An Opponent Based Statistical Exploitation Module
And It's Benefits When Used With an Approximate Nash Equilibrium Strategy

Kevin Norris
October 2012

Supervisor: Ian Watson

Abstract

Opponent modeling strategies provide the ability to exploit weak players, but have the disadvantage of being exploitable to strong players. An approximate Nash equilibrium strategy on the other hand is difficult for opponents to exploit, but it is not able to exploit opponents. This dissertation examines the effects of combining an approximate Nash equilibrium strategy with an opponent based strategy. The system proposed in this dissertation uses a comprehensive frequency statistical opponent model and statistical exploits to provide opponent based actions. Statistical exploits were chosen because each individual exploit has few preconditions and it can easily be proven whether it applies to a given statistical model or not. This allows the system to only use opponent based actions when it is sure the action is not exploitable, ensuring the agent is difficult to exploit even while it exploits the opponent. For each opponent we build up a counter-strategy against them using all the exploits which apply to their statistical model. This counter-strategy is able to change during the match as the preconditions for new exploits are met and the preconditions for the exploits being used are no longer satisfied. This approach has shown promising results in our initial tests and could lead to a champion level player once the system is improved.

Contents

Glossary

Weak player: A player who's strategy can be captured in a generalized opponent model, these are usually players whose strategy is static, who play predictably, and who don't react to their opposition.

Strong player: A player whose strategy cannot be captured in generalized models, usually players who play non-static, unpredictable strategies and who react to their oppositions play style.

Fully opponent based strategies: Strategies employed by poker bots in which each action is dependent on the opponent model.

Partially opponent based strategies: Strategies employed by poker bots in which only some actions are dependent on the opponent model.

Statistical exploits: Ways in which one can change their strategy to take advantage of a poor play from the opponent, which you deduce from the frequency statistic

1 Introduction

This dissertation proposes an improvement to current poker bots. The concept is based upon the game of Heads Up no limit Texas Hold'em which is described briefly in section 2. Our concept looks at the problem of creating a Heads Up no limit Texas Hold'em agent which is able to exploit opponents while being difficult to exploit. We approach this problem by using an approximate Nash equilibrium strategy as the underlying strategy for its attribute of being difficult to exploit. To this we then add an opponent based component which utilizes an opponent model and expert defined exploitations. This addition creates a partially opponent based strategy which is difficult to exploit and is able to exploit opponents without making itself more vulnerable to exploitation than the underlying strategy.

We begin the dissertation by providing an overview of the field of opponent modeling, outlining several of the techniques that have been attempted in producing opponent based strategies and the challenges associated with opponent modeling. In this section we also present new terminology, as we feel the term opponent modeling is being used as an umbrella term. We define opponent based strategy and the components which are needed to make up a opponent based strategy, opponent modeling and opponent based action. Next we define our central research question and the motivation behind the approach taken. The motivation is split into two parts. Our initial motivation that prompted us to delve into the literature, and then the motivation we acquired after understanding the challenges and techniques explored in the fields of opponent modeling and computer poker to date. We then describe the implementation of the system, how the individual components operate independently and with one another. We describe our experimental methods and the results of our experiments. The dissertation ends with the conclusions that were reached, and possible future work to improve and extend the system.

2 No Limit Texas Hold'em

We describe briefly the game of Texas Hold'em focusing on two-player no limit Hold'em as our system has been specialized for this domain. If a game consists of only two players, it is described as being a heads-up match.

The game of heads-up no limit Texas Hold'em consists of four stages: pre-flop, flop, turn and river. During the pre-flop stage each player is dealt two hole cards, which only they can see. Two forced bets are contributed to the pot, these being the small blind (SB) and the big blind (BB) before any betting takes place. The big blind is usually double the value of the small blind. In the game of heads-up Texas Hold'em the dealer contributes the small blind and the non-dealer contributes the big blind. The dealer signifies the player who is first to act during the pre-flop stage of the game and last to act for each of the other stages of the game. The betting actions, which are common to all variations of poker, are described as follows:

- **Fold:** When a player abandons their hand, no longer committing any chips to the pot and giving up any right to contest the chips that make up the pot.
- **Check/Call:** When a player commits the minimum amount of chips with which he/she is able to continue to contest the pot. A check requires zero chips to be committed, and a call requires an amount greater than zero to be committed.
- **Bet/Raise:** When a player commits a larger number of chips than the amount necessary to continue to contest the pot, this is known as a bet. If a player is in the position where he/she must call a bet to continue, but then decides to invest more than the call amount in the pot, this is known as a raise.

In a no limit game a player may bet any amount they desire up to the total value of chips they possess. Once the betting in one stage of the game is complete and as long as no players have folded, play continues on to the next stage. Each further stage after the pre-flop stage involves the drawing of community cards from the shuffled deck of cards as follows:

- **Flop:** 3 community cards
- **Turn:** 1 community card

- **River:** 1 community card

In a standard heads-up no-limit poker game the chip stacks of each player would fluctuate between hands depending on who won the previous hand. To reduce the variance of this structure a variation known as Doyle's Game is played in this dissertation where the starting stacks of both players are reset to a specified amount at the beginning of every hand.

3 Literature Review

When considering the attributes required for successful poker play as identified by Billings [2002], we see that poker research has progressed by leaps and bounds in all areas but opponent modeling. Schauenberg [2006] "Of these required attributes, the one that remains the biggest obstacle to world-class play is opponent modeling", a similar sentiment was echoed by world-class poker player Gautam Rao after testing PsOpti [Billings, 2003]. There have been a number of opponent modeling methods attempted over the years and though they have differed greatly in some respects they have all had one thing in common: They all used statistics on the opponent's action frequencies to create their models. We will begin by introducing a few suggested terminology alterations. From there we will discuss statistics in regards to the Texas Hold'em poker opponent modeling domain. We will then go over seven opponent modeling methods that have been tried, giving a brief overview and discussing the merits of each. After which we will describe the Polaris system, which took a game theory approach to altering a Nash equilibrium strategy such that it was able to exploit opponents.

3.1 Opponent Modeling

The term "Opponent Modeling" has been ambiguously used in the literature to mean playing an opponent based strategy. This is not intuitive; intuitively one would think that the opponent model would be the collection of information about the opponent that represents the way in which the opponent plays. From this definition opponent modeling would just be the creation of this model, but in the literature it has been used to also incorporate the way in which the models are used to play an opponent based strategy. I believe that for ease of understanding, the ambiguous usage of the term "Opponent Modeling" should be broken into three terms:

1. **Opponent Model:** the set of information which represents the opponent's play style.

2. **Opponent based actions:** the way in which you use the given Opponent Model to determine actions that are suited towards the opponent you are facing.

3. **Opponent based strategy:** The combination of creating an opponent model for your opponent and using this model to determine opponent based Actions. There are two types of opponent based strategies that can be employed:

    a. **Fully opponent based strategy:** The agent uses the opponent model to play an opponent based action for every action.
    b. **Partially opponent based strategy:** The agent plays opponent based actions only some of the time.

3.2 Statistics

Poker is an incomplete information game, as one does not have the ability to see the opponent's hole cards, it is also a stochastic game as players have no control over which cards are dealt. This makes it very difficult to know where you stand in the hand, what your opponent has, what they will do or how they will react to your actions. If you look at a poker game state in isolation the only information you have is your cards, the community cards and the size of the pot. This is not enough information to make a decision with any confidence about it being the correct decision. To make a better decision we need hindsight; what actions have occurred that led to this state? With this information we can already make a much better decision. What if we look even further back? What did the opponent do the last time they were in this situation? What are the opponent's action frequencies in this situation? This is the basis of opponent modeling, keeping statistics on the opponent for use in later similar situations to improve the quality of decision making, to maximize profit.

The observations you make during your game play are the only indications you have about your opponent's play. Keeping statistics on your opponent's action frequencies gives you a much better idea of how your opponent will act and react in any given situation as you will have a

probability distribution for each action your opponent can make in each situation. Schauenberg [2006] describes an observation model as a statistical model based on all that is observed. He uses this type of model in [Schauenberg, 2006] and brings up the major concern with statistical models, that without enough observations of a situation the action distribution statistics can be nowhere near their true value. When one has observed a limited number of hands at a specific game state it is very likely that the frequencies observed are skewed and do not correlate with the opponent's true frequencies at that game state. Thus far there have been two main ways of dealing with this problem, using priors and waiting to use the frequencies of a game state to impact the action decision until that game state has been observed enough times to make the probability distribution statistically significant. We will discuss examples of these as we describe opponent modeling implementations in section 2.3.

There is a second major problem with statistical models, the fact that opponents will often alter their strategies throughout the game, invalidating the statistics that have been observed. If the opponent's play does not match the frequencies we are using to make our decisions they will be exploiting our incorrect information and causing us to lose money. This is known in poker as “exploiting your image”: for example you play very tight to start with, only showing down big hands and very rarely bluffing, then once you believe the opposition has pegged you as a tight player you start bluffing to exploit the fact that they believe you only raise with big hands, giving you much more fold equity for your bluffs. This can also be done in the reverse, creating a loose image and then tightening up or on a smaller scale for example re-raising pre-flop only with your high value hands and then later bluff re-raising pre-flop a lot of low value hands. The methods for dealing with this problem are usually “simple methods such as decaying histories to accommodate opponent drift” [Southey, 2005].

Full observation models are rarely used in opponent modeling research. The statistics used are usually limited in number and simplistic in nature, each generalizing greatly over a large set of game states. Researchers are using only the simple statistics such as: raise, call, fold percentage per street, VPIP (voluntary put money in pot) how often you call or raise pre-flop, PFR(pre-flop raise) and aggression factor. Commercial heads up displays or HUD’s, which are prevalent these days in online poker, provide a full observation model and can provide one with any statistic that

can be thought of because they save every hand. Online players have been using HUD's for some time now and have discovered many more statistics that are very helpful for determining an opponent's play style.

We believe opponent models can be greatly improved by adding some of the statistics that have become popularized in online play through the HUD's. For example 3bet, how often the opponent re raises a pre-flop raise, and fold to 3bet, cbet, how often an opponent bets the flop after raising pre-flop, and fold to cbet and various other statistics are now a staple for online player's HUD's to allow them to get a better idea of opponent's play styles. Adding in more statistics, while possibly making opponent models more expressive and better at capturing the nuances of an opponent play style, will also increase either the computation or complexity or both of the program needed to extract relevant data from the statistics to make betting decisions. This leads to the tradeoff between program complexity and required computation verses opponent model expressivity.

3.3 Opponent based play techniques

In this section we will review a selection of Opponent based strategy techniques organized by bot. We will first go over each bot's opponent model, then the way in which the bot uses the opponent model to determine opponent based actions. We will end the review of each bot with an overview of the entire bot and a discussion about the opponent based strategy technique employed by the bot.

Loki-1

We begin by looking at the first iteration of the Loki bot to use specific opponent modeling as described by Billings [1998]. Specific meaning that the opponent model used for each opponent is unique and based on their action frequencies.

Opponent Model:

The opponent model consists of 4 components:

1. A weight table of each of the possible combination of hole cards. The table gives the probability that the opponent would have played the hand to the present point in the game. The weights are updated after each opponent action.
2. A table of betting frequencies for various stages of a hand. This is computed by counting the number of times each player folded, called, raised in each of 12 contexts, the contexts being a combination of the betting round, and the number of bets to call.
3. A threshold or median hand strength calculated using the betting frequencies after each opponent action.
4. A posteriori probability of each possible holding, given its connection to the community cards. Calculated using the previously calculated threshold.

How Opponent based actions were determined:

A hand evaluator which is given the game state and the opponent model uses enumeration techniques to compute hand strength and hand potential for any hand. A hand assessment of the bots current hand provided by the Hand Evaluator is then fed into the Bettor which uses a set of expert-defined rules along with the hand assessment data provided and the public game state to determine whether to fold, call or raise.

Overview of Loki-1:

Loki-1 uses a weight table of each possible combination of hole cards which pre-flop is based on the opponent's call, raise, and fold frequencies. Once the flop has occurred re-weighting begins. After each opponent action the probabilities for all hands are updated in a process called re-weighting. The opponent modeler calls the hand evaluator once for each possible hand and increases or decreases the weight for that case based on the new information. During this process the Hand Evaluator uses the Weight Table values to bias the calculation, giving greater weight to

the more likely hands. When the bot is at a decision point it has the Hand evaluator evaluate its hand and call the Bettor which uses this evaluation along with the set of expert defined betting rules to make a betting decision. Loki-1 plays a fully opponent based strategy since the Hand Evaluator is used to determine each action, and the hand evaluator uses the opponent model to determine its evaluation.

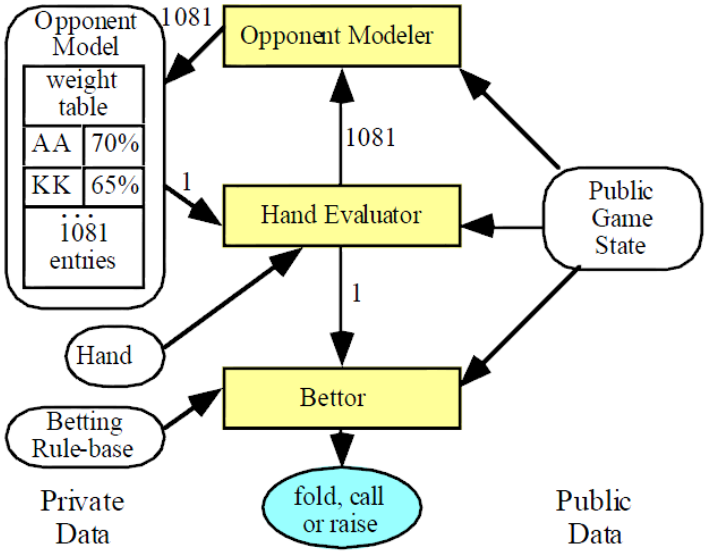

Figure 1. The architecture of Loki-1 after Billings [1999].

Discussion about the opponent based strategy technique employed by Loki-1:

Loki-1's general concept seems to be a very good idea. Narrowing down the range of hands an opponent can have at a particular game stat based off the opponent's previous actions and play style is popular among human players and is known as hand reading. However the individual components used to allow for opponent based play were very simplistic. [Billings, 1998] acknowledged themselves that the way in which they used and collected observed statistics was crude, and that "... much of the relevant context was ignored for simplicity". Billings [1999] said there were a number of problems with the original approach; these being: Loki was difficult to maintain and improve due to the use of expert knowledge in much of the bot. "The Bettor returned a single value (fold, call, raise), which does not reflect the probabilistic nature of betting decisions". Also the Opponent Modeler does not distinguish between the various actions that an opponent might take. [Davidson, 2000] adds that the framework was simplistic in the fact that it

did not account for many relevant details. These included the number of active opponents, and betting positions. These are all major concerns but many of them were rectified at least to a degree as the university of Alberta poker group improved Loki over the years. Bellow, I will be describing the various improvements to Loki-1 that produced the new bots Loki-2 and Poki.

Loki-2

Loki-2 described in [Billings, 1999] improved upon Loki-1's Opponent based action determination by introducing the probability triple and simulation with selective sampling to the Loki bot. I will give an overview of both improvements starting with the probability triple.

The probability triple has three values [f,c,r] which equate to the probability of choosing the fold, check/call and bet/raise actions. Each of the three actions has a value between 0 and 1 and the triple together sums to 1. The probability triple has been incorporated in most aspects of Loki-2 and the Triple Generator is at the heart of Loki-2. The triples are used by the Action Selector to choose a course of action, by the Simulator to choose actions for simulated opponent hands, and by the Opponent Modeler to update the opponent weight tables. To generate a triple the Triple Generator takes a two-card hand and calls the Hand Evaluator which evaluates the hand in the current context. The Triple Generator then uses the resulting hand value along with the current game state and expert-defined betting rules to compute the triple. To update the Weight Table using probability triples the Opponent Modeler calls the Triple Generator for each possible two-card hand. It then multiplies each weight by the entry in the probability triple that corresponds to the action the opponent just took.

Loki-2 uses a Simulator to enhance the quality of the actions selected. Every time Loki-2 faces a decision it calls the Simulator to get an estimated expected value (EV) of each betting action. When the Simulator is used it replaces the Action Selector and the action chosen is the one with the highest EV. A simulation plays out a hand a number of times from the current state of the game through to the end. Two trials are considered, the first performing a check/calling action at the first decision point and the second bet/raising; it is not necessary to simulate folding

as it is considered to have an EV of 0 as the bot makes no future profit or loss. For each trial the hand is simulated to the end and the amount won or lost is determined. The average over all the trials is taken as the estimated EV of each action. Simulating each hand that each opponent might have is far too computationally expensive as the branching factor is extremely large due to there being multiple players, 3 action choices per decision point, many possibilities for community cards, etc. Loki-2 solves this problem by using Selective Sampling from the Weight Table to determine which of the possible opponent hands will be simulated. Using the Weight Table also allows the Triple Generator, which determines the opponent's actions during the simulation, to compute actions which the opponent is likely to perform. The use of simulations in this way was shown, by Billings [1999], to greatly magnify the quality of the evaluation function of Loki and allowed higher performance to be gained with minimal expert knowledge. Loki-2 as Loki-1 before it, plays a fully opponent based strategy since the opponent model impacts each action, meaning the action is always opponent based.

Poki

Davidson [2002] introduces the original Poki bot along with a number of later versions of Poki that improve upon the original. Poki_1 is a modular object-oriented reimplementation of the Loki AI which allows it to easily use different implementations of the major modules. Poki's later variants do just that, using different betting strategies and predictors. I will discuss the general updates that have been added to the Loki framework to create Poki and will then go over the various betting strategies and predictors available to Poki.

Where Loki's pre-flop strategy was based on the opponent's pre-flop action frequencies, Poki's pre-flop strategy is an expert based system built on a set of tables containing the expected income results for each hand. The reweighting process has also been updated; each value in the weight table is now updated by the probability that, given those cards, the opponent would have taken the action the opponent took. Poki also added a noise-factor to the reweighting process. The noise-factor represents the amount of uncertainty the bot has about the opponent's actions reflecting the information about the cards they hold. Loki had a large source of error in multi-

way pots as it used a field array as the weight table. This field array represented the average of all opponents' weight tables. Poki has replaced this method with computing the hand strength for each opponent and multiplying these values together. If this is too computationally expensive, Poki creates a field array which is the combination of the maximum of each opponent's weight tables instead of the average.

The first hint at Poki was in [Davidson, 2000] where Davidson described improvements upon Loki-2 through the addition of two features, these being previous action and previous amount to call and renamed the bot Poki. An artificial neural network (ANN) was created with the goal of predicting an opponent's next action. All information available to previous Loki bots was used as input for this neural network along with numerous new inputs that included previously unused contextual information. The network was trained and the weights for each of the various inputs were examined to determine which inputs affected the prediction the most. It was found that the statistics of previous action and previous amount to call were particularly strong features when determining opponents' actions. This lead to the addition of these new features into a new bot called Poki, which was shown to outperform Loki-2 significantly. As the Loki bots before them, the Poki bots play a fully opponent based strategy.

Betting Strategies:

Poki's basic betting strategy is a formula-based system which uses the hand evaluation as input and outputs an appropriate betting action. It is sensitive to input accuracy; if the opponent modeling is poor the strength and potential will be badly estimated. It is not very different to the formula based system employed by Loki-2.

Selective sampling and Simulation Betting is another of Poki's betting strategy's. The idea behind this strategy is to probe, to the leaves, for the most probable hands instead of doing a comprehensive but shallow search. At the time of simulation, Poki does not know the cards of the opponents, or the future board cards, so it probabilistically assigns cards to each opponent in each trial of the simulation. The weight table for each opponent is used to bias the selection of

cards to match the hands the opponent is likely to hold. Davidson [2002] claims that “after simulating several hundred trials, the average amount won or lost by calling or raising will typically converge to a stable estimate of the expected values (EV’s) of each action” For each trial the hand is simulated twice to determine the expected value of a check/call or a bet. Folding does not need to be simulated because folding gives an expected value of zero. This technique is beneficial as it is not a fixed strategy, it depends on the opponent model allowing it to shift dramatically and adapt to any mix of different opponents. It also saves computation in that properties such as implied pot-odds, draw odds and the like don’t have to be computed as the simulations uncover this information naturally, producing only a simple EV for each action. Simulations can uncover complex strategies without the need for any expert knowledge. The difficulty with this betting strategy is that it is entirely dependent on having a quality opponent model to ensure good results, but then so is the previous betting strategy.

Predictors:

The expert system predictor uses a set of rules to make rational choices on behalf of the opponent during simulation. This assumes the player will play somewhat according to the rules in the system, and is referred to as a "generic opponent model" as it is the same for all opponents.

The statistical predictor uses the opponent's history of actions to make predictions, using opponent action frequencies like Loki-2, and is a "specific opponent model" as it is unique for each opponent.

The neural network predictor takes 18 inputs which give it the contextual, mathematical and opponent based information it needs, from "Poki is in the hand" to "estimated Hand strength for the opponent" to "immediate pot odds" and everything in between. A new neural network is trained for each opponent and it can either: remember opponents and continue to use the previously used neural net, or it can treat each session as different and start from random weights again.

The Meta-Predictor uses a combination of five different predictors, uses multi-predictor voting and selects the action which has the highest weighting. Each predictor in the Meta-Predictor has an accuracy which is dynamically tracked by Poki. The accuracy of the last n actions of a player are used to evaluate each predictor. The votes of each predictor are weighted based on the predictor's accuracy. The predictors used by the Meta-Predictor are as follows:

- Statistics I: A statistical predictor which uses basic action frequencies as in Loki-2 but with enhanced context found in [Davidson, 2000]
- Statistics II: Similar to Statistics I but is more sensitive in that it only keeps statistics for the last 40 observed actions in each context class
- Expert Formula: Poki's formula based betting strategy
- Neural Network I: A neural network predictor who's training data consists of all actions of the opponent that have ever been observed
- Neural Network II: A neural network predictor who's training data consists of only the actions of the opponent that have been observed in the current session of play

Discussion of the predictor results:

Predictor	Classic	Incremental
Always Call	57.4%	57.4%
Statistics Loki-2	57.5%	57.5%
Statistics I	60.4%	60.6%
Statistics II	60.7%	60.9%
Expert Formula	54.7%	54.9%
Neural Network	68.4%	66.1%
Meta-Predictor	70.4%	69.5%

Table 1. Poki, average predictor accuracy after Davidson [2002].

The experiments presented by Davidson [2002] show that the Meta-Predictor is able to predict the opponent's actions more accurately than the others. This is as one would expect, the use of a combination of various predictors and the marginalization of the inaccuracy found in them

through weighting should outperform any of the individual predictors. The results, however, conclude that the neural network predictor performs only slightly worse than the Meta-Predictor and this is somewhat surprising. In order for a neural network to produce relevant outputs it would have had to have seen a significant number of training examples. As the number of nodes and levels in a neural network increase, the number of training examples that need to be seen before the weights are distributed meaningfully also increases. The neural network predictor has 18 input nodes 4 hidden nodes and 4 output nodes, due to this I would imagine that a rather large amount of training cases would have to be observed before the network returns any meaningful results. The paper, however, does not discuss how many training cases the network has seen from the opponent before the evaluation took place. The network's accuracy would most likely be far lower if the opponent was previously unseen.

The results show that getting an accuracy of 57.4% is trivial as a bot that always predicts a call is able to do that. Only the Neural Network predictor is significantly better than the trivial amount. If this accuracy is suspect due to an incomplete description of the learning rate and amount of training data the neural network has received, then the accuracy of the Meta-Predictor is also suspect as in this example it would be weighting the neural networks votes much higher than the others.

Bays Bluff

Southey [2005] describes an opponent specific strategy technique based on Bayesian probability and the use of priors. This is the only opponent specific strategy discussed in this literature review that does not use frequency statistics. This distribution is similar to what can be found in the literature, the overwhelming majority of research on opponent based strategies use some form of statistics in the opponent model and or the opponent based actions. The opponent based strategy outlined by Southey [2005] highlights why this is the case. If one does not use statistics, it is necessary to have very good prior knowledge of the opponent, which is not feasible in the real world.

How the opponent specific strategy is carried out

The opponent specific strategy has two key problems which it looks to solve. The first one being: "inferring a posterior over opponent strategies given a prior distribution and observations of their play" [Southey, 2005]. The posterior distribution summarizes the current state of knowledge about all the uncertain quantities in a Bayesian analysis, this is the opponent model. To do this a tuple was created which is able to denote any possible hand, it includes all hand information, the community cards, the betting and the opponent's cards if a showdown was reached. Using this tuple they came up with algorithms to calculate the probability of a particular showdown hand occurring given the opponent's strategy, and the probability of a particular fold hand occurring given the opponent's strategy. The opponent's strategy is given by the prior distribution. Given a set of observations they apply Bayes' rule to update the prior distribution using the previously mentioned algorithms to determine how each observation updates the prior distribution. The second problem the paper solves is "playing an appropriate response to that distribution" [Southey, 2005]. They give a number of options: Bayesian Best Response, Max A Posteriori Response, and Thompson's Response. The response is computed at the beginning of each hand and played for the entirety of the hand. We will only cover Tompson's Response as it is the only one they were able to use on the domain of Texas Hold'em, due to it being the least computationally expensive. The rest were only used on the far smaller domain of Leduc Hold'em which is described in detail in [Southey, 2005]. Thompson's Response samples a strategy from the posterior distribution and plays a best response to that strategy. Sampling the posterior directly is indicated to be potentially difficult. To overcome this difficulty they "... sample a set of opponent strategies from the prior, compute their posterior probabilities, and then sample one strategy according to those probabilities" [Southey, 2005]. The best response is determined in regards to the selected opponent strategy, which determines the opponent specific actions. Since a best response is calculated and the played against the opponent, each action is an opponent based action, so this bot plays a fully opponent based strategy.

Prior's

Bayesian approaches require priors, a prior should capture our beliefs of the opponent's strategy. The resulting performance and efficiency of a Bayesian approach depends on the choice of prior. Southey [2005] states that "The form of the prior also determines the tractability of (i) computing the posterior, and (ii) responding with the model". Due to this and the considerable size difference between the domains of Texas Hold'em and Leduc Hold'em they used a different prior for each domain. For Leduc Hold'em an Independent Dirichlet prior was used and for Texas Hold'em they used an Informed prior. Southey [2005] found that for opponents drawn from their priors the posterior captures them rapidly and the responses are able to exploit the opponent quickly in both Leduc and Texas Hold'em. The trouble with this is that a lot of prior knowledge is needed either in the form of accurate priors or many observed hands. This is unrealistic as one does not normally play against only known opponents in a poker game.

Vexbot

Vexbot is described in [Schauenberg, 2006], it is a poker bot built around a game tree, an expectimax search tree to be exact. An expectimax search tree can be thought of as a compacted imperfect information game tree. This is because the tree only considers the game from one player's perspective, allowing all nodes making up each of the player's information sets and the subtrees rooted at those nodes to be merged together, according to which information is observable. Each of the expectimax tree's nodes in essence represents a group of nodes in the imperfect information game tree that are indistinguishable to the player, and the edges represent a group of edges which are also indistinguishable. This merging creates an implicit probability distribution at each node over the information that occurred but has been kept hidden and remains unknown.

Opponent model

Schauenberg [2006] states "For opponent models to be used within the expectimax action-selection search ..., they have to be able to supply three different types of information as seen from the decision-maker's perspective:

1. probability distribution over observed opponent actions at an opponent decision node,
2. probability distribution over observed chance event outcomes at chance nodes, and
3. probability of the decision-maker winning a showdown"

The opponent model provides the first type of information by keeping count of each observed action at that decision node. These counts are then used to construct a probability distribution for observed opponent actions by dividing each possible actions count by the sum of all three. The second type of information needed is assumed to be uniform in frequency. Although this is not always the case against all opponents this assumption is stated to be "small compared to errors present in the opponent model" [Schauenberg, 2006]. To determine the third type of information counts are kept of which hands the opponent has revealed at the specified node in the past. The probability of wining is estimated by summing up the total number of observed hands that are worse than the player's current hand along with half of the observed hands that tie and then dividing this sum by the total number of hands observed.

Generalization

Schauenberg [2006] recognized that the number of distinct scenarios in which opponent modeling information is needed in Texas Hold'em is very large and that this would make it difficult to play an opponent based strategy early in the session as the opponent model would need many games to be played before it becomes effective at modeling the opponent. This would lead to possibly substantial losses until the opponent model got enough experience to be effective. This problem was combated through generalization of the data observed in one situation to other similar situations.

Generalization of action frequencies

To calculate an approximate opponent's action frequency at a given opponent decision point a context tree was created. This tree is able to represent all possible betting strings, where a betting string is the sequence of actions that have occurred thus far in the game. Each node in the tree represents an action and the frequency of the various actions which can occur after it. To determine the action frequency one starts at the root of the tree and traverses it according to the current betting string, the node that you arrive on then has a count for the number of times each of the three actions have occurred after it and the frequency is calculated from these counts. This method disregards all board card information and is simplistic in that it does not allow for a high degree of generalization.

Generalizing the estimation of winning at showdown

To generalize the estimation of winning at showdown a histogram is kept for each possible unique betting string and position combination for the opponent that represents a show down. These histograms represent the hand strengths that the opponent has shown down in the past, given the betting string and positions. The player’s hand strength is computed and compared to this histogram to determine how often he would have won, lost or tied at this showdown in the past. A distance function was created to define how similar observations in one showdown context are to observations in another to allow for generalization of the data early on. To keep things simple only ten levels of similarity where defined. The bot combines show down observations by starting with the most similar observations and successively adding more distant ones until the total number of observations at a showdown reaches a predefined threshold. The data is then combined using weights, where the weight an observation is assigned depends on its similarity, with the highest similarity being given the most weight.

Defaults

Although there are generalization strategies in place, there is still going to be a period of time when there are very few or no observations. This is a large problem for this type of modeling which is overcome by using default opponent modeling information in these cases. Four

showdown hand rank distributions are defined, which are chosen depending on the amount of betting an opponent did leading to a particular showdown. A simple heuristic rule base is used to assign default observation probability triples when there is no data for observed opponent frequencies. Schauenberg [2006] focused on opponents that do not change their strategy, so opponents that switch strategies once vexbot has built up a model can easily exploit it.

Using Expectimax trees to implement an opponent based strategy is a novel and interesting idea, but we believe it relies too much on the opponent model. Every action is based only on the opponent model and this is a problem they have recognized themselves, it greatly hampers the bots performance while it is building up enough information to create a meaningful opponent model and although the defaults help they are little more than a band aid on a gaping wound. Vexbot also required the opponents to have stationary strategies, which is unlikely when playing against humans. This could be fixed by information decay, but information decay without a strong strategy to fall back on weakens your opponent based strategy overall. This may still be the better option as strategies switching could exploit Vexbot tremendously once the bot has acquired enough history. Although Vexbot has a period of time where it is not playing an opponent based strategy while it populates its model, it still plays a fully opponent based strategy once it's model has been sufficiently populated. The goal of Vexbot it to play a fully opponent based strategy and the defaults are only there to help with the initial play when the opponent model is sparse. Due to this we classify Vexbot as playing a fully opponent based strategy.

Game-Theory based opponent modeling

Ganzfried [2011] describes an algorithm created to play opponent based strategies in large extensive-form games of imperfect information. They wanted to build an algorithm which could easily generalize to other settings and therefore relied on game-theoretic concepts such as Nash equilibrium and best response instead of game-specific information and features. Their algorithm assumes it has no prior knowledge of the opponent's strategy and operates online (in real time). The algorithm showcased in this paper is called Deviation-Based Best Response (DBBR). It creates an opponent model based on the deviations it finds between the opponent's strategy and a

pre-computed approximate equilibrium strategy. It then computes and plays a best response to this model in real time. The construction of the opponent model and the computation of a best response both take linear time in proportion to the size of the game tree, and they stated these processes "... can be performed quickly in practice" [Ganzfried, 2011].

Computing the Opponent model, DBBR

The first step in the DDBR algorithm is the creation of an approximate equilibrium of the game it will be used on, which is done offline. When the game begins, the opponent's action frequencies are recorded at various public history sets. These are then used to compute posterior action probabilities for the opponent, which are the probabilities the opponent chooses each action at each public history set. The probability the opponent is in each of the hand rank buckets at a given public history set is computed given the model of the opponents play so far, which consists of the opponents action frequencies and how they differ from the equilibrium strategies frequencies. These probabilities are referred to as the posterior bucket probabilities. The algorithm then computes the full opponent model by comparing the difference between the opponent's posterior action probabilities and those of the equilibrium strategy. It does this by iterating over all the hand rank buckets and based on the differences noted previously, shifting weight away from the action probabilities in the equilibrium associated with each bucket until a strategy is obtained that is consistent with the model of the opponent's action probabilities. The opponent model now also consists of the strategy which has been calculated for the opponent, which is in the same format as the equilibrium strategy.

There are a number of algorithms for computing the opponent strategy which alter the weights for the buckets differently and are discussed in the paper, but the one that was selected is their own custom weight-shifting algorithm. It works by greedily adding or removing buckets to the equilibrium strategies range for performing each action in each public history set until the opponent's observed frequency is produced. Once an opponent model has been created the algorithm iterates over all the public history sets, and computes a best response to the opponent models strategy computed earlier.

In order to improve the speed of the algorithm they suggest it could be run only every k hands. Ganzfried [2011] also states "we may want to start off playing the equilibrium σ* for several repetitions so that we can obtain a reasonable number of samples of the opponent's play, rather than trying to exploit him immediately".

Discussion

It is difficult to comment on the algorithm because everything is given at a very high level and few details are discussed. What frequency statistics they are using to define the equilibrium and opponent strategies would be interesting, since they say they wish to stay away from any game specifics yet they must recorder game specific statistics. The way in which the individual statistical differences alter the strategy is also game specific and would be useful to know when attempting to compare this opponent based strategy to others. The paper however keeps everything very high level and Ganzfried [2011] remarks that their algorithm "... achieves significantly higher win rates against several opponents - including competitors from recent AAAI computer poker competitions - than an approximate equilibrium strategy does". So however they handled all the game specific considerations, it worked. The bot utilizes a fully opponent based strategy, since it plays a best response computed against the opponent model, which is the calculated opponent strategy.

Opponent type adaptation

Rubin [2012], builds on earlier work where he describes the case based reasoning poker bot he built [Rubin, 2009]. He has updated it by adding in an opponent based strategy through the use of frequency statistics and adaptation. The opponent model is made up of 16 numerical values, one for the frequency of each of the four actions: fold, check, call, and bet, for each of the four betting rounds. This vector of 16 values represents the entirety of the opponent model. The opponent based play is created through adapting the solution vectors which are returned by the

case based reasoning (CBR) poker bot during the hand. The solution vectors are probabilistic action vectors, they are made up of three values (f, c, r) where f is the probability of performing the fold action, c is the probability of performing the check or call actions and r is the probability of performing the bet or raise actions. The adaptation shifts these values to make the solution vector more aggressive by increasing the probability to make aggressive actions or more defensive by increasing the probability to make more defensive actions.

To determine how the solution vector should be altered based on the opponent model to produce the best outcomes, the paper introduces the concept of opponent types. Classifying opponent into various opponent types is a popular abstraction used by online poker players to determine how they should play against an opponent when they have little experience against the player. Opponent types are in essence generalized play styles which can be determined through some limited statistical information. A number of opponent type models are created for which the aggression response trends are known. The aggression response trends are models which describe the impact various levels of adaptation has on the performance against a particular opponent model. During game play the betting frequencies for the opponent are recorded and the opponent model is used to classify the opponent as a particular opponent type. Once the type is known, an aggression trend is associated with the opponent and an algorithm is used to determine the adaptation factor that will be used for each hand.

The algorithm probabilistically selects the adaptation factors based on how much they were able to improve upon not applying adaptation at all. To make sure the bot is able to cope with opponents that shift their play styles only the latest M hands of play effect the frequencies. A learning period is also given to the bot, during the first M hands no adaptation occurs, to make sure the frequencies are statistically significant. With the addition of adaptation the bot plays a fully opponent based strategy. This is because once the initial M hands have been played it alters every action based on the aggression trend of the opponent type which the opponent model is most similar to, meaning it uses opponent based actions for every action thereafter.

This is a novel idea, based upon the case based reasoning concept of abstraction which fits nicely with the rest of the bot. This implementation adds a very generalized opponent based strategy on

top of a strong poker bot. Due to its generalizations of opponents through the limited frequency statistics of the opponent model and the classification of opponent types it increases its win rate reasonably for the most part over all of the bots it tested against. It was not able to fully exploit the weak players and it was not able to guarantee a positive or neutral result against all opponents but then again such results would not have been expected as they are not possible with this degree of generalization. The results that were shown indicate that even generalized information can give good insights into how an opponent plays and simple adaptation can have a largely positive impact on win rate.

3.4 Polaris

Polaris incorporates a collection of four poker bots. The first uses an approximate Nash equilibrium strategy. The second computes an abstract best response against an opponent. The third is a compromise of the first two and the fourth is a team of agents consisting of several of the previous types of agent.

Counterfactual Regret Minimization

The approximate Nash equilibrium strategy is computed using a new technique described by Johanson [2007] called Counterfactual Regret Minimization. It is different from other equilibrium approximations in that it requires memory linear in the number of information sets not game states. This allows them to solve much larger abstractions than were possible with the previous methods, which allows them to produce equilibrium strategies that are closer to the real game's Nash equilibrium. These strategies do not try to exploit opponents instead they try to minimize their own exploitability, making them robust in that they are difficult to defeat.

Frequentist Best Response

The method for calculating an abstract best response used in [Johanson, 2007] is called Frequentist Best Response. It is a new technique created by Johanson et al. and it addresses the three drawbacks associated with calculating an abstract game best response, these being:

1. Knowledge is required of the abstraction in which the opponent plays.
2. The strategy of the opponent is required.
3. The abstract game best response must be constructed in the same abstraction that the opponent plays in.

The counter-strategies created by this method are brittle. Brittle meaning they perform well against the opponents they are designed to defeat, but can lose badly to others including opponents which are weak or similar to those trained against.

Restricted Nash Response

The first method for creating a poker agent creates an agent that is difficult to defeat but does not win very much. The second method creates and agent that can exploit specific opponents, but is easily defeated by opponents it is not designed to defeat. Regret Minimization is a technique created by Johanson [2007] which attempts to find a compromise between these two extremes, creating agents that exploit particular opponents or classes of opponents and still providing a bound on their exploitability. This strategy is constructed by finding a Nash equilibrium in a restricted game, where the opponent must play according to a fixed strategy with probability p. p is chosen when creating the strategy and determines the proportion of time the opponent must use the fixed strategy. p ranges between zero and one, a p of zero meaning the opponent never plays the fixed strategy so a Nash equilibrium is computed, and a p of one meaning the opponent only uses the fixed strategy so a best response is computed. All values for p in-between zero and one represent a tradeoff between exploitation and exploitability.

Instead of constructing the usual two agents who play and adapt to one another for millions of hands to approach a Nash equilibrium, three agents are used for computing the Restricted Nash

Response strategy. One that will learn the Restricted Nash Response, and two for the opponent: a learning agent and a static agent. During the millions of games the RNR agent tries to minimize its regret against both the learning and the static components of the opponent. p is used to determine the amount of weight put on each part of the regret.

A Team of agents

In competitions the opponents will be unknown, so which of the previously discussed agents should be used against each opponent? Each of the previous agents have had their pros and cons, to get the benefits of each a team of agents is created, comprised of a number of agents of various types, creating a meta-agent. One of the problems faced by the meta-agent, which of the team of agents to choose when selecting an action, is solved by expert algorithms. In this case the algorithm UCB1 was chosen, it is designed to trade off exploration and exploitation when choosing which agent to use. Since there are various types of agents in the team, the UCB1 algorithm should use different costs of exploration for different types. For example the cost of using a Frequency Best Response agent that was not designed for the opponent you are playing is very high whereas the cost of using a Restricted Nash Response that was not designed for the opponent you are playing is much lower and the cost of using the Nash equilibrium agent is the lowest. Johanson [2007] found that, using a team of agents provided better results than using only one of its parts.

Discussion

The attempt to incorporate exploitation into a Nash equilibrium strategy while still remaining difficult to exploit is the idea this dissertation is based on. Johanson [2007] addresses this through the creation of Restricted Nash Response (RNR) strategies. RNR strategies are static and so are only able to exploit opponents with similar strategies to the strategy they were trained against. Due to this, if you match an RNR strategy against an arbitrary opponent it is unlikely the RNR strategy will be able to exploit the opponent. An RNR strategy gives up some of its

difficulty to be exploited in order to be able to exploit opponents. If the strategy then plays an opponent whom it can't exploit it is more exploitable to that opponent than a Nash equilibrium strategy would be. This coupled with the fact that a single RNR will not be able to exploit most of its opposition makes it worse to use an RNR strategy than a Nash equilibrium strategy.

To address this problem Johanson [2007] created a team of agents. If the team consists of a Nash equilibrium strategy and a number of different RNR strategies it is able to exploit a larger subset of opponents than a single RNR, and against the opponents whom it can't exploit it plays the Nash equilibrium strategy. The "coach" algorithm of the team must learn throughout a match which agent to use against the opponent it is playing. During this exploration stage there is a cost for each exploration which uses one of the agents that is not the best against the current opponent. RNR agents that were created with low p values will be difficult to distinguish from one another and from the Nash equilibrium strategy, prolonging the exploration period. This is bad because the longer it takes for the "coach" to find the best agent, the longer the meta-agent is not playing at its fullest potential.

To be able to exploit the majority of opponents using this method would require a meta-agent which consisted of many RNR strategies. This would be difficult since each RNR requires a different opponent strategy to train on and the strategies take a relatively long time to compute. Johanson [2007] states "We estimate that competitive strategies that use larger abstractions with 10 or 12 buckets can be computed in less than a month". Due to this it seems it would be difficult to use this technique to create a bot which is able to exploit a large percentage of opponents.

3.5 Summary

In this section the field of opponent modeling was examined, the terminology for opponent modeling was clarified, and the importance of frequency statistics to the idea of modeling an opponent was explained. The agent that make up Polaris were also covered, to show another technique which was used to add the ability to exploit while remaining difficult to exploit. In the following sections the research question and motivation for the research will be presented.

4 Research Question

In light of the above discussions, this dissertation's central research question is:

> Is it possible to significantly increase the win rate of an equilibrium style poker bot against weaker players through the use of exploits, without decreasing the win rate against strong opponents?

5 Motivation

5.1 Initial Motivation

The idea of exploits and a complex statistical frequency model was derived from the way humans play online poker. People use Heads Up Displays or HUD's that store every hand in a database and use this information to provide statistics for any opponent you have played or are playing. The statistics that can be displayed are vast, practically anything that can be thought of, since all the hands were recorded there is no limit as to which statistics can be calculated. These are very popular among online players, the two most popular being Poker Tracker [Poker Tracker] and Hold'em Manager [Hold'em Manager]. Humans use these statistics to determine how to play against unseen or unremembered opponents and they use them quickly. This dissertation seeks to investigate if the numbers help a human gain an edge then will they give the computer a significant advantage since computers are much better at number crunching than humans.

The difficulty is giving the statistics meaning to the computer. Human online players tend to classify people into opponent types and then make notes of any unexpected plays or tendencies. They start with the opponent type and quickly deviate based on the opponents play to create a specialized opponent model. This would be very difficult for computers to do. Humans are able to make intuitive leaps from only a few hands and restructure their entire model whereas computers would have to see enough hands to make a statistically significant difference.

So instead of trying to have the computer play like an experienced online human player we thought what does a novice do? They play a generally static poker strategy and only step away from this strategy when they see significant statistics. For example they are playing an opponent and they notice that the opponent folds to a three bet (re-raise) 70% of the time pre-flop. Through this statistic they realize that if they re-raise the opponent pre-flop the opponent will fold the majority of the time. From this they begin to form what we have defined as statistical exploits, or ways in which you change your strategy to take advantage of poor play from the opponent which you deduced from the frequency statistics. The most obvious exploit to deduce here is, "I can

three bet bluff this opponent profitably since he will fold 70% of the time". One could also deduce "If I have a good hand I should not three bet since he will fold most of the time", and "If he calls my three bet he must have a good hand". The novice uses the information gained from the statistical exploit to alter their play and play a more opponent based strategy. Obvious statistical exploits will not be found as easily when looking at good players statistics, but being able to exploit obvious weaknesses greatly increases the novices win rate. A bot should be able to benefit equally from a similar reaction to statistical exploits.

5.2 Motivation From the literature

An approximate Nash equilibrium strategy is a static strategy that is minimally exploitable, where exploitability is defined as how much a player could take advantage of another player's strategy. This means that an agent playing a Nash equilibrium strategy has a guaranteed upper bound on its exploitability. Schnizlein [2009] states that one way of creating a champion level poker agent is by computing a Nash equilibrium in an abstract version of the poker game and using the resulting strategy to play the full game. Having an upper bound on the exploitability can create a champion level poker agent because, in essence, it means you should not loose. The agent may not win much or it may only tie but it will not lose. This is a very desirable trait to have, to create a champion level poker bot it must be difficult to exploit.

The downside of using a static Nash equilibrium strategy is that it does not exploit weaknesses in the opponent's play. While it plays not to lose, it does not play to win. A champion level poker agent should be able to exploit weak players. A novice human player is able to exploit an opponent who only calls but an approximate Nash equilibrium agent would not be able to. If the agent is not able to do something that novice human players are able to, how can it be considered a champion level player?

Opponent based strategies on the other hand play to win. Fully opponent based strategies attempt to maximally exploit the opponent, examples of these are Loki and its variants, Poki and its variants, Vexbot, the Bayesian bot, DBBR, and SartreNL with adaptation, all discussed in the

literature review. These bots play fully opponent based strategies where the opponent model impacts the action for every hand played. Playing to win is important as the goal of any game is to win. The more you can win from you opponent the better, and fully opponent based strategies are created to win as much as they can from weak players. The downside of this strategy is that it makes the bot exploitable to strong players.

Each of the two approaches, Nash equilibrium and fully opponent based have a positive attribute and a negative attribute. To make a true champion poker agent, however, it would require the agent to have the positive aspects of both. The agent should be very difficult to exploit so that it does not loose against strong players and should be able to exploit weaknesses in opponents play so that it is able to win a large amount from weaker players. We feel the logical step here is to combine the two approaches to attempt to get the best of both worlds. Johanson [2007] combined a Nash equilibrium strategy with a best response strategy and had to face similar challenges. The difficulty with this approach is that you cannot keep both positives as strong as they were in their individual models. This means we have to choose which of the two attributes is most important and which we can sacrifice to a degree to keep the other as it was before the combination. To create a champion poker agent I believe it is most important not to lose. Winning is the primary goal of the game, and you can't win if you lose, so it seems reasonable to decrease the win rate to a degree against weak opponents to ensure that you do not loose against strong opponents. We will still have the ability to exploit players but we are giving up maximal exploitation of opponents to ensure that we remain difficult to exploit.

The statistical exploitation model is an approach to add an opponent based portion onto a Nash equilibrium strategy. It mixes the positive aspect of the Nash equilibrium strategy with that of the fully opponent based strategy with an emphasis on retaining difficulty to exploit over exploitation, creating a partially opponent based strategy. To ensure difficulty to exploit we only want to perform exploitative actions when we are sure that they will not be exploited by the opponent. This is very difficult for most approaches to do since their exploitation approaches are general, trying to exploit the opponents general play style, and cover every action, as seen in the approaches that were covered in the opponent based techniques literature review.

The statistical exploitation model, however, is a collection of exploitations which only apply to specific situations. This means the bot must only be sure of the small number of preconditions associated with each individual exploit in order to start applying it. Since the exploits are specific it is also possible to calculate the expected value of each exploit, making it possible to ensure that each exploitation has a positive expected value. Taking advantage of specific weaknesses makes it possible to perform exploitative actions while remaining difficult to exploit. Trying to remain difficult to exploit while attempting to exploit the opponent's general play style would be far more difficult, if not impossible. The model also allows the exploitation of an opponent with a strong general play style but who has a few small weaknesses, for example the Hyperborean bots as shown in section 6.

The new bot created from SartreNL and the statistical exploitation module incorporates a Nash equilibrium strategy and an opponent based strategy well, but it does have a downside, the exploitations are not easy to produce. The exploitations must be created using expert knowledge, this is a down side because it is time consuming to create exploitations by hand and it requires knowledge of poker, statistical frequencies in poker, and the ability to code. Automation of the identification and creation of exploitations would be highly beneficial but is also extremely non-trivial. It requires a large amount of prior knowledge of poker, and the ability to make inferences about what statistics express about an opponent's play style and what type of actions would be able to exploit that.

6 Statistics based opponent exploitation system design

6.1 Overview

We have created an addition to SartreNL [Rubin, 2011] that exploits weaker players through the use of statistical exploits. This significantly increases the bots win rate against them, without decreasing the bots win rate against strong players. We have done this by creating a statistical model that records detailed frequency statistics of an opponent in many contexts, and a number of exploits that provide highly profitable actions in the situation they apply to. The addition also has an opponent exploiter that keeps track of the exploits that apply to a given opponent model at any given time and provides the underlying bot with actions from the exploits when a situation arises in which one of the exploits applies. We have used SartreNL as the underlying agent but this addition could be applied to any underlying agent through minor changes to the opponent exploiter and the underlying agent chosen.

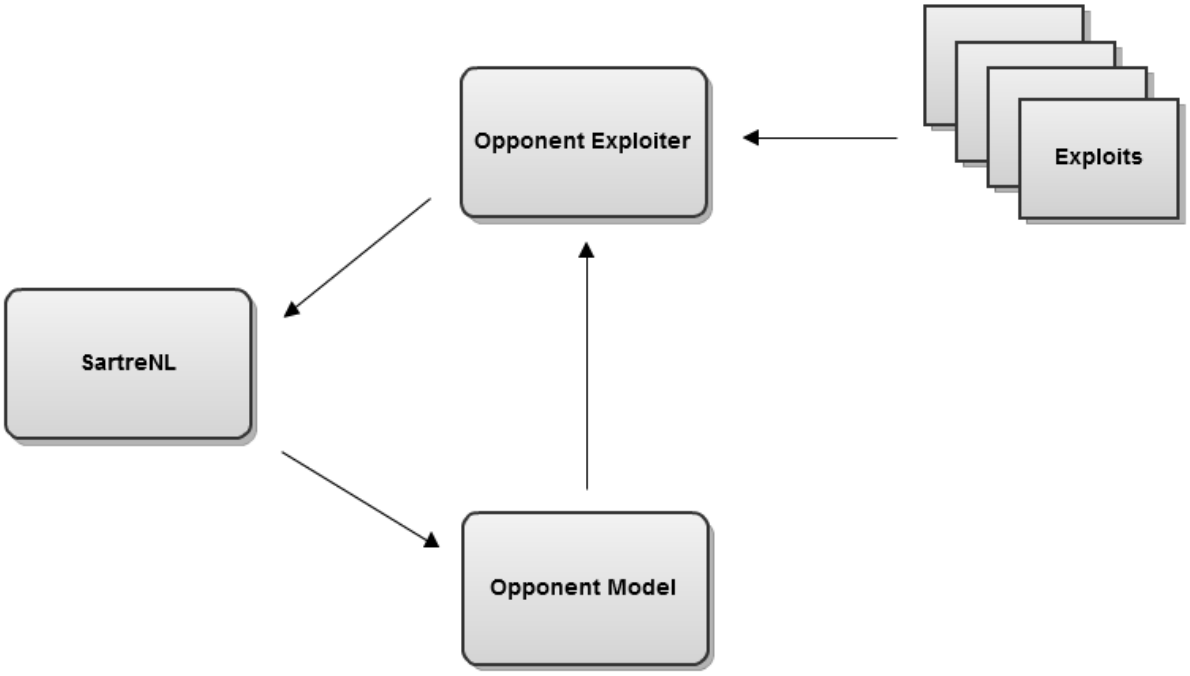

Figure 2. Model of exploitation system design

The above Figure depicts the way in which the individual parts work together to provide the addition of a partially opponent based strategy to SartreNL. Partially opponent based because the

actions determined by the bot/addition combination are not always dependent upon the opponent model, they are only dependent on the opponent model when an exploit applies to both the game state and opponent model.

6.2 SartreNL

Rubin [2011] describes the alterations applied to transform the Sartre limit poker bot into SartreNL; a case based reasoning bot for no limit Texas Hold'em. Rubin [2011] states that the focus of this transformation was "... determining a suitable **action abstraction** and resulting **state translation** that is required to map real-value bet amount into a discrete set of abstract actions". The similarity metrics also had to be redefined to allow identification of similar scenarios between complicated no limit betting sequences accurately, through which generalization of decisions is possible.

SartreNL uses four attribute-value pairs to depict the state of a match. Three of these (hand strength, betting sequence, board texture) are also used in the limit variant of Sartre [Rubin, 2009]. The fourth attribute, stack commitment, is unique to the no limit variation of Sartre. Due to the bet amount being mapped into discrete categories proportional to the pot size, the information about the total amount a player has contributed to the pot relative to their starting stack is lost. However Rubin [2011] states "Once a player has contributed a large proportion of their stack to a pot, it becomes more important for that player to remain in the hand, rather than fold, i.e. they have become pot committed". The attribute stack commitment was added to allow the bot to account for being stack committed or not.

In no limit Texas Hold'em were a raise can be of any value, action abstraction is required to help restrict the size of the state space. This was not necessary in the limit version of Sartre since in limit there are only three actions to choose from: fold, check/call, or bet/raise. SartreNL, however, uses the following action abstraction:

f	fold
c	call
q	quarter pot
h	half pot
i	three quarter pot
p	pot
d	double pot
v	five times pot
t	ten times pot
a	all in

Table 2. The action abstraction used by SartreNL after ref.

To map an action to one of the action abstractions listed in Table 1 requires a translation process. SartreNL uses translations in three areas:

1. Case base construction - encoding hand history logs into cases
2. During game play - observed actions during a hand are mapped to abstract actions
3. Reverse translation - mapping a chosen abstract action into a real value to use during game play

SartreNL uses two different translation approaches, which were originally formulized by Schnizlein [2009]:

1. Hard translation - uses a distance metric to map unabstracted betting values it abstract actions. Given a unique unabstracted betting value it will always map into the same abstract action.
2. Soft translation - uses normalized weights as a similarity measure to probabilistically map an unobstructed betting value into an abstract action.

Which type of translation occurs depends on where the translation occurs in the system. SartreNL uses hard translation during case base construction, and soft translation during game play and reverse translation.

The new similarity metric formulas are given in [Rubin, 2011] along with some examples. I have not summarized this section as it should be read in full to gain full understanding.

SartreNL is used as the underlying poker agent because it uses an approximate Nash equilibrium strategy which provides the trait of difficult to exploit, and is the only poker agent that was readily available to us. SartreNL plays the style of the player whom its case base was trained on. In this case the case base was trained on the Hyperborian bot's hand histories from the 2011 AAAI Heads-up No-Limit Texas Hold'em competition. This bot plays an approximate Nash equilibrium strategy, so the equilibrium strategy should emerge in SartreNL's play.

6.3 Opponent Model

The opponent model is a collection of counters that are updated after each hand is played out, as well as a variable for each statistic, which is also recalculated after each hand has been played out and all the counters have been updated. In the poker framework used in the AAAI competitions messages are passed to and from the bots. The messages sent to the bots include all of the contextual information from the game and the messages sent from the bots contain the actions they wish to make. The bulk of the code for the opponent model consists of methods which update the counters from the message sent from the server. The statistics which are used in the model are often context based and so to update the counters one has to check numerous conditions for each.

I will illustrate how these counters are used to determine statistics and how the counters are updated with an example. This example is for statistics concerning a flop continuation bet. A flop continuation bet or flop cBet is when a player is the last to raise in the pre-flop betting round and is the first to raise in the flop round of betting. To calculate the statistics associated with flop continuation bets the model uses seven counters:

1. Could cBet Total: couldcBetTotal

2. cBet Total: cBetTotal
3. Where cBet Total: wherecBetTotal
4. Fold to cBet Total: foldTocBetTotal
5. Raise cBet Total: raisecBetTotal
6. cBet Raised Total: flopcBetRaised
7. Fold to cBet Raise Total foldTocBetRaiseF

The method is a series of nested if statements which update counters as they apply. So if the player is the dealer and they were the last to raise before the flop and the non-dealer checks to them on the flop we can update the counter for could cBet total. If the player did raise after being checked to the cBet total counter is updated. If the cBet is then re-raised the flopcBetRaised counter is updated and if the response to the re-raise was to fold the foldTocBetRAiseF counter is updated. The full code for the updatecBet method can be found in appendix A. After these counters have been updated the statistics for cBet, fold to cBet, raise cBet and fold to cBet raise are calculated by dividing the number of times the action was performed by the number of times it was possible for the player to perform the action so for example:

$$cBet = cBetTotal \ / couldcBetTotal \ * 100$$

6.4 Exploit specifications

Exploits are basically rule modules that adhere to the generic exploit template. The generic exploit template includes two methods: applysToStats and getAction. applysToStats is the same for every exploit, it takes a stats model object and returns whether the given exploit applies to the stats model. The getAction method is given the game context and returns an action if the exploit applies to the specified context. There are two versions of this method, one for pre-flop exploits and one for post-flop exploits. They differ in that the pre-flop exploits are given the two card hand ranking and the post-flop exploits are given a hand ranking calculated by SartreNL which takes into account the community cards. For getAction to be called applysToStats must have been called previously and must have returned true. To give a more complete overview of how

individual exploits work I will go over an example of an exploit, but first I will explain equity and expected value, which are used to determine the profitability of an exploit.

Equity: The chance that a hand will win the pot.

Calculating the equity of a single hand against another single hand is relatively difficult in and of itself due to all the probabilistic calculations one would have to carry out and the numerous considerations one would have taken into account. For example, when calculating the equity of AdKc verses 5h5c you would have to calculate the chance you have of improving to the best hand after the five community cards have been shown minus the chance your opponent has of improving to a better hand. This may seem simple at first but when you break it down you can see that there are a large number of possibilities. If your hand is AdKc (i.e., Ace of Diamonds and King of Clubs) your hand improves if:

- One of the community cards is an Ace or a King giving you a pair
- One of the community cards is an Ace and one of them is a King giving you two pair
- Two of the community cards are Aces or Kings giving you three of a kind
- The community cards include a Queen, a Jack and a Ten giving you a strait
- Four of the community cards have the suit diamonds or four have the suit clubs giving you a flush
- Two of the community cards are Aces and one is a King, or two are Kings and one is an Ace, or there is a pair in the community cards that is not Aces or Kings and there are also two Aces or Kings, or there are three of a kind in the community cards and one of the community cards is an Ace or a King, all giving you a full house
- Three of the community cards are Aces or Kings giving you four of a kind

There are a lot of ways to improve to the best hand and the opponent also has many ways to improve to a better hand so for each of these cases one has to take into account the chance that the opponent improved to something better. This calculation becomes difficult quickly for only one specific hand against another specific hand and it becomes exponentially more complicated when considering a range of hands against another range of hands. To calculate equity easily

tools are used that run large numbers of simulations and then calculate the equity of a hand based on the number of times it won or tied compared to the number of outcomes considered. These tools are able to calculate the equity of a range of hands versus another range of hands by running simulations.

We used Poker Stove [Poker Stove] to calculate the equities that were used in our expected value calculations used for calculating the profitability of exploits. The way in which Poker Stove [Poker Stove] calculated equity is described in their FAQ section as “the fraction of the pot that a hand will win on average over many repeated trials, including split pots".

Expected Value (EV): The long term expected outcome of a given hand or situation.

Expected value is calculated using the following equation:

*EV = [Our Equity] * [What we win] - [Opponent's equity] * [What we lose]*

For example if we are playing 100/200 no limit, we have AdKc and we are in position, our opponent has 5h5c, and we go all in pre-folp knowing our opponent will call. We have already posted the small blind and our opponent has already posted the big blind so there is 300 in the pot, and we have bet 19,900. Using Poker Stove [Poker Stove] we know that our equity is 45.382% and our opponent’s equity is 54.618%. If we win, we win the pot plus our opponents 19,800 call, which is 20,100. If we lose. we lose the amount we are betting, 19900. So the calculation is:

*EV = 0.45382 * 20100 - 0.54618 * 19900 = -1747.2*

This shows that over the long term every time you go all in with AdKc against 5h5c you will lose 1,747.2, it’s a losing play and you don't want to be doing it. However, it is a very unusual circumstance in which you know exactly what cards your opponent has (for example, your opponent’s hand was flipped during the dealing), most of the time you will be calculating your

EV against a range of hands which you believe your opponent is most likely to have. In the case of the exploits that I have created I have never made them as specific as to only include one hand so the EV calculations are done between ranges of hands. This does not change the EV calculation, only the equity calculation.

All in pre-flop exploit example

The all in pre-flop exploit is an exploit that raises all in pre-flop if the statistics of the opponent are applicable and our hand rank is high enough. There are two thresholds for the hand rank, one for when the opponent is the dealer and one for when they are not, this is because it is possible for opponents to play vastly different hand ranges from different positions. For example some opponents may play close to all hands when they are the dealer and very few hands when they are not. The numerical values for these thresholds are determined during the applysToStats method. I will go through how the rank threshold is calculated for when the opponent is the dealer.

```
//If the opponent has been raised all in at least 11 times
if(model.whereRaisedAllinPre > 10){
   //if they have played at least 31 hands in position
   if(model.vpipInPosition > 30){
      if(model.vpipD > 30 && model.vpipD < 50){
         if(model.folded2AllinPre < 10)
            //lowest EV = + 3224 ~16BB
            rankThresholdD = 25;
         else if(model.folded2AllinPre < 30)
            //lowest EV = +3346 ~16BB
            rankThresholdD = 20;

      }else if ...
   }
}
```

The first thing checked in the appliesToStats method is whether the opponent has been raised all in pre-flop a significant number of times such that the frequency statistics are reasonably

accurate. At least 11 instances of the opponent being raised all in was chosen on the assumption that since going all in is such a big decision, players will not be doing it for deceptive purposes, and the statistics will be accurate within a small number of trials. If the opponent has been raised all in enough times before, the exploit looks at how often the opponent voluntarily puts money in the pot in position (VPIP dealer). First it checks that the opponent has voluntarily put money in the pot enough times such that the statistic is significant, if this is the case it then looks at the statistic to determine the threshold value. The value of in position VPIP tells us the range of hands the opponent plays in position. So if an opponent had an in position VPIP of 30% we would assume their range consisted of the top 30% of cards. If this was not the case it would only benefit us more since it would give our range of hands more equity. The value of folded2AllinPre tells us the percentage of times the opponent folded when raised all in pre-flop. When calculating their range this percentage of their range is removed, the hands removed are from the bottom of the opponents range.

A number of different thresholds have been created which take effect depending on the frequency statistics of the opponent. For each threshold the minimum expected value of going all in pre-flop was calculated by using the smallest range of hands for the opponent that still applies. For example the minimum EV for the threshold 25 was calculated as follows:

- Our hand range was set to the top 25 hands
- The opponent's hand range was set to the top 27% of hands, since he plays 30% of all hands and folds 10% of these to a pre-flop all in
- Using PokerStove [Poker Stove] our equity was 57.811% and our opponents was 42.189%

The calculation was then:

$$EV = 0.57811 * 20100 - 0.42189 * 19900 = 3224.4$$

The calculations for determining the non-dealer rank threshold are similar and can be viewed in appendix A. When creating the thresholds for the all in pre-flop exploit, we made sure that there was an expected value of at least +15 big blinds since the risk when going all in is your entire

stack. We wanted to make sure that there was a reasonably high expected value not just a marginal one.

The getAction method evaluates the current context to determine whether the exploit applies to this context or not. In the all in pre-flop exploit it checks to see who the dealer is, if the bot is the dealer it uses the rankThreshold for non-dealer and if the opponent is the dealer it uses the rankThreshold for dealer. The hole card rank given in the context is then checked against the threshold which applies, if the hole cards are within the acceptable range getAction returns the string "a" for "all in action" otherwise it returns null.

6.5 Opponent Exploiter

The opponent exploiter is the module which provides the interaction between SartreNL, the statistical model and the exploits. The opponent exploiter has a statistical model associated with it and has four lists of exploits. Each list represents all the exploits that apply to the associated statistical model for each of the four betting rounds: pre-flop, flop, turn, and river. These lists are populated through the findApplicableExploits method which goes through each exploit, calling its applysToStats method, and supplying the statistical model the opponent exploiter is associated with as the parameter. If an exploit returns true it is added to the list of exploits for the betting round it applies to. The opponent exploiter also has a method for each betting round which takes the game state information and calls the getAction method for each exploit in the list for the given betting round. If an exploit returns a non-null value this is passed along to the bot which uses the action.

6.6 Summary

This section gave a brief overview of the statistical exploitation addition and how the components work together before delving into the implementation of each of the parts. It first

covered the underlying agent, which the statistical exploitation was added to. It then described the implementations for the opponent model, the exploits and the opponent exploiter.

7 Results

7.1 Testing Methodology

We required several opponents to challenge to evaluate the results of using exploits. Optimally we would evaluate exploitation against a variety of competences, ranging from easily exploitable to unexploitable. The participants in the Annual Computer Poker Competition (ACPC) represent a good variety of computer players. While it is not possible to challenge the agents submitted to the competition directly, due to them not being publicly available, the hand history information is available for each agent that participated. Jonathan Rubin created expert imitator case bases for several of the no limit Texas Hold'em participants from the 2011 ACPC that imitate and generalize the opponent's style of play from their hand histories. Rubin created these to be used by the expert imitation based framework described in [Rubin, 2010], training each expert imitator on the decisions made by each particular agent in the competition. The agents that where imitated are shown in Table 3. These agents cover a variety of exploitability, ranging from very exploitable to difficult to exploit. Table 3 shows the author's views on the exploitability of the various agents.

Player	Exploitability
POMPEIA	very exploitable
Kappa	very exploitable
Hugh	high-moderately exploitable
Lucky7	moderately exploitable
Hyperborean-iro	difficult to exploit
Hyperborean-tbr	difficult to exploit

Table 3. Exploitability of the opposition

POMPEIA was categorized as very exploitable because it always calls no matter what. Kappa was also categorized as very exploitable because it raises all in with the majority of its hands. Lucky7 and Hugh do not have such glaring holes in their play, however Hugh has a very exploitable playing style in that it plays very aggressive. Both bots have numerous statistical

exploits that apply to them, but due to Hugh's exploitable play style it was categorized as high to moderately exploitable and Lucky7 was categorized as moderately exploitable. The Hyperborean bots are approximate equilibrium distribution bots and are therefore categorized as difficult to exploit. They were not categorized as unexploitable because, as will been seen, exploits were successfully used against both showing there are some statistical exploits present in the Hyperborean bots.

These six opponents were challenged against SartreNL without exploits, and SartreNL with the addition of exploits which will henceforth be denoted as SartreNLExp. Each of the six bots played two seeded duplicate matches against both SartreNL and SartreNLExp. A duplicate match consists of 20,000 hands in total. 10,000 hands are initially played, the players then switch seats and the same 10,000 hands are played again. This way each of the players receives the cards that their opponents received before. The duplicate match style was used to decrease the variance that is normally involved in poker. To decrease the overall variance further, the same seed value was used for each of the duplicate matches played between each of the variants of SartreNL and the various opponents.

Overall 24 duplicate matches were played; SartreNL played two duplicate matches against each of the opponents and SartreNLExp also played two duplicate matches against each opponent. To determine the effectiveness of the addition of the partially opponent based strategy based on exploits on SartreNL the duplicate matches were split into two subsets: Run 1 and Run 2. Run 1 consisted of the first duplicate match SartreNL played against each of the opponents and the first duplicate match SartreNLExp played against each opponent. Run 2 consisted of the second duplicate match that SartreNL and SartreNLExp ran against each of the opponents.

In each run the match SartreNL played against a particular opponent is used as the base-line. The difference in performance between SartreNL and SartreNLExp can then be taken as the effect of the exploits. The final scores can be viewed in appendix B. However, looking at the final scores in each case is not going to give an accurate depiction of the effect of the exploitations. Some of the bots have some randomness associated with their strategies, so although the same hands and community cards come up in all matches this does not mean the bots will choose the same action

each time. Due to this it is likely that the scores will fluctuate between matches. There is also the fact that an exploit is not applied to every hand and SartreNL chooses the actions as it normally would for hands exploits are not applied to. This means that the situations can occur where exploitations are used and the overall score for SarterNLExp is lower than SarterNL's. This would not be due to the exploits loosing, since exploits always have a large positive expected value, and they all but in the rarest cases do better than SarterNL's chosen action. It would be caused by the fact that there is randomness in SarterNL's action selection process. This means that although the exploits had positive effects, SarterNLExp chose less profitable actions in the hands in which exploits where not used, causing SarterNLExp's overall score to be lower than SarterNL's.

To combat these problems we will only compare the resulting scores of the hands in which exploits were used. When a match is run a number of output files are produced along with a log of each of the hands. SartreNLExp notes, in an output file, every time it uses an exploit, and the corresponding hand numbers. From this output file we create a list of all the hand numbers in which exploits where used. In each run there exists a duplicate match between SartreNLExp and each opponent and a corresponding duplicate match between SartreNL and each opponent. The score for the exploited hands are found for both the SartreNL and SartreNLExp match by going through the log files and tallying up the result for each of the hands in which exploits were used. The difference between these two scores then shows the impact of the exploits much more accurately than the difference between the overall scores.

7.2 Effect of exploitation

In this section we will present the effects the exploits have on the hands they affected. To do this we will highlight the difference of the score for this subset of hands when played as SartreNL would and when exploits are used. The score is the amount that was won or lost by the bot over all the hands in the subset. We will present the finding for each opponent and then discuss the implications of the results. The findings are tabulated so the first row shows the number of hands that are present in the subset of the hands, which were exploited. The second row shows the score of this subset of hands when played by SartreNL with no exploits being used. The third

row shows the score of the subset of hands when exploits were used, and the fourth row shows the difference between the score when exploits were used and the score when exploits were not used. The first column labelled Run 1 represents the duplicate matches between SartreNL/SartreNLExp and the opponent in the first subset of duplicate matches. The second column Run 2 represents the matches in the second subset of matches. The opponents are addressed in the same order as Table 3, going from most exploitable to least exploitable.

POMPEIA

	Run 1	Run 2
Number of exploits	2090	2090
not exploited	441,139	460,782
exploited	10,220,000	10,220,000
enhancement:	9,778,861	9,759,218

Table 4. POMPEIA exploited hand scores

As seen in Table 4 POMPEIA is very exploitable, this is apparent in the results as well. In the first run we see an increase in winnings by a factor of 23 and in the second we see an increase by a factor of 22. This is an exaggerated case in that POMPEIA always calls, which is not something a human player would ever do, but it is a good example to shows just how much value an equilibrium approximation strategy can miss out on when dealing with an extremely weak player. Not only is SarteNL missing a lot of value in this case, it is missing a lot of easy value. By this we mean that even the weakest human player who is paired against an opponent who always calls will be able to easily exploit them. If one wishes to create a strong poker bot we feel it should not lack the ability to do what any human novice can.

Kappa

	Run 1	Run 2
Number of exploits	4160	4160
not exploited	12,240,800	12,240,800
exploited	19,635,200	19,635,200
enhancement:	7,394,400	7,394,400

Table 5. Kappa exploited hand scores

The kappa bot raises all in with the majority of its hands and so is also very exploitable. The approximate equilibrium strategy does a much better job coping with this than with only calling. However it still often folds good hands against kappa due to its equilibrium strategy, decreasing its win rate. Although the equilibrium strategy does relatively well against kappa, it still misses a lot of value. Like POMPEIA, kappa is not only missing value, but it is missing value which is very simple for human players to get.

Hugh

	Run 1	Run 2
Number of exploits	4703	5122
not exploited	-591,465	-606,192
exploited	765,423	598,245
enhancement:	1,356,888	1,204,437

Table 6. Hugh exploited hand scores

Hugh is an extremely aggressive bot. Although this is difficult for novice players to handle it is also a very exploitable strategy. Since Hugh is betting, raising and re-raising so often the average hand strength it has when it makes these actions is relatively low. The high aggression makes a number of statistical exploitations possible making Hugh high to moderately exploitable.

Lucky7

	Run 1	Run 2
Number of exploits	6392	6353
not exploited	143,293	139,951
exploited	644,291	319,774
enhancement:	500,998	179,823

Table 7. Lucky7 exploited hand scores

Lucky7 does not have as obvious of an exploit as POMPEI or kappa or even Hugh, it has a number of statistical exploits that apply to it, but the one which is applied the majority of the

time is an exploit which bluff raises when Lucky7 calls pre-flop as the dealer. This exploit does not add a large amount of value every time it is used but it does add up. This exploit is why the number of exploited hands is so high yet the increased amount is less than for the previous opponents for Lucky7.

Hyperborean-iro

	Run 1	Run 2
Number of exploits	1265	1265
not exploited	22,678	7,398
exploited	122,988	113,239
enhancement:	100,310	105,841

Table 8. Hyperborean-iro exploited hand scores

Hyperborean is a static bot produced through counterfactual regret minimization. This is a technique for solving large games to compute a Nash equilibrium and results in an approximate Nash equilibrium strategy. Nash equilibrium strategies are meant to be un-exploitable however this is not the case with approximate Nash equilibrium strategies as seen in the results.

Hyperborean-tbr

	Run 1	Run 2
Number of exploits	1357	1357
not exploited	21,626	-4,961
exploited	116,023	120,032
enhancement:	94,397	124,993

Table 9. Hyperborean-tbr exploited hand scores

The difference between this version and the iro version of the Hyperborean bot is that this version does not perform actions which are performed less than 10% of the time, distributing the action percentage across other more frequently used actions.

As we can see in each case the use of exploits significantly increases the score of SartreNL. In fact in some cases it massively increases it. This was expected as SartreNL plays an approximate

equilibrium strategy and so does not exploit players but instead attempts to be un-exploitable. Static equilibrium strategies acquire the ability to not lose by giving up the ability to exploit weak players. It is however not necessary to lose the ability to exploit weak play. Through the use of a partially opponent based strategy based on exploits, we are able to first identify if an opponent is exploitable and only exploit them if this is the case. Through the use of expert made exploits we are able to guarantee a positive EV as long as the opponent model is accurate. Through this we keep the quality of being difficult to exploit and are also able to exploit weaknesses in the opponents play. The opponent model may become inaccurate if an opponent switches strategies after a long period of playing one strategy. To deal with this decaying history is necessary and this is part of the future work.

8 Conclusions

The aim of this dissertation was to create an opponent based strategy that can be added to an underlying Nash equilibrium bot. This addition should be able to retain the attribute of being difficult to exploit, provided by the underlying strategy, while having the ability to safely exploit opponents. In this respect the exploitation system created is successful. The underlying strategy is used unless the bot is sure that an exploit is applicable without the possibility of being exploited, in which case the exploit is used.

The system consists of three distinct parts, the opponent exploiter, the statistical model and the exploits. The opponent exploiter uses the information in the statistical model along with the exploits to provide the underlying system, SartreNL, with opponent based actions when they are applicable. The addition can however be used with any Nash equilibrium bot through changing the opponent exploiter such that it can communicate with the underlying system chosen. Expert designed exploits where used because they provide the ability to segment the exploitation of an opponent into small parts, each of which can easily be proven to apply to the statistical model or not. The fact that each exploit applies to a specific situation also allows you to calculate the expected value of each, to ensure that it has a positive expected value.

8.1 Comparison between the Polaris Meta-agent and our statistical exploitation Model

Polaris's game theory approach to adding exploitation to a Nash equilibrium strategy has a number of down sides. The meta-agent comprised of a number of agents and a coach incorporates various RNR agents to add exploitation abilities against opponents. RNR strategies are only able to exploit opponents with a similar strategy to the strategy they were trained on. This means that to be able to exploit a large percentage of the opponents the meta-agent encounters the meta-agent must consist of many RNR agents. Creating all these agents is time consuming, each taking something near a month. Also the more agents added the longer it takes for the "coach" algorithm to learn which one is best against the opponent it is currently playing. RNR agents trade off their difficulty to exploit to gain the ability to exploit some opponents. Due to this they are easier to exploit than the Nash equilibrium strategy. During the exploration stage when the "coach" is learning which is the best against an opponent it will be trying these more exploitable agents, costing it more than just playing a Nash equilibrium strategy. The RNR agents are created in an abstract game and so are not be able to exploit opponents fully as they will not have the full nuances of Texas Hold'em during training. Due to this they will play more generalized exploitation, not being able to pick up on very specific situations in which the opponent plays exploitable. In addition, the exploitation abilities are greatly hindered by the characteristics which make the RNR strategies difficult to exploit. If the goal is to create a meta-agent which can exploit a high percentage of the opponents it encounters it would be slow to produce, not able to fully exploit opponents, and take a long time to learn which agent to use while it is playing an opponent, during which time it would be more exploitable then a Nash equilibrium strategy.

The technique presented in this dissertation has problems similar to Polaris, but they are less severe. The statistical exploit approach plays the underlying Nash equilibrium strategy if no exploit pre-conditions are met and so does not have the exploitability problem faced by Polaris during its exploration phase. Instead of a large number of agents needed to exploit a high

percentage of opponents this technique requires a large number of exploits. Ss seen in the results, however, the number of exploits does not have to be that large to exploit a number of opponents with very different statistics, but the more that are added the more SartreNLExp can exploit the opponents it comes across. This seems a more scalable approach than Polaris's technique due to not needing expensive exploration stages. Fully exploiting most opponents would require a lot of exploits and would be difficult as someone would have to come up with all of them, but unlike Polaris SartreNLExp is still able to exploit most opponents to a degree because exploits apply to situations and therefore span opponent strategies. Creating exploits requires an expert and is somewhat time consuming but compared to creating an RNR agent it is far less time consuming. The statistical exploitation model also allows for exploitation of very specific scenarios which a generalized exploitation strategy created from an abstract game such as the one in Polaris would not be able to do. The fact that one can calculate exploitation's expected values means every exploitative action can be guaranteed to have a positive expected value. This is not the case when using the exploitation from Polaris since it was created in an abstract game. In comparison the technique outlined in this dissertation is more scalable, faster to include more exploitation, does not suffer from costly exploration procedures, and spans opponents better than the Polaris meta-agent.

9 Future Work

There are several improvements that could be made to increase the performance of the current system and increase the scalability. The most obvious improvement is the addition of more exploits. The more exploits that are in the system the better it is at exploiting opponents, and the system currently does not have a large number of exploits. Currently the frequency statistics model does not use any form of decaying history and is unable to remember opponents. Decaying history is an important upgrade as the system is currently vulnerable to exploitation by players who play a strategy for a long period of time, saturating the opponent model, and then switch strategies to exploit the image they have built up. Another improvement would be to re-implement the statistical model to store hand histories. This would improve the performance and scalability of the statistical model, by making it easy to include and compute further frequency statistics, and allowing the ability to recognize previously played opponents again. Decaying history would also be easier to implement for a system that stored hand histories than for the current system which has only counters. If one only has a counter, it is difficult to determine how to alter the counter to implement decaying history, if one had the hand histories stored it is simple to determine what the last M hands were and how they affected each counter.

Currently the main limitation in the systems exploitative power is the number of exploits that are in the system. These exploits must currently be created by experts, which is time consuming. Also as more exploits are added to the system, assuming the easiest and most obvious exploits are added first, each following exploit will be more complex and difficult to come up with. So as the system is populated the quality of the experts required to create more exploits must also increase. Due to this it would be hugely beneficial if a way to automate the discovery of exploits was created. This is a very non-trivial task, since the discovery of exploits requires large amounts of poker knowledge and knowledge of what differences in frequency statistics mean about an opponent's play. Exploit are mostly created through intuition, which is not something computers are particularly good at. Although it does not seem like an achievable goal it would be the best way to improve the system's performance.

Bibliography

1. D. Billings, A. Davidson, J. Schaeffer, D. Szafron " The Challenge of Poker," *Artificial Intelligence Journal*, Vol. 134(1-2) , 2002.

2. T. Schauenberg, "Opponent Modelling and Search in Poker," *M. Sc. Thesis*, 2006.

3. D. Billings, N. Burch, A. Davidson, R. Holt, J. Schaeffer, T. Schauenberg, D. Szafron, "Approximating Game-Theoretic Optimal Strategies for Full-scale Poker," *Proceedings of the 2003 International Joint Conference on Artificial Intelligence*, 2003.

4. F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, C.Rayner, "Bayes' Bluff: Opponent Modelling in Poker," *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

5. D. Billings, D. Papp, J. Schaeffer, D. Szafron "Opponent Modeling in Poker," *Proceedings of the Fifteenth National Conference of the American Association for Artificial Intelligence (AAAI)*, 1998.

6. D. Billings, L. Pena, J. Schaeffer, D. Szafron, "Using Probabilistic Knowledge and Simulation to play Poker," *Proceedings of the Sixteenth National Conference of the American Association for Artificial Intelligence (AAAI)*, 1999.

7. Davidson, D. Billings, J. Schaeffer, D. Szafron, "Improved Opponent Modeling in Poker," *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI\'2000)*, 2000.

8. Davidson, "Opponent Modeling in Poker: Learning and Acting in a Hostile and Uncertain Environment," *M. Sc. Thesis*, 2002.

9. S. Ganzfried, T. Sandholm, 1st Initial. , "Game theory-based opponent modeling in large imperfect-information games," *Proceeding AAMAS '11 The 10th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 2,, 2011.

10. J. Rubin, I. Watson, 1st Initial. , "Opponent Type Adaptation for Case-Based Strategies in Adversarial Games," *ICCBR 2012*, Vol. , no. , 357-368, 2012.

11. J. Rubin, I. Watson, 1st Initial. , "A Memory-Based Approach to Two-Player Texas Hold'em," *Australasian Conference on Artificial Intelligence 2009*, Vol. , no. , 465-474, 2009.

12. M. Johanson, "Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player," *M. Sc. Thesis*, 2007.

13. (2012), Poker Tracker, . *Available from: , : http://www.pokertracker.com/ [Accessed: Oct 12, 2012]*.

14. (2012), Hold'em Manager, . *Available from: , : http://www.holdemmanager.com/ [Accessed: Oct 12, 2012]*.

15. (2012), PokerStove, . *Available from: , : http://www.pokerstove.com/blog/ [Accessed: Oct 12, 2012]*.

16. D. Schnizlein, "State Translation in No-Limit Poker," *M. Sc. Thesis*, 2009.

17. J. Rubin, I. Watson, 1st Initial. , "Successful Performance via Decision Generalisation in No Limit Texas Hold\'em," *ICCBR 2011*, Vol. , no. , 467-481, 2011.

18. J. Rubin, I. Watson, 1st Initial. , "Similarity-based retrieval and solution re-use policies in the game of texas hold'em," *18th International Conference on Case-Based Reasoning, ICCBR 2010*, Vol. , no. , 465-479, 2010.

Appendix

.1 Appendix A: Code

Update cBet:

```
private void updatecBet(final String preflopBetting, final String flopBetting, final boolean
isDealer){
   if(isDealer){
      //the dealers last action preflop was a raise and the non-dealer called
      if(preflopBetting.length() % 2 == 0 && preflopBetting.endsWith("rc")){
         // if nondealer check to dealer on flop
         if(flopBetting.charAt(0) == 'c'){
            couldcBetTotal++;
            //if dealer cbets flop
            if(flopBetting.charAt(1) == 'r'){
               cBetTotal++;
               //if dealer's cbet is raised
               if(flopBetting.charAt(2) == 'r'){
                  flopcBetRaised++;
                  //if dealer fold to that raise
                  if(flopBetting.charAt(3) == 'f'){
                     foldTocBetRaiseF++;
                  }
               }
            }
         }
      }
      //the nondealers last action preflop was raise and the dealer called
      if(preflopBetting.length() % 2 == 1 && preflopBetting.endsWith("rc")){
         //if nondealer cbets flop
         if(flopBetting.charAt(0) == 'r'){
            wherecBetTotal++;
            //if dealer folds to cbet
            if(flopBetting.charAt(1) == 'f'){
               foldTocBetTotal++;
            }
            //if dealer raises the cbet
            if(flopBetting.charAt(1) == 'r'){
               raisecBetFTotal++;
            }
         }
      }
```

```
    } else {
        //nondealers last action preflop was raise and the dealer called
        if(preflopBetting.length() % 2 == 1 && preflopBetting.endsWith("rc")){
            couldcBetTotal++;
            //nondealer cbets the flop
            if(flopBetting.charAt(0) == 'r'){
                cBetTotal++;
                //dealer raises the cbet
                if(flopBetting.charAt(1) == 'r'){
                    flopcBetRaised++;
                    //nondealer folds to cbet raise
                    if(flopBetting.charAt(2) == 'f'){
                        foldTocBetRaiseF++;
                    }
                }
            }
        }

        //dealers last action preflop was raise and nondealer called
        if(preflopBetting.length() % 2 == 0 && preflopBetting.endsWith("rc")){
            //nondealer checks and dealer cbets flop
            if(flopBetting.startsWith("cr")){
                wherecBetTotal++;
                //nondealer folds to flop cbet
                if(flopBetting.charAt(2) == 'f'){
                    foldTocBetTotal++;
                }
                //nondealer raises flop cbet
                if(flopBetting.charAt(2) == 'r'){
                    raisecBetFTotal++;
                }
            }
        }
    }
}
```

Applies to Stats for All in pre-flop:

```
public boolean appliesToStats(statsModel model){
    rankThresholdD = 0;
    rankThresholdND = 0;

    //Want to have an EV of at least +15BB always as we risk a lot with allin's
    //If the opponent has been raised all in at least 11 times
    if(model.whereRaisedAllinPre > 10){
```

```
//if they have played at least 31 hands in position
if(model.vpipInPosition > 30){
   if(model.vpipD > 30 && model.vpipD < 50){
      if(model.folded2AllinPre < 10)
         //lowest EV = + 3224 ~16BB
         rankThresholdD = 25;
      else if(model.folded2AllinPre < 30)
         //lowest EV = +3346 ~16BB
         rankThresholdD = 20;

   }else if(model.vpipD >= 50 && model.vpipD < 70){
      if(model.folded2AllinPre < 10)
         //lowest EV = +3483 ~17BB
         rankThresholdD = 35;
      else if(model.folded2AllinPre < 30)
         //lowest EV = +3109 ~15BB
         rankThresholdD = 30;
   }else if(model.vpipD >= 70){
      if(model.folded2AllinPre < 10)
         //lowest EV = +3654 ~18BB
         rankThresholdD = 45;
      else if(model.folded2AllinPre < 30)
         //lowest EV = +3109 ~15BB
         rankThresholdD =40;
   }
}
//if they have played at least 31 hands out of position
if(model.vpipOutOfPosition > 30){
   if(model.vpipND > 30 && model.vpipND < 50){
      if(model.folded2AllinPre < 10)
         //lowest EV = + 3224 ~16BB
         rankThresholdND = 25;
      else if(model.folded2AllinPre < 30)
         //lowest EV = +3346 ~16BB
         rankThresholdND = 20;

   }else if(model.vpipND >= 50 && model.vpipND < 70){
      if(model.folded2AllinPre < 10)
         //lowest EV = +3483 ~17BB
         rankThresholdND = 35;
      else if(model.folded2AllinPre < 30)
         //lowest EV = +3109 ~15BB
         rankThresholdND = 30;
   }else if(model.vpipND >= 70){
      if(model.folded2AllinPre < 10)
         //lowest EV = +3654 ~18BB
```

```
                rankThresholdND = 45;
            else if(model.folded2AllinPre < 30)
                //lowest EV = +3109 ~15BB
                rankThresholdND =40;
        }
    }
}


return((rankThresholdD > 0 || rankThresholdND > 0));
}
```

.2 Appendix B: Results

Run 1:

SartreNL			
Players	Total Winnings	Win Rate	In BB per hand
hugh	1006756	50.3378	0.251689
Lucky7	802484	40.1242	0.200621
kappa	9227800	461.39	2.30695
POMPEIA	3609474	180.4737	0.9023685
Hyperborean-iro	-223621	-11.1811	-0.05590525
Hyperborean-trb	-433292	-21.6646	-0.108323

SartreNLExp			
Players	Total Winnings	Win Rate	In BB per hand
hugh	2036747	101.8374	0.50918675
Lucky 7	1232532	61.6266	0.308133
kappa	16622200	831.11	4.15555
POMPEIA	13386262	669.3131	3.3465655
Hyperborean-iro	44744	2.2372	0.011186
Hyperborean-tbr	-50017	-2.50085	-0.01250425

Difference between SartreNLExp and SartreNL			
Players	Total Winnings	Win Rate	BB per hand
hugh	1029991	51.49955	0.25749775
Lucky7	430048	21.5024	0.107512
kappa	7394400	369.72	1.8486
POMPEIA	9776788	488.8394	2.444197
Hyperborean-iro	268365	13.41825	0.06709125
Hyperborean-trb	383275	19.16375	0.09581875

Run 2:

SartreNL			
Players	Total Winnings	Win Rate	In BB per hand
hugh	961177	48.05885	0.24029425
Lucky7	561309	28.06545	0.14032725
kappa	9227800	461.39	2.30695
POMPEIA	3636399	181.82	0.90909975
Hyperborean-iro	-281192	-14.0596	-0.070298
Hyperborean-trb	-197896	-9.8948	-0.049474

SartreNLExp			
Players	Total Winnings	Win Rate	In BB per hand
hugh	2011427	100.5714	0.50285675
Lucky 7	868812	43.4406	0.217203
kappa	16622200	831.11	4.15555
POMPEIA	13442772	672.1386	3.360693
Hyperborean-iro	-96297	-4.81485	-0.02407425
Hyperborean-tbr	-222377	-11.1189	-0.05559425

Difference between SartreNLExp and SartreNL			
Players	Total Winnings	Win Rate	BB per hand
hugh	1050250	52.5125	0.2625625
Lucky7	307503	15.37515	0.07687575
kappa	7394400	369.72	1.8486
POMPEIA	9806373	490.3187	2.45159325
Hyperborean-iro	184895	9.24475	0.04622375
Hyperborean-trb	-24481	-1.22405	-0.00612025