So the opponent could not exploit the fact that the blind player would sometimes fold a 10 by raising the maximum.

Maximal strategies on the other hand try to maximize their winnings against a specific strategy. For this to be possible we need a good model of the opponents strategy. In a real world environment the task of creating the opponent model would be the main weakness of a maximal strategy approach. The strength is that it is able to win large amounts against very simple strategies. This paper deals mainly with pseudo-optimal strategies so maximal strategies will not be discussed much more.

With this information it is easy to see that a good optimal strategy will only perform moderately well against both strong and weak opposition because it lacks the ability to take advantage of the opponents mistakes. A maximal strategy takes advantage of the opponents mistakes and will therefore perform very well against weak opposition and moderately well aginst strong opposition. But for this to work a good model of the opponents strategy is needed and the better the model the better the results.

5.1.2 The Non-Transitive Property of Strategies

Strategies in poker are not transitive. That means that if strategy A beats strategy B and strategy B beats strategy C, it does not necessarily imply that strategy A beats strategy C. This causes trouble when we want to measure the performance of a specific strategy. We cannot simply match it against another strategy and determine it's strength based on the winnings or losses. Even if we create two pseudo-optimal strategies (A and B) with different abstractions of the poker game, both designed to lose as little as possible, it might happen that strategy A is in fact a very effective counterstrategy to B and beats it even though A is more exploitable than B.

The criteria for a very strong real world strategy should at least include good performance against: a strong optimal strategy, a strong maximal strategy, and very simple strategies such as always betting or always calling. We need to include very simple strategies as well because a pseudo-optimal strategy might be almost impossible to exploit but would not win any reasonable amount from bad players that call all the time, and this would not be very useful in reality. We want strategies that are able to extract

more money the worse the opposition is.

So the conclusion is that because of the non-transitive property we need to use several strategies to measure the strength of any specific strategy.

5.1.3 The Variance of Poker

A common unit, and the unit that will be used in this paper, for measuring how two strategies perform against each other is the number of big blinds won or lost on average per hand (bb/hand). A big blind is the amount player 2 puts in the pot at the start of the game.

As most poker players know, poker is a game of extremely high variance. A thousand hands might not even be enough to simply determine if a player is winning or losing. The smaller the edge one strategy has against the other, the more samples are needed to get an accurate value. For the simulations in this paper several thousands of hands will be used to determine the average outcome of two strategies against each other. It is not enough to get a exact result but will at least be an indicator of how much a strategy is roughly winning or losing.

5.2 Results

The created model of the game uses the following abstractions: It uses four buckets for each street. Each betting round is limited to three raises. It uses a preflop and a postflop model. The bets and raises are discretized to four categories: half pot, pot, two times the pot and all-in. When the model was finished it was transformed to the sequence form [10] and an optimal solution was found using an interiorpoint algorithm. The creation of this model used up to 2.5Gb of address space (with stack size equal to 20 big blinds) and since it was done on a regular 32-bit OS it was not possible to create any larger models. The generation of the model and the solution to the linear programming problem took a couple of hours to compute on a P4 2.8Ghz with 1.5Gb RAM.

The pseudo-optimal strategies was implemented as a poker playing agent named AggroBot. The name was chosen because of it's tendencies to be very aggressive, it bets and raises with a high frequency. A software package called "Poker Academy Pro 2.5" [12] was used to perform all the simulations. AggroBot was matched one by one against a row of different opponents. The opponents used to test AggroBot were:

Amanda This bot is based on the Xenbot engine from Poker Academy. It uses opponent modelling and plays an average game in terms of looseness and aggressiveness.

Avery This bot is based on the Averybot engine from Poker Academy. It uses basic opponent modelling and plays an average game in terms of looseness and aggressiveness.

Gus Xensen The bot is based on the Xenbot engine from Poker Academy. It uses opponent modelling and plays a loose aggressive style of poker.

Sklansky A rule-based bot from Poker Academy. It uses a set of rules to determine if it should fold or go all-in depending on position, stacksizes and cards.

Allin This player goes all-in whenever it is possible. If the other player has gone all-in already it calls the bet. Any strong player should be able to defeat this opponent easily.

Caller This player always check/calls and never raises. Any strong player should be able to defeat this opponent easily.

Opponent	Hands played	Won	bb/hand
Amanda	15607	1082.50	0.07
Gus Xensen	25500.00	3185.00	0.12
Avery	9015	1933.00	0.21
Sklansky	9160	3674.50	0.40
Allin	33874	15102.00	0.45
Caller	28326	30682.50	1.08

Table 5.1: Results for AggroBot using four buckets for each street. Each player's stack is 15 big blinds at the start of every hand.

Opponent	Hands played	Won	bb/hand
Amanda	89871	-1999.50	-0.02
Gus Xensen	54835	7013.00	0.13
Avery	20055	7916.00	0.26
Sklansky	8005	2161.00	0.27
Allin	9886	2372.50	0.24
Caller	18108	30601.00	1.69

Table 5.2: Results for AggroBot using four buckets for each street. Each player's stack is 20 big blinds at the start of every hand.

Hands played is the number of rounds the players played. **Won** is the the number of big blinds AggroBot won. **bb/hand** is the average number of big blinds won by AggroBot in a round. Observe that the number of hands needed to get a statistically significant result depends on the relative strength of the two opponents. Two players who are close to equal in toughness need many simulations to determine the better player. If one player is much better than the other it is possible to see that after much fewer hands.

Chapter 6

Conclusions

AggroBot is clearly much stronger than the other poker bots when the stack sizes are 15 big blinds. When the stack sizes are increased to 20 big blinds the overall performance is slightly worse and AggroBot is actually loosing a small amount against Amanda. One possible reason is that four buckets for each street is not enough with a larger stack. The agent is unable to distinguish between a good hand and a really good hand which becomes more and more important as the stakes increases. It is also important to understand that the Poker Academy bots were originally designed for full-ring or shorthanded games and not heads-up. We will hopefully see stronger heads-up bots being developed in the future to provide tougher competition. Until then, playing against human players might be an alternative but a very time consuming one.

One large obstacle in this project has been to find a suitable linear programming package to solve the sequence form matrices. There exists many open source or otherwise free packages but most of them lack the performance needed for these kind of problems. The matrices created of the previous described abstraction techniques of fullsize poker game are pushing the limits of what is possible to solve with our current technology. To be able to solve really large problems (often very expensive) commercial software is needed. In this project a trial of the optimization software MOSEK was used.

Most game theory litterature are talking about the Simplex algorithm when explaining how to solve imperfect information games. But when a matrix is in the sequence form interior-point based algorithms were found to be much more effective on the larger problems.

Another limitation in this project was the hardware. A regular pc with a 32-bit OS was used for all the calculations. The virtual address space was therefore limit to around 2.7GB. Generating and solving the largest model with stack-sizes of 20 big blinds used close to all of the available virtual address space. To create even bigger models more memory would be needed. If the available memory was doubled models with stack sizes of 30 big blinds or models with five buckets on each street would be possible to solve.

The techniques in this paper only found strategies for heads-up situations where the players have a stacksize of a small predetermined fixed amount. These situations occur regularly online in the end of the small tournaments called "Sit and Go's" when there are only two players left. These are also some of the easiest situations to analyze. On the other spectrum, the task of finding close to optimal solutions in a game with ten players with big stacks playing no-limit poker is infeasible with today's techniques. Hopefully more research on the subject will be done as no-limit presents some interesting problems not available in fixed-limit games.

Chapter 7

Further Research

The interest in poker related research has never been more widespread than today. The possibility for a person to create a strong poker AI and put it online to, possibly illegally, win money to it's creator probably makes the subject very appealing to non-academics as well. The best agents still looses to a human expert player but with the current pace of new ideas in the field it will probably not be long before the best fixed-limit poker player in the world is a computer. Research on no-limit poker still has a long way to go.

This approach of abstracting the game and then finding the optimal solution to the abstracted game is only one of many approaches but a very interesing one. The basic problem is to find abstraction that makes the game tree as small as possible while preserving as much as possible of the strategic attributes of the game. In this project memory requirements was the bottleneck when trying to create more complex models. But it is possible to create more accurate models without increasing the size. In this project the categorization of the hands into buckets was the same independent on previous actions. The river buckets are for example {0-40%, 40-60%, 60-80%, 80-100%} (in terms of chance to win against a random hand). As the initial stack size increases for the players these categories are not enough. If the players have done a lot of betting before the river it is reasonable to assume that they both have a pretty good hand and in that case we would like the categories to like something like {0-60%, 60-75%, 75-93%, 93-100%}. Adjustment of buckets depending on previous actions could be done without increasing the size of the game tree.

Another way to keep the game tree small is to force the agent to do a certain action that does not affect the expected value of the game by much but reduces the game tree. One example would be if we force the player in the small blind to either fold or raise five times the big blind. This would effectively reduce the game tree as fewer raises will be possible in the rest of the hand. The small blind could also be forced to fold every hand in the worst bucket which would reduce the game tree with another 25% if four buckets are used.

Games of incomplete recall does not really exist in the real world but if we are able to

solve them as fast as complete recall games they might be useful for creating abstractions of normal games. If we imagine that we have a game tree of a poker game and the tree splits in 50 different subtrees at the preflop stage, if we now make the players forget what happened preflop at the river stage and we make changes in the game tree to reflect this[1] we have effectively cut the size of the game tree's leaf nodes by 98%. The problem is that we today do not know of any effective way of solving the general case of games with incomplete recall [2].

Another interesting approach would be to create abstracted games that are even bigger than what we are able to solve using linear programming techniques and instead finding strategies using fictitious play. The basic idea in fictitious play is that each player starts with a simple strategy and knows exactly the other player's strategy. The players then step by step adjust their own strategy (this adjustment has to follow a specific set of rules) to be maximal against the opponent's strategy. Fictitous play converges to a Nash equilibrium in every two person zero-sum game[13] but the algorithm is much slower than linear programming algorithms. The advantage of fictitious play is that it is not using extreme amounts of memory as linear programming techniques do on really large problems. It is also possible to stop the algorithm at anytime and get the best solution found so far. So what we loose by only finding an approximation of the optimal solution to the abstracted game we might gain by being able the create a much larger abstracted game that capture more of strategic attributes of the real game.

[1] For every pair of nodes that has the exact same history except for the preflop stage are combined into one node. The process is repeated until no more nodes can be combined. A node in a game tree of incomplete recall can therefore have more than one parent.

[2] To simplify things, the fact that some nodes in the game tree can have more than one parent is basically why it is so hard to find a solution to these kind of games

Chapter 8

Acknowledgements

This project would not have been possible to do alone. So my deepest thanks goes out to:

The people on the forums at poker-academy.com. Without their talent and interest in poker research I would never have gotten the ideas and inspiration for this project.

Michael Minock, supervisor at Umeå University, for his advice and helping me structuring this report.

Last by not least, my family and friends for always being there and keeping my spirit high.

References

[1] D. Billings, N. Burch, A.Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-gheoretic optimal strategies for full-scale poker. In *Proceeings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[2] Darse Billings, Denis Papp, Jonathan Scaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of AAAI-98*, 1998.

[3] Emile Borel. *Applications aux jeux des hazard*. Gautier-Villars, Paris, 1938.

[4] Thomas S. Ferguson Chris Ferguson. On the borel and von neumann poker models. Working paper, University of California at Los Angeles, 2004.

[5] Aaron Davidson. Opponent modeling in poker: Learning and acting in a hostile and uncertain environment, 2002.

[6] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'2000)*, pages 1467–1473, 2000.

[7] Andrew Gilpin and Toumas Sandholm. Finding equilibria in large sequential games of imperfect information. Technical report, Carnegie Mellon University, 2005.

[8] Andrew Gilpin and Toumas Sandholm. Optimal rhode island hold'em poker. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 1684–1685, 2005.

[9] Oskar Morgenstern John Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[10] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *STOC*, pages 750–759, 1994.

[11] John F. Nash. Equilibrium points in n-person games. In *National Academy of Sciences*, pages 48–49, 1950.

[12] Poker academy, http://www.poker-academy.com, 2006.

[13] Julia Robinson. An iterative method of solving a game. In *Annals of Mathematics*, pages 298–301, 1951.

[14] Jiefu Shi and Michael L. Littman. Abstraction methods for game theoretic poker. In *Computer and Games*, pages 333–345. Springer-Verlag, 2001.

[15] K. Takusagawa. Nash equilibrium of texas hold'em poker. Undergraduate thesis, Stanford University, 2000.

[16] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Boston, 1996.