Jbot: An Advanced Poker Bot Implemented with a Specialized Distributed Architecture
Abstract: Poker is an advanced artificial intelligence problem that requires a deep understanding of probability, game theory, and deception. This paper describes an implementation of a poker bot aims to play no-limit texas hold’em at an expert human level. It utilizes a distributed architecture that combines both expert based rule systems with an adaptive learning system.

Table of Contents

I. Objective...........................................................................................................................3

II. Introduction....................................................................................................................3

III. Poker Background........................................................................................................5
    a.) Texas Hold’Em........................................................................................................5
    b.) Limit or No-Limit......................................................................................................6

IV. The Software Platform................................................................................................7
    a.) Poker Academy Pro..............................................................................................7
    b.) Bot Implementation..............................................................................................8

V. Performance Evaluation..............................................................................................8
    a.) The Problem to be Solved....................................................................................8
    b.) The Testbed...........................................................................................................9

VI. Bot Design..................................................................................................................10
    a.) Utilities...............................................................................................................10
        1. Pre-Flop Hand Evaluator.................................................................................10
    b.) Decision Modules..............................................................................................11
        1. Pot Odds Decision Module............................................................................11
        2. Deception Module.........................................................................................12
        3. Opponent Modeling Decision Module.......................................................13
    c.) Strategies (Arbiters).........................................................................................14
        1. Tight Arbiter.................................................................................................14
        2. Loose Arbiter...............................................................................................15
        3. David Sklansky Arbiter................................................................................15
        4. Doyle Brunson Arbiter..................................................................................16

VII. Evaluation Results and Analysis...............................................................................16
    a.) Baseline Testing: Pot Odds Only.........................................................................16

VIII. Future Work and Conclusion................................................................................17

IX. References..................................................................................................................18
Jbot: An Advanced Poker Bot Implemented with a Specialized Distributed Architecture

I. Objective:

To develop an implementation of a poker bot that plays at an expert human level utilizing a
distributed architecture designed specifically for the task.

II. Introduction:

I recently was watching an old episode of The World Series of Poker when they did a
segment about creating the ‘Perfect Poker Player’. As they interviewed poker pros I was struck
by how ill-defined this ideal seemed to be. Many professionals could not even define their own
idea for the perfect poker player. Others gave answers in terms of their opponents or themselves.
Still others pieced together a poker Frankenstein made up of different parts of different players.
I thought then that the question of the perfect poker player seems to be an interesting problem.

The perfect poker player is so ill-defined for a host of different reasons. For one, given
any hand there are multiple correct ways to play it. Different betting strategies are better for
certain game situations and multiple factors affect the correctness of calling a bet, raising, or
folding. Different game types such as tournament play or cash-game play also have a huge
impact on what the ideal moves are. The past actions of the opponents need also be taken into
consideration by a perfect poker player if they are to read the situation properly. With these
things in mind evaluation of a poker player’s skills becomes another formidable problem. What
is an ideal move? Is one move better than the other? Why?

But this difficulty is what keeps people playing cards. Poker has become extremely
popular in the past decade; poker rooms in casinos have grown exponentially and professional
poker players have been thrust into the limelight. People strive to understand the mathematical
aspects of the game as well as master the arts of the bluff and the read. Card players read books
on technique, strategy, opponent reading, and bluffing voraciously for fresh insights on how they
can increase the strength of their game. Every poker player is looking for the missing link in
their game, the one thing that will turn them from a casual player and sporadic winner to a
“rounder”, a professional who cashes out ahead nearly every time they sit down at a table.

One game above all card games “No Limit Texas Hold’em” has established its
dominance in the poker world. It is referred to by pros as the hardest game to play, one that
requires a brilliant mind, an almost compulsive attention to detail, and a cast-iron will to be
successful at. It is the game that is played at the main event of The World Series of Poker and it
is by far the most popular game played in the world.

With the advent of the Internet, poker and poker strategy has spread like wildfire all
around. Millions of web pages provide tips, tricks, and strategies to the most popular of games
and others still allow users to play online whether for fun and learning purposes or for real
money. In the beginning these online poker games were mostly only on IRC chat rooms but
recently have spread to the mainstream with companies such as PokerStars, PartyPoker, and
many others providing both online gambling in poker form as well as educational fun play for
all. This online poker industry has become a multi-billion dollar one which is a testament to the
popularity of the game in today’s world.
It seems inevitable that with the digitizing of the poker room and poker cards computer scientists would want to get their hands in the game. Whether it is the mathematical rigor of the game or the promise of a near intractable problem to solve, computer science and poker have shared a long history together. Many computer scientists have used simple examples of poker hands to develop basic game theoretical algorithms and used odds calculation and simulations to supplement their algorithms. More recently computer scientists have attempted to take on the bigger games such as Limit Texas Hold’em, a game similar to No-Limit with only a simplified betting strategy. Their goal is simple: to develop a computer player (poker bot) that can play at an expert level.

The epicenter of the Texas Hold’em playing computer revolution has been the University of Alberta’s Poker Research Group. It was here that Loki and its next generation bot Poki were conceived of, brainstormed, and coded. Poki is by far the most advanced Limit Hold’em bot in existence and its success against both human and other computer players is second to no other bot. It includes mathematical odds calculations as well as advanced neural networks for opponent modeling combined with some simulations based results that allow it to make intelligent decisions.

But Poki is in no way the solution to the expert poker playing software problem. Though it is far and away the best computer player it has yet to achieve its goal of defeating the best human poker players in the world. It has not yet become a master level player and the steps to make it so seem fairly vague and difficult. Poki also does not play what is widely recognized as the hardest form of poker, “No Limit” Texas Hold’em where at any time a player can bet any amount of money they possess in front of them. The high variance in betting causes a lot of changes in the ideal behavior of a bot as it needs to be much more accurate with its decisions. In No Limit a single wrong move can cost a player his entire chip stack, a problem not present in Limit.

I intend to pick up where Poki has left off in bot development. I feel that Poki’s weakness was in its all-around approach to the problem. Its architecture is not nearly robust enough to handle the full components of what would be an ideal poker playing bot. I have previously developed what I believe to be a more intelligent distributed architecture that better captures the game of poker. It allows multiple expert systems to be laid on top of the intelligent game-theoretical approach that Poki takes. This synergy of simulation, expert, and adaptive learning systems makes the potential strength of my bot far higher than that of Poki’s.

My explanation of my architecture here will be brief only to help those reading understand my plans for implementation of my bot. For those interested further I suggest reading my previous paper on poker bot architecture (see “Beyond Poki: A Distributed Architecture for an Advanced Poker Bot”).

Basically my architecture allows for a set of strategies to be combined and selected for use during the course of a poker game. Each strategy is a complete expert type system representing a set of ideas on how best to play poker. A strategy is really a collection of weights on a set of smaller modules. These modules represent small ways to understand a poker hand. These small modules are not complete in that a bot playing just by utilizing these small tools would fail miserably at defeating even the weakest player. But they are combined together in a unique way in each strategy to create a unique and complete way of playing.

The implementation I plan to work on this semester involves creating four basic strategies for my bot to follow and developing the modules necessary for them to work. Then
the challenge will be to give the bot the intelligence to choose between the 4 strategies depending on the game situation.

Evaluation of my bot’s performance once these strategies are built and running is a very important problem to solve. This needs to be done carefully and in depth to ensure that I can have an accurate idea of how strong my bot actually plays. Also, it is necessary to establish baseline performance so that I can assess future upgrades and new features.

Poker is clearly an exciting computer science problem, one that can be explored from thousands of different angles. It holds a lot of potential for showing a lot more than smart heuristics for solving a complicated problem. A computer program that can reason like this is solving the problem in much of the way that a human would. It is for that very reason that poker is more exciting than chess or backgammon. How a person plays poker is a reflection of how they think, and a robot that plays poker just as well as a human and that utilizes many of the same strategies as a human could ultimately be seen as a reflection of how humans think.

In this paper I have attempted to outline the theory behind all of poker decisions in my bot as well as general information on how these ideas are implemented. While I’m sure a more thorough analysis of the implementation would be helpful (class structures, code segments, algorithms) for the bot-maker, there is simply not enough room or time to do such a thing. The ideas are the fundamentally important things in my mind and how specifically I implemented them is less of a concern. My previous work on the architecture I am employing outlines a vast majority of this information and is worth reading if you are interested.

III. Poker Background

**Texas Hold’em**

My bot as well as the University of Alberta’s Poki was designed to play the poker game of Texas Hold’em. It is one of the most popular games in existence and presents interesting computational and game theoretical problems because community cards are held in common among players. While I try not to focus on the small details surrounding the game in my descriptions, a brief introduction to the game and its rules is in order to avoid confusion.

The game is played between as few as two players (“heads-up” play) and as many as 10 or so opponents at a table (any more just becomes unwieldy and will most likely be split into two tables). At the start of each hand, each player in the game is dealt two private cards face down. These are known as “hole” cards, or “pocket” cards. These cards are kept hidden from the other players. A round of betting occurs based solely on the strength of these two cards. This is known as pre-flop betting, because it occurs before the flop which consists of the first three community cards to be dealt.

In this first round of betting, players must-based only on their two pocket cards-decide on whether or not to call (stay in the hand by matching the bet), fold (throw their cards away and sit out until the next hand), or raise (equal the bet and then bet on top of the current bet). When all players have had a chance to bet or call the game continues.

The flop consists of three community cards dealt in the center of the table face up so that everyone can see them. These cards can be used by any player still currently in the hand. The object of the game is to make the best five card hand possible using the community cards and your hole cards. After the flop another round of betting occurs; this is known as post-flop

---

^1 Langhauser, “Beyond Poki: A Distributed Architecture for and Advanced Poker Bot”
betting. Post-flop betting is usually more aggressive than pre-flop betting because the players now can make a full five card hand and have a much better idea about the strength of their hand with only two community cards left to fall.

After the post-flop betting round another single community card is dealt face up in the center of the table. This is known as the turn or fourth-street. The players can now make their best five card hand from any of the six cards on the table (their 2 hole cards and the 4 community cards). It should be noted that there is no requirement as to how many of each different sets of cards a player can use. If for example at the end of the hand the five community cards makeup a player’s best hand, then this is completely fine and the player is said to be “playing the board”. This is of course far from favorable as everyone at the table has at least this strong of a hand for obvious reasons but it is nevertheless a valid hand.

After the betting round on fourth-street a final 5th community card is dealt face up in the center of the table. This is known as the river or fifth-street. This now gives the remaining players a total of seven cards to make their best five card hand from (2 hole cards and 5 in the community). After the river is dealt there is a final round of betting. At the end of this round the remaining players flip their cards and the winner is determined. This is called the showdown. The best five card hand wins the entire pot (the accumulation of all the bets throughout the hand). In the case of a tie the money is divided up among the winners (also known as a chop pot or split pot).

**Limit or No-Limit**

Texas hold’em can be played in one of two varieties, either limit or no-limit. In limit poker there is a maximum bet that can be placed for each round. It varies from game to game but the basic idea is that each player cannot bet any more than this maximum each round. An example would be $1 in the pre-flop betting round and $2 in the subsequent rounds. This means that player A before the flop can only bet $1 and then player B can only increase A’s bet by another dollar. Generally there are rules as to how many times players can raise and re-raise in any given round to guard against infinite re-raises that generate no-limit type situations.

Limit games require smaller amounts of skill in terms of bluffing and reading your opponents. After all, the results of any given hand should not make or break the player, but rather a string of poorly played hands will end up costing them big money. This means a player can survive simply playing by odds and almost entirely ignoring the game theoretical aspects of the game (bluffing, reading opposing players, trap-setting, etc). For this reason, while limit games still require a great deal of skill, they are less interesting from a computational standpoint. Odds calculations are things that computers excel at and it would make sense for them to play such a game well that is more reliant on these calculations. Getting them to play no-limit well is far more challenging.

In no-limit poker, there is quite simply no limit on how much a person can bet at anytime during the course of the hand. A player can bet all the money they have in front of them (known as going “all-in”) at any time during any betting round. What this creates is a game where reading an opponent and evaluating your hand strength relative to what you believe them to have is of the utmost importance. Whenever a player goes all-in there is the opportunity to take down a large pot and a player must be willing to call these bets at certain times. What this requires is a much deeper understanding of the odds and the player betting than in limit games. No-limit is also a dangerous game in that if you were to play one hand poorly you can end up flat broke immediately. The “swings” (the shifting of money around the table) are enormous in no-limit to
the point that even some professional players refuse to play because of the frustration level it creates.

No-limit also introduces some interesting game theoretical aspects that aren’t as prevalent in limit poker. If Player A has a large chip advantage over Player B it might make sense for Player A to go all in on a lower percentage hand for a couple reasons. Perhaps A does so anticipating that B will fold (a bluff) or that because A is so far on top of Player B he is willing to gamble a little (semi-bluff) for the opportunity to knock Player B out. Players with large chip advantages can “push their stack” around in no-limit, using their money to gamble and get lucky or to push weaker players out despite having a worse hand.

No-limit poker is a much more interesting game and much more fun to play at times because it is more action-packed than limit. It is also more difficult for a computer to play at an expert level which is why I have chosen to focus on no-limit for my poker bot. While I have not ruled out the possibility of my bot also playing limit, I want to make sure that I have the functionality of a no-limit bot. Other bots that I will discuss (Poki) has gone the other way, choosing to play exclusively limit texas hold’em and ignore the problems that no-limit can produce.

IV. The Software Platform

Poker Academy Pro

The software platform I have chosen for my poker bot is called Poker Academy Pro (PAP). It is an entirely java based application that is available for purchase at http://www.poker-academy.com/. Priced at $129 for a multi-computer license, it is reasonably cheap considering the wide variety and high level of its functionality. It contains a host of different poker analysis tools that have aided me infinitely in designing and testing my bot. These include above all player statistics, programmable dealers, and the ability to rewind hands to visualize errors in my bot’s judgment. The player statistics played a particularly big role in analyzing my bot’s play. Instead of having to track its decisions in each hand, I only had to review the logs kept by PAP on every player. They also have here cumulative analysis tools that evaluate the relative luck in drawing cards of players and the overall tightness and looseness of the player. They have been extremely useful. Another fantastic part about PAP are the built-in bots that it comes with. These bots play both limit and no-limit hold’em and are of varying strategies and skill levels. But using them to play my bot as part of the testing process was a huge help. They saved me hundreds of hours of writing my own test bots and devising what they should look like and how they should play. These bots (specifically the xenbots which are what I used for my testing) play solid all-around poker and made for great competition for my bot as it played and as I improved it.

Aside from the actual features of PAP that made it a great software platform to develop for, it also provided some important things that any software vendor should. The online documentation for the PAP API was phenomenal and the online help forums were filled with similar questions to mine and timely responses with well-thought out answers. The availability of free updates and how in touch the creators were with the users really taught me something about how to run a software company. Suggestions were taken from any source, debated, completely developed, and then implemented into the next version of PAP which is a great way to run any business. Their relationship with the University of Alberta and their bot Poki was another reason why PAP was a great software platform. Many of their limit bots utilize the Poki
AI and it also just provided another access point of information for my queries. Poki is by far the best limit bot out there and to be able to play against it and learn from its documentation was really valuable.

That being said, PAP is far from perfect and the problems with the way the creators implemented it impeded my progress on several occasions. Their secret player log files that are not meant to be read by humans or other applications (only PAP) made it much more difficult than necessary to find/store player log information for opponent modeling. It doesn’t seem to make sense why in a piece of learning software like PAP they would be so secretive about the way they store player statistics. I don’t see how it would jeopardize their business at all. The rigidity of the application’s interface also hurt my bot’s development in a few different ways. For one, a graphical user interface, while a great idea for humans, is completely unnecessary for 9 bots playing each other. There should be a way to switch out of it to speed up the games and in the current version there is not. This would have saved me a lot of trouble. Instead I was forced to build crufty macro scripts that actually manipulate the GUI. I would have liked to have methods I could implement to do these things but because of the GUI and the restraints of the platform I could not. Should there be a way to programmatically add money to a bot’s payroll? I think so, but PAP disagrees.

Bot Implementation

The API for implementing a poker bot in PAP is very simple to use as mentioned earlier and loads of online documentation is available. In an attempt to keep my poker AI separate from the specific platform it is built for I have divided up the code into two different packages. The PokerAcademyInterface package consists of one java source file that implements the Player interface required by the API for a poker bot. This class then sets up the main AI of the poker boy by referencing the jbotarch package and calling the appropriate methods.

In this way my AI is (for the most part) completely platform independent. The architecture and implementation of jbot can be applied to any platform by changing the interface package and making minor changes in the other classes. The reason the other changes must be made is because jbot relies on some information gathering methods that PAP has. A platform change would require gathering this information in potentially different methods and would need to be changed.

V. Performance Evaluation

The Problem to Solve

As I developed this poker bot it became increasingly clear to me that evaluation of its performance needed to be a primary concern of mine. Because part of the goal of developing a poker bot that plays at a level equivalent to a human is unpredictability, testing to ensure that the bot is functioning properly needs to be done carefully and thoroughly. It is important to know whether the bot is working properly and behaving unpredictably or it is malfunctioning.

When it comes to evaluation of poker players, the only true way to effectively test the player’s skill is to play as many hands against them as possible. Because there is an element of chance in what cards a player will receive, the larger the sample size in terms of hands played, the more accurate the evaluation statistics will be. I define accuracy here as being reflective of the player’s skill rather than their luck. The number of hands necessary to play before statistics
begin to emerge that are reflective of a player’s skills is difficult but a safe estimate would be on
the order of 10,000 hands.

Playing this many hands is easy for the computer to do, but if I were to try to play this
often against my poker bot I would need years for this project to get done. I needed a way to
play hands quickly and to get consistent levels of play from the opponents playing against my
bot. Luckily, the poker playing software I have chosen (Poker Academy Pro) comes with a wide
variety of bots itself that play cards at fairly intelligent levels. If I had my bot play against these
bots (called Xenbots) I could play a large number of hands fairly quickly.

Having played against Xenbots myself a few times I can tell you that they are definitely
not world class players, but they are certainly smart enough to establish solid testing results.
Their play will not be identical each time obviously as they are designed for unpredictability as
well, but as I discussed above, given a significant sampling size their play will take on certain
patterns that are reflective of their skill. For this reason I feel comfortable that different versions
of my bot will face off against relatively similar levels of competition for testing.

There was a time when I believed that I should play my bot against only one other test
bot in an attempt to speed up testing and simplify results. I have decided that this is not a wise
way to evaluate my bot for a few different reasons. For one, so many poker decisions are made
on the basis of the number of opponents in the hand. To eliminate this variable altogether by
making that number a constant would create testing results that weren’t really reflective of my
bot’s actual skill. Also, there is very little reason to speed up testing. Because PAP’s license is
for multiple computers I was able to spread out my testing on multiple machines and do testing
in parallel. The actual time testing took was not an issue.

I am stressing the importance of testing because honestly I did not appreciate it as much
as I should have and it cost my bot a great deal. Each week I did not assign testing near enough
priority – I was too busy writing code without testing each moving part of jbot. This is one of
my greatest regrets about this project, and it is something I will think about whenever doing
another major programming project.

The Test Bed

The test bed itself is a PAP poker room I have created that I refer to as “The Proving
Grounds”. It consists of 9 poker players: 8 test bots and my jbot. The test bots are the most
advanced kind of no-limit bots that PAP offers, called Xenbots. There are a variety of different
types of Xenbots and I have picked 9 that I deemed to be the most talented. I did this evaluation
just by playing against them for a dozen games or so. Each one plays different from the others,
but they share the same core AI and poker evaluation tools.

Because there are no human players at this table I have configured PAP to deal and play
much faster than usual. There is no time delay to play cute animations of cards being dealt or
time delays to build suspense during a showdown. The dealing and playing process is
streamlined as much as possible to ensure a large number of hands played. The poker bots make
decisions very quickly and the bottleneck to speed is really the GUI dealing animation which
cannot be eliminated. On average with 9 players at the table I was able to play 400 hands per
hour.

The game itself is a no-limit texas hold’em cash game. In a casino a cash game means
that everyone buys in for what they want and then cashes out (gets up and leaves) whenever they
feel like it. This differs from a tournament style of play where all the players start out with the
same amount of chips and they play until there is only one player left with chips – the winner. For testing purposes however all the bots start with the same amount of money and nobody ever cashes out. They play until they run out of money. The problem this presents is what if my bot runs out of money? What happens if the game ends and I’m not there to restart it? To fix these problems I wrote a few macro scripts that perform necessary functions. One such macro reloads my bot’s bankroll (gives it more money when it runs out), and another restarts the game once it has ended. Others are used to set things up when the game first begins. These macros are called from inside the code of my bot but only if a certain flag for testing mode is enabled. This flag is switched on and off from within Poker Academy Pro. This allows for my bot to be easily tested while still playing fairly in a real game situation.

VI. Bot Design

Utilities

Pre-Flop Hand Evaluator

Theory:
The basic idea between any hand evaluator is to give a ranking for a given hand based on (in texas hold’em) the player’s two hole cards and the community cards available. In general a hand evaluator is an essential part of any poker bot because it is necessary at each stage of the game to understand where your bot stands in terms of their hand. This allows them to play according. It is obvious that you want to play very differently (usually) if you have the best hand possible than if you have the worst hand. Hand evaluators in poker bot software are designed to be as fast as possible because if done naively they could prove to be a huge bottleneck in making decisions. It takes some computational intensity to determine what hand a player has and where it falls on the rankings of possible hands given the community cards. Evaluating your hand at every point where a decision is necessary with a slow evaluator can be very time consuming which is why in general a hand evaluator is a tough thing to write and must be done with great care.

Luckily for me hand evaluators have been around since poker was first put on a computer and there are literally thousands of open-source versions. The poker platform I am using to design my bot (Poker Academy Pro) in fact comes with a built in hand evaluator that is very fast and easy to use. But what this evaluator does not do is provide pre-flop hand rankings. These are rankings of the best possible hands when there are no community cards out – simply evaluating the two cards that a player holds in his or her hand. Luckily because there are no community cards and thus less hand possibilities to iterate through this is a relatively simple task. It is one that in fact doesn’t need to be done dynamically. A table of hand rankings can be determined off-line and hard coded directly into the program. This is because a pre-flop hand ranking is static. It is the same every hand and is thus much easier to determine than a hand ranking that takes into account community cards.

Implementation:
The method I chose to do the pre-flop hand evaluation is based on starting hand rankings determined by two professional poker players: David Sklansky and Mason Malmuth. They have, based on their own experience and simulations, created a hand ranking system that groups starting hands into 8 groups. Every hand can be assigned a number from 1-8 where the best
starting hands have a ranking of 1 and the worst starting hands have a ranking of 8. These rankings are based on a nine or ten person table.\textsuperscript{2} I also discovered a very useful algorithm known as the “Chen Point Count” for determining these rankings based on the two cards. This algorithm was not perfect and required some special cases to implement a complete and accurate pre-flop hand ranking.\textsuperscript{3}

**Decision Modules**

1. Pot Odds Decision Module

**Theory:**

Pot Odds are one of the most basic but most crucial ways to analyze any given poker situation. Simple pot odds are the odds that any player is given just based on the amount of money in the pot and the amount of money necessary to stay in the hand. A simple example with $90 in the pot and a bet of $10 to call gives you 1:9 pot odds or 10%. This is because you must pay 10 dollars for the possibility of winning 90 dollars. Pot odds themselves do not give you a good grasp of whether or not you should fold but if you couple pot odds with some sort of evaluation of your hand and chances at winning the pot then they are an invaluable deciding factor in any poker player’s mind.

The evaluation of the probability of winning is a more difficult calculation and can be done a few different ways. This calculation is then compared to the Pot Odds calculation. If the probability of winning the hand is greater than the odds the pot is giving you it is a wise decision to stay in the hand, otherwise it is wise to fold.

Hand potential is one way of assessing hand strength which is in essence what you need to know to make a decision. Poki provides a good description of hand potential that I will summarize: [hand potential](http://www.cs.ualberta.ca/~jonathan/Grad/papp/node40.html). Hand potential can be thought of as consisting of two parts: positive and negative. Positive hand potential is the probability of a hand improving as new community cards come out and negative hand potential is the probability of a hand being overtaken as community cards are dealt. Positive hand potential gives a good safe estimate of winning a hand if it is properly combined with the hand rank generated by the hand evaluator. Hand rank is a way of evaluating where a player’s current hand (taking into account their two hole cards and the community cards) ranks in terms of all the possible hands.

**Implementation:**

My Pot Odds decision module takes into account both hand ranking and a one-card look-ahead positive hand potential to make its decision given the immediate pot odds. The algorithm for doing this hand potential calculation is described in the above Poki link. The algorithm assigns an opponent every possible two cards they could have and then simulates every possible card coming out in the community. It keeps track of how often the new community card makes the player’s hand better than the opponent’s. It also keeps track of the number of times the community cards hurt the player (give the opponent a better hand) and cause them to tie (both hands have same rank). There is then a formula that turns this into the positive and negative


hand potentials. The positive hand potential, because this algorithm enumerates every possible opponent hand, provides a good estimate of the probability that the player will win the hand.

The module first makes some checks on the hand ranking. Obviously if the bot has the best possible hand it recommends raising, foregoing any sort of pot odds analysis. Otherwise it calculates the hand potential using the poki algorithm and compares it to pot odds. Based on this it makes the decision whether to call, fold, or raise.

2. Deception Module

Theory:

Any poker player that is expecting to win has to practice the art of deception in some form or another. Deceiving the other players at the table, bluffing, semi-bluffing, setting traps, and slow playing are the essence of intelligent poker play and cannot be ignored.

Most players would consider deception an art form and it comes in a few different varieties. Bluffing is the act of portraying a stronger hand than you actually possess. The goal of any good bluff is to get your opponent to fold so it involves betting enough money that they believe you to have a better hand than them even though in reality you probably have a worse hand. A semi-bluff is a somewhat similar strategy. A semi-bluff is generally done when a player has very good hand potential. Hand potential is an evaluation of the probability of your hand improving a great deal. For example if you have the Ace of spades and there are three spades in the community cards another spade will give you the strongest possible flush on the table. This hand has excellent draw potential and is certainly worthy of a semi-bluff. The goal of a semi-bluff is still to get your opponent to fold and thus involves betting as if you had a better hand than you do. The interesting part about semi-bluffs is that if you are unsuccessful in getting the other player to fold there is still the possibility that the next community card will come out and put you ahead of him or her. The last crucial deceptive strategy is known as slow-playing. It differs from the previous two in that it is done when a player has a very strong hand rather than a weak one. It is a form of trap setting that can be extremely effective when done properly, but devastating when not thought out. The purpose of slow-playing isn’t to get your opponent to fold. In fact at times it is to get your opponent to bet. When you firmly believe that you have a better hand than the other player(s) then it is probably a good time to slow-play a hand. Simply checking (not betting) will make your opponent believe that you have a weak hand and they will either bet or stay in the hand. An ideal slow-play is done when you cannot be beaten regardless of what the next community cards may bring, but this is rarely the case. The biggest danger with slow-playing is that you allow your opponent to draw a better hand than you because you let them see more community cards for free. For this reason it is necessary to be very careful when slow-playing.

Implementation:

My implementation of the Deception Decision Module provides a deceptive move for every situation. It uses hand rank and positive hand potential to evaluate the hand and then recommend an appropriate bet. A hand rank above a certain threshold (meaning the player has a very good hand) is slow-played meaning that the bot should just call. The decision of whether or not to semi-bluff is determined by a combination of hand rank and hand potential. Flat out bluffing is done with a little more complexity. Two random cards are drawn until the random hand picked is ranked higher than the player’s actual hand. The bot then acts as though it has this hand and determines its bet based upon that. This is done pre-flop as well as post-flop. The
thing to keep in mind about the deception module is that it will never suggest a fold which is why it must be used with caution. It has a deceptive move for every situation and if not utilized intelligently by the arbiter it could cost the player a lot of money.

3. Opponent Modeling

Theory:
The theory behind any opponent modeling system is relatively simple: to predict the actions of the other poker players at the table. The methods of doing so are varied, but they all rely on some sort of statistical analysis of the prior moves and decisions made by the opposing players. Poki utilizes a neural network to achieve this. Poki trains this neural network by feeding it the values of a dozen or so inputs and an output (the decision made by the opponent). After training these same inputs can be fed in and will predict the output (the decision the opponent will most probably make) based on the training inputs.

Implementation:
An artificial neural network certainly has its benefits (unlimited inputs, easily trained and utilized, standard AI tools with many libraries for it), I decided against using one for a couple main reasons. For one, it seemed like it was overkill for my purposes and that I could use a lighter method that didn’t require so much overhead to setup. Secondly, I frankly had a lot of trouble not only comprehending the fundamentals of programming with neural networks, but I also had a lot of problems turning my problem into one that can be solved by neural networks.

I decided instead to use a fairly simple table to keep track of an opponent’s past decisions. There are two main versions of this table, one for modeling pre-flop decisions and one for modeling post-flop decisions. The pre-flop table is reproduced below:

<table>
<thead>
<tr>
<th></th>
<th>Early Position</th>
<th>Late Position</th>
<th>Dealer</th>
<th>Small Blind</th>
<th>Big Blind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Active Player</td>
<td>BetFreq:2</td>
<td>CallFreq:5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FoldFreq:2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less than 5 Active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Players</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greater than 5 Active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Players</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greater than 7 Active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Players</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Basically the purpose of the table is to represent the interaction between two different variables and the decisions the opponent makes given the values of the variables. The two main variables considered pre-flop are the player’s position at the table and the number of active players in the hand when they make their action. The post-flop table is similar but is three-dimensional and
takes into account a third variable: community card “type”. This is a measure of what kind of community cards are on the board (pairs, flush draws, high cards, low cards, straight draws, etc).

The strength of this modeling system is the easy lookup after it is trained. It is a simple table lookup to get the probabilities of a player folding, calling, or betting in a given situation. The obvious weakness of this modeling system is that the variables have discrete quantized values instead of being given a wider range of values. This is what makes neural networks an appealing option – the inputs can be of any values. In my model they must be first translated into these quantized values and then can be evaluated. I think that in a second version of jbot I would definitely make a push to better understand neural networks so I could use them. I just didn’t have the proper amount of time to digest the literature on them and understand it completely enough to implement one.

**Strategies (Arbiters)**

1. **Tight Play Arbiter**

Theory:

A player playing tightly is going to play less hands as he/she will play only hands that he/she has a relatively good chance of winning. A player can certainly play too tightly and see themselves laying down a large number of winning hands, but more often than not, a player does not play tight enough and ends up losing a great deal of money because they play hands that do not win or do not pan out in terms of drawing good community cards. An arbiter represents a complete poker strategy and so a tight play arbiter should be complete in the sense that if one were to play only adhering to its decisions they would be playing smart poker. It should favor more conservative forms of hand evaluation and not take chances with risky reads or “hunches” about other players. It should really only play a hand when there is a very good chance of winning it.

Implementation:

In terms of my architecture and implementation, a loose arbiter can be best visualized as a weighted decision of what modules to trust. My poker bot consists of a pot odds decision module, a deception module, and an opponent modeling module and these three all have varying degrees of certainty in the calculations they perform. While these certainties aren’t necessarily quantifiable (and if they were it still wouldn’t be extremely helpful) there is a clear sense about which decision modules produce more conservative evaluations of hands. Pot Odds vs Hand Potential is by far the most conservative and foolproof decision module and so in my tight arbiter that is given an 80% weight when weighted sum is performed of the decisions generated by the modules. The Opponent Modeling module is probably the least reliable of any decision module because it involves extrapolation based on players’ past moves. While this is done intelligently and is worth trusting at times if a player is playing tightly or conservative, then this should be trusted a lot less frequently than if a player were playing loosely. This module is given only a 10% weighting. The deception module is also given a 10% weighting but not necessarily because its decisions are unreliable in the same way that the decisions of the opponent modeling could be unreliable. Bluffing and other forms of deception are crucial parts of good poker play but they should not be done as frequently if a player is playing tight. Conservative players will rarely bluff or slow play because it puts them at risk of losing more money than necessary.
2. Loose Player Arbiter

Theory:
A loose player is much less conservative with their betting and is willing to gamble much more often on bad hands or hands that have strong draw potential. Playing loosely can be a severe detriment at times for obvious reasons. But looseness can be a powerful tool for a skilled player who knows when to use it. Loose players put themselves in hands and whenever you are in a hand you give yourself a chance to draw cards and take down big pots. Looseness is also a tool to be used in response to opponents’ play. If an opponent is playing extremely tightly then the best way to defeat them is to counteract their conservative play with loose play. A loose player will be able to bluff more successfully against tight players as well as set traps for them. A loose player will be more willing to trust their instincts about an opponent’s strategy and make decisions based more on intuition than odds.

Implementation:
My loose arbiter simply puts more trust in the Deception Module and the Opponent Modeling decision module than the Pot Odds Decision Module. Both the Deception Module and the Opponent Modeling DMs are given a weighting of .45 each while the Pot Odds DM is given the remaining .1 weighting when the sum is computed. This makes the bot play much more hands that it normally would because decisions are more heavily influenced by the Deception Module (where a fold is never recommended). It also introduces more of an element of luck because the bot is playing more bluffs and hopefully drawing miracle community cards if called.

3. David Sklansky Arbiter

Theory:
David Sklansky is an American poker player and writer who has published two books detailing his philosophy on poker strategy. His approach is more mathematical that most poker players as he supports “The Fundamental Theorem of Poker” above all else. This Theorem states that “Every time you play a hand differently from the way you would played it if you could see all your opponents’ cards, they gain; and every time you play your hand the same way you would have played it if you could see all their cards, they lose.” Aside from this general axiom a great deal of Sklansky’s strategy is based on pot odds, both immediate pot odds and implied pot odds. Immediate are those previously described that can be calculated knowing only the amount of money in the pot and the amount of money necessary to stay in the hand. Implied odds involve some opponent modeling as you try to envision the bet that your opponent will make in the future and then calculate the pot odds based on that number.

He spends a lot of time thinking about bluffing, semi-bluffing, and slow playing. He discusses in detail determining the appropriate bluffing frequency and predicting traps set by opponents. He is able to combine deception and opponent modeling extremely well to turn himself into a complete poker player.

Implementation:
My implementation of the Sklansky arbiter did not turn out as well as I hoped. I originally saw it as operating outside of the normal standard for arbiters (that utilize percentages

---

4 David Sklansky “The Theory of Poker”, Nevada, Two Plus Two Publishing 1999 Pg 17
to weight the DMs). I saw it as using special heuristics that Sklansky thinks about and utilizing the decision modules in creative ways. I unfortunately was not able to finish this kind of advanced arbiter in time and was forced to settle on a percentage weighting system of the decision modules that I believe to be reflective of Sklansky’s theory of poker. It weights the Pot Odds DM at 60%, the Deception Module at 30%, and the Opponent Modeling DM at 10%.

4. Doyle Brunson Arbiter

Theory:
Doyle Brunson is a living legend in the poker world. In his career he won two back to back World Series of Poker Main Event titles as well as turned himself into a millionaire playing cards. He is also by far the most widely-read poker writer, as his book Super System is considered almost unanimously to be the no-limit texas hold'em bible. He has a distinct style of play, which he coins as “tight-aggressive”. He plays more by feel than most, able to read his opponents very well, but also a student of the game and able to make tough decisions mathematically. He puts tremendous pressure on players by betting heavily and forces a great number of uncertain hands to be folded. His game is very fluid however and he is able to change and adapt to his opponents better than almost anyone else. He is widely considered the best poker player to have ever existed and a strategy based on his teachings is a challenging but important component to consider.

Implementation:
Similarly with the Brunson arbiter, I did not get a chance to create it as I wished to. I failed to even create a Brunson arbiter because I felt that if I did not have time to do it properly, I shouldn’t do it at all. It would be disrespectful to Doyle Brunson himself. I hope to work on this at a later date. I see it as being similar to my ideal for the Sklansky arbiter in utilizing heuristics and special strategies of Doyle’s to make decisions.

VII. Evaluation Results and Analysis

Baseline Testing: Pot Odds Only

Hands Played: 10688
Dollars won: -15270.40
Games: 14

Preflop: Fold: 64%
Call: 20%
Check: 12%
Bet: 4%

Pre-Flop Played: 25%
Pre-Flop Aggression: .2 on a 0-4 scale
Post-Flop Aggression: 1.8 on a 0-4 scale
Went to Showdown and won: 60%

Flop: Fold: 22%
Call: 9%
Check: 51%
Bet: 17%

Turn: Fold: 21%
Call: 12%
River: Fold: 17%
    Call: 10%
    Check: 55%
    Bet: 18%

These statistics paint a very good picture of how jbot (using only the pot odds dm) plays and the
improvements that need to be made. The large percentage of decisions that were checks shows
how tight jbot is playing which really cost the bot in later rounds as he allowed his opponents to
catch too many cards. A more aggressive player would have bet opponents with weak hands out
and not had so many tough beats on the turn or river. While it is fine to play 25% of hands, the
pre-flop aggression of .2 is horrible. Hands that are worth playing are generally worth raising
and specific hands need to be raised to protect them in later rounds. Jbot did not do this and this
is part of the reason why it lost so much money.

Post-flop jbot plays much better in terms of aggression. Betting is done more frequently
and he projects much more strength. Unfortunately this is to his detriment many times. By not
betting pre-flop he allows players to stay in the hand with bad cards only to catch something on
the flop. Jbot’s bet post-flop does nothing but sweeten the pot for the player who caught two
pair, 2’s and 3’s for example. Even so there is still an overwhelming percentage of checks on
every round and jbot needs to play much more aggressively if he is to find success.

More thorough testing of each part of jbot is required before further analysis can be done.
I have tested the deception module and portions of the opponent modeling decision module but
not nearly enough to have meaningful data that can be analyzed. The debugging process
significantly slowed testing and I’m still searching for a few bugs that are causing bad but non-
fatal runtime errors in my deception module. This debugging must be completed before
complete testing and analysis can be done of the other parts of jbot.

VIII. Future Work and Conclusion

Poker as a research field within computer science is wide and complicated, its potential to
advance the science so far from being tapped, that it is vital that computer scientists constantly
attempt to leverage it. And that is what I was trying to do in this bot. I think while I certainly
did not achieve my lofty goals of creating an expert level poker player, I think I did make
progress towards that goal. I think the architecture that I developed and utilized here is a large
step forward and can be built upon with later work.

The answer to the problem of expert level poker playing computers is in part a question
of tweaking the architecture that I have worked hard to enumerate and utilize. My architecture is
far from perfect and thought I was aware of this before I began work, I realized it much more as I
worked on jbot. The problems are less philosophical as they are mechanical in my opinion
which is a good thing. What I mean by this is that the idea of my architecture is crucial to
solving the problem. The basic premise of overlaying expert systems on top of adaptive learning
systems is crucial to the development of any extensive AI program. It is the crux of what we
need, as computer scientists, to utilize in order to build advanced systems. The main problem
with my architecture is that the adaptive learning element is not as strong or clearly defined as it
should be. Switching between expert systems is the most vital part of building a poker bot that is unpredictable and successful. This is true for any system that is looking to build complicated behaviors that build on one another, systems that truly do learn.

Aside from my architecture, there were other problems that plagued my work and caused problems with jbot. Despite being told again and again how important evaluation of my bot’s performance was, I did not attach enough significance to it. My test bed was well-thought out but I did not follow through with enough testing of my bot’s inner-workings. Testing seemed to get thrown by the wayside each week which was a huge problem. I think my future work will take more of an extreme attempt to this by testing every small part until I am completely satisfied that it is working as I want it to.

But summing up my project, I do believe that the work done was very important, albeit perhaps sloppily at points. It was certainly difficult to cram so much work into a short semester and perhaps the speed at which I was forced to complete certain elements contributed to the lack of completeness and lack of total performance. I see it as an important first step, one that myself and others can build upon. The next step I foresee is to revamp the architecture, correcting the problems that were evident in the first generation bot and then rebuilding jbot. This bot served as both a good first test of my previously developed architecture and a good case study for the development of a poker bot. I still have visions of jbot becoming more and more powerful with every new decision module that is added. I see it as being the sum of hundreds of smaller bots and the more different strategies I can enumerate, the better it will become. But there needs to be some architectural changes before these additions can be made.

IX. References


University of Alberta’s Artificial Intelligence/Probability Archive
http://www.cs.ualberta.ca/~jonathan/Grad/papp/node1.html

University Of Alberta’s Poker Research Group <http://www.cs.ualberta.ca/~games/poker/>

Wikipedia’s Poker Probability Entry