Opponent Modeling in Texas Hold'em

Nadia Boudewijn, student number 3700607,

Bachelor thesis Artificial Intelligence 7.5 ECTS,

Utrecht University, January 2014, supervisor: dr. G. A. W. Vreeswijk

ABSTRACT

Many of the current approaches to opponent modeling research in the domain of poker focus on building an explicit model that captures the opponent's behavior. Unfortunately, all of these approaches face the same problems for which no solution has yet been found. In this paper, the properties of explicit opponent models and the difficulties that they introduce will be discussed and compared to the properties of implicit opponent models. Recently, Bard et al. proposed an implicit approach that seems promising: the agent that is described in their paper is shown to have won the 2011 Annual Computer Poker Competition and recently they entered an agent based on this implicit modeling framework in the 2013 Annual Computer Poker Competition that won (shared) second place. Maybe the time has come to favor implicit models over explicit models for opponent modeling. To be able to make a fair judgment on this we will also discuss the possible problems that are introduced by the implicit modeling framework.

"Imagine working in an adversarial environment, trying to determine your next course of action. You know what your goal is, but there are others operating here too. Others who are not on your side. Others who may know things that you do not. As you decide on actions to take, the others are simultaneously plotting their next actions, hoping to make the best of their situation which may involve hindering your progress. The environment itself is full of uncertainty: you don't know whether luck will be in your favor or not but you must decide what to do next. "

- Brett Jason Borghetti

Contents

1 Introduction

The opening quote, by Brett Jason Borghetti, illustrates vividly that succeeding in an adversarial environment takes (at least some form of) intelligence [6]. In an adversarial environment the utility of an agent increases when the utility of the other agents (the adversaries or opponents) is reduced. An agent's utility can be seen as his well-being. If you are in an adversarial environment, there are no previously written rules you can rely on in every situation. Furthermore, if you gain something, another agent is losing it. Needless to say that goes both ways. As the other agents are gaining you will be set back in your winnings. This creates interesting opportunities to take advantage of specific opponents and forces us to consider our own exploitability.

To be able to take advantage of specific opponents, agents can create and maintain an opponent model. This is known as opponent modeling. The difficulty with opponent modeling is that in many real-world situations the search for an optimal action is computationally prohibitive. The enormous amount of possible action sequences and interactions between agents combined with the presence of chance simply leads to too many possibilities. Opponent models can be used to reduce the size of the search space by altering the like hood of certain action sequences. The field of research that examines these topics is known as machine learning. Machine learning is a subdomain of Artificial Intelligence which considers itself with getting computers to act without being explicitly programmed. Recently a lot of work has been done in the area of multi agent systems and specifically on opponent modeling in the domain of poker. This thesis willl focus on opponent modeling and how it is implemented in computer programs, called 'poker agents' or 'poker bots'.

1.1 Relevance to AI

Games are a natural choice for AI research. Games can usually be defined by a set of simple rules and yet present challenging situations that require prediction, simulation, reasoning and decision making to solve. Performance is easily measured as each game has it's own performance measure by definition. A major issue in game programming is opponent modeling. The creation of an accurate model to predict the opponent's actions turns out to be quite difficult.

Opponent modeling can be a challenge in perfect information games (where the full state of the game is known to all players at all times), like checker or chess, due to the huge size of the search space. With the help of game tree search algorithms, world-class computer players have been developed that defeated human world champions. But computer programs for poker have not been as succesfull. The random shuffeling of the card deck makes poker a non-determinisitic game. It is also a game of imperfect information as a player does not know which private cards are handed to the opponent. This means that a player has to make decisions without knowing the precise state gamestate. Therefore, with each decision the payer makes, he has to consider the alternative possibilities. The more information is hidden, the more alternative situations the player will have to consider. At some point in the game, the player has to consider so many alternative possibilities that the computations become intractable. This makes brute force search to determine a coarse of action a highly impractical option, leading to poker being a perfect testbed for AI.

We must use this perfect testbed for AI provided by poker to our advantage and learn as much as we possible can from it. The fact that none of the current poker programs are able to defeat a human world class player should not discourage us. On the contrary, it should motivate us to develop new (or enrich current) algorithms, learning methods, and search techniques. It is very imporant for AI research that we keep improving the level of perfomance of pokerbots as the ultimate goal for much of AI reserach is to develop useful systems that can adaptively make intelligent decisions in a world like ours: a huge complex, hostile environment that is very unpredictable. The development of agents that are able to act in complex, unpredictable and hostile environments will bring us one step closer to the goal of creating intelligent agents and has numerous applications ranging from economic endeavors to military operations.

1.2 Purpose and structure of this thesis

The goal of this thesis is to give insight in the properties of the current two main approaches to opponent modeling in the domain of poker, and the issues that arise in this area. I will try to reach this goal by answering the following question:

What are the main differences between explicit and implicit modeling, and how do these differences affect the usability of a model for opponent modeling in poker?

I will approach this topic as follows: section II provides the necessary background information. In section III the properties and problems of explicit modeling are discussed. Section IV does the same for the implicit modeling approach. Section V will discuss the implicit modeling framework proposed by Bard et al.: the first subsection will stepwise explain the steps that are taken offline, whereas the second subsection handles the online steps and pays specific attention to the bandit-style algorithm Exp4. Finally, section VI will conclude this thesis.

2 Background

This section begins with a short overview of the rules and goal of heads-up limit Texas Hold'em poker. Next I present some concepts from game theory including the framework of extensive form games which can be used as a model of multiagent interaction in the domain of Texas Hold'em. I then discuss methods for computing behavior policies, called strategies, in this framework. Finally I will address the multi-armed bandit problem.

2.1 Heads-up limit Texas Hold'em Poker

There are many variants of poker. The variant we will be focusing on, Texas Holdem, is one of the most popular variants and represents the main event of the World Series of Poker (which in 2012 had over $220 million dollar in total prize money [20]). Texas Holdem Poker is a very popular game with many interesting properties and just a few simple rules. The goal is to win as much money as possible from the opponent by the end of the session. The game is played with a standard 52-cards card deck. The heads-up variant means there are only two players. 'Limit' refers to the fact that there are pre-specified bet and raise amounts and the number of bets each player can make in a single round is bounded. At the beginning of a betting round the players alternate between being small blind and big blind. Before any cards are dealt the small blind contributes one chip (chip represents a fixed betting amount) to the pot and the big blind contributes 2 chips. Each player then receives two private cards. The small blind can then choose from 3 options:

- **Fold**: quitting the game, the pot goes to the opponent
- **Call**: follow through with the game and thus matching the highest bet currently placed on the table
- **Raise**: the player calls and raises his bet with the allowed number of chips

If the player decided to call or raise, the other player gets to choose between fold/ call / raise. Next is the Flop. Three cards, visible to both players, are dealt face-up. The big blind then starts a new betting round. After this round one card, the turn, visible to both players is dealt face-up after which a new betting round takes place as on the flop. Finally the last card, the river, is dealt face up for both players to see. The last betting round takes place as on the flop and turn. If none of the players at this stage in the game have chosen to fold it is time for the showdown. Both players try to make the best possible 5-card combination (the rules for these combinations can be found on [18]). The player with the highest hand earns the chips from the pot (in case of equal hands the players split the pot).

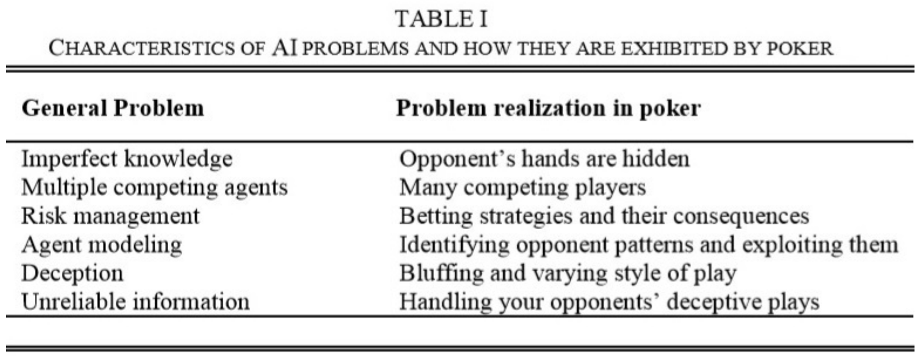

TABLE I
CHARACTERISTICS OF AI PROBLEMS AND HOW THEY ARE EXHIBITED BY POKER

General Problem	**Problem realization in poker**
Imperfect knowledge	Opponent's hands are hidden
Multiple competing agents	Many competing players
Risk management	Betting strategies and their consequences
Agent modeling	Identifying opponent patterns and exploiting them
Deception	Bluffing and varying style of play
Unreliable information	Handling your opponents' deceptive plays

Table 1: Source: Poker as a Testbed for AI Research, table 1 [5].

A strong poker player requires several skills:

- **Hand Evaluation**: the probability that a hand is the best, given the opponent and the context of the game. It is crucial to be able to make an accurate assesment of the current and potential strength of a hand.
- **Unpredictability**: actions must not give away any information about the strength of the hand the player is holding. The agent must hide information about his hand by playing deceptively and mixing strategies.
- **Opponent Modeling**: this is necessary to exploit the opponent's weaknesses and to defend our play from possible attacks.

Poker is not only popular amongst humans: The AAAI Annual Computer Poker Competition (ACPC) takes place since 2006 and tries to benefit the field of Artificial Intelligence by providing a test bed for poker research. In this paper we consider a poker variant called two-player limit Texas Holdem, which is the smallest variant played in the ACPC (extra rules or restrictions that are enforced during the ACPC can be found on the official website) [1]. Besides the ACPC providing a perfect venue for testing and demonstrating poker-playing software systems there are several aspects that make this immensely popular game an interesting field of research. The game has an enormous strategy space (10^{18} game states for limit Texas Holdem and 10^{71} for no-limit Texas Holdem) and exhibits several characteristics of AI problems that are listed in Table I. In theory it is possible, when playing against perfect opponents, to find an optimal strategy based on the underlying mathematical structure of the game. Unfortunately this is not possible in reality because determining the optimal strategy appears to be computationally infeasible. Even if we were to find this optimal strategy for perfect opponents it does not have to maximize our utility against most typical opponents. This makes opponent modeling a topic that cannot be overlooked when writing a poker program.

Results for computer poker agents are usually expressed in millibet, a one thousandth of a small bet. In [13] Johanson et al. provide a nice example that might give some intuition for this unit of measurement: a player that always folds will lose 150 millibet per hand while a typical player that is 10 mb per hand stronger than it's opponent would require over one million hands to be 95 percent certain to have won overall.

2.2 Game Theory Concepts

Poker is a game, where the agents are known as players. Players take actions that result in utilities, i.e. their scores. Each player may develop a **strategy** which consists of a collection of actions for each possible decision, with respect to different conditions, to be made in the game. A strategy may also consist of a collection of distributions over actions (this is called a **mixed strategy**), e.g. in the game Rock, Paper, Scissors the chance of playing each action being 1/3. The game can be represented in it's **extensive form**. This is an intuitive model for representing actions between multiple agents and their environment, that also makes it possible to represent chance events. It can be viewed as a tree, in which each non-terminal node represents a state where one of the players (or chance) has to act. The available actions at each node are represented by the direct edges. Each terminal node (the leafs of the tree) assigns a utility to each player. Each action may be observed by one or both players. Since players are not observing all information they cannot determine the precise game state. Instead, they observe an **information set** that contains all nodes that differ only in that they all exists under a different hidden condition which is unknown to the player. This makes an information set a set of games indistinguishable to the acting player. For example, when the cards are being dealt at the beginning of a round each player will get 2 cards that are not visible to the opponent. There are $52 \cdot 51$ possibilities for receiving the two first cards which when divided by 2 (because it does not matter which card was received first) gives 1326 possibilities for the two first cards. In the same fashion, $50 \cdot 49$ divided by 2 gives 1225 options for the two cards the other player holds. This means that there are $1326 \cdot 1225 = 1{,}624{,}350$ branches form the initial chance node that represents the dealing of the cards. There are 1225 different states which are not distinguishable to the player since he has no way to know the cards in the hand of the opponent. These states are grouped together in an information set. The same decision policy has to be applied to all states in this information set, since it is not possible to know exactly which of those states we are in. Information sets can be used when abstracting the game by merging information sets that result from similar chance outcomes.

Now we can define a players strategy to be a function mapping the player's information sets I to a probability distribution over the available actions A(I). A **strategy profile** is a tuple containing one strategy for each player in the game. Given a strategy profile σ, we define the **best response** for an agent to be the strategy that maximizes the expected payoff, assuming all other agents play according to σ. A strategy profile σ is a Nash Equilibrium if no agent has anything to gain by changing is strategy. That is, by deviating from the equilibrium strategy, assuming the other agents are playing according to σ, the agent cannot enhance its winnings in any way. Simply said, with a Nash Equilibrium every agent's strategy is a best response to all the other agent's strategies. Nash showed that all zero-sum imperfect information games have an equilibrium in which ever player can ensure the optimal outcome with an appropriate randomized mixed strategy [15]. An ϵ-equilibrium is a strategy profile in which each agent receives a payoff within ϵ of his best response. A strategy profile's **exploitability** represents the expected loss. We define it to be the average of the best response values of its strategies against a worst-case opponent. A Nash Equilibrium strategy profile for poker (which is a two-player zero-sum game) has an exploitability of 0.

2.3 Robust Counter Strategies

When facing an arbitrary opponent, creating a hundred percent accurate model of it's behavior is usually not possible within reasonable time limits. Therefore, when modeling an opponent the agent makes assumptions about the opponent. When these assumptions differ from reality the agent's strategy can become a victim to exploitation. A **minimax** strategy minimizes the possible maximum loss (it can also be thought of as maximizing the minimum gain, in which case its called a **maximin** strategy). The minimax theorem from von Neumann says that in any finite, two player , zero-sum game, in any Nash equilibrium each player receives a payoff that is equal to both his maximin value and his minimax value. Any maximin/ minimax strategy profile is a Nash equilibrium. Minimax strategies are intended to be as un-exploitable as possible. The worst-case scenario is guaranteed to provide some maximum loss and any non-optimal choices on the part of the opponent only increase the payoff to the minimax player. Therefore, minimax strategies can be viewed as a safe strategy. A signicant advantage of the minimax strategy over other algorithms is that it is independent of the policy played by the opponent. This means that a minimax solution can be calculated ahead of time for any game, and this strategy can be put into effect regardless of the actions of the opponent. Unlike learning algorithms, such as opponent modelers, there is no initial period of low effectiveness while the model is being built. Like the opponent modelling strategies, it is the assumptions made by minimax agents that are their main weakness. Minimax strategies assume that the opponents are optimal, and that the goals of the opponents are opposite to the goals of the agent. In cases where these assumptions are not true, minimax players can end up settling for much lower payoffs than what could be achieved by exploiting non optimal opponents.

Counter strategies are able to maximize utility by taking advantage of the opponent's flaws. But Johanson et al. have shown that when the opponent's behavior deviates from the approximation, or when the opponent deliberately changes his behavior, counter strategies are not very robust and can become victim to exploitation [12]. McCracken and Bowling propose the use of **ϵ-safe responses** to create robust counter strategies [8]. These strategies can guarantee to be exploitable for no more than ϵ in the worst case, and win much more than a Nash equilibrium (minimax) strategy by exploiting non perfect opponents. Two algorithms for the creation of these robust responses will be discussed in this paper: The Restricted Nash Response (RNR) algorithm and the Data Biased Response (DBR) algorithm.

2.4 The Multi-Armed Bandit Problem

The well-known multi-armed bandit problem provides a simple model for the trade-off between exploration and exploitation. In the multi-armed bandit problem, a gambler tries to maximize his winnings from playing a row of slot machines in a sequence of trials (slot machines are also known as one-armed bandits). When played, each machine provides a random reward from a distribution specific to that machine. The gambler constantly has to make a decision between keep playing on the slot machine that has the highest payoff at the moment (exploiting a single arm) or trying out new slot machines that might give a higher pay off (exploring other arms). In the adversarial multi-armed bandit problem the payoffs of each arm are not generated by a well behaved stochastic process. Instead, they are influenced by an adversary (which in the game of heads-up poker would be the opponent).

3 Explicit Opponent Modeling

An opponent model can be either implicit or explicit. Most of the existing approaches to agent modeling in poker fall in the explicit category. With explicit modeling, an agent tries to infer the opponent's strategy by observing his actions in different situations. This is achieved by building a model for the opponent. From this model the agent tries to predict te opponent's actions and tries to choose a best response given the current conditions in the environment. Thus, the oppent's actions are analyzed seperately from the state of the world. This might be done by building a static opponent model: once the opponent model is created the agent keeps using it during the entire game. It is easy to see that this is not very realistic. Especially not for the game of poker where the opponent might change strategy, or may have been playing deceptively, hoping that our agent infers the wrong strategy. When the agent plays a counter strategy to the strategy that he believes the opponent is playing he will make himself vulnerable for exploitation. Therefore, almost all recent approaches favor a dynamic (learning) model which is able to adjust when an opponent changes his strategy during the game.

In the first subsection I will discuss several different methods that may be used for learning and using such a dynamic explicit model. Although there is a lot of work done in the area of explicit poker agents, in subsection 2 we see that all approaches encounter two problems. The most applied technique to overcome these two problems is applying a state-space abstraction algorithm to the game which constructs a smaller game that preserves as many of the strategic properties as possible. The solution for the smaller game is mapped to a strategy profile in the original game. This technique introduces some problems of its own; abstraction pathologies may rise or we might see our solution overfitting the abstract game [19]. This so calles state-space abstraction technique will be discussed near the end of this section. We will finish this section by looking at some of the most promising explicit opponent modeling frameworks.

3.1 Building an Explicit Model

An opponent strategy can be modeled with anything that maps game states to moves (or move distributions). Some frequently used tools are:

- **Desicion trees**: learning based on a predictive model using decision trees.
- **Artificial Neural Networks**: learning based on biological neural networks, like our brain.
- **Bayesian Networks**: learning based on Bayes' rule.
- **Clustering**: in large sets of unlabeled examples, examples get grouped together in a cluster if they are more similar.

An example of a poker program that combines several methods is Poki[9]. This meta predictor approach performed better than all single methods. Poki combines decision trees, neural networks and expert formulas. It plays at the level of an average human player and uses opponent modeling to predict whether opponent wil raise, fold, or call/check on each round of betting. Poki uses a meta-predictor: it runs the neural network, decision tree and other methods (such as expert formulas) on the available data. Each predictor votes on which action it thinks the opponent will take and votes are weighted on each predictor's accuracy so far.

3.2 Difficulties

In order to be able to create an opponent model, an agent has to observe his opponent. These observations can then be used to model the opponent's behavior. But in such a complex domain as poker, the building of an accurate model requires a prohibitive number of observations. This is the first problem that all explicit modeling frameworks encounter. Second, even if the agent is able to build a model, computing a response strategy that is robust to modeling error may be impractical to compute online. Online calculations have to obey tight time constraints which make it quite impossible to perform extremely heavy calculations in time.

The most used solution for these problems is applying a state-space abstraction technique. **State-space abstraction** is a many-to-one mapping between the game's information sets and the information sets in a smaller, artificially constructed game. A large amount of possible poker situations have to be translated to a relatively small amount of abstraction classes. The agent observes the abstract game information set, and uses the strategy for that information set for all of the real information sets mapped to it (simply put: the agent applies his knowledge of similar situations to the current situation). If the opponent changes his style, previous observations lose their value. Therefore we must acquire knowledge very quickly, and incorporate a bias towards more recent observations. The goal is to construct a game small enough that an optimal strategy can be found and can be used in the original game where it is hoped to closely approximate a Nash equilibirium strategy.

The size of the abstraction is very important. If the abstracted game remains quite large it is more likely to be an accurate representation of the full scale game, but at the same time calculations for this game may still be to large to be performed online. If the game is abstracted to a very small version, important information might get lost but performing online calculations will not be a problem. Another important factor that determines the succes of this technique are the domain features used to decide which information sets can be mapped together.

The abstracted game can be created in many different ways. A common metric used in early work is a player's expected hand strength. Expected hand strenght is the expectation of hand strenght over all possible rollouts of the remaining public cards (in the final round when all public cards are revealed, a players hand strength is the probability that their hand is stronger than a uniform randomly sampled oponent hand). The expected hand strenght squared computes the expectation of the sqaured hand strength values, and assigns a relatively higher value to hands with the potential to improve. These expectation based metrics can than be used to create abstract chance events in a number of different ways. An example of this can be found in section 5.1.1, where an abstraction is used in the

Restricted Nash Response algorithm to find ϵ-safe best responses.

3.3 Related Work

In [11], Ganzfried and Sandholm build an explicit poker agent that observes the opponent's action frequencies. The agent then uses these observations to build a model based on the deviations from a pre-computed equilibrium strategy. Next the agent computes and plays the best responses to this model. This gives the advantage of being able to identify weak opponents by observing their actions, and exploiting them with best responses to their weaknesses. When faced with a strong opponent, the agent plays the equilibrium strategy. However, the approach has not been tested against strong opponents and may be highly exploitable because the best response is calculated against a current model, and the model must use a relatively coarse abstraction of the game for the agent to act quickly enough.

Rubin and Watson apply adaptation to a pre-computed, static case-based strategy in order to allow the strategy to rapidly respond to changes in an opponents playing style [17]. A case-based strategy looks at similar situations in the past to select successful actions. To classify the current opponent type online they build a low-dimensional explicit model. This approach overcomes the problem of needing many observations to build a representative model by using pre-computed strategies. But there is still no guarantee that the explicit model build online that is consulted for adaptations is a hundred percent accurate. Modeling error can lead to choosing a bad adaptation. Unfortunately, there are no results for this approach against actual ACPC agents.

3.4 Summary

It seems that none of the recent efforts to use explicit modeling in this complex domain are able to overcome the challenges that come from building a model and computing a robust response online, and are not able to provide agents that are capable of defeating strong opponents in a full-scale game. The state-space abstraction technique that many researches apply to overcome challlenges also introduces some delicate problems. It is therefore exciting to explore an implicit modeling agent built by Bard et al. of which they promise that it overcomes these challenges. It is shown in their paper that their agent would have won the heads-up limit opponent exploitation event in the 2011 ACPC, which proves that their agent is capable of defeating strong opponents in a full-scale game. In the next section we will first look at implicit opponent modeling in general, followed by a section that discusses the implicit opponent modeling framework by Bard et al. in detail.

4 Implicit Opponent Modeling

With implicit modeling the agent tries to maximize it's utility with respect to it's own observations and actions. The agent tries to find a good counter strategy without having to identify the opponent's strategy. Thus, unlike explicit modeling, the opponent's actions are not analyzed seperately from the state of the world. Remember that with explicit modeling, online data from the opponent playing the game is used to estimate a model and determine a response. With implicit modeling, the agent first computes a portfolio of responses offline and then uses the data from playing against the opponent online to estimate the utility of the responses. By not having to construct an opponent model online the two main problems seen in explicit modeling are completely avoided.

In their paper, Bard et al. illustrate several other benefits of this approach. Because prior work can be performed offline Bard et al. claim that they are able to use computationally demanding techniques which enable the creation of robust responses for the portfolio. To create the portfolio they make use of existing algorithms that can guarantee a maximum loss. By limiting the actual behavior of the agent during play to be from this portfolio of responses they maintain a safety guarantee for the maximum loss. Furthermore, the dimensionality of model parameterization for implicit modeling is reduced to the size of the portfolio regardless of the complexity of the domain or certain behavior. This is quite the improvement on explicit modeling where this called for a prohibitive number of observations.

4.1 Difficulties

We have seen that the implicit modeling framework does not have to deal with the two main problems introduced by the explicit modeling approach. Unfortunately, implicit modeling introduces a challenge of its own: how to decide when to switch between the two phases of the modeling process. Simply put, implicit modeling consist of two phases:

- exploration of various counter strategies
- exploitation of the highest scoring strategy

When an agent exploits a single strategy to soon, there is a very high risk that he is exploiting a non-optimal strategy. On the other hand, when the agent stays in the exploration phase for too long, there might not be enough time to recover from the losses that are build up in this phase.

As mentioned before, this thesis will focus on the implicit modeling framework propesed by Bard et al. In the next section we will see which methods they have chosen to deal with the difficulties that we have discussed. To emphasize that there are other approaches to implicit modeling we will first mention some related work before discussing the implementation by Bard et al. in detail.

4.2 Related Work

In [16], Rubin and Watson investigate an implicit agent modeling approach quite similar to the approach from Bard et al. that we are considering. They use the UCB1 algorithm to select from a portfolio of expert imitators. Johanson et al. also applied the UCB1 algorithm in a similar fashion to select from a portfolio of RNR strategies [12]. Unfortunately, these approaches do not take into account the fact that UCB1s regret bounds are for the stochastic bandit problem (see section 5.2.2). Because poker is an instance of the adversarial bandit problem, this might be inappropriate.

5 Online Implicit Poker Agent

To avoid the two main challenges introduced by explicit models, Bard et al. propose using an implicit model instead of an explicit model for the creation of an agent for heads-up limit Texas Hold'em [4]. This implicit approach seems promising: the agent that is described in their paper is shown to have won the 2011 Annual Computer Poker Competition and recently they entered an agent based on this implicit modeling framework in the 2013 Annual Computer Poker Competition that won (shared) second place. Their method consists of two steps (see Figure 1, page 16):

1. generation of a portfolio of strategies offline
2. choosing the best suitable response from the portfolio online

We will now discuss this implicit framework in detail, starting with the offline creation of the portfolio.

5.1 Offline Portfolio Generation

The portfolio with response strategies is build offline. Offline computation has a major advantage on online computation: its not bounded by tight time constraints. This extra time allows the building of more sophisticated responses. But what kind of responses do we want in our portfolio? Ideally, we want a portfolio with strategies that maximize utility for all opponents that we will be facing. But we do not want these strategies to become exploitable by any of the opponents. It seems that only when we have access to a perfect model of the opponent, we can exploit them safely by a best response. Otherwise, it is best to play a Nash equilibrium strategy. This is a little bit disappointing since its far from likely that when facing a new opponent during a game we willl have access to a perfect model of his behavior. Of course we can build a model, but it will not be a hundred percent accurate (due to the fact that it is formed from a limited number of observations of the opponents actions, or the opponent is known to be changing strategy). We could compromise: accepting a lower worst-case utility in return for a higher utility if the model is approximately correct.

Such a compromising strategy can be created very easy. You could let a biased coin decide the probability p with which we will play the best response, and the Nash equilibrium will then be played with probability (1 - p). Bard et al. have decided to create their compromising strategies between Nash strategies and counter strategies with ϵ-safe responses. ϵ-safe responses are the utility maximizing strategies from the set of strategies exploitable for no more than ϵ, where ϵ represents the maximum loss we are willing to accept. To produce these ϵ-safe responses two existing algorithms are considered: the Restricted Nash Response algorithm and the Data Biased Response algorithm. These algorithms will be discussed in the upcoming two subsections. Figure 1 on the next page illustrates the complete implicit modeling process. The reader might notice that the offline creation process for the portfolio involves two more steps that need some explanation: the application of the CFR algorithm and submodular optimization. These steps are discussed in subsection 3 and 4 from this section.

5.1.1 RNR

The RNR algorithm [12] is applicable if you want to find ϵ-safe best response strategies for a known adversary strategy. The algorithm creates a modified game where it finds the Nash equilibrium. This modified game is created using a hand strength squared abstraction. Hand strength is the expected probability of winning given only the cards a player has seen, hand strength squared is a metric that gives a bonus to card sequences whose eventual hand strength has higher variance (higher variance receives a bonus because it eventually makes the player more certain about the ultimate changes of winning even prior to showdown). The abstraction groups card sequences (combinations of a players private and public cards) into bucket sequences. Each bucket maps the sequences to a number between 0 and 1. First, all private card pairs are partitioned into five equally sized bucket based upon the hand squared metric. Next, all public card pairs that got placed in the same bucket in round one are partitioned into five equally sized buckets based on the metric now applied to round two. This is repeated after each round, continuing to partition card sequences that agreed on the previous rounds' buckets into five equally sized buckets based on the metric applied in that round [12]. The resulting abstract game has approximately $6.45 \cdot 10^9$ game states (which is a nice improvement with respect to the 10^{18} game states for Limit Texas Holdem).

In this modified game the opponent is forced to play according to a fixed

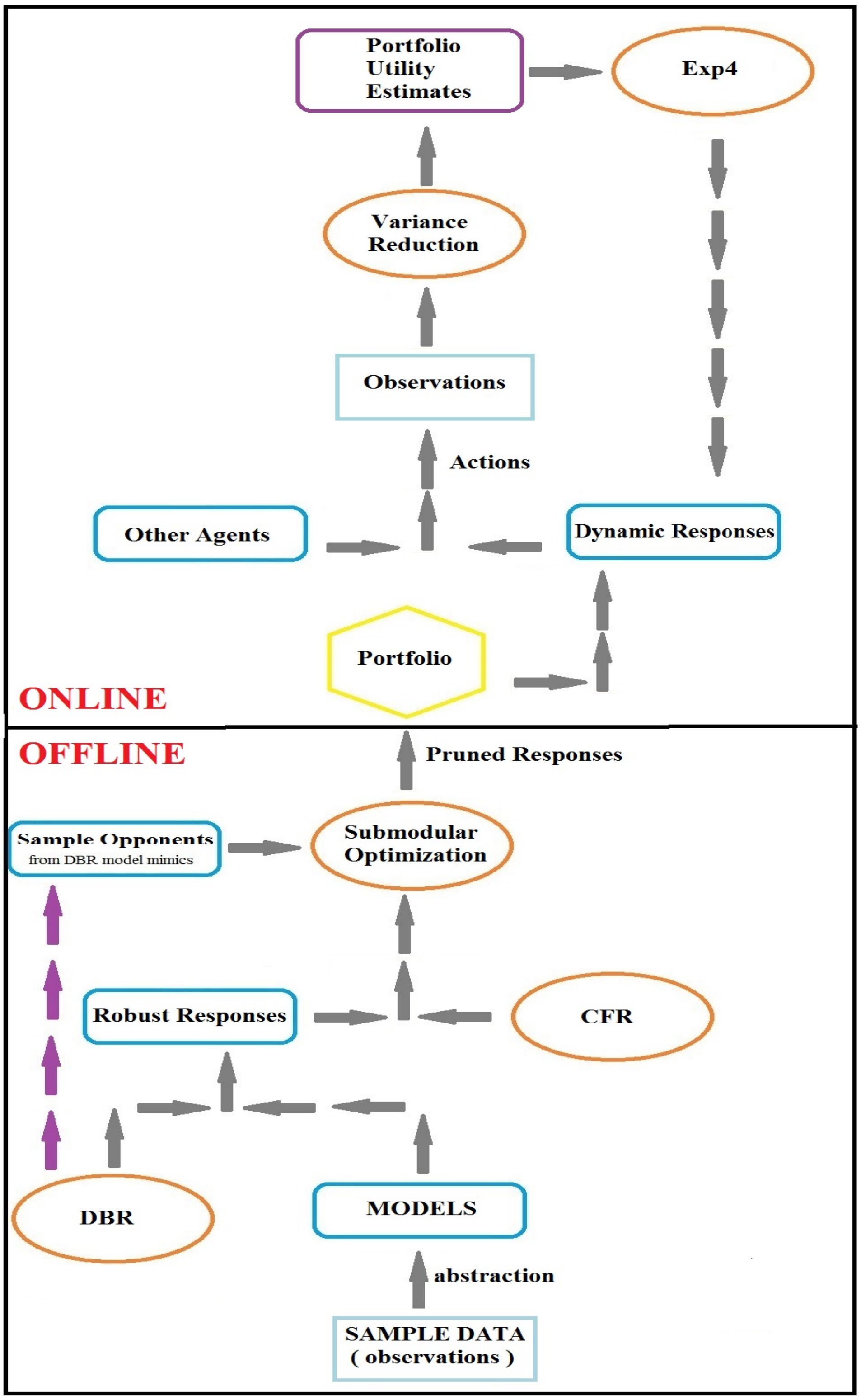

Figure 1: Implicit Modeling Process. This figure is based on figure 2 in [4].

strategy with some probability p. The value of p controls the proportion of time the opponent must use the fixed strategy. This value is chosen when creating the strategy. If p is 0, the opponent never plays the fixed strategy, meaning the agent plays a Nash equilibrium, and if p is 1 the agent plays a best response to the opponent model of the opponent's fixed strategy. When p is any value between 0 and 1 a counter strategy is played with different trade-offs between exploiting and preventing exploitability. These trade-offs are important to consider when facing a particular opponent. Setting p closer to 1 creates the opportunity to gain more utility from an agressive response to that opponent but one must consider the exploitability of the response itself. Given a value of p, the modified game can then be solved (that is, a Nash equilibrium strategy can be approximated) using any game solving algorithm, such as CFR (see section 5.1.3) [21]. The counter strategies are ϵ-safe best responses. The best response between these ϵ-safe best responses can be found by varying p, making the RNR strategies the best possible counter-strategies, assuming the model is correct.

Keep in mind that the assumption of a correct model is quite dangerous: for the model to be correct the opponent's strategy has to be known up front and as pointed out before, there are not many real life situations where full opponent strategies are available. If the opponent model is not correct it may lead our RNR strategies to not being the best possible counter strategies. This answers the question why RNR is not depicted in Figure 1 on the previous page as the algorithm used to create robust responses. The DBR algorithm, explained in the next subsection, is favored instead of RNR because it does not need a known adversary strategy to produce ϵ-safe best responses.

5.1.2 DBR

The DBR algorithm [13], an extension of the RNR algorithm, is applicable when we only have a set of observations of the opponent playing the game and have to construct a model of his behavior. It constructs an opponent model by counting the frequency of each action at each information set over the set of observations. Instead of a single probability p that is set at the root of the game tree as with RNR, the DBR strategy chooses a probability p(I) at each information set I, with p scaling with the number of observations at I. p is varied at each decision: if there are many observations of the opponent's actions available, a higher value of p makes the agent play more exploitive strategies. In cases where there are no observations available p is set to zero, the agent plays a Nash equilibrium and the opponent is free to choose any action. To summarize: if not enough data is provided, the algorithm defaults towards a Nash equilibrium and when observations are present, it moves towards exploitive strategies that also limit their worst-case loss.

A nice feature of the DBR algorithm is that while it computes a robust response to data, it also computes a robust strategy that mimics the data. At each information set, the mimic will, with some probability based on the amount of data available, choose its play so as to prevent exploitation by the DBR strategy. This mimic strategy behaves increasingly like the agent which produced the data as more observations are available. We will use these mimics in the process of determining the portfolio's exploitive power.

5.1.3 CFR

CFR [21] is an algorithm for approximating Nash-equilibrium strategies in two-player zero-sum perfect recall extensive form games.[1] CFR requires too much computation for real scale poker and is therefore usually applied to an abstract game. This abstraction is generated by partitioning card sequences based on the hand strength squared metric (section 5.1.1). The CFR algorithm minimizes counterfactual regret in order to minimize the total regret. It is an iterative self-play algorithm. Each player begins with an arbitrary strategy. On each iteration, the players examine every decision, and for each possible action compare the observed value of their current policy to the value they could have achieved by making that action instead. This is the regret for playing an action, and the accumulated regret is used to determine the strategy used on the next iteration. The average strategies used by the players converge to a Nash equilibrium.

5.1.4 Selecting for the portfolio

We have seen that based on observations of agents playing poker we are able to create robust responses offline. We want to bundle these responses in a portfolio and determine online which of the strategies from our portfolio will maximize our utility. In theory, it is possible to generate a robust response from every past interaction. In reality, it may not be wise to include all these responses in our portfolio. The portfolio would become very large which would slow down our online calculations. After every hand the agent has to estimate the utilities of every strategy in the portfolio. Too many strategies will add too much computational burden. We must also realize that both in theory and in practice, bandit-style algorithms show regret growing with the number of available bandit arms. As we try to estimate the utility of each response from our portfolio by using a bandit-style algorithm, having many responses will require too much exploration before exploitation can reliably occur (the bandit algorithm that is used is called Exp4 and is explained in section 5.2.1). Furthermore, each additional response may not be adding

[1]Although CFR is only proven to converge to a Nash Equilibrium in two-player zero-sum perfect recall games, in practice it appears robust when these constraints are violated as it has been succesfully applied to multi-player games, non-zero-sum games, and imperfect recall games [14].

much to the overall exploitive power of the portfolio if other similar responses are already included.

We want to find a subset of the robust responses which maximizes the resulting portfolio's exploitive power. The mimics, generated by the DBR strategy, can provide interaction to determine our portfolio's exploitive power. Bard et al. now define the objective as the total expected utility achieved against all of the generated mimic strategies, when the portfolio's utility-maximizing response for each mimic can be optimally chosen. Using greedy approximation, responses are repeatedly added to the portfolio one at a time, with each one maximizing the marginal increase in our proxy objective function. We stop adding responses once the marginal increase becomes too small or when computational resources run out.

Bard et al. have demonstrated their implicit modeling agent using two different portfolios. A portfolio with all responses (Big-Portfolio) and a smaller portfolio with four responses that were generated using this greedy approximation to submodular optimization (Small-Portfolio). The Small-Portfolio agent outperformed their Big-Portfolio agent three times:

- When playing against the four mimics generated by DBR for the four responses of the Small-Portfolio.
- When playing against the entire field of 2010 ACPC competition mimics.
- Against all agents from the 2011 ACPC.

These empirical results support Bard et al.'s intuition for the benefits of using a submodular optimization to prune back the portfolio to a manageable size.

5.2 Online Adaptation

When playing online, we want to know the expected utilities of the responses in our portfolio so we can select the response that generates the highest utility. The expected utility of each response in our portfolio is estimated using a multi-armed bandit algorithm. As the number of observations that are needed for a confident utility estimation might grow dramatically due to the element of chance that is present in the game of Texas Hold'em, the agent also makes use of variance reduction techniques. Variance reduction techniques can help eliminate some of the noise induced by chance and reduce the number of observations needed to generate a reliable utility estimation. This section starts with a thorough explanation of the Exp4 algorithm [3] that Bard et al. have chosen to use to determine the utility of each response from the portfolio online. Some of the previously mentioned related work used the UCB algorithm for this task so I will discuss some points that support the decision for Exp4. Two small adjustments to the Exp4 algorithm are made that allow the usage of Bard et al.'s off-policy importance sampling and imaginary observations as variance reduction technique [7]. A short description of this variance reduction technique concludes this section.

5.2.1 Exp4

Exp4 stands for Exponential-weight algorithm for Exploitation and Exploration using Expert advice. It provides a solution for the adversary multi-armed bandit problem where the player has a set of strategies for choosing the best action. Expert advice refers to the fact that Exp4 combines the choices of N strategies (experts) which all select a different action from K actions at each iteration. This is where Bard et al. make their first adjustment to the Exp4 algorithm. Since they play a mixture of extensive form strategies instead of a distribution over single actions, the strategies' action sequence probabilities need to be averaged. With another small adjustment that will be discussed further on this algorithm is directly applicable to the problem of selecting the strategy with the highest utility from our portfolio. To create a better understanding of this process the original Exp4 algorithm will now be explained in detail. Let us start with a formal definition of the adversarial bandit game, given by Figure 2 on the next page.

Exp4 is an extension of Exp3, which stands for Exponential-weight algorithm for Exploitation and Exploration. Exp3 uses a subroutine called the Hedge algorithm. The Hedge algorithm from Auer et al. is a variant of the Hedge algorithm for full information games introduced by Freund and Schapire [10]. This variant from Auer et al., described in Figure 3, works with gains [0,M] instead of losses [-1,0]. This adjustment makes Hedge applicable to partial information games and thus usable as a building block for the Exp4 algorithm.

Notation and terminology
The adversarial bandit game is formalized as a game between a player choosing actions and an adversary choosing the rewards associated with each action.

The game is parameterized by the number K of possible actions with integer $i : 1 \leq i \leq$ K.

All rewards belong to a unit inverval [0, 1].

The game is played in a sequence of trials t = 1, 2, ..., T.
On each trial t:

1. The adversary selects a vector $\mathbf{x}$(t) $\in [0, 1]^K$ of current rewards. The ith component x_i(t) is interpreted as the reward associated with action i at trial t.
2. Without knowledge of the adversary's choice, the player chooses an action by picking a number $i_t \in \{1,\ 2,\ ...,\ \mathrm{K}\}$ and scores the
corresponding reward x_{it}(t).
3. Since we consider a game with partial information (poker) the player observes only the reward x_{it}(t) for the chosen action i_t (in a full information game the player would observe the entire vector $\mathbf{x}$(t) of current rewards).

Let $G_A \doteq \sum_{t'=1}^{T} x_{it}(t)$ be the total reward of player A choosing actions $i_1, i_2, ..., i_T$.

We formally define an adversary as a function mapping the past history of play $i_1, ..., i_{t-1}$ to the current reward vector $\mathbf{x}$(t).

The measure of performance is *regret*, which is the difference between the total reward of algorithm G_A and the total reward of the best action (Exp4 measures against the total reward of the best expert instead of the best action).

Formally, we define the expected total reward of algorihm A by:
$\mathbf{E}[G_A] \doteq \mathbf{E}_{i_1,...,i_T}\left[\sum_{t=1}^{T} x_j(t)\right]$,
the expected total reward of the best action by:
$EG_{max} \doteq max_{1 \leq j \leq K} \mathbf{E}_{i_1,...,i_T}\left[\sum_{t=1}^{T} x_j(t)\right]$,
and the expected regret of algorithm A by $R_A \doteq EG_{max} - \mathbf{E}\left[G_A\right]$.

Figure 2: Formal definition of the adversarial bandit problem as defined by Auer et al. [3].

```
Algorithm Hedge
Parameter: A real number $\eta > 0$.
Initialization: Set $G_i(0) := 0$ for i = 1, ... ,K.

Repeat for t = 1,2, ... until game ends

    1. Choose action $i_t$ according to the distribution $\mathbf{p}$(t), where

       $p_i(\mathrm{t}) = \frac{\exp(\eta G_i(\mathrm{t} - 1))}{\sum_{j=1}^{K} \exp(\eta G_j(\mathrm{t} - 1))}$.

       Note: At time t, action i is chosen with a probability
       proportional to $\exp(\eta G_i$(t - 1)), where $\eta > 0$ is a parameter
       and $G_i(\mathrm{t}) = \sum_{t'=1}^{t} x_i(t')$ is the total reward scored by action i
       upto trial t.

    2. Receive the reward vector $\mathbf{x}$(t) and score gain $x_{i_t}$(t).

    3. Set $G_i$(t) := $G_i$(t - 1) + $x_i$(t) for i = 1, ... , K.
```

Figure 3: Hedge algorithm, source: [3], Figure 1 (slightly adjusted)

To see how Hedge works one simply follows the steps described in Figure 3. In step 1, Hedge chooses an action using the current weight vector, after normalizing. With the exponential function the Hedge algorithm makes sure that actions yielding high rewards quickly gain a high probability of being chosen. The η parameter controls how much emphasis is put on playing the action with the highest cumulative reward. A small η will cause the player to choose among all of the actions with near-uniform probability, a large η makes the player choose among the most succesfull actions with high probability. At trial t, Exp3 (Figure 4) receives a probability vector from Hedge which represents a distribution over the available strategies. According to the distribution vector, which is a mixture of the probability vector from Hedge and the uniform distribution, an action is selected. Mixing in the uniform distribution is done to make sure that the algorithm tries out all k actions to get good estimates of the rewards for each. Otherwise, the algorithm might miss a good action because the initial rewards and observations for this action are low and large rewards that occur later are not observed because the action is never selected. Parameter γ set to 1 corresponds to all actions being chosen uniformly on every round, $\gamma = 0$ corresponds to the most exploitive case where experts that were previously well-rewarded are highly favoured. After Exp3 receives the reward associated with the chosen action, it generates a simulated reward vector for Hedge. Hedge requires full information so all components must be filled in, even for the actions that were not selected. For the chosen action the simulated reward is calculated

```
Algorithm Exp3
Parameters: Reals $\eta > 0$ and $\gamma \in$ (0,1].
Initialization: Initialize Hedge($\eta$).

Repeat for t = 1,2, ... until game ends
    1. Get the probability vector p(t) from Hedge which represents
       a distribution over the available strategies.
    2. Select action $i_t$ to be j with a probability according to the
       distribution vector $\hat{p}_j$(t), which is a mixture of the vector p(t)
       from Hedge and the uniform distribution:
       $\hat{p}_j$(t) = (1 - $\gamma$)$p_j$(t) + $\frac{\gamma}{K}$.
    3. Receive the reward for the chosen action: $x_{i_t}$(t) $\in [0, 1]$.
    4. Generate a simulated reward vector $\hat{x}$(t) and feed it back to
       Hedge, where $\hat{x}_j$(t) = $\frac{x_{i t}(t)}{\hat{p} i_t(t)}$ if j = $i_t$ and $\hat{x}_j$(t) = 0 otherwise.
```

Figure 4: Exp3 algorithm, source: [3], Figure 2 (slightly adjusted)

by dividing it by the probability with which it was chosen. This compensates the rewards of actions that are unlikely to be chosen. The other actions all receive a simulated reward of 0 to make sure the expected simulated gain for an action is equal to the actual gain.

The goal of Exp3 is to get the total reward as close to the reward of the best single action. The goal of Exp4 is to get the total reward as close to the reward of the best expert. Exp4 achieves this by running Exp3 over the N experts using estimates of the experts' losses. The performance of any player is measured in terms of regret (see Figure 2[2]). Exp4 mixes the expert advice with the probability distribution over experts maintained by Exp3 (Figure 5, step2). In step 3 the second adjustment by Bard et al. is made: instead of the uniform exploration over action sequences (in the Exp3 case: actions) maintained by Exp4, the weight of each expert is forced to be at least some minimum value, bounding each experts weight in the mixture away from zero. This allows the guarantee that the acting strategy has non-zero probability on every action sequence played by any response in the portfolio. The necessity of this will be explained in section 5.2.3.

[2]The definition of regret compares the total reward of the algorithm to the sum of rewards that were associated with taking some action j on all interations. However, had action j acutally been taken, the rewards chosen by the adversary would have been different than those actually generated since the variable x(t) depends on the past history of play. This is quite difficult to interpret and will not be discussed any further in this paper. The interested reader is referred to the official paper by Auer et al. [3].

```
Algorithm Exp4
Parameters: Reals $\eta > 0$ and $\gamma \in$ [0,1].
Initialization: Initialize Hedge (with K replaced by N).

Repeat for t = 1,2, ... until game ends
    1. Get the distribution (over N strategies) $\mathbf{q}$(t) $\in [0,1]^N$ from Hedge.
    2. Compute the vector $\mathbf{p}$(t) as a weighted average (with respect to $\mathbf{q}$(t)) of the strategy vectors $\xi^j$(t): get advice vectors $\xi^j$(t) $\in [0,1]^K$, and let $\mathbf{p}$(t) := $\sum_{j=1}^{N} q_j$(t) $\xi^j$(t).
    3. Select action $i_t$ and assign it to be j with a probability according to the distribution vector $\hat{p}_j$(t), which is a mixture of the vector $\mathbf{p}$(t) from Hedge and the uniform distribution:
       $\hat{p}_j$(t) = (1 - $\gamma$)$p_j$(t) + $\frac{\gamma}{K}$.
    4. Receive reward $x_{i_t}$(t) $\in [0,1]$.
    5. Compute the simulated reward vector $\hat{\mathbf{x}}(t)$ as
       $\hat{x}_j$(t) = $\frac{x_{i_t}(t)}{\hat{p}_{i_t}(t)}$ if j = $i_t$ and $\hat{x}_j$(t) = 0 otherwise.
    6. Feed the vector $\hat{\mathbf{y}}$(t) $\in$ [0,$\frac{K}{\gamma}$]$^N$ to Hedge where $\hat{y}_j$(t) $\doteq \xi^j$(t) $\cdot$ $\hat{\mathbf{x}}$(t).
```

Figure 5: Exp4 algorithm, source:[3], Figure 4 (slightly adjusted)

Exp4 steps described in a high-level fashion:

- **Step 1 and 2**: Generate a probability distribution over a collection of expert strategies using a normalized exponential function of the expected total rewards for each expert(see Figure 3, step 1).
- **Step 3**: Select an action according to the weighted mixture of the experts.
- **Step 4**: Receive reward.
- **Step 5 and 6**: Compute expected reward for each expert which is added to the vector of expected total rewards.

We have seen that the balance between exploration and exploitation in Exp4 is controlled by the two parameters γ and η. The η parameter controls the amount of exploration among potential counter-strategies. The γ parameter controls the amount of exploiting the highest-rated strategies during the exploration phase.

Although it is beyond this paper to provide a proof its worth noticing that Exp4 comes with finite-time regret bounds on the expected per-time-step regret, guaranteeing the expected utility of the selected actions performs nearly as well as the best expert in hindsight (In [3] the reader finds exact bounds and their validation).

5.2.2 UCB

The UCB family of algorithms has been proposed by Auer et al [2]. The simplest algorithm, UCB1, can be summed up by the principle of optimism in the face of uncertainty. This principle simply states that even if we do not know the exact payoff of each action we can just make an optimistic guess, and pick the action with the highest guessed payoff. We want to be (almost) certain that the true expected payoff of an action is less than the prescribed upper bound. When choosing the arm that has the largest Upper Confidence Bound (= the highest expected utility) the agent is able to exploit that action while obtaining minimal regret. If the pick of the largest UCB is not correct the optimistic guess of the expected utility will quickly decrease and cause the agent to switch to a different arm. This creates a balance between exploitation and exploration. UCB1 maintains the number of times that each arm has been played in addition to the empirical means. Initially, each arm is played once. Afterwards, at random the algorithm greedily picks an arm.

We have seen that in some previous work upper confidence bounds are used in implicit modeling to select from a the portfolio of strategies. There are several problems with this approach. To start with, it might not be appropriate as UCB's regret bounds are for the **stochastic** bandit problem. Poker is an **adversarial** game where the opponent might manipulate the value of bandit arms. Furthermore, since UCB selects and acts according to a single strategy from the portfolio rather than a mixture, it cannot ensure that the support of the strategy being played is a superset of the supports for the portfolios individual responses. Therefore we cannot estimate the utilities of the off-policy strategies in the portfolio without potentially introducing bias. Bard et al. have shown that their small portfolio agent which uses Exp4 for strategy selection improves on an implicit modeling UCB-based agents when playing against the 4 small-portfolio mimics and when playing against all 2010 ACPC mimics, which is consistent with the notion of the apparent shortcomings in applying UCB to an adversarial bandit problem.

5.2.3 Variance Reduction Techniques

Since poker is a game that is heavily influenced by chance, the number of observations needed to generate a reliable utility estimation might become very large. Variance reduction techniques can help eliminate some of the noise induced by chance and reduce the needed number of observations. Bard et al. present a technique in [7] that is based on importance sampling and can be used to simultaneously evaluate many strategies while playing a single strategy in the context of an extensive game (importance sampling is a general technique from statistics that can be used to estimate the properties of a particular distribution, based only on samples generated from a different distribution). The estimator variance is reduced by imagining alternate observations taken by other agents. Observations are created for all possible private cards and early folding opportunities. Early folds are provably unbiased, and although the all-cards technique can create bias under partial information (due to card replacement effects and not knowing which cards the other agent holds) Bard et al. point out that prior results suggest this bias is small while providing substantial variance reduction ([7], table 3).

Being able to simultaneously evaluate many strategies while playing a single strategy is called off-policy estimating. It enables us to use each observation of the opponent's actions and outcomes to update estimates for the entire portfolio. Bard et al. have chosen to apply this technique of 'imaginary observations' and importance sampling to return low variance estimates of each response's utility. To be able to use this technique for off-policy estimating they have to make sure that the support of the acting strategy is a superset of the strategies whose utility is being estimated. This means that for any sequence of actions that can be realized by the off-policy strategy, the acting strategy must take this sequence of actions with some probability >0. Otherwise, there will exist some sequence of actions that the acting strategy will never take but the off-policy strategy will, leading to bias. This could be prevented by acting according to a strategy that mixes between all of the strategies in the portfolio (if any individual strategy would play an action, then the mixture will necessarily play it with non-zero probability).

6 Conclusion

This section provides an overview of the answers to the research question that were given in this thesis. Next, these results and their impliciations for the broader perspective of AI are reviewed. To conclude, the contribution of this thesis is discussed and some suggestions for future research are made.

6.1 Answering the research question

The goal of this thesis was to give insight in the two main approaches to opponent modeling in the domain of Texas Hold'em Poker, and the issues that arise in this area. We have discussed the differences between the two approaches and the way these differences affect the usability of an opponent model. These differences can be captured in one main point of difference: the usage of a specific model for the opponent's strategy. We have seen that with explicit modeling, an agent tries to discover weaknesses from the opponent by identifying the opponent's strategy. The agent then uses the model from the opponent's strategy to create an effective counter strategy. With implicit modeling, no such model is created. An implicit agent tries to find the most effective counter strategy for an opponent by using different counter strategies against the opponent. These counter strategies try to maximize the agent's utility by exploiting the opponent's weaknesess, but the exact nature of these weaknesses is of no concern to an implicit agent (as opposed to an explicit agent that builds an entire model to discover the nature of his opponent's weaknesses). This difference has a major influence on the usage of the opponent models. The online building of a model that represents the opponent's strategy introduced two major difficulties for explicit modeling:

- In such a complex domain as poker, the building of an accurate model requires a prohibitive number of observations.
- Computing a response strategy that is robust to modeling error may be impractical to compute online.

In section 3 we discussed the most-used solution for these two problems: applying a state-space abstraction technique. It became clear that this solution, although making explicit models usable in complex domains, is no perfect-fix and introduces some difficulties as well. Although having an explicit model that allows us to predict the opponent's actions and reactions is the ultimate goal, the question remains if it is possible to capture the full behavior of a dynamic opponent based on a few observations (in comparison to all decision points in the game). None of the efforts of applying explicit modeling to opponent modeling in poker have been able to overcome these difficulties for a full scale envrionment. Models always deviate from the more complex reality with all kinds of possible negative influences on the players utility.

Implicit modeling does not concern itself with the creation of a model for the opponent's strategy. This makes implicit modeling frameworks better suitable for complex online environments, where agents have to make decisions within tight time constraints.

The most important problems that arise within this approach are:

- Which (and how many) counter-strategies to consider for usage against the opponent.
- How to balance between searching for the most effective counter strategy and exploiting a single counter strategy to gain utility.

In the implicit framework, a lot of the heavy calculations can be performed offline. This is probably the property that has the main effect on the usability, it makes implicit modeling better suitable for large complex domains such as online poker than explicit modeling.

6.2 Implications towards the field of AI

As indicated in the introduction: poker presents a very interesting research field for Artificial Intelligence. This thesis tried to inform the reader about the two major approaches to opponent modeling in the domain of Texas Hold'em Poker. In the previous subsection I have reached the conclusion that implicit modeling seems the best approach to opponent modeling in large complex domains. To support this statement, an overview was presented of the implicit modeling approach as proposed by Bard et al. This approach avoids building an explicit model online, with positive results. They trained two agents on data from 2010 ACPC agents. These two agents are validated using the 2011 ACPC benchmark server. Note that both agents are unknown to the 2011 agents that they will encounter as they are trained against the 2010 agents. Both implicit agents would have won the competition. The Big-Portfolio agent did not win by a statistically significant margin. However, Small-Portfolio won the event by greater than the 95 percent confidence margin. The fact that the Small-Portfolio outperformed the Big-Portfolio agent seems to indicate that pruning the portfolio of redundant responses results in higher utilities.

Building on these positive results, Bard et al. have entered an agent, named Hyperborean_tbr, in the 2013 ACPC. Hyperborean_tbr is based on the implicit modeling framework that is discussed in this paper and performed very well: it placed second in the Total Bankroll event (together with a French agent named Feste). First place went to Marv, submitted by Marv Andersen. This agent consists of a neural net that is trained to imitate previous ACPC winners. These results certainly indicate that there is a lot to gain from the implicit modeling framework. For the field of Artificial Intelligence, this might mean a grand shift from the most used approach (explicit modeling) to the implicit approach. By leaving the explicit approach behind, we no longer have to deal with the problems that came with this framework. This might be a big chance on improvement. The improvement of pokerbots will help bring AI research closer to it's goal: creating intelligent agents. If the positive results shown by Bard et al. for implicit modeling provide any indication for future research contributions, there is a lot of progress to be expected for AI.

6.3 Contribution

This thesis was written in such a way as to be helpful for anyone that has an interest in opponent modeling, but has no idea where to begin with the existing literature. It should provide them with a solid basic knowledge about the two main approaches to opponent modeling. Also, I hope that this thesis has been able to raise appreciation for the advantages that were shown to come from favoring implicit modeling over explicit modeling in complex domains such as poker.

6.4 Future Research

Although it might very well be possible that most attention in this research area will be payed to implicit modeling in the upcoming period, I do not think explicit modeling has to be dropped just yet. But, in order to compete with implicit modeling agents, future research on explicit poker agents will need to focus on solving the problem of building and maintain a high dimensional model in real time without requiring many observations or significant prior knowledge.

Future research on implicit poker agents might concern itself with optimal methods for pruning the portfolio. As the Small-Portfolio agent outperformed the Big-Portfolio agent it might be fruitful to think of new ways of building a Small*-Portfolio agent that performs even better.

References

[1] The Annual Computer Poker Competition webpage, Retrieved January 20, 2014, from *http://www.computerpokercompetition.org*

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multi-armed bandit problem. In *Proc. of Machine Learning*, pp. 235 - 256, 2002.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund and R.E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, pp. 322-331, 1995.

[4] N.Bard, M. Johanson, N. Burch and M. Bowling. Online implicit agent modelling. In *Proc. of the 12th International Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 255-262, 2013.

[5] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Poker as a testbed for AI research. In *Proc. of Canadian Conference on AI*, pp. 228-238, 1998.

[6] B. J. Borghetti. Opponent modeling in interesting adversarial environments. Dissertation, ProQuestt, August 2008, Retrieved on January 28, 2014, from *http://conservancy.umn.edu/bitstream/11299/46071/1/Borghetti_Brett* %20*August*%202008*.pdf*

[7] M. Bowling, M. Johanson, N. Burch, and D. Szafron. Strategy evaluation in extensive games with importance sampling. In *Proc. of the 25th Annual Int. Conf. on Machine Learning (ICML)*, pp. 72-79, 2008.

[8] P. McCracken and M. Bowling. Safe strategies for agent modelling in games. In *AAAI Fall Symposium on Artificial Multi-agent Learning*, pp. 103 - 111, October 2004.

[9] A. Davidson. Opponent modeling in poker: learning and acting in a hostile and uncertain environment. Master Thesis, University of Alberta, 2002, Retrieved on January 28, 2014, from *http://poker.cs.ualberta.ca/publications/davidson.msc.pdf*

[10] Y. Freund and R.E. Schapire. A decision-theoretic generalization of online learning and application to boosting. IN *Journal of Computer and System Sciences*, pp. 119-139, 1997.

[11] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *Proc. Of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 533-540, 2011.

[12] M. Johanson, M. Zinkevich, and M. Bowling. Computing robust counter strategies. In *Advances in Neural Information Processing Systems 20 (NIPS)*, Proc. of the 21st Annual Conf. on Neural Information Processing Systems, pp. 721-728, MIT Press, Cambridge 2007.

[13] M. Johanson and M. Bowling. Data biased robust counter strategies. In *Proc. of the 12th Int. Conf. on Artificial Intelligence and Statistics*, pp. 264-271, 2009.

[14] M. Johanson, N. Burch, R. Valenzano, and M. Bowling. Evaluating state-space abstractions in extensive-form games. In *Proc. of the 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 271-278, 2013.

[15] J. F. Nash. Non-Cooperative games. In *The Annals of Mathematics*, Second Series, Volume 54, Issue 2, pp. 286-295, September 1951.

[16] J. Rubin and I. Watson. On combining decisions from multiple expert imitators for performance. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 344-349, 2011.

[17] J. Rubin and I. Watson. Opponent type adaptation for case-based strategies in adversarial games. In *Proc. of the 20th Int. Conf. on Case-Based Reasoning*, pp. 357-368, 2012.

[18] Texas Hold'em hands. The Poker Practice webpage. Retrieved January 20, 2014, from *http://www.thepokerpractice.com/hands/*

[19] K. Waugh, D. Schnizlein, M. Bowling, and D. Szafron. Abstraction pathologies in extensive games. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 781-788, 2009.

[20] 2012 World Series of Poker webpage. Retrieved January 20, 2014, from *http://www.wsop.com/2012/*

[21] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, Proc. of the 21st Annual Conf. on Neural Information Processing Systems, pp. 1729-1736, MIT Press, Cambridge 2007.