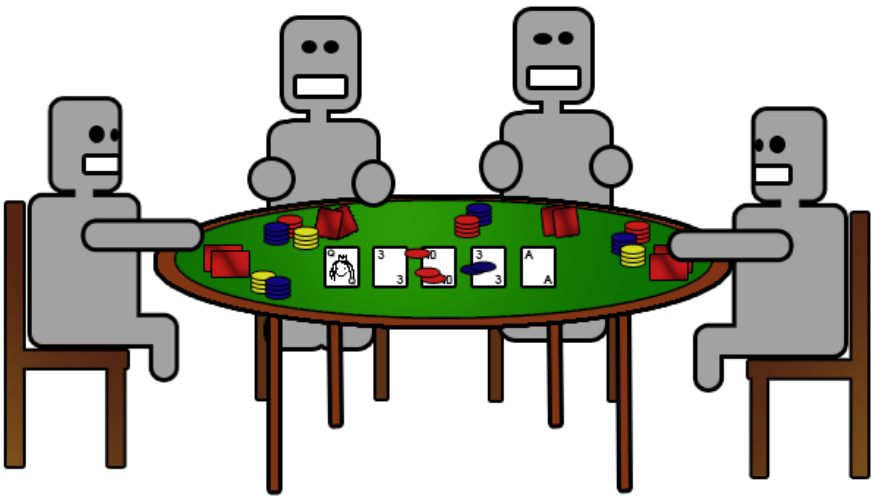
YAPA

Yet Another Poker Agent

Dat3 semester project in Computer Science fall 2010

Group:

d303a



Department of Computer Science
Aalborg University

Department of Computer Science
Computer Science
Selma Lagerlöfs vej 300
https://intranet.cs.aau.dk/education/computerscience/dat3/

Title: YAPA

Theme: Modelling

Project period:
1/9 – 17/12, 2010

Project group:
d303a

Group members:
Kaj Richard Nielsen
Kim Sø Pedersen
Casper Nicolaj Sparre
Mikkel Holm Søgaard
Christian Høgh Pedersen
Marius Pallisgaard Olsen
Heidi Selmer Nielsen

Supervisor:
Ricardo Gomes Lage

Circulation: 9

Number of pages: 60

Finished at: 17/12-2010

Summary:

This is a Dat3 report detailing the development of a poker agent. The report contains theory for estimating poker odds, and the creation of a learning poker agent. The agent is created through the use of a Bayesian network for a decision network, and a k-nearest neighbour algorithm for learning through pattern recognition. Learning data is acquired by storing poker hands in XML format, for analysis. The agent is tested against human players, and a previously developed poker agent. The agent finally achieved a 48.3% win rate against human test persons, 76% a previously developed agent and 68% against a randomly acting agent. It is concluded that the agent is functional, but could be improved upon. Firstly through the addition of bluffing capabilities, strategic understanding and modified probabilities for multiple players.

Preface

This report, which concerns the creation of a poker agent, is written by group d303a in the time span between September 1 and December 17, 2010, during the Dat3 semester of the Computer Science education at Aalborg University.
The reader is assumed to have background knowledge on computer science topics at least equivalent to a Dat3 student.

Reading Guide

Reading guide for the report.

Abbreviations

Some terms are repeated often in the report and therefore they are abbreviated. The first time such a term is written, it will be written in full followed by a parenthesis that contains the abbreviation. Subsequent occurrences will be written as the abbreviation alone.
Example of first time use: This is a Test Abbreviation (TTA).
Example of a subsequent use: TTA.

Citation

Citations are written as a number in square brackets. The source associated with the number can be found in the bibliography on page 59.
Example of a citation: [1].

Quotes

Quotes will be written in quotation marks followed by the citation to the source with page number, e.g.:
"*The fundamental idea of decision theory is that an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action.*"[1, page 483]

Code

Code will be written in listings like this:

```
1 This
2 could
3 be
4 code
```

Listing 1: Listing.

Typewriter Font

In lined text that contain specific implementation words are written in typewriter font, e.g. `main`.

Acknowledgment

A great many thanks to Dion Bak Christensen, Henrik Ossipoff Hansen, Anders Hesselager, Lasse Juul, Kasper Kastaniegaard and Michael Skoett Madsen for kindly lending us their poker engine and agent, which has both saved an enormity of time and provided a comparison with which to evaluate our own poker agent. Also thanks to our supervisor, Ricardo Gomes Lage, for his assistance. And thanks to Thomas D. Nielsen for supplying guidance in regards to decision networks and pattern recognition. Also thanks to the people who where so kind to play against YAPA for the tests: Klement Johansen, Jesper Rosenkilde, Jakob Errebo, Anders Hesselager, Martin Mortensen and Christian Kjaer.

Table of Contents

Chapter 1

Introduction

Poker is an exciting game, with its high-paced strategy and the tension as players try to exude confidence in a bad hand, or frantically try to adapt and change their playstyle in a desperate effort to stay ahead of their opponents. Identifying what their opponents are doing without solid knowledge of what cards they possess. Doing all of the above without showing any outward signs of what is going on in one's head. And, of course, there is often great big piles of money to be won.
Then there is trying to teach all that to a machine; strategy, deceit, unpredectability and a desire for personal enrichment.
Poker fits into a specific niche within Artificial Intelligence (AI) known as an "imperfect knowledge" scenario. These scenarios differs from other well-known applications of AI, such as chess programs, in that the AI never possesses complete knowledge of the resources and possibilities available to its opponent. This prevents the computer from simply calculating every outcome of a given move and picking the most favourable one, forcing it to perform based on calculations of probabilities, which infers new problems and possibilities for the AI. The general differences between perfect- and imperfect knowledge scenarios are listed in greater detail below.

Perfect Knowledge
: Perfect knowledge describes the situation where the AI has the perfect information set. That is, when the AI can determine the exact state of the situation at any given time. An example of such a scenario could be checkers or chess. These types of games have been targets for AI agent development for quite some time. e.g. IBM's Deep Blue that could evaluate 200 million chess moves a second, this lead to a win in the epic "Deep Blue vs. Kasparov, 1996, Game 1". A Perfect Knowledge scenario needs a technique for calculating all possible scenarios deep into the game. Of course, actually creating such a technique, may be much more difficult than simply knowing that it must the created..[1]

Imperfect Knowledge
: In an imperfect knowledge scenario, there will be states that the AI agent know nothing about. Staying within the game domain, we consider Texas Hold'em Poker. Here the opponent's cards are unknown and the agent has to make guesses based on the partial knowledge that it possesses. This is often done by calculating the probability of all possible outcomes and then make a decision as to which is probably the most likely one.[1]

There are two sides to poker: playing the cards, and playing your opponents; It is not enough to simply make decisions based on the probabilities of a good hand. decisions must

be based on the cards and the actions of the opponents. The interpretation of these factors are further modified by the amount of money that one is investing at a given time. The probabilities of the card game itself is not the greatest challenge, neither is it the most significant aspect. Predicting the actions of other players is a more difficult challenge. It is challenging enough to begin with, even with the natural tools for evaluating them, that evolution has provided humans with. Programming an AI which possesses the ability to identify human behaviour would necessitate that it could perform the same estimations as a human, without the knowledge that our senses intuitively provide us with. An AI capable of playing poker needs to be able to evaluate its opponents, based solely on their actions. Even if one develops an AI capable of correctly assessing human opponents, this would not be sufficient, since with poker, predictability is the bane of success. So even an AI capable of flawless and efficient analysis of its opponents is far from guaranteed victory if it does not conceal its own intentions.
This report will explore the imperfect knowledge scenario through the creation of Yet Another Poker Agent, or YAPA for short. Since many possibilities and decisions have to be considered and made during a game of poker, YAPA will need a decision-model which will enable it to take intelligent actions. But if YAPA is going to be a good poker agent, it needs to be able to recognise patterns of its opponents and predict their actions. This could be achieved through data mining and pattern recognition. Since recognising a bluff and being able to bluff oneself is an essential part of a poker game, YAPA will need to learn how to identify these so it can make better decisions, and hopefully bluff by itself to achieve a decent win-rate. Since poker is often played for money, it seems that YAPA's success would hinge on it winning more money than it loses.

1.1 Aim of the Project

This project aims to produce a functioning poker agent capable of performing basic analysis of opposing players' play style, and adapt its own game accordingly.
We will achieve this goal by utilising a decision network to model the backbone of the poker agent's behaviour, and supplementing this network with pattern recognition techniques to enable the more advanced capabilities of the agent; The agent should not just act based on the probabilities of the poker game itself, but modify its behaviour according to the play style of its opponents'.

1.2 Constraints of the Project

Developing an AI capable of poker is a complex endeavour, but there are some aspects which are more integral than others, and some which are essential for the implementation of YAPA, but not in themselves particularly relevant to the subject of the report. There is also an upper limit to how much work can be dedicated to this project, restricted by the duration of the semester. These two has lead to some predetermined restrictions to the work of this project. These are listed below.

- Developing an engine within which YAPA can be implemented and tested would be a time-consuming and manpower-heavy endeavour. Therefore, it has been decided to utilise an existing poker engine, created by another university group for their own work with creating a poker agent.

- YAPA will be developed with a single opponent in mind, adapting it for play against multiple opponents will not be a standard goal for this project.

- As the complete set of possible poker hands run well into the millions, there will be some abstraction over the considered card combinations in order to reduce the complexity of decisions that YAPA will perform.

1.3 Structure of the Report

The report is organised into seven chapters.
Chapter 1 introduces the motivation, goals and limitations for the project.
Chapter 2 contains the rules for Texas Hold'em poker, as well as considerations of poker strategy and psychology. Chapter 3 contains theory regarding machine intelligence, probability and decision modeling as is relevant to the development of YAPA. 4 describes the process of designing YAPA; The construction of a functional decision network.
Chapter 5 consists of the implementation chapter, which details the coding of YAPA. The code of the agent is not shown in its entirety. Rather, the discussion of the code is limited to some of the most interesting aspects of it.
Chapter 6 contains the test criteria for YAPA, the test results and the evaluation of the test results.
Chapter 7 summarises the state of YAPA, concludes the report and also lists possible advances for the future.

Chapter 2

Poker Rules and Strategies

In the following section there will be a brief description of the rules and game types of Texas Hold'em Poker. Furthermore there will be an overview of some poker strategies and profiles.

2.1 Rules For Texas Hold'em Poker

The game of poker is comprised of a number of rounds called **hands**. The dealer in the game is determined by the **dealer-button**, which moves clockwise between the players and is moved to the next player at the start of each hand. Play begins with each player being dealt two cards face down from the dealers left and then clockwise, one card at a time. The two players to the left of the dealer make **blind** bets. The blind bets (or **blinds**) are determined beforehand. The player to the dealers immediate left pays the **small-blind** and the player to the left of the small-blind pays the **big-blind**. The big blind is twice the small blind, e.g. if the small blind is \$10, the big blind is \$20. The player to the left of the big-blind is the first player to make a betting action and this first stage of the game is called the **pre-flop**. The players can choose to **call** (**check**), **call**, **bet** (**raise**) or **fold**. The other players then make their choice of action one by one, going clockwise between the players.

- Check: If no bet has been made the player can check. Here no money is added to the pot.
- Call: The player matches the previous bet. If a player do not have enough chips to entirely match the bet, a side-pot is made.
- Bet (Raise): Is to increase the bet. If the other players want to continue to play the hand, they have to match this amount.
- Fold: To fold is to give up the hand. The player sits out until the next hand is dealt.

After all players have either called (checked) all bets (raises), or have folded, the first three community cards, of five total, is dealt. This is called **the flop** and is done by first removing the top card of the stack, which is called burning the card. This is done to make sure no one saw the top card and to help prevent cheating. After burning the top card, three cards are dealt face up. The player to the left of the dealer now starts the second betting round following the same structure as before. When betting is done, the last two community card are dealt, again after burning the top card before each. These cards are called **the turn** and **the river**. Between and after dealing the turn and the river there are betting rounds.

At the end of the river's betting round all the remaining players turn over their two cards and the player who has the best five card combination is the winner. This is called **the showdown**. However, if at one point, the second to last player folds, the last player automatically wins.

Below is a list of all valid hand types, in decreasing order of rank/value.

- Straight Flush - Five cards with sequential values, all of the same suit. Since it is not possible to build a straight flush from King over Two. A hand that has Queen, King, Ace, Two and Three of the same suit. This is just a regular Flush with Ace as the high card. Hence the highest straight flush combination is Ace to Ten, and the Lowest Five to Ace.
- Four-of-a-Kind - Four cards with the same value.
- Full House - Three-of-a-Kind and a Pair.
- Flush - Five cards of the same suit.
- Straight - Five cards with sequential values, regardless of suit. As with a straight flush, it is not possible to build a straight with a King over Two combination.
- Three-of-a-Kind - Three cards with the same value.
- Two Pairs - Two Pairs.
- One Pair - Two cards with the same value.
- High Card - Your highest card determines your hand.

If two players have the same hand, the player with the highest excess card or cards, also called **kickers**, wins the hand, each player can only use five cards to build his or her combination. [2]

2.1.1 Different types of Texas Hold'em

Texas Hold'em Poker comes in a few different varieties. The basic rules are the same, but the betting structure can be different. They are described below.

Limit Texas Hold'em In a limit game, the size of the bets are predefined. Often the same size as the big-blind. Also each player can make a maximum of four bets in each betting round.

No Limit Texas Hold'em In a no limit game, there are no upper limit on the bets. However there are a minimum bet size, which is the same as the big-blind. There are also a minimum raise size. The raise have to be at least the same as the previous bet.

Pot Limit Texas Hold'em In a pot limit game, the size of the pot defines how much a player can bet in a single action. The maximum raise size is limited by the size of the pot plus the cost of calling the previous bet.

Mixed Texas Hold'em The last common type is the mixed game. In this type the game varies between limit and no limit rules.

The Texas Hold'em type considered in this project is No Limit Texas Hold'em.

2.2 Poker Profiles and Strategy

When playing no limit Texas Hold 'em Poker, relying just on luck will rarely be enough to win a tournament game. Literature and interviews with professional poker players, will describe in detail how they categorise their opponents, how they observe them and how they vary their playing style according to what they observe.[3][4][5] There are many different parameters a professional poker player use during a game. This section will describe the parameters that YAPA will observe and use.

2.2.1 Basic Poker Playing Styles

There are four basic playing styles in no limit Texas Hold'em Poker, or just poker for the rest of this section. The four styles that are shown in Figure 2.1 on the following page are as many other things subject to interpretation. Figure 2.1 displays the general strategy and view on poker, as it is going to be used in YAPA. The four playing styles will be defined in the following listing:

- Loose-Passive: This playing style is the most common for amateurs, its a category for people who buys into the flop, but very rarely raises their bet. It is a quick way to lose many chips fast in the game, these players often reason their playing style, saying that what comes in the flop is luck, therefore they might as well see it.
- Loose-Aggressive: Many of the people who use this playing style, are either amateurs or very experienced players. The amateurs who use it, mostly have a problem with not being in control at the table. They have to bet, raise and even re-raise other players, this is a weakness that can be exploited. The experienced players that use this style, use it to set up traps for their opponents.
- Tight-Passive: Players that use tight-passive, do not buy into the flop very often, and do not bet, raise or re-raise very often, this style can get players into trouble when the blinds are getting big.
- Tight-Aggressive: Many poker professionals agree that this is the most winning playing style over time, a tight-aggressive player won't buy into the flop or go beyond the flop unless he or she has excellent cards, when this is the case they are usually very aggressive players.

To expand, loose and tight refers to how many hands the player will participate in. Loose players will buy into the flop, and possibly beyond quite often, regardless whether they have a good hand or not. This means they depend a lot on luck, to get the good combinations and winning. Tight players are the opposite, they wait for the good pre-flop cards, or a good flop if they can see it fairly cheaply, and tend to only play the hand out, when they feel they have a good chance of winning.

Passive and aggressive refers to the player's betting strategy. Passive players often just goes along with whatever the other players are bidding, and will rarely be the one to throw in big bets, and drive up the pot. Aggressive players likes to be in control by raising and re-raising a lot, and push the opponents to throw more and more money into the pot.

An important thing to remember when reading the above, is that any of these playing styles are subject to analysis and it is possible to use your opponent's playing style to set up traps for him or her, or simply just avoid getting trapped by a better hand. Another important

consideration is that there are no absolutes in poker. You might believe that you have identified your opponent's playing style, but he or she might change it over time and shift into another playing style, giving your opponent a possibility to bluff or entrap you.

This means that even though profiling opponents can give a general idea of how and opponent is expected to play, and can help predict what cards he is holding, one should be careful not to rely too heavily on this. If a player trusts he has identified an opponent accurately, and lets that decide his actions there is a possibility that an opponent can exploit that fact, if he realises what is going on.

Figure 2.1: Basic poker styles[6]

2.2.2 Profiling Parameters

As stated above, there are four archetypes of poker players, comprised of just two characteristics. The first being betting strategy; passive and aggressive. And the second being the amount of hands the player will actually decide to play or fold, being loose and tight. The parameters available for determining styles are the bets: fold, check, call and raise, and the actually cards, especially the final hands.

When categorising a player's betting style, one should look for a very low number of raises, for the passive players, and a high number of raises and calls for aggressive players. Passive players tend to just go along with the current bet, by checking and calling, but they rarely raise the bet themselves. Aggressive players like to take control, and will often decide to raise the stakes, by raising the bet. So for deciding if a player is passive or aggressive, especially raises, or the lack thereof, are interesting.

As for the tightness of the player, you will need to see a high number of final hands, in order to determine if a player is tight. Loose players will usually buy into most hands, deciding to pay to see at least the flop, and often play a hand all the way, relying on a high degree of luck to win their hands. Obviously, this depends on the bets, in the sense that a hand can become too expensive for the loose player to play at some point. Tight players are much more selective about the hands they will buy into, and most of the time they will wait for a hand with a high winning chance. In essence, this will manifest in a lot of early folds, but again it depends on the bets.

In order to categorise players as loose/tight, you need a high number of final hands. Otherwise the player could easily be wrongly categorised, due to the high influence of chance in poker. Specifically a tight player which happens to hit a string of good hands, could easily

be thought to be a loose player, without seeing the final hand, to determine the actually strength of the hand. Likewise this needs to be correlated with the betting, as even loose players might fold a lot of hands early, if there is some very aggressive betting going on.

However, once the player has been classified as being tight or loose, we can start making qualified predictions, towards the player's current hand, based on his history. Per definition, a tight player will participate actively in fewer hands, therefore when he does play a hand, we can predict a higher probability that he has a hand with a decent winning chance. Of course there is the possibility that a player shifts his styles so often, that he can never be classified. These opponents should be played against very carefully, relying more on good hands, to consistently beat the opponent straight up. Furthermore, once a player has been classified, we have to watch out for shifts in his play, and constantly be ready to re-evaluate the classification.

Bluffs are the real challenge, and are hard to impossible to predict based on history. A bluff is when a player tries to trick an opponent. This could either be by betting aggressively, to make his opponent think his hand is much stronger than it actually is. Obviously if he gets called and ends up showing his hand, he will likely lose, so the objective of this bluff is to get the opponent to fold.

Another kind of bluff is the trap, which is the situation, where the player has a very strong hand, but he still decides to play somewhat passively. The objective here is to get the opponent to put more and more money into the pot, to the point where the opponent is so invested, that he feels he has to call, so not to lose an unaffordable sum of money. This generally works best when the opponent has a fairly decent hand himself, so he will be willing to raise, and feel like he is dictating play.

Statistically, one could claim a loose player bluffs more often. If you define a bluff as playing a hand out, even if it have a limited chance of winning, in the hope of the opponents will eventually fold, if he is not successful in hitting a good combination. However, as loose players decide to play out a lot of hands, that could almost be the definition of bluffing, and therefore this is not a very accurate description.

As for tight players, by categorising them as tight, we predict a high probability of them having a good hand if they buy in to the flop. This basically means that, as long as tight players do not bluff too often, they could ensure successful bluffs every now and again, by disguising it within the tight pattern.

Likewise, bluffing in regards to betting can be hidden under the passive and aggressive patterns. Passive players might set up traps, and aggressive players could be perceived as often bluffing, as they like to dictate play in the hands they participate in.

2.2.3 Stack Size

This section is based on [7]. The amount of chips/money each player has, also called stack size, can strongly influence the way they play. Early in a game, when every player has about the same, they will usual fall into whichever style they play by. However, when one player has a very big stack (a lot more chips than average at the table), or a very small stack (much

less chips than average), they tend to change their playing style.

Big stacks allows for a more aggressive playing style, as the extra amount of chips over opponents, allows the big stack holder to dictate play, and even at times to buy the pot. Buying the pot essentially means betting many more chips, than the opponents are willing to, and forcing them to fold. But buying the pot is of course always risky, as an opponent might have a better hand, and decide to play out the hand. Also the big stack style tends to be more loose, as it becomes more affordable to see the flop, turn and river, in the sense that it will cost the big stack player a smaller percentage of his chips to call, compared to the other players.

Small stacks tends to pull players in the opposite direction. The shortage of chips means the player will need to conserve his resources, and only play hands with a decent winning chance. Therefore the playing style will lean towards the tight-passive style. However the blinds pose a problem for the small stack player, as they will slowly eat away at his resources, if he becomes too passive. Therefore the small stack player will often bide his time, and end up going all in or at least play very aggressively, when he gets a good hand, or he is forced to before the blinds themselves eliminate him, in order to make a last ditch effort. This effort will often either eliminate the player from the game, or result in a more competitive stack size.

2.2.4 Table Positions

This section is based on [8]. Table positions in poker can play an important role in a player's strategy. Table positions are the order in which players get to act during each round. Generally, three categories of positions are considered, and the number of players in each category depends on the number of players around the table. The positions are; early position, middle position and late position. Because of advantages and disadvantages to the different positions, the positions change for each hand played during the game.

Early positions are roughly the first third of the players that get to act, the players positioned directly after the big blind. Being in the early position is generally regarded as a disadvantage. This is because the later positioned opponents can await your move and act accordingly.

Middle positions, the second third of the players, are regarded as being more neutral than the early positions, but still slightly at a disadvantage. A middle position can await the early position, but is still at the mercy of the late positions. Also a difficult situation can arise, if an early position player is very aggressive, where the middle position gets caught between the aggressive play of the early position, and the responsive play of the late position.

Late positions are the last players to act on a hand. This is a definite advantage, as they are able to gather information, by observing the opponents before making a decision. Especially the late position allows the player to execute a strategy known as stealing the pot. This is when the earlier positioned players have only checked or otherwise played passively, displaying no sign of strength in their cards. The late position player then attempts to steal the pot, by raising, and hoping the other players will fold. This is essentially a kind of bluff, but one that is hard to pull off in an early or middle position.

2.2.5 Basic Poker Psychology

So far a number of strategic aspects of poker have been mentioned, but another important aspect that sets poker apart from many other games of chance, where luck is often regarded as a primary factor, is the psychology of the game. This is especially true when playing in a live setting, where you are able to look your opponents in the eye, and even speak with them.

When playing in a virtual setting, some of the game's psychological aspects are diminished, and playing the odds of your hand becomes more important. But they are still there, and are mainly manifested through varying strategies, to keep opponents guessing at your playing style, and enabling the player to set up successful bluffs.

Showing and Hiding Hands

The showing and hiding of cards is one psychological aspect of the game, that is not allowed at all virtual tables, but can make a huge impact when available and used correctly. This is a decision of showing your cards, or hiding them, in a situation where you have won the hand, by opponents folding their cards.

By showing cards in this situation, you could for instance display a bluff, to let the opponents know they can never be sure of what you are holding, by just looking at your bets, which could possibly make it easier to set up traps later in the game. In contrast showing a strong hand, could make opponents less inclined to bet against you, making it potentially easier to win hands by bluffs and trying to buy the pot.

In the same regard, hiding your cards, will keep your opponents guessing, making it harder to figure out your playing style, and therefore harder to guess when to play passively and aggressively against you.

Chapter 3

Predictions and Decisions - Bayesian Networks and Pattern Recognition Theory

This section contains the theoretical background knowledge, which is the basis this project. Understanding the concepts that follow is necessary in order to completely understand the contents of this report. Each of the sections below explains the basics of the relevant theoretical knowledge, and discusses its usefulness for this project.

3.1 Basic Probability Theory

Since poker is an environment with imperfect knowledge, YAPA must handle the uncertainties through probability calculations. This section will pin out the basics of probability calculus, and is based on [9] and [1].
Probability calculations are useful when dealing with uncertainties, since an agent will be able to calculate what actions will have the highest probability of enabling the agent to achieve its goal.
A simple example of probability calculations can be drawn from the experiment of throwing a six-sided die. The probability that the die will show 6 is $\frac{1}{6} = 0,1666$, and the probability is the same for 1, 2, 3, 4 and 5. From this experiment we can lay down some terms in probability theory.

- **Sample space:** The set of possible outcomes of an experiment, denoted S. In the die example $S = \{1, 2, 3, 4, 5, 6\}$.
- **Event:** A subset of a sample space. The event of throwing a die showing an even number would be $\{2, 4, 6\} \subseteq \{1, 2, 3, 4, 5, 6\}$.
- **Sum to 1:** The probability of an event $\mathcal{A} \subseteq S$ is denoted $P(\mathcal{A})$ and must be a number between 0 and 1, where 0 is certainly false and 1 is certainly true. The probability of the sample space, $P(S)$, must sum to 1, since it is certainly true that the outcome of the experiment will be in the sample space. E.g. it is certainly true that the outcome of throwing a six-sided die will be either 1, 2, 3, 4, 5 or 6. In formal notation:

$$0 \leq P(\mathcal{A}) \leq 1 \text{ for every } \mathcal{A} \text{ and } \sum_{\mathcal{A} \in S} P(\mathcal{A}) = 1. \tag{3.1}$$

In the die example: $P(1)+P(2)+P(3)+P(4)+P(5)+P(6) = \frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6} = 1$.

- **Unconditional/prior probabilities:** The probabilities that are current in the absence of any other information. E.g. for the experiment of throwing two dice, the prior probability of getting a total of 11 eyes would be $P(Total = 11) = P(5,6) + P(6,5) = \frac{1}{36} + \frac{1}{36} = \frac{1}{18}$.
- **Conditional/posterior probabilities:** The probabilities that are current after granting some information, also known as **evidence**, that influences the experiment. E.g. when throwing two dice, a posterior probability could be that of getting a total of 11 eyes given that the first die is a 5, $P(Total = 11|Die_1 = 5)$. Posterior probabilities can be calculated from prior probabilities:
"*Conditional probability. For two events $\mathcal{A}$ and $\mathcal{B}$, with $P(\mathcal{B}) > 0$, the conditional probability for $\mathcal{A}$ given $\mathcal{B}$ is*

$$P(\mathcal{A}|\mathcal{B}) = \frac{P(\mathcal{A} \cap \mathcal{B})}{P(\mathcal{B})}."[9, page\ 4] \tag{3.2}$$

Using Equation (3.2) we would get the following posterior probability in the two-dice example: $P(Total = 11|Die_1 = 5) = \frac{P(Total = 11 \cap Die_1 = 5)}{P(Die_1 = 5)} = \frac{\frac{1}{36}}{\frac{1}{6}} = \frac{1}{6}$.

3.1.1 Probability Calculus

The rule known as **the fundamental rule** can be deduced by rewriting equation (3.2):

$$P(\mathcal{A}|\mathcal{B})P(\mathcal{B}) = P(\mathcal{A} \cap \mathcal{B}). \tag{3.3}$$

Which can be expressed for $P(\mathcal{B}|\mathcal{A})$ as well, and since $P(\mathcal{A} \cap \mathcal{B}) = P(\mathcal{B} \cap \mathcal{A})$ it is legal to write $P(\mathcal{A}|\mathcal{B})P(\mathcal{B}) = P(\mathcal{B}|\mathcal{A})P(\mathcal{A})$ which yields **Bayes' rule**:

$$P(\mathcal{A}|\mathcal{B}) = \frac{P(\mathcal{B}|\mathcal{A})P(\mathcal{A})}{P(\mathcal{B})}. \tag{3.4}$$

Bayes' rule is used widely for calculating probabilistic inference regarding AI systems. The three terms on the right side of the equation is exactly what is given in many cases. E.g. in medical diagnosis when calculating $P(disease|symptoms)$ – $P(symptoms|disease)$, $P(symptoms)$ and $P(disease)$ are known in many cases. Bayes' rule can also be applied when conditioned on more than one event:

$$P(\mathcal{A}|\mathcal{B},\mathcal{C}) = \frac{P(\mathcal{B}|\mathcal{A},\mathcal{C})P(\mathcal{A}|\mathcal{C})}{P(\mathcal{B}|\mathcal{C})}. \tag{3.5}$$

Independence

Events are independent of each other when evidence of one event does not change the probabilities for the other event.

"*Independence. The events $\mathcal{A}$ and $\mathcal{B}$ are independent if*

$$P(\mathcal{A}|\mathcal{B}) = P(\mathcal{A})".[9, page\ 6] \tag{3.6}$$

Given two events that are independent, Equation (3.6) allows for the fundamental rule to be rewritten into:

$$P(\mathcal{A} \cap \mathcal{B}) = P(\mathcal{A}|\mathcal{B})P(\mathcal{B}) = P(\mathcal{A})P(\mathcal{B}). \tag{3.7}$$

I.e. the probability that both event $\mathcal{A}$ and $\mathcal{B}$ will occur can be calculated by multiplying the individual probabilities $P(\mathcal{A})$ and $P(\mathcal{B})$.
Independence also applies when conditioned on more than one event, this is called **conditional independence**.

"*Conditional independence. The events* $\mathcal{A}$ *and* $\mathcal{B}$ *are conditionally independent given the event* $\mathcal{C}$ *if*

$$P(\mathcal{A}|\mathcal{B} \cap \mathcal{C}) = P(\mathcal{A}|\mathcal{C})".[9, page\ 6] \tag{3.8}$$

Again this entails a multiplication rule:

$$P(\mathcal{A} \cap \mathcal{B}|\mathcal{C}) = P(\mathcal{A}|\mathcal{C})P(\mathcal{B}|\mathcal{C}). \tag{3.9}$$

3.2 Bayesian Network Theory

Bayesian networks are well suited to the type of controlled environment that a poker game is. The attributes of Bayesian networks are discussed in the this section. An optional type of network which could be used would be an Artificial Neural Network (ANN), which is well suited for learning and pattern recognition, but generally adapt slowly. A more thorough listing of the attributes of ANNs can be found in appendix A.2. There is a greater availability of resources for using Bayesian networks, as they are the focus of this semester's machine intelligence course. The greater availability of relevant resources for Bayesian networks coupled with the weaknesses of ANNs in regards to swift adaptations, makes Bayesian networks appear more suitable for this project, and one will be used as the decision model for YAPA. This section will explain the theory regarding Bayesian networks that is needed in order to understand the Bayesian networks in this project. First, the technical terms will be explained, followed by the formal definition of a Bayesian network. Lastly, the functionality of Bayesian networks will be discussed. This section is based on [9].

3.2.1 Technical Terms and Definition

The technical terms regarding Bayesian networks will be explained in this section. Examples are drawn from Figure 3.1 which illustrates the scenario of who is going to win the poker game, assuming that there is a poker agent and one opponent player.

- **Variable:** The set of possible outcomes for any process for which the outcome is uncertain. It is illustrated as a node in the network. In Figure 3.1, "Winner", "Opp_player_hand" and "Agent_hand" are variables.
- **State:** A state is one of the possible outcomes. For the variable "Winner" in Figure 3.1 the states would be "agent_win", "tie" and "agent_lose". The states are mutually exclusive and exhaustive, meaning that the variable can be in only one state at a time.

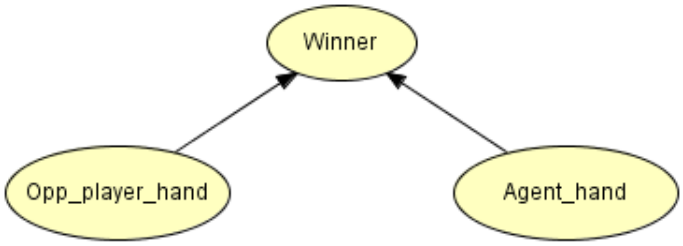


Figure 3.1: Bayesian network example.

- **Dependency:** Dependencies are illustrated with directed edges between the variables. We say that a variable is *conditioned* on its parent. In Figure 3.1 the arrows from "Agent_hand" and "Opp_player_hand" to "Winner" tell that the state of "Winner" depends/is conditioned on the state of "Agent_hand" and the state of "Opp_player_hand".
- **Conditional probability table:** A table that represents the probabilities for a variable given its dependencies. For the variable "Winner", the notation for the conditional probability table would be $P(Winner|Agent\_hand, Opp\_player\_hand)$, meaning "the probability of Winner, given Agent_hand and Opp_hand". An example of a conditional probability table can be seen in Table 3.1. This table is not complete, it serves only to illustrate the idea of a conditional probability table. The "..." represents the remaining states for the valid variable. The states of "Winner" are present at the left side while the states of "Agent_hand" are present at the top. For every conjunction of states from those two variables, the states for "Opp_hand" are present, since "Winner" depends on the two variables "Agent_hand" and "Opp_hand". Probability 1 means 100% chance, and 0 means 0% chance for the given outcome to happen.

	agent_flush			agent_straight			...
agent_win	opp_flush	opp_straight	...	opp_flush	opp_straight	...	...
	0	1	...	0	0	...	
tie	opp_flush	opp_straight	...	opp_flush	opp_straight	...	...
	1	0	...	0	1	...	
agent_lose	opp_flush	opp_straight	...	opp_flush	opp_straight	...	...
	0	0	...	1	0	...	

Table 3.1: $P(Winner|Agent\_hand, Opp\_player\_hand)$.

With the technical terms defined, the definition of a Bayesian network can be seen on Table 3.2.

Definition of a Bayesian network
A Bayesian network consists of the following:

- A set of variables and a set of directed edges between variables.
- Each variable has a finite set of mutually exclusive states.
- The variables together with the directed edges form an directed acyclic graph (traditionally abbreviated DAG); a directed graph is acyclic if there is no directed path $A_1 \rightarrow ... \rightarrow A_n$ so that $A_1 = A_n$.
- To each variable A with parents $B_1, ..., B_n$, a conditional probability table $P(A|B_1, ..., B_n)$ is attached. [9]

Table 3.2: Definition of a Bayesian network.[9, page 33]

3.2.2 Functionality of Bayesian Networks

Bayesian networks are used for calculating new probabilities when receiving evidence on variables. Evidence may influence other variables in the network, in the sense that evidence on one variable may change the probabilities for another variable. Influence may "float" through a network, depending on the connection types for the variables. When a variable initially gets evidence, it is called hard evidence. If a variable gets evidence through influence from another variable, it is called soft evidence. There are three types of connections, through which evidence can flow:

- **Serial connection** A serial connection is illustrated on Figure 3.2. Evidence may "float" from A to B and from B to C or the other way around. If the state of B is known, then the connection is closed and evidence on A cannot influence C (or the other way around). Evidence on B deduces independence between A and C. This is known as d-separation – A and C are d-separated given B. Hard evidence is required in order to close a serial connection.

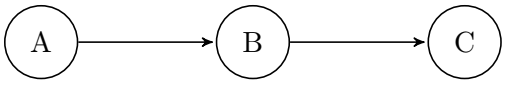


Figure 3.2: Serial connection.

- **Diverging connection** A diverging connection is illustrated on Figure 3.3. The children $B, C, ..., E$ of A can pass influence to each other. If the state of A is known, then the connection is closed and the children can no longer pass influence to each other. $B, C, ..., E$ are d-separated given A. Hard evidence is required in order to close a diverging connection.
- **Converging connection** A Converging connection is illustrated on Figure 3.4. In a converging connection, the parents $B, C, ..., E$ can only influence each other given evidence on A. Both hard and soft evidence may open a converging connection.

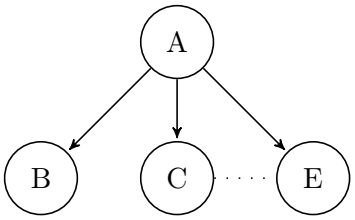


Figure 3.3: Diverging connection.

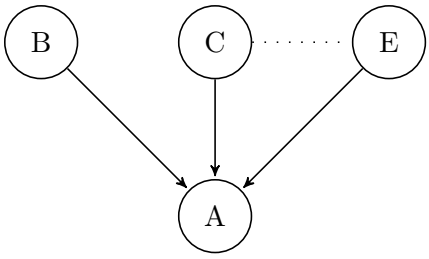


Figure 3.4: Converging connection.

3.3 Decision Theory

Since an agent must make intelligent and rational decisions in a given context, probabilities are not enough, to make a smart agent. When making decision, various circumstances will often give a bias towards certain decisions. In poker, an agent could simply play based on probabilities. However, in order to implement strategic play, and making an agent capable of adapting to various play styles of opponents, as well as a certain degree of unpredictability of the agent for the opponents, it is necessary to add more than just probabilities. Here decision theory comes into play.

Decision theory add utilities to the probabilities[1]. Utilities in this context, should be understood as the usefulness of something. By adding utility nodes to Bayesian networks, which are then called influence diagrams or decision networks, which allow the results to be subjective, or biased, towards certain outcomes. For instance in poker one may have a high probability of getting a pair, and a much lower probability of getting a flush or a straight. But the usefulness, the utility, of a flush or straight is much higher than that of a pair, and therefore if one sees the possibility for the better combination, one might be willing to bet more on that, than just a pair.

So in essence utilities makes certain outcomes preferable over others, and combining these with probabilities, results in decision theory. The goal is to have the agent make rational decisions, by weighting the utility of a certain possibility, with the probability of it happening. In poker, most very high utility outcomes, have a low probability, but by combining the two, one can make the agent ”understand“ that sometimes a longshot might just be worth it, because the possible gain can outweigh the risk.

An example of an utility table can be seen in Table 3.3. It states a high utility for the action of raising, and some utility for checking or calling, when YAPA expects to win the hand(based on probabilities). However it actually states negative utilities for the same actions, if YAPA expects to lose the hand, since throwing money into the pot, when you expect to loose, would just be a waste. In the final model, described in section 4.2.5, several other utilities are used

YAPA Decisions	Expect a winning hand	Expect a losing hand
Fold	0	1
Check/Call	3	-1
Raise	20	-20

Table 3.3: Possible utility table for YAPA's decisions.

as well, and this is just a simple example, from an early YAPA model.

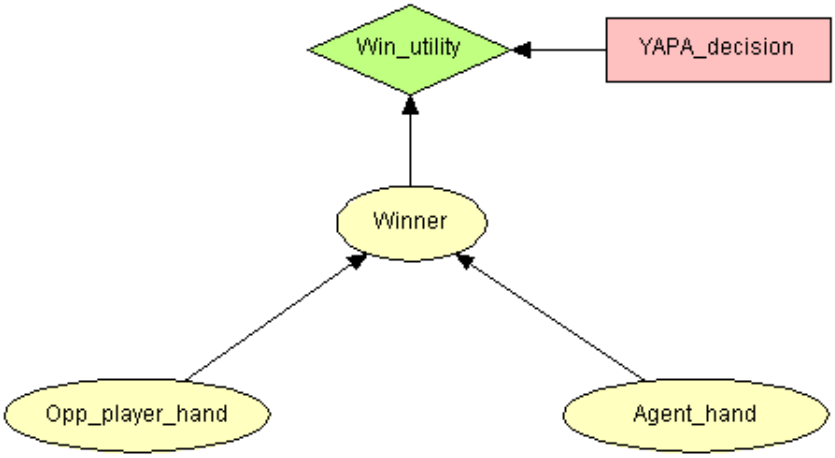


Figure 3.5: Decision Network Example.

In Figure 3.5, an example of a decision network can be seen. The oval nodes are chance nodes(for example "Winner"), and operate like in a normal Bayesian network. The diamond shaped box, "Win_utility", is a utility node, and adds the above described utility to the network. Finally there is the square node, "YAPA_decision", which is a decision node. This node is able to take on different values, for each decision one is considering, and then choose the best decision in any given situation. Further explanation of the nodes and their specific uses in Hugin, can be read about at the Hugin website.[10]

3.4 Data Mining in Agents

Modern society is evolving faster and faster, and we collect more and more information of all kinds: How customers shop in stores, on the internet, how traffic evolves given any number of parameters, how the industry records, examines and update their business processes, and much more. To accomplish these tasks of analysis, data is required, often staggering amounts of data. The reason for this, is of course that to find any of the patterns one seeks through data mining, a case has to be generalisable. An everyday example of data mining that few people think about are social network, like e.g. Facebook. It utilises every piece of information committed by its users, and tries to link it to other data, e.g. finding old friends. It compares data that two users supplied independently of each other and the algorithms Facebook uses try to link these two users together and suggest that they become friends. Another good example is how Amazon for many years have had a function where it looks at a

product a user is looking at or has bought, and suggests other potentially interesting products.

Data mining has a strong relationship with machine learning, it could be said that they are almost dependent on each other. The more one look at data mining, the more one has to look at machine learning. The reason for this is of course, that the amounts of data collected is growing exponentially, thus making it less and less possible to understand all the gathered data. Therefore machine learning helps extrapolating the information we seek, or discover new patterns of the raw data. While data mining needs machine learning to understand data, machine learning also often needs data mining to perform anything usefull. When using machine learning to help with decision making, the more information of the same kind there are at hand, the more accurate your decision process will be, in cases where the purpose is to predict future events from recorded data.

Data mining is concerned with the extraction and analysis of existing data for problem solving. An example could be how businesses that operate in highly competitive markets, keep track of customer loyalty. How would a business make sure that their customers return to them for further sales, rather than buying a competitor's product? Behaviour patterns of former customers can be analysed, to identify distinguishing parameters of those likely to switch products and those likely to remain loyal. Once the relevant parameters are identified, it is possible to find existing customers that are likely to switch to a competitors product, and offer them special deals or terms, that would be to expensive to offer to the entire customer base. It would be possible to tweak this customer model and look at former customers that already left for a competitors products, and find out what it might take to convince them to return. In today's highly competitive, customer-centered, service-oriented economy, data are the raw materials that fuel business growth, if only it can be mined.[11] To this end, data mining is defined as an automated or semi-automated process for discovering meaningful patterns in data, these have to lead to a predefined goal, e.g. an economic advantage or optimisation.

3.4.1 Data Mining Aspects

Data mining can be divided into isolated sections of specialised activity.[12] Relevant in regards to the scope of this project is: The collection of data, the storage of data, the transformation of data into a format that can be processed by a pattern recognition algorithm and detection of patterns in the data. In section 3.5, it is described how to find, describe and understand patterns. The collection, preprocessing and storage of data, so far as it is relevant to this project, is discussed in section 5.3, which concerns the implementation of these aspects in YAPA's data mining capacities.

3.5 Pattern Recognition

Pattern recognition is, in machine learning, the idea of assigning some sort of value or label to a given input, so that it can be either sorted into pre-labelled categories or be used to form categories for future input. The discipline relies on using algorithms to achieve these goals. There is a variety of categories within pattern recognition, each suited to different problems and data availabilities. And each of these have specifically suited algorithms associated with them. Some examples include classification or clustering algorithms. The difference in the operation of these two algorithm types stems from the data they work with, and how they process it. Classification tries to divide the data into predefined classes such as the four poker playing styles, whereas clustering does not have any predefined categories it attempts to fit the data into, but rather orders the input into "clusters" of seemingly congruent data.

These general algorithms are fitted and optimised for specific tasks through training, which is done through the use of one or more learning techniques. Such as supervised learning for classification, and unsupervised learning for clustering.

3.5.1 Pattern Recognition Defined

Pattern recognition is the scientific discipline whose goal is the classification of objects into a number of categories or classes.[13]

Table 3.4: Definition of pattern recognition.

To put it in "layman's terms", this means that any type of operation where the goal is to solve a task, that can either be tedious, time demanding or perhaps a task where humans are limited by our vision. An example could be sorting mail at postal centers, and distributing them out to the correct local post offices for delivery. Another example could be the analysis of radiographic recordings from radio telescopes receiving radio waves from outer space, the number of possible applications are countless.

3.5.2 Learning Techniques

This section is based on [1].

This section contains a short summation of the two learning techniques supervised- and unsupervised learning, as well as the unification of the two, called semi-supervised learning, and how they are relevant to the development of YAPA. As mentioned under the introduction to pattern recognition for the algorithms to fit a specific task the algorithm has to learn how to divide the different inputs into the correct categories. This is done by running the algorithm on training data, making the algorithm generalise a function that describes what the result of a given input should be.

Supervised Learning

This section is based on [1] and [14].

In supervised learning one takes a given training set with only labeled examples; these consist of sets of input data matched with the output data resulting of calling an unknown function on the input data. The goal here is to generate some function that approximates the results of the training set. Then, using another set of data, called a test set, it is evaluated how well the function performs. As a general rule, the simpler an approximated function is, the better it will categorise a data set. Of course, a function for a given training set is not necessarily strictly mathematical, it may also, as would be the case in a poker game, be a stochastic function, where the output is not strictly a function of the input, but rather a conditional probability distribution. Depending on whether the output belongs to one of the two following categories, the learning problem in question is categorised of one of two possibilities: If the output is a member of a finite set, such as the possible poker profiles, the problem is categorised as a classification problem. If the output is a number, for instance matching the temperature of the following day, the problem is categorised as a regression problem. Since the aim of YAPA is to attempt to predict opponent moves based on profiles coupled with current behaviour, classification seems a clear-cut approach.

Unsupervised Learning

This section is based on [1] and [14].

In unsupervised learning, one takes a given training set, but this time without labels. Now the goal of the algorithm is to organise the training set's elements into groups, based on some shared attributes of the elements. Since the performance of the trained algorithm is wholly dependent on the connections it draws during it's iterations over the training set, it is vital that one devises a training set with both a sensible number of elements, and relevant attributes to the elements. The training set elements should be evenly distributed across all possible elements, to avoid skewing the learning, and contain a proportional amount of set elements and element attributes, so that the algorithm does not risk drawing unfounded conclusions on the relation of attributes. There are a number of sub disciplines within unsupervised learning, one of these is clustering, which deals with ordering similar elements into larger conglomerations of elements. There exists a variety of clustering algorithms, called k-means clustering, which sorts the supplied elements into k clusters. This could be used to split recorded hands into four categories, to which the four basic Poker play styles can be matched, in order to enable the identification of play styles for individual players.

Semi-supervised Learning

As the name implies, semi-supervised learning attempts to straddle the fence between supervised- and unsupervised learning. This typically means extending a method associated with one of the paradigms with information or attributes typical of the other.[14] A common semi-supervised algorithm is semi-supervised classification. It is essentially the same as the original classification algorithm, but the training set now consist of partially labelled data; that is both labelled and unlabelled data. The goal for the algorithm is now to approximate a function from both the labelled and unlabelled data. This alleviates the possibly prohibitively restrictive cost in time and resources to label an entire training set. Data on poker games are readily available, both through the game currently being played and databases of saved games. However, unless there is a competent analyst engaged with matching individual player moves to an overall strategy, it is unlikely that much useful information can be obtained through an arbitrary set of data. So a clustering algorithm would probably not be entirely applicable, and it would be prohibitively expensive to label enough data to use a classification algorithm. However, if one could identify just a few key moves, and label the data accordingly, and then let the algorithm sort out the rest, there would be a great deal of resources to be saved.

Similarly, as an example of a semi-supervised algorithm built from an unsupervised algorithm there exists the constrained clustering algorithm which takes a load of unlabelled data, similarly to unsupervised clustering, but now the algorithm will have some manner of constraints, such as must-link or cannot-link. "Must-link" means that two instances of input data must be in the same cluster, and "cannot-link" specifies that instances may not be in the same cluster. Another rule could be the number of clusters allowed. These constrains helps the clustering algorithm obtain a better clustering or more constrained clustering. If one possesses a data set of poker hands, constrained clustering could be used to "mark" certain hands as anchors or limiters for the clusters and it would then be possible to shape the clustering

Summary

There exists a multitude of different paradigms within pattern recognition. The few which have been described above, seem to be most applicable for work with a learning poker agent.

The algorithm that will be used for this project, will be drawn from within these paradigms. The work with choosing and implementing an algorithm, is described in the following section and in the implementation chapter.

3.5.3 K-Nearest Neighbour Algorithm

The k-nearest neighbour (KNN) algorithm is one of the simplest supervised learning algorithms. As the name of the algorithm implies, it attempts to classify a given input vector by comparing it to a number "k" of pre-labelled vectors. Whichever label is most common amongst the k pre-labelled neighbours, is applied to the input vector. The steps of the algorithm, are listed below.

1. Determine parameter K = number of nearest neighbours.
2. Calculate the distance between the query-instance and all the training samples.
3. Sort the distance and determine nearest neighbours based on the K-th minimum distance.
4. Gather the category of the nearest neighbours.
5. Use simple majority of the category of nearest neighbours as the prediction value of the query instance.

The method for choosing the parameter k depends on how the data set looks. If the classification is binary, it is sensible to select an uneven k. Later in the algorithm, this will help avoid tied votes in the majority vote. A high value of k reduces noise in the classification, but may reduce the distinctions between classifications.

Calculating the nearest neighbours requires a way to represent the distance between the vectors in the data set. This can be done with the Euclidean distance, which is a specialisation of the Minkowski distance norm.[1] The formula can be viewed in Equation 3.10

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{3.10}$$

Where p is the query-vector and q is a vector from the training set. And p_1, q_1 through p_n, q_n are the attributes.

As seen in the KNN-algorithm, the Euclidean distance calculation is done for every vector in the training set, so as the training set increases, the algorithm gets computationally heavier, this is usually fixed by modifying or simplifying the algorithm, so that the number of distance evaluations performed are reduced. If the training set contains a class which is more frequently represented than others, the majority vote step may result in equally skewed labelling of the input.

Advantages of KNN

- Robust to noisy data; noisy data is quelled by the majority vote.
- Effective if there is a large training set, since a high number of neighbours yields a better chance that the k nearest neighbours are actually near, resulting in a correct label.
- Simplistic approach, relatively straightforward to implement.

Disadvantages of KNN

- It is needed to first determine the "exactness" of the algorithm, by setting a k.
- There is no implicit way to evaluate which attributes should be considered, nor which type of distance metric should be used.
- Computational costs swiftly become cumbersome, as it is needed to compute the distance of each input to all units in the training set.

Chapter 4

Design of YAPA

This chapter deals with the more theoretical aspects of the poker engine; choice of decision a network and implementation of a Bayesian network in the chosen decision engine.

4.1 Decision Engine

The decision engine is the "framework" through which the Bayesian network is made implementable in code. A number of decision engines were considered for use in this project, a short description of each, follows in the section below.

4.1.1 Considered Decision Engines

Hugin

Hugin is a decision engine developed by the Hugin Expert ApS group, located in Aalborg. Hugin has a framework that enables construction of decision structures; Bayesian networks and influence diagrams. It also has APIs that allows implementation through a number of programming languages. The licensed version of Hugin has the rather steep price of 13.000 DKK. for a single-user academic license.

Netica

Netica is a widely used tool for implementation of decision networks, developed by the Norsys Software Corporation. Like Hugin, it enables the construction of belief networks and influence diagrams. It checks in at a rather more affordable 285 USD.

infer.NET

Infer.net is a Microsoft Research framework for running Bayesian inference in graphical models. The framework provides algorithms needed for machine learning applications.A full license is free of charge.

4.1.2 Selection Parameters for the Decision Engine

Each of the above decision engines will be evaluated on the following points:

Cost The monetary cost of acquiring the decision engine for use.

Use How the decision engine can be implemented in the project.

Adaptation How great an investment of time it will require to adapt the decision engine to the resources of this project, or vice versa.

Hugin

Cost The price of acquiring a Hugin license, even the cheapest available research license is to expensive prohibitive for a project of this size. However, Aalborg University owns a researcher licence, that can be used for the project.

Use Hugin has a GUI, that can be used to design and implement a Bayesian network. This network can then be used in, amongst others, C# through the Hugin API. The Hugin API consists of several DLL's which are added to the project and referenced through the Visual Studio IDE. Tests with supplied examples showed that the process of integration works without problems.

Adaptation Hugin should be easier to adapt to the project; it seems easy enough to use, and the previous project that inspired this project also used Hugin, which should make the understanding and integration to create a better poker agent an easier task.

Netica

Cost Netica would cost 285 US$ dollars for a single educational license. It should be well within the ability of the university to supply these funds, and is therefore not heavily prohibitive. It does, however, put Netica a bit behind both Hugin and Infer.net, which are both effectively costless as far as this project is concerned.

Use It has proven difficult to test Netica, as the setup instructions have not yielded a functional result. Neither have exploratory attempts to make it work. Presumably, it may be that a licensed version is needed for the setup, despite the instructions claiming otherwise. This means that it will not be possible to test Netica beforehand. The examples supplied by the developer indicates a functionality similar to Hugin; Bayesian networks can be created with a GUI, and they can then be accessed through a relevant C# API. However, this cannot be guaranteed without testing.

Adaptation Again, it is difficult to ascertain this as long as a proper test of Netica cannot be carried out. However, as there does not exist a tool to convert Hugin files to Netica files, there would be some extra work involved in using Netica, since the older project that YAPA is based on uses Hugin.
Netica floats in a somewhat indeterminate state of usefulness, as it has proven hard to test.

Infer.net

Cost The full license carries no cost, it is free. Strong point, on par with Hugin.

Use Being a Microsoft product, Infer.net is integrates directly into Visual Studio. This also means that there is no GUI for constructing the Bayesian Networks, it can only be written directly in code. Furthermore, Infer.net is still in Beta, and some unexpected errors can probably be expected to occur.

Adaptation This is a sore spot. Not only will writing complex networks in code, would be an arduous process, translating existing networks, constructed in Hugin, is likely to be a time-consuming affair. Even more so than duplicating them using a GUI, as would be necessary with Netica.
Infer.net would be a much stronger candidate if it was not for the fact that a Hugin license was already available and that the project is to be based on existing work.

4.1.3 Summarisation and Selection of Decision Engine

A summarisation of the impression of each considered decision engine follows below:

- All in all, Hugin seems to be the stronger candidate; it is easily adaptable to the other resources of the project, and it's prime disadvantage, the cost is negated.
- Netica is not a strong candidate at this time; it is unlikely to be as adaptable to existing resources, as the API is different, and it would also carry a monetary cost. Furthermore, it has been impossible to test Netica.
- Infer.net seems to be a decent candidate, but it's beta status coupled with lack of a GUI places it behind Hugin.

The evaluation of the different decision engines has resulted in Hugin appearing as the most applicable decision engine for this project, hence this project will use Hugin to develop the Bayesian network.

4.2 The Bayesian Networks

This section includes the iterations of Bayesian decision networks, where we calculate what actions YAPA should take, based on probability tables. In each of the following subsections, a network and a description will be shown and written, what the idea behind each network is, what whet well and why we iterated further, finally there will be a short talk on what features the network could be expanded with, and how it was to build the network with the chosen Hugin framework.

The used nodes in the Bayesian networks, is being explained in Section 3.3 on page 18.

4.2.1 Network Iteration 1

The initial plan for the network, was to represent every card-combination possible in Texas Hold'em Poker. This choice proved to cause an exponential increase in the number of derived states for each variable; a variable depending on other variables with a multitude of states, will have a state for every combinatorial possibility. In an effort to reduce the exponential impact, some variables were removed, which would mean shorter dependency chains. To be exact, this meant the merging of all the "hole card" variables into one single variable. Furthermore, it turned out that the Hugin GUI was not able to handle networks of this size.

While the first network was nor able to run, it showed us that looking at all the possible hand combinations in poker, is a combinatorial problem, as there in Texas Hold'em Poker are $52 \times 51 \times 50 \times 49 \times 48 \times 47 \times 46 = 674.274.182.400$ hand combinations. The solution to this problem will in the next iteration of the network, be to combine some of the variables that make up the network, in order to reduce the potential exponential states that are being

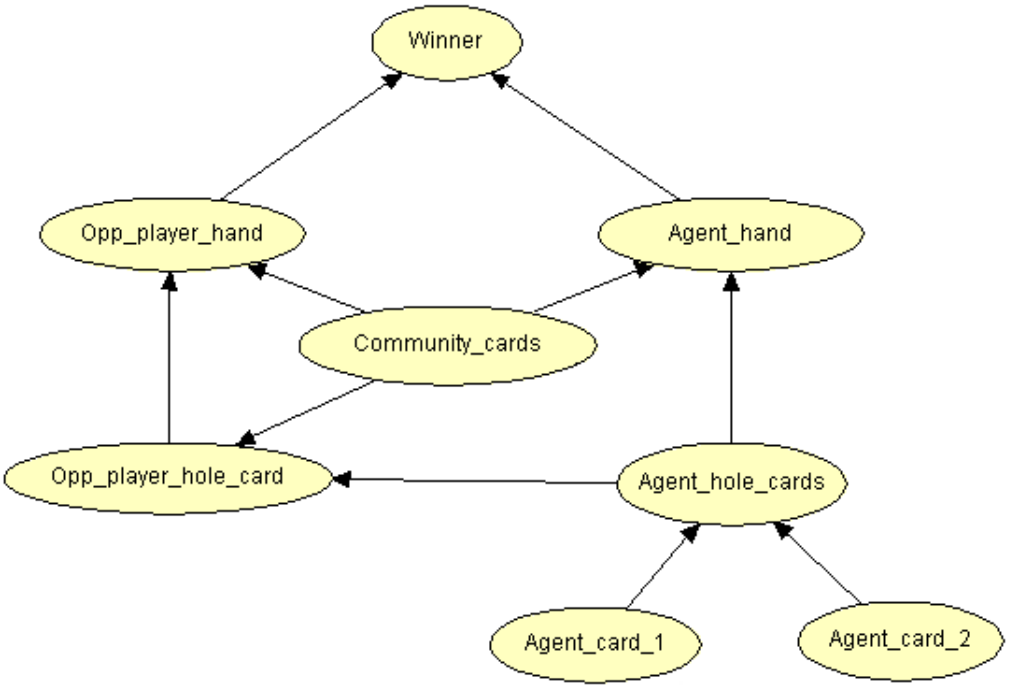


Figure 4.1: Network Iteration 1.

derived.

4.2.2 Network Iteration 2

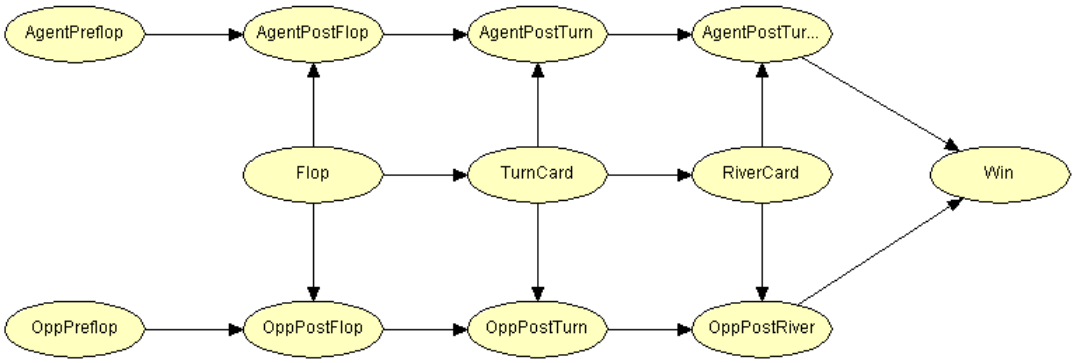


Figure 4.2: Network Iteration 2.

It should be obvious that the second network is very different from the first network, instead of inserting values for each of the 52 cards in the deck, all the values in the network are predefined combinations of cards. The "AgentPreflop" and "OppPreflop" nodes, contains the values "HighCard" and Pair. The states "Flop", "TurnCard" and "RiverCard", all contain the possible valid combinations that are specific to the nodes place in a game, combinations with three, four and five cards. As the game progresses, the states of the community cards are combined with the preflop cards, for the agent we know what we have, and for the opponent,

we can calculate the chances of what possible combinations the opponent can have, and find out after the river card has been shown, who has the better chance of winning. This network as the previous, does not have any utility or decision nodes that can help YAPA decide what actions to take at each stage of the game.

Like network iteration 1 did, this model generates too many states. There are too many derived states which makes the final state space in "AgentPostRiver" and "OppPostRiver" exponentially large. What we learned from this network, that could use for the next iteration. Is that predefined combinations, together with fewer states which receive derived states, is a good possible final solution.

4.2.3 Network Iteration 3

With the knowledge gained from the two previous networks, it is clear that more use of predefined combinatorics is a way of limiting the size of the networks that needs to be created and filled with data. With reference to Section 3.1 on page 13, the following will look at combinatorics as they are being used in the networks.

4.2.4 Combinatorics in Poker

Combinatorics is a mathematical discipline, that concerns itself with the study of finite and countable discrete structures. For the purpose of YAPA, it is applicable to how many ways it is possible to combine different cards to give accepted values in Texas Hold'em Poker. It is possible to calculate the probabilities of any possible event in poker. There are 52 cards in a deck used for Texas Hold'em Poker, this means that for a two player game, player one, who is not the dealer, is dealt the first card, the dealer the second, player one the third and the dealer the fourth card. If each card in the deck is said to be different from any other card in the deck, e.g. the two of spades is different from the two of hearts, then the number of possible starting combinations would be 52*50 for player one in a two man game, and 51*49 for the dealer. No suit is stronger than another suit, therefore a pair of two's made up from the two of spades and hearts would have the same value as the pair of twos made up from two of clubs and diamonds. This would mean that there are only 169 distinct holecard combinations in the deck (13×13), where 13 of these would be pairs, rather than 2.600 distinct combinations calculated by 52*50. YAPA uses the assumption presented above, where there are only 169 distinct holecard combinations, and maintains this assumption throughout the game.

The third network iteration uses the ssumption explained above. The node, "CommunityCards" contains the states of each of the possible hand combinations of Texas Hold'em Poker, from high card to straight flush. "CardPos" contains four states, "Preflop", "Flop", "Turn" and "River", these four are as with the "CommunityCardsť" node, derived down to the "Opponent" node, where they together with its states, forming the backbone where calculations are being inserted. The "Agent" node contains the same states as the "Opponent" node, where information about the best possible current hand YAPA can have, is inserted as evidence. The "Win" node is used to show what the chance is for YAPA to win. The Utility node "WinT" is used together with the decision node "Yapa_decision", to find out what YAPA's next action should be.

This network is using the approach described earlier, using predefined combinatorics instead of derived combinatorics. This gives a much lighter network, that can express values for decision making, that are different from the full set of all possible combinations of cards, but

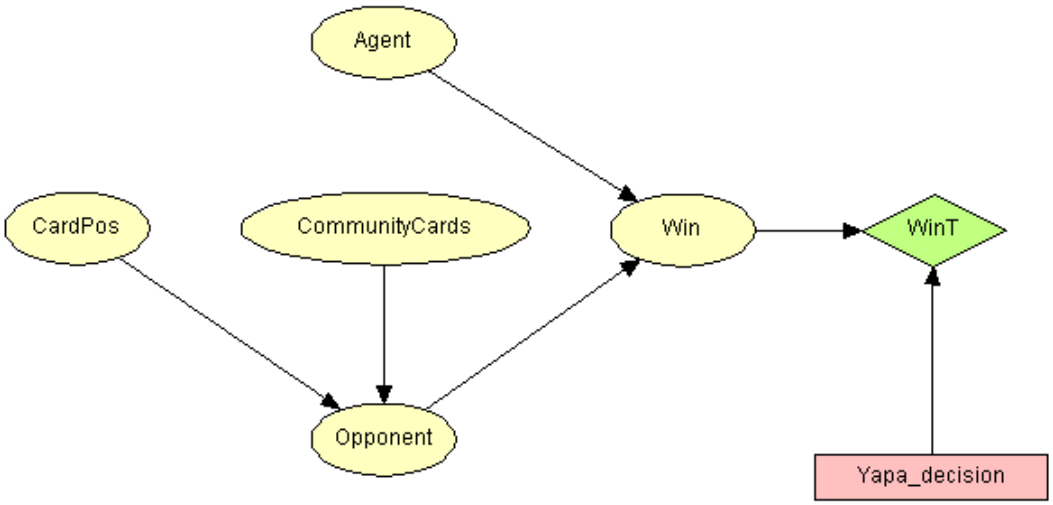


Figure 4.3: YAPA with full use of combinatorics.

still contain the essence of the numbers. This means that even though there are a lot less

For the next version of the network, everything from this third iteration can be used. There is just a need for more states and more utility and decision tools, to help YAPA do its task with a higher accuracy.

4.2.5 Final Network

This final Bayesian network uses the initial knowledge learned from network-iteration-3, based on the assumptions described in section 4.2.4. It has twice the number of nodes, compared to the so far best working model, Figure 4.3. The fourteen nodes in this final network, counts one decision node, four utility nodes and nine discrete chance nodes. Figure 4.4 on the facing page displays the full network of the working YAPA. For this final network, every node will be explained and the most interesting nodes will be further examined. The nodes that are being examined, will be accompanied with a screen dump of either the entire node, or a singled out part of it, in case it's state space is too large. A screen dump of all nodes can be found from page A.2 on page 52 in the Appendix.
States of the final network explained:

The "CardPos" node is the only node that has the same states, as the same named node in the last network iteration, it still has the four states, preflop, flop, turn and river. Node can be seen in the Appendix at page A.1 on page 52

The "CommunityCards" node, contains all the same states as the previous network, but also many new ones. These additions to the network, have been made to make it more precise, calculating the winning chances in some cases.

Where the previous networks "CommunityCards" node, only had the states of the regular combinations of Texas Hold'em Poker, Figure 4.5 on page 32 shows that there are several states for each acceptable hand combination. E.g. the state "3K" is short for the community card value three of a kind. The two other version of the "3K" state, has been implemented

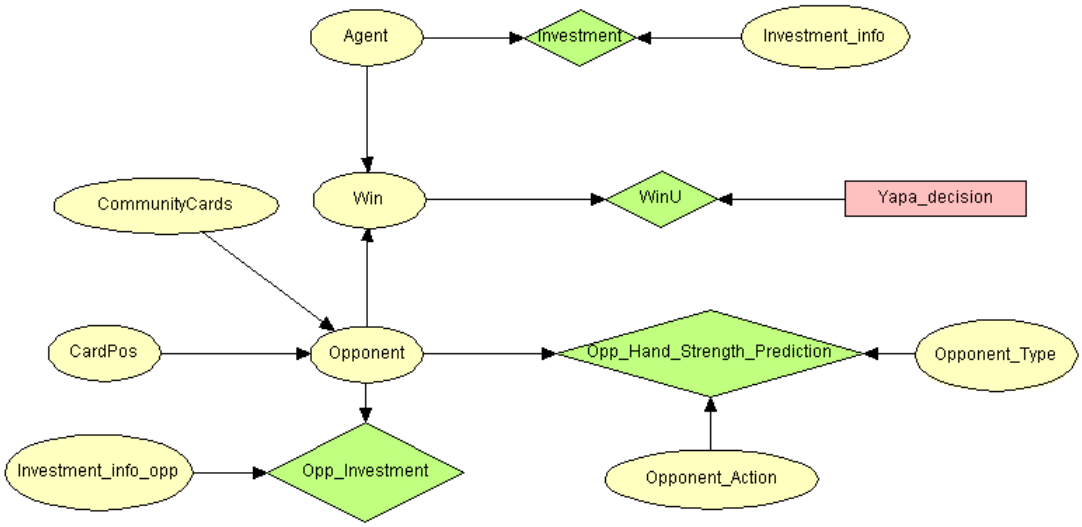


Figure 4.4: The final YAPA network.

into the node, to help calculate the odds for getting either a flush or a straight, where "3K_3" is a state that says we have three of a kind, but also three of same suit in the community cards, a situation that can arise in the "CardPos" state "River", when there are five cards in the community cards stack. The same can be said about "3K_3con", it means there is three of a kind, but also three consecutive cards in the community card stack.

Figure 4.6 on the following page shows a small cutout from the "Opponent" node, where the states 'river' from "CardPos" and '3K' from the "CommunityCards" node are joined in with the states of the "Opponent" node itself. The full "Opponent" node can be seen in the Appendix, in four figures, starting at page A.4 on page 54.

Node "Investment_info_opp" has the states "High" and "Low", that derives its states down to the "Opp_Investment" utility node, The usage of this node is explained in the text hereunder. This node can be seen in the Appendix at page A.3 on page 53.

The "Opp_Investment" utility node, derives states from the "Investment_info_opp" and "Opponent" nodes; the utility node helps YAPA decide, what the opponents betting strategy actually signifies. Figure 4.7 on page 33 shows us a compact version of the derived states in "Opp_investment". This utility node looks at the opponent's chip investment in the current hand. If the investment is low, then YAPA ignores it. Though if the investment is high, YAPA will look at its other believes. If YAPA already thinks the opponent has a strong hand, YAPA will amplify its belief. Therefor; if YAPA's hand, is not very strong, it will be more likely to fold. If YAPA has a strong hand, it will call. If YAPA believes that the opponent has a weak hand, there is a likelihood that YAPA will raise to call the opponents possible bluff. It is the opponents starting chip stack that determines weather YAPA will look at any bet as a high or low investment. YAPA will remember how many chips the opponent had at the beginning of a hand, if the opponent , bets more than twenty percent of the chips over the entire length of the hand, then the investment utility will be set as high.

The "Win" node is simply a node that determines if one hand can beat another hand, e.g. a

CommunityCards

Straight_Flush	0.041667
4K	0.041667
Full_House	0.041667
Flush	0.041667
Straight	0.041667
Straight_4	0.041667
Straight_3	0.041667
3K	0.041667
3K_3	0.041667
3K_3con	0.041667
Pair_Two	0.041667
Pair_Two_3	0.041667
Pair_Two_3...	0.041667
Pair	0.041667
Pair_4con	0.041667
Pair_3con	0.041667
Pair_4	0.041667
Pair_3	0.041667
High	0.041667
High_4con	0.041667
High_3con	0.041667
High_4	0.041667
High_3	0.041667
NoCards	0.041667

Figure 4.5: YAPA CommunityCards Node.

				River	
3K	3K_3	3K_3con	Pair_Two	Pair_T...	Pair_
0	0	0	0	0	0
0.001009	0.00101	0.00101	0.001932	0.001932	0.00
0.269425	0.269697	0.269697	0.182828	0.182828	0.18
0	0.045454	0	0	0.045454	0
0	0	0.032323	0	0	0.03
0.729566	0.683838	0.69697	0	0	0
0	0	0	0.81524	0.769785	0.78
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Figure 4.6: YAPA Opponent River.

flush compared to three of a kind. The table can be found in the Appendix at page A.5 on page 54.

The "Agent" node contains the nine hand combinations. Is used to insert evidence into, depending on what YAPA knows it has at any given stage of the game. The node is displayed in the Appendix at page A.6 on page 55.

Figure 4.8 on the facing page displays how YAPA looks at its own investment using the "Investment" node. If there has been a low investment, again under one fifth of the starting chips per hand, then YAPA won't care about its own investment, when determining which

Investment...	High									
Opponent	Straight_Flush	4K	Full_House	Flush	Straight	3K	Pair_Two	Pair	High	None
Utility	-15	-15	-10	-5	-5	-1	2	5	10	0
Investment...	Low									
Opponent	Straight_Flush	4K	Full_House	Flush	Straight	3K	Pair_Two	Pair	High	None
Utility	0	0	0	0	0	0	0	0	0	0

Figure 4.7: Opp_Investment Cutout

Investment																		
Investment...	low									high								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Utility	0	0	0	0	0	0	0	0	0	20	18	15	10	5	4	2	0	0

Figure 4.8: Investment node

action to take. If YAPA has more than one fifth of its chips invested into the pot, then YAPA will be more likely to stay in the game and play it out.

The "Investment_info" node, has the same properties as the "Investment_info_opp node", it is displayed in the Appendix A.14 on page 57.

Opponent_Type	
LA	0.2
LP	0.2
TA	0.2
TP	0.2
Unknown	0.2

Figure 4.9: Opponent_Type

The "Opponent_Type" node, is where evidence is insert, into the network, with our beliefs about what type of player the opponent is, Figure 4.9 shows the different possibilities. The acronyms of the node are explained here:

LA Loose-Aggressive, this players type buys into most games and raises most hands.

LP Loose-Passive, this player type buys into most hands but won't raise much

TA Tight-Aggressive, this player type only buys into a hand when he or she believes that the cards they have are a favorite to be the winning hand. Once they buy in they often raise the bets.

TP Tight-Passive, this player type only buys into a hand when he or she believes that the cards they have are a favorite to be the winning hand. When they buy into the hand, they will seldom raise the bet.

Unknown Unknown, this is the default player type assigned to all opponents until they have been classified, furthermore, if YAPA can't find a suiting profile, it will continue to keep or even move players back to this profile.

As the above listing shows, its is important for YAPA to classify its opponents. Since it will make it easier for YAPA to play against them and predict what they will do and how good their cards are.

The Opponent_Action node, lists the last action taken by the opponent. The node is displayed in the Appendix A.12 on page 56.

WinU

Win	Yes			No		
Yapa_decision	Fold	Check/Call	Bet/raise	Fold	Check/Call	Bet/raise
Utility	0	3	40	1	-1	-20

Figure 4.10: WinU

Figure 4.10 shows how YAPA will behave when it believes that it either has a winning or losing hand. This utility node, together with the other utility nodes in the network, is what YAPA really uses to make a decision on what action to take next.

The decision node "Yapa_decision", contains three states, Fold, Check/Call and Bet/Raise. These are the actions YAPA may take in any given situation. The decision node is displayed in the Appendix A.10 on page 55

Opp_Hand_Strength_Prediction

Opponent	Full_House														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Figure 4.11: Opp_Hand_Strength_Prediction Cutout

Figure 4.11 is a cutout from the "Opp_Hand_Strength_Prediction" node. It displays how YAPA uses its knowledge of the opponent player type, the opponents current action and the predicted possible hand combination available. In the current example, YAPA knows that the opponents predicted hand is a full house, furthermore the opponent will be classified as a Tight-Aggressive player type. If the opponent raises a bet, YAPA will see it as a higher chance for the opponent to actually have the Full House. Depending on what the other utilities are in the network, and YAPA's own hand, YAPA will make a decision to either Fold, Call or Raise the opponents Bet.

Chapter 5

Implementation - The Creation of YAPA

This chapter will describe the implementation of YAPA. This includes a description of the decision model and its underlying elements of data mining and pattern recognition. The agent itself is implemented in an previous poker project which was created for use of the Caroline poker agent [15], which also used Hugin with Bayesian networks for its decision making. This previous project includes a complete game engine, a gaming interface and the Hugin Application Programming Interface (API) with a tree-structure implemented.

5.1 Decision Making

The YAPA agent will, when playing poker, always have to decide between the three basic actions of raise, fold or call. The main element it uses in the decision of which one to pick, is its decision network. The decisions are determined by having a decision node in the network, connected to a utility node, and then simply prompting for which decision would result in the highest utility for the network. It should also be mentioned that the model is designed for use only against a single player. If playing against several opponents, it will react to the last opponents actions.

A description of a typical game could look like this:

1. Hand start:
    - Initialise the Bayesian network.
    - Calculate strength of current hand.
    - Check if opponent has been seen before, and if so, try to classify him.
    - Insert initial evidence into the network.
2. Pre-Flop, Flop, Turn, River:
    - Calculate own hand and community card combinations.
    - Insert evidence of card combinations and opponent action.
    - Search for best decision in Bayesian network and perform an action accordingly.
    - Record all players actions.
3. Hand end:

- Record hand winners and save hand for later player analysis.

Before an action can be decided upon, the agent needs to be implemented properly in the game engine so it can communicate. This is done by extending a class with the "`Player`" class of the engine, which subscribes to the events the engine sends out and contains event handles which are simply overriden by YAPAs own. Basicly YAPA watches the events and either reacts or record where necessary.
While working, the agent constantly inserts evidence in the Bayesian network, which is used when the game engine requires an action from the agent. This evidence gathering mainly concerns:

- Finding best hand.
- Finding what card combination the community cards are.
- Finding the opponent player type.
- Recording opponent and player raises.
- Seeing opponent actions.

This information gathering, and how it is analysed, will be further described in the following sections.

When a game is started, the Bayesian network is loaded from the hugin file and converted into a tree structure of objects. When inserting evidence, a dictionary containing the node names as keys and the nodes of the network as values, is queried for the correct node, and when retrieved, the state of that node is changed or set to an indicated statename. A boolean is also set to true, to indicate that the network should be propagated next time a decision utility is requested.

When the game engine requires a decision from YAPA, YAPA will start evaluating which decision is the best choice, by simply getting the utilities for the decision node "Yapa_decision", and then picking the decision which has the highest of the utilities.
The three actions it will be choosing between are the usual "Fold", "Check/Call" and "Bet/Raise". In case of "Fold", it will obviously result in the action fold, unless YAPA is able to check (i.e. it should not fold if it is free to continue playing at the moment). "Check/Call" is even simpler, it just calls. "Bet/Raise" on the other hand, is a bit more subtle. First of all, since YAPA should be able to bet an amount which is dependent upon its current situation, the raised amount is calculated by taking the amount of chips on hand, divided by a number equal to ten times the expected utility of the decision, see Listing 5.1.

```
1  switch (model.GetBestDecision("Yapa_decision"))
2  {
3    case "Fold":
4      if (info.ToCall == 0)
5        action = new PokerAction(PokerAction.PokerActionType.Call, null);
6      else
7        action = new PokerAction(PokerAction.PokerActionType.Fold, null);
8      break;
9    case "Check/Call":
10     action = new PokerAction(PokerAction.PokerActionType.Call, info.ToCall);
11     break;
12   case "Bet/raise":
13     int raise = Chips / (10 * Convert.ToInt16(model.
           GetExpectedUtilityForDecision("Yapa_decision", "Bet/raise")));
```

```
14      if (raise < info.MinRaise)
15        raise = info.MinRaise;
16      else if (Chips == 0 || Chips < info.MinRaise)
17      {
18        action = new PokerAction(PokerAction.PokerActionType.Call, Chips);
19        break;
20      }
21      action = new PokerAction(PokerAction.PokerActionType.Raise, raise);
22
23      //do not raise in an endless loop
24      if (maxRaise >= rand.Next(1, 4))
25        action = new PokerAction(PokerAction.PokerActionType.Call, null);
26      maxRaise++;
27      break;
28    default:
29      action = new PokerAction(PokerAction.PokerActionType.Fold, null);
30      break;
31  }
```

Listing 5.1: YAPA decision formula.

There were however, a problem with YAPA when both it and its opponent kept raising, they could potentially raise until they both ran out of chips. This has been fixed by setting a random number of turns between 1 and 4 after which YAPA will call instead of raising.

5.2 Classification and Learning using K-Nearest Neighbour

This section will describe how the KNN algorithm is implemented for YAPA. Recall from Section 3.5.3 that the KNN classification algorithm selects the "k" closest vectors from a training set and classifies a vector by choosing the category which is in majority, from the "k" selected training data. The vectors are used to symbolise a poker player's playing style through a hand, and have the following parameters, which are always calculated as a number from 0 and 10 (see Listing 5.2 for an example):

- `handStrength` - The strength of the player's hand (e.g. a straight flush would give a hand strength of 9, and a high card would result in a 1).
- `raisePercentage` - How often the player has raised in a hand (e.g. a hand where 3 out of 10 actions were first raises would result in 3, and another with 1 out of 4 would result in 2).
- `reraisePercentage` - How often the player has raised two or more times in a hand (e.g. a hand where a player raised 4 times in a row and then folded, would result in 6 (3 out of 5 actions reraise)).
- `raiseAmountPercentage` - How much of his currency the player has betted in a hand (e.g. a player starts a hand with 400 chips and throughout the hand throws 300 into the pot through calls and raises, would result in 7).
- `continuedFlop` - Did the player continue into the flop, or fold before? (e.g. a player folding before the flop results in a 0 and a player continuing results in a 10).

```
1 //(handStrength, raisePercentage, reraisePercentage, raiseAmountPercentage,
      continuedFlop)
2 KNN_Vector(9, 3, 6, 7, 10, KNN_Vector.Classification.LP)
```

Listing 5.2: Example af a vector in KNN.

There are five different categories; Tight-Passive (`TP`), Tight-Aggressive (`TA`), Loose-Aggressive (`LA`), Loose-Passive (`LP`) and `Unknown`.
When attempting to classify a player, the first thing that happens is that all hands the player has played will be analysed and a list of vectors, as above, will be created from the data. These vectors will then, one by one be analysed and classified with the KNN algorithm (see Listing 5.3), at the moment using a "k" value (the number of closest vectors looked at) to find the majority category. In case of a tie, the `Unknown` category is used.
After each vector has been classified, the classification of the player will start. The player's type is classified by finding the category of the analysed vectors associated with him, this time only looking at the latest "k", which are in majority. The `Unknown` category is used when either there is none of the other four categories which are in majority or when there is no data recorded about a specific player yet.

```
1  public KNN_Vector.Classification ClassifyVector(KNN_Vector sample)
2  {
3      //Calculate euclidian distance
4      foreach (KNN_Vector trainer in trainingSet)
5      {
6        trainer.distance = Math.Sqrt(
7            Math.Pow((trainer.handStrength - sample.handStrength), 2) +
8            Math.Pow((trainer.raiseAmountPercentage - sample.raiseAmountPercentage)
                 , 2) +
9            Math.Pow((trainer.raisePercentage - sample.raisePercentage), 2) +
10           Math.Pow((trainer.reraisePercentage - sample.reraisePercentage), 2) +
11           Math.Pow((trainer.continuedFlop - sample.continuedFlop), 2));
12     }
13
14     //Count categories for the k nearest training vectors
15     List<KNN_Vector> clonedSet = trainingSet;
16     int TA = 0; int TP = 0; int LA = 0; int LP = 0;
17     clonedSet.Sort(delegate(KNN_Vector v, KNN_Vector v2) { return v.distance.
           CompareTo(v2.distance); });
18         for (int i = 0; i < k; i++)
19         {
20             switch (clonedSet[i].playerType)
21                 {
22                     case KNN_Vector.Classification.LA:
23                         LA++;
24                         break;
25                     case KNN_Vector.Classification.LP:
26                         LP++;
27                         break;
28                     case KNN_Vector.Classification.TA:
29                         TA++;
30                         break;
31                     case KNN_Vector.Classification.TP:
32                         TP++;
33                        break;
34                 }
35         }
36
37      //Find majority and classify
38      if (TA > TP && TA > LA && TA > LP)
39        sample.playerType = KNN_Vector.Classification.TA;
40      else if (TP > TA && TP > LA && TP > LP)
41        sample.playerType = KNN_Vector.Classification.TP;
```

```
42      else if (LA > TP && LA > TA && LA > LP)
43        sample.playerType = KNN_Vector.Classification.LA;
44      else if (LP > TP && LP > LA && LP > TA)
45        sample.playerType = KNN_Vector.Classification.LP;
46      else
47        sample.playerType = KNN_Vector.Classification.Unknown;
48
49       return sample.playerType;
50  }
```

Listing 5.3: KNN Implementation.

5.3 Data Mining

To be able to perform the classification described in Section 3.5.3, data regarding previously played hands is needed. Because of this, YAPA records every action for every opponent it plays against. All the hand data is saved in an Extensible Markup Language (XML) file. An example of the data stored for a hand is shown in Listing 5.4.

```
1  <Hand>
2    <GameInfo>
3      <Player Name="YAPA" SeatNumber="0" StartingChipStack="2990" Winner="no"
           Cards="3c 9s " />
4      <Player Name="Foo" SeatNumber="1" StartingChipStack="2980" Winner="yes"
           Cards="7h 3s " />
5      <Action Type="SmallBlind" Player="YAPA" Amount="10" />
6      <Action Type="BigBlind" Player="Foo" Amount="20" />
7    </GameInfo>
8    <PreFlop>
9      <Action Type="Call" Player="YAPA" Amount="20" />
10     <Action Type="Call" Player="Foo" Amount="" />
11   </PreFlop>
12   <Flop Cards="As 6c 4d ">
13     <Action Type="Call" Player="Foo" Amount="" />
14     <Action Type="Call" Player="YAPA" Amount="" />
15   </Flop>
16   <Turn Cards="5h ">
17     <Action Type="Call" Player="Foo" Amount="" />
18     <Action Type="Call" Player="YAPA" Amount="" />
19   </Turn>
20   <River Cards="9h ">
21     <Action Type="Call" Player="Foo" Amount="" />
22     <Action Type="Call" Player="YAPA" Amount="" />
23   </River>
24 </Hand>
```

Listing 5.4: Format of saved hand data.

Each round, methods in YAPA are called according to the current stage within the hand, and the actions YAPA and the opponent take. This information is temporarily saved until the end of the hand. At this point the gathered information is appended to the existing XML file.

With all the previously played hands saved, it is possible for YAPA to learn how a single opponent plays. This is done by extracting data for a single player from the XML file. For each hand the strength of the player's hand, the percentage of times the player raises and reraises, the percentage of the player's money used to raise and how much money the player raised, is calculated. It is also registered wether the player bought into the flop, how much money the player had at the start of the hand and which cards the player had. All this

information are then used to create a vector describing the hand. Every vector created are added to a hashtable for the specific player.

When doing the classification, the vectors in the hashtable are compared to vectors in a training set, which contains vectors describing the different classes. After the comparisons, all the vectors for the player are examined and the player's class is determined by the KNN algorithm.

Chapter 6

Testing YAPA

This section contains an evaluation of YAPA's poker play. YAPA is tested against other agents and a number of human players in an attempt to evaluate the strengths, weaknesses and flaws of YAPA. YAPA will be tested against each of the agents below in 100 full games (playing until only one player has any chips), but the human number of tests will be lower, since human opponents take longer to finish a full game. Therefore, the number of human games played will change according to what is being tested. At the very least 20 tests full games will be conducted.

6.1 Test 1

YAPA will be tested with the direct approach of running a number of poker games with YAPA as one of the players. These games are all one-on-one games, called "Heads up" in poker, as YAPA is not able to consider the altered probabilities of multiple players. The resulting statistics and remarks on the playing tendencies of YAPA are contained in the sections below.

6.1.1 YAPA Versus Software

Caroline

The data in this section stems from games which had YAPA and the poker agent "Caroline" version 3b, developed by Computer Science group d3003a in the autumn 2009 [15]. A total of 100 games were played. At the end, YAPA had won 73 of these games.
Caroline plays by considering the opponents last action, and otherwise play directly based on the probabilities of the cards. When it raises, the amount is set to a random number within it's remaining funds. The tendency seems to be that YAPA is capable of outplaying Caroline to a lesser degree; It is definitely ahead, but it still makes mistakes. Reviewing the game logs, it is apparent that YAPA more often than not correctly identifies that Caroline has a strong hand when it bets. Since Caroline mainly plays based on probabilities, it is fairly uniform in its behaviour and therefore easy to predict.

Randomiser

The data in this section stems from games which had YAPA and a randomly behaving poker agent. The randomised agent simply picks a number from 0 to 2 each turn, which decides which action it should take. If the raise action is chosen, it uses yet another random variable to decide upon the amount. Against the randomiser, YAPA achieved a winning rate of 64 percent. Considering the unpredictable nature of this opponent, it is not unsurprising that YAPA performed worse than against Caroline. This is an improvement over Caroline's 58

percent win rate against the randomiser.[15]. The improvement is probably caused in part by YAPA being a more aggresive agent, and that YAPA is more concerned with opponent behaviour, and not just probabilities.

Caller

Finally, YAPA was tested against an agent which was programmed to always call its opponent, thereby flaunting odds and analysis in favour of blind luck. Against this variety of AI, YAPA achieved an 81 percent win rate.

6.1.2 YAPA Versus Humans

This section contains data obtained from games YAPA played against human opponents, these are further divided into games against the developers and games against non-affiliated persons.

Developers

When performing this internal test, YAPA obtained a 52 percent win rate. This was significantly worse than its performance against the other poker agents, but also better than its performance against test persons who had never played against YAPA before. The result of this test indicates that YAPA is not yet sufficiently advanced to correctly assess a human opponent. And that the testing developers either overthink their actions when playing against YAPA, or are generally low-quality poker players. 26 tests were conducted in all.

Non-Developers

This was the weakest area for YAPA, it achieved a 40.9 percent win rate. However, this was unevenly distributed against the people who played. There was a tendency for YAPA to steer closer to 50 percent win rates against opponents who were not particularly skilled at poker, or had a tentative grasp of the rules. Against expert players, YAPA remained uniformly incapable of winning. The greatest revealed issue, was that players who played tight remained unbeatable. One player was also able to figure out a lacking obfuscation in YAPA; if he called twice, without YAPA following up with a raise, it had poor cards. 32 tests were made.

6.1.3 Evaluation of Test 1

YAPA outperforms Caroline and the Caller, both in direct confrontation and against a randomiser. It does not perform nearly as well against human players, just barely breaking even on its best round of testing. One definite flaw in YAPA was revealed; due to problems with YAPA's feature extraction and skews in the training set, YAPA classified every single opponent as loose passive. This means that YAPA played solely according to the probabilities of the cards and the actions of the opponent, rather than properly enterpreting this through the prism of the opponent's playstyle. This lead to increased losses, mainly against the better human players, who swiftly adjusted their own game accordingly. The statistics for this test can be seen in Table 6.1. For the next test, the errors in the feature extraction of the classifier will be corrected, which will enable correct classification. The training set has been improved and enlarged, which should further improve classification.

YAPA		
AI	Caroline	73%
	Caller	81%
	Random	64%
Humans	Developers	52%
	Non-developers	40.9%

Table 6.1: Test statistics for YAPA test 1.

Error Sources

- The value for k in YAPA's KNN algorithm might be making KNN use data from too far back, meaning it gets locked into its beliefs about playstyle for too long, or it might not use enough data, meaning it changes its beliefs too often or with too low accuracy.
- Utility nodes in the Bayesian Network could be incorrectly calibrated.
- Only a few hundred games were played, so results may be skewed due the randomness of the cards.
- Since there was no actual risk involved with losing the game, the human players may not have played naturally.

6.2 Test 2

The purpose of this test is to verify whether the improvements made to the classification aspect of YAPA, has lead to improved play. It will be performed similarly to test 1, with YAPA being pitted against a number of artificial and human opponents. Due to there a lesser number of testers available for this test, both developers and non-developers will be defined simply as humans

6.2.1 YAPA Versus Software

Caroline

YAPA achieved a victory rate of 66% against Caroline. This is a poorer result compared to the first round of testing, but might be caused by fluctuations in pure luck. Review of the game logs shows that YAPA is now able to alter its classification of an opponent. It classifies Caroline as shifting between loose-passive and loose-aggressive playstyles. Additional testing will be necessary to state with finality, whether YAPA's poorer results are caused by the fluctuations of the low number of games it played, or inaccuracies in it's decision model.

Randomiser

Against the randomiser, YAPA won 73% of its games, which was an improvement over the first test. Again, it is not completely obvious whether this is caused by the improvements to YAPA, or fluctuations due to luck. Review of the game logs, show that YAPA's classification of the randomiser primarily switches between unknown and tight-passive, loose-aggressive being a less represented factor. On the occasions where the randomiser performs a number of similar moves a few times in a row, YAPA is capable of performing a classification, but most of its moves are too confused to be ordered according to a scheme, and this causes the YAPA to return to the "unknown" classification throughout the games.

YAPA		
AI	Caroline	66%
	Caller	73%
	Random	73%
Humans		65%

Table 6.2: Test statistics for YAPA test 2.

Caller

Against the caller, YAPA achieved a 73% win rate. In all but the initial few hands, YAPA classified the caller as loose-passive, a playstyle which the caller fits to the letter. The fact that YAPA has not achieved a greater win-rate, could be attributed to YAPA not correcting its play sufficiently to match the playstyle of its opponent.

6.2.2 YAPA Versus Humans

In this round of testing, YAPA achieved a 65% win rate, which is a clear improvement over the previous round of human testing. However, the availability of human opponents for this test was unfortunately limited to the poorer of previously participating players. It had some difficulty identifying the playstyle of one player who interspersed a generally loose-aggressive style with "bursts" of tight-aggressive play when he received good cards, and YAPA generally lost money to this maneuver.

6.2.3 Evaluation of Test 2

YAPA was capable of performing classifications of its opponents during this round of testing. Against other artificial agents, this did not result in much improvement, and its total victory rates were actually slightly lower, despite the fact that it appears to correctly classify its opponents. This could indicate that YAPA does not have an optimal array of proper responses to playstyles. On the other hand, YAPA had improved when facing human opponents. The statistics for test 2 can be found in Table 6.2.

6.3 Test 3

The purpose of this test is to decide whether adjustments to the "k" value of the KNN algorithm and calibrations to the utility nodes of YAPA's Decision Network, will make YAPA more susceptible to alterations in the opponent's playstyle. As in test 2, the developers and non-developers will be categorised as humans, but the developers will not be told what is being changed. The effect of the calibrated "k" value upon classification will also be monitored closely.

6.3.1 Calibration

Changed utility nodes:

- `Opp_Hand_Strength_Prediction` - Doubled all utilities to make them have more effect on the decision.
- `WinU` - Lowered the utility of raises a bit, to make it less aggressive and a bit more versatile (40 to 30).

- `Opp_Investment` - A change to the prediction of how YAPA should react if it believes the opponent has two pairs and he is betting high. Instead of being a possitive utility (to account for bluffs) it was changed to a negative (2 to -2), to reflect that this is a hand which often wins the game (and therefore should not by default be considered a bluff).

Changed "k" value of the KNN algorithm to 3 (standard was 5), which should result in faster reactions to changed playstyles, but might also be too low a number, and therefore result in more cases of the "Unknown" classification.

6.3.2 YAPA Versus Software

Caroline

YAPA achieved a 76% win rate agains Caroline. This result is an improvement over test 2, and in fact more similar to the results of the first test. Assuming that this improvement is not simply an illusion caused by an insufficient number of tests, it would seem that being more readily adaptable is an advantage against Caroline.

Caller

YAPA reached an 85% win rate, which is an improvement over the results from test 2. This improvement can probably be attributed to the alterations to the utility nodes of YAPA's decision networks, which causes it to consider its opponents playstyle to be a more important factor in the predicted strength of its hand.

Randomiser

YAPA achieved a 68% victory rate against the randomiser. This is a reduction from test 2, but not a great one. It it not surprising that the speed of the classification, is largely irrelevant when playing against an opponent who does not follow any sort of strategy. The fact that the win rate diverges from test 2 at all, can be attributed to fluctuations caused by the luck of the draw.

6.3.3 YAPA Versus Humans

As a whole, YAPA did not improve as a result of the reduced value of k. As expected, it developed a tendency towards more spontaneous modifications of its classifications. However, this does not appear to have resulted in an improvement in the quality of its game. It achieved a 48.3% win rate against human opponents, most of which it managed a greater win rate against during test 2. When reviewing the logs, it was apparent that YAPA's inability to maintain a coherent image of its opponent's playstyle was a detriment to its playing ability.

6.3.4 Evaluation of Test 3

The results of the reduction to the set of considered hands, through "k", used to classify the opponent's strategy were mixed. It had swifter reactions to alterations in playstyle, but perhaps it reclassified too eargerly, which resulted in YAPA being less able to maintain a correct classification of a player's overall strategy. The results of this test are listed in Table 6.3. There was one player who YAPA previously was completely unable to beat, whom it could now achieve a 40% win rate against. This is a marked improvement. Unfortunately, the player was not available for test 2, so it is not completely clear whether the modifications

YAPA		
AI	Caroline	76%
	Caller	85%
	Random	68%
Humans		48.3%

Table 6.3: Test statistics for YAPA test 3.

to the value of k and the utility nodes resulted in a detriment or an improvement to YAPA's play against this player.

6.4 Final Evaluation

The original version of YAPA was decidedly incorrect in its implementation. Through evaluation of its performance during the tests, it has been possible to correct some flaws in its behaviour, and recognise others that have yet to be properly eradicated. Amongst these are an inability to recognise the playstyle tight-aggressive. YAPA also erroniously believes a player is having a good hand or bluffing, when the opponent is low on money, as it interprets the opponent's buy-in to the blind as aggressive play or high belief in his hand. On the other hand, this means that YAPA has been taking most of the opponents chips.
Generally, YAPA has improved its talent for playing poker. Primarily, it could be seen from test 1 that there was a fairly sharp limit to how talented humans YAPA was able to beat. When the testing had progressed to test 3, YAPA had overcome this limit and could actually win some of its games against these previously unbeatable opponents.
An overview of the results of the three tests can be seen in Table 6.4.

		Test 1	Test 2	Test 3
AI	Caroline	73%	66%	76%
	Caller	81%	73%	85%
	Random	64%	73%	68%
Humans	Developers	52%	65%	48.3%
	Non-developers	40.9%		

Table 6.4: Overview of test results.

Chapter 7

Conclusion and Future Works

This section will contain the conclusion of the project. Followed by details of the possible avenues of development that YAPA could take in the future, from basic improvements of the classification training set to emulation of human attributes.

7.1 Conclusion

YAPA is able to play poker and even classify opponents' playstyles according to the parameters we have defined. Against other agents YAPA performs very well, with a high win rate over Caroline and randomly acting agents. Against human players, the playing field is more even, with most players able to achieve 50 percent or higher win rate against YAPA.

YAPA is able to evalute hands based on probabilities, and play a full game of poker correctly. This in itself satisfies one of the base goals for this project. Further refinement, especially betting patterns, could however be useful in regards to making YAPA act more like a human player.

However, at this point in time, there are some issues with YAPA's classification of opponents. It is very basic, and straight out too simple. Even though the framework for classification is in place, it needs alot of refinement to really achieve what we initially hoped. One specific example of this could be how YAPA evaluates the opponent's hand strength, which it uses to decide if a player is playing a tight or loose game. Currently YAPA only evaluates the final hand combination, the 5 card combination used to decide who wins the hand. To make any sort of accurate classification, you need to consider the two initial cards the player was dealt as well. They are extremely important as a deciding factor, as to whether the player buys into the flop or not. Furthermore, correlating the player's hand at pre-flop, flop, turn and river, with betting patterns could also be very helpful. These are just some examples of how much the classification could, and should be improved, to have a really competitive and intelligent agent.

Finally, YAPA does not have any strategic knowledge. As described in section 2.2 on page 7, there are many strategic influences to consider, besides just playing styles of opponents. Big stack and small stack play for instance, is not something YAPA understands, and in fact an opponent simply paying the big blind, when that opponent is sitting with a small stack, can cause YAPA to think he is playing aggressive, since the blind now counts as a considerable percentage of his remaining stack size. So not only is YAPA not able to make any sort of strategic play itself, but it is also not able to counter strategic play from it's opponent. Of course right now, YAPA can only handle one opponent at the time, which eliminates the

importance of table positions. But still, one further goal of making YAPA a good poker agent, is the implementation of several opponents.

In conclusion, we have succeeded in achieving our basic goals for this semester. We have made YAPA able to play poker correctly, and deliver some opposition and challenge to both other agents and human players. However, most advanced goals have not been reached, either at all, or only partially. Player classification and profiling, is implemented at the moment, but only in a very crude version, and it needs further work. An understanding of poker strategy is not implemented at all, and YAPA can only handle one opponent at a time in its current form.

7.2 Future Works

YAPA may be functional now, but it has yet to reach a satisfactory level of poker ability. So far it is generally successful when compared to the other agents it was tested against, but it has not displayed a convincing ability to consistently defeat human opponents. Improving on this lack, should be the most immediate task, to be considered.

7.2.1 Immediate Technical Improvements

There are several aspects of YAPA that could do with improvement. Currently, some of its functionality is implemented in an unsatisfactory and haphazard manner; It uses a simple classification algorithm, the training set is small and the viability of its attributes could do with additional testing and editing. Most significant of current aspects that require improvement, is the feature selection/extraction for the classifier. There has not been sufficient testing to determine that the current setup is optimal. There are indicators pointing towards possible improvements, specifically to which features to use.

Another existing aspect of YAPA that could be tuned and improved is the Bayesian Network that is used for decision calculations. The current network abstracts over the poker game in a manner that causes YAPA to ignore certain card combinations, it could be possible to modify the network in a manner that will enable it to consider the full suite of available poker moves. Another limitation of YAPA, its inability to consider more than one opponent at a time, stems from the current Bayesian Network. Amongst other things, it does not have the capacity to express the alterations of probabilities that multiple opponents infer. The utility nodes, which are an important factor in YAPA's decisions, have not been sufficiently tested, and their values are unlikely to be optimal at the moment. Lastly, the complexity of considerations that the network enable YAPA to perform, is limited to four main factors. Addition and enlargement of the network would enable YAPA to make more informed decisions.

7.2.2 Advanced Learning and Play

So far, the pattern recognition of YAPA has only been implemented with one simple algorithm for player classification. But there exists a number of other algorithms, some of which are more advanced, that might be more applicable for recognising and exploiting the behaviour and psychology of poker players. This is, particularly, algorithms for semi-supervised learning, which would enable learning with a reduction to the cumbersome labeling of large amounts of training data, and might also cause an improvement in the overall training set. Furthermore, possessing an extensive knowledge base of player behaviour would provide YAPA with

an improved ability to create swift counter-moves to the actions or playstyles of its opponents.

The next step in YAPA's travel towards being a proficient poker agent, should be the implementation of an ability to emulate human behaviour and obfuscate its own nature, for instance by bluffing. This would serve to deter attempts from human players to ferret out its behaviour patterns. Also implementation of strategies into YAPA, such as small and big stack play, table position strategies when more than one opponent has been implemented, and the possibility for YAPA to perhaps assume any of the player profiles, to further emulated human behaviour in poker, could be beneficial.

A

Appendix

A.1 Artificial Neural Networks

This section contains theory in regards to Artificial Neural Network (ANN) , a learning model based on the functionality of biological learning systems, e.g. the human brain.

Unit A unit is analogous to a neuron in a brain; the ANN consists of a number of interconnected units, each taking a number of inputs, and producing a single resulting output, which may then again be directed to a number of other units as inputs.

Perceptron The Perceptron is one type of unit which an ANN can be constructed from. The Perceptron takes some input and outputs either 1 or -1. It reaches its decision by weighing each input by some fixed amount, and deriving the output by adding all inputs and running them through its activation function, if the result is larger than a given threshold, it outputs a 1, otherwise a -1.

Weights The weights in an ANN is the most essential part of the whole network, these are used for manipulating the Perceptron's input so that the activation function computes the correct $\pm$ 1 output for that given training example. If the output is incorrect, the weights are modified, this is done until the perceptron computes the correct or the best-fit output.

Activation Function The Activation Function computes whether the output of the unit, given it's input modded with the weights, should between +1 and -1. Different kinds of activation functions exists, whereof the most common is the Sigmoid function and the Sign function. The sign function produces a boolean output, +1 or -1, whereas the sigmond function produces output between +1 and -1.

A.1.1 Principles

The principal idea behind an ANN is to emulate a biological brain. To this end, ANNs are made up of a number of simple units passing input amongst each other, much like a brain consists of a number of interconnected neurons. These units can become quite numerous in advanced systems, again mimicking the large number of neurons in a brain, e.g. there are approx. 10^{11} neurons in a human brain, each of them connected to approx. 10^4 other neurons. In reality, ANNs may diverge somewhat in functionality from their biological roots. This is especially prevalent with ANNs dedicated to implementation of machine learning algorithms, where efficient results are a greater concern than absolute adherence to their biological origins.

A.1.2 Appropriate Problems

ANNs are particularly suited to solving problems that include a large amount of input, where not all is necessarily relevant to the solution of the problem, e.g. video, images or auditory input. They can also be successfully used for more strictly controlled scenarios, such as a poker game, but they are not more suited to this than other decision models, such as Bayesian Networks or Decision Trees.

A.1.3 Functionality

Each of the nodes(units, neurons) receives an input from an outside source, or another node in the network. Depending on the weights assigned to each node, the intermediate and final outputs will differ.

A.1.4 Learning

It is an integral aspect of a decision network is the ability to alter, learn and adapt, since they would not bring much new to computing if they required the user to supply both input, functionality and output. There exists a number of different algorithms for ANN learning. They will not be discussed in detail, as they are not relevant to this project.

A.2 YAPA Nodes

This appendix section will display screen dumps of all the states in all the nodes of the final YAPA Bayesian network. The full picture of all the nodes in the final YAPA network can bee seen in Section 4.2.5 on Page 31

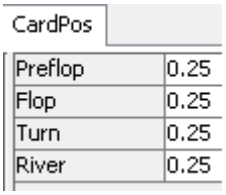


Figure A.1: CardPos.

CommunityCards

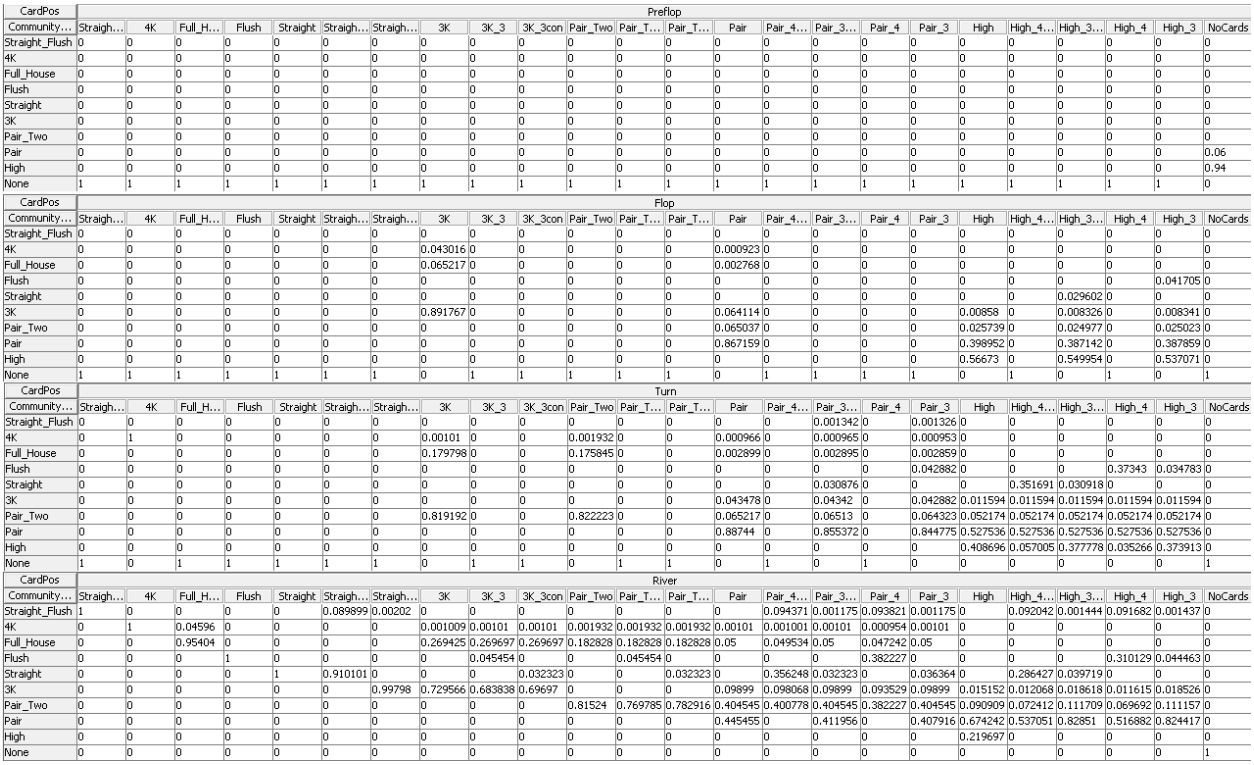
Straight_Flush	0.041667
4K	0.041667
Full_House	0.041667
Flush	0.041667
Straight	0.041667
Straight_4	0.041667
Straight_3	0.041667
3K	0.041667
3K_3	0.041667
3K_3con	0.041667
Pair_Two	0.041667
Pair_Two_3	0.041667
Pair_Two_3...	0.041667
Pair	0.041667
Pair_4con	0.041667
Pair_3con	0.041667
Pair_4	0.041667
Pair_3	0.041667
High	0.041667
High_4con	0.041667
High_3con	0.041667
High_4	0.041667
High_3	0.041667
NoCards	0.041667

Figure A.2: CommunityCards.

Investment_info_opp

High	0.5
Low	0.5

Figure A.3: Investment_info_opp.



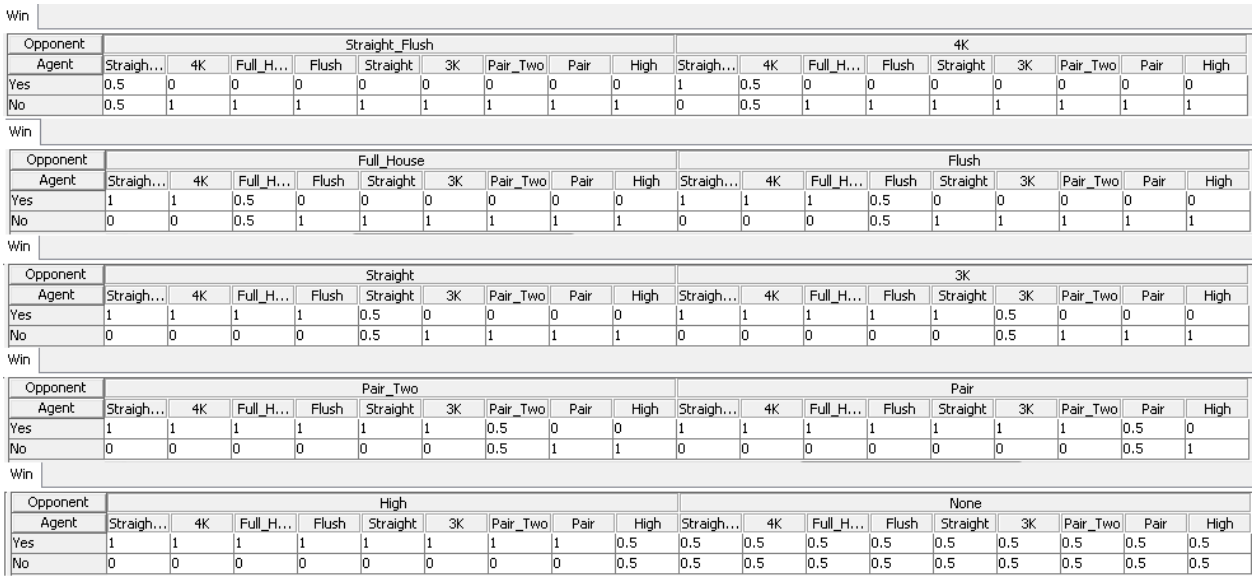
CardPos	Preflop																							
Community...	Straigh...	4K	Full_H...	Flush	Straight	Straigh...	Straigh...	3K	3K_3	3K_3con	Pair_Two	Pair_T...	Pair_T...	Pair	Pair_4...	Pair_3...	Pair_4	Pair_3	High	High_4...	High_3...	High_4	High_3	NoCards
Straight_Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Full_House	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Straight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pair_Two	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pair	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.06
High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.94
None	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

CardPos	Flop																							
Community...	Straigh...	4K	Full_H...	Flush	Straight	Straigh...	Straigh...	3K	3K_3	3K_3con	Pair_Two	Pair_T...	Pair_T...	Pair	Pair_4...	Pair_3...	Pair_4	Pair_3	High	High_4...	High_3...	High_4	High_3	NoCards
Straight_Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4K	0	0	0	0	0	0	0	0.043016	0	0	0	0	0	0.000923	0	0	0	0	0	0	0	0	0	0
Full_House	0	0	0	0	0	0	0	0.065217	0	0	0	0	0	0.002768	0	0	0	0	0	0	0	0	0	0
Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.041705	0
Straight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.029602	0	0
3K	0	0	0	0	0	0	0	0.891767	0	0	0	0	0	0.064114	0	0	0	0	0.00858	0	0.008326	0	0.008341	0
Pair_Two	0	0	0	0	0	0	0	0	0	0	0	0	0	0.065037	0	0	0	0	0.025739	0	0.024977	0	0.025023	0
Pair	0	0	0	0	0	0	0	0	0	0	0	0	0	0.867159	0	0	0	0	0.398952	0	0.387142	0	0.387859	0
High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.56673	0	0.549954	0	0.537071	0
None	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	0	1	0	1	0	1

CardPos	Turn																							
Community...	Straigh...	4K	Full_H...	Flush	Straight	Straigh...	Straigh...	3K	3K_3	3K_3con	Pair_Two	Pair_T...	Pair_T...	Pair	Pair_4...	Pair_3...	Pair_4	Pair_3	High	High_4...	High_3...	High_4	High_3	NoCards
Straight_Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001342	0	0.001326	0	0	0	0	0	0
4K	0	1	0	0	0	0	0	0.00101	0	0	0.001932	0	0	0.000966	0	0.000965	0	0.000953	0	0	0	0	0	0
Full_House	0	0	0	0	0	0	0	0.179798	0	0	0.175845	0	0	0.002899	0	0.002895	0	0.002859	0	0	0	0	0	0
Flush	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.042882	0	0	0	0.37343	0.034783	0
Straight	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.030876	0	0	0	0.351691	0.030918	0	0	0
3K	0	0	0	0	0	0	0	0	0	0	0	0	0	0.043478	0	0.04342	0	0.042882	0.011594	0.011594	0.011594	0.011594	0.011594	0
Pair_Two	0	0	0	0	0	0	0	0.819192	0	0	0.822223	0	0	0.065217	0	0.06513	0	0.064323	0.052174	0.052174	0.052174	0.052174	0.052174	0
Pair	0	0	0	0	0	0	0	0	0	0	0	0	0	0.88744	0	0.855372	0	0.844775	0.527536	0.527536	0.527536	0.527536	0.527536	0
High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.408696	0.057005	0.377778	0.035266	0.373913	0
None	1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	1

CardPos	River																							
Community...	Straigh...	4K	Full_H...	Flush	Straight	Straigh...	Straigh...	3K	3K_3	3K_3con	Pair_Two	Pair_T...	Pair_T...	Pair	Pair_4...	Pair_3...	Pair_4	Pair_3	High	High_4...	High_3...	High_4	High_3	NoCards
Straight_Flush	1	0	0	0	0	0.089899	0.00202	0	0	0	0	0	0	0	0.094371	0.001175	0.093821	0.001175	0	0.092042	0.001444	0.091682	0.001437	0
4K	0	1	0.04596	0	0	0	0	0.001009	0.00101	0.00101	0.001932	0.001932	0.001932	0.00101	0.001001	0.00101	0.000954	0.00101	0	0	0	0	0	0
Full_House	0	0	0.95404	0	0	0	0	0.269425	0.269697	0.269697	0.182828	0.182828	0.182828	0.05	0.049534	0.05	0.047242	0.05	0	0	0	0	0	0
Flush	0	0	0	1	0	0	0	0	0.045454	0	0	0.045454	0	0	0	0	0.382227	0	0	0	0	0.310129	0.044463	0
Straight	0	0	0	0	1	0.910101	0	0	0	0.032323	0	0	0.032323	0	0.356248	0.032323	0	0.036364	0	0.286427	0.039719	0	0	0
3K	0	0	0	0	0	0	0.99798	0.729566	0.683838	0.69697	0	0	0	0.09899	0.098068	0.09899	0.093529	0.09899	0.015152	0.012068	0.018618	0.011615	0.018526	0
Pair_Two	0	0	0	0	0	0	0	0	0	0	0.81524	0.769785	0.782916	0.404545	0.400778	0.404545	0.382227	0.404545	0.090909	0.072412	0.111709	0.069692	0.111157	0
Pair	0	0	0	0	0	0	0	0	0	0	0	0	0	0.445455	0	0.411956	0	0.407916	0.674242	0.537051	0.82851	0.516882	0.824417	0
High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.219697	0	0	0	0	0
None	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure A.4: Opponent node.



Win

Opponent	Straight_Flush									4K								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Yes	0.5	0	0	0	0	0	0	0	0	1	0.5	0	0	0	0	0	0	0
No	0.5	1	1	1	1	1	1	1	1	0	0.5	1	1	1	1	1	1	1

Win

Opponent	Full_House									Flush								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Yes	1	1	0.5	0	0	0	0	0	0	1	1	1	0.5	0	0	0	0	0
No	0	0	0.5	1	1	1	1	1	1	0	0	0	0.5	1	1	1	1	1

Win

Opponent	Straight									3K								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Yes	1	1	1	1	0.5	0	0	0	0	1	1	1	1	1	0.5	0	0	0
No	0	0	0	0	0.5	1	1	1	1	0	0	0	0	0	0.5	1	1	1

Win

Opponent	Pair_Two									Pair								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Yes	1	1	1	1	1	1	0.5	0	0	1	1	1	1	1	1	1	0.5	0
No	0	0	0	0	0	0	0.5	1	1	0	0	0	0	0	0	0	0.5	1

Win

Opponent	High									None								
Agent	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High	Straigh...	4K	Full_H...	Flush	Straight	3K	Pair_Two	Pair	High
Yes	1	1	1	1	1	1	1	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
No	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Figure A.5: Win node.

Agent

Straight_Flush	0.111111
4K	0.111111
Full_House	0.111111
Flush	0.111111
Straight	0.111111
3K	0.111111
Pair_Two	0.111111
Pair	0.111111
High	0.111111

Figure A.6: Agent node.

Investment...	High									
Opponent	Straight_Flush	4K	Full_House	Flush	Straight	3K	Pair_Two	Pair	High	None
Utility	-15	-15	-10	-5	-5	-1	2	5	10	0
Investment...	Low									
Opponent	Straight_Flush	4K	Full_House	Flush	Straight	3K	Pair_Two	Pair	High	None
Utility	0	0	0	0	0	0	0	0	0	0

Figure A.7: Investment Low.

Investment_info

low	0.5
high	0.5

Figure A.8: Investment Info.

WinU

Win	Yes			No		
Yapa_decision	Fold	Check/Call	Bet/raise	Fold	Check/Call	Bet/raise
Utility	0	3	40	1	-1	-20

Figure A.9: WinU.

Yapa_decision

Fold	0.333333
Check/Call	0.333333
Bet/raise	0.333333

Figure A.10: Yapa_decision.

Opp_Hand_Strength_Prediction

Opponent	Straight_Flush														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	4K														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	Full_House														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	Flush														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	Straight														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	3K														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	Pair_Two														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

Opp_Hand_Strength_Prediction

Opponent	Pair														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	-5	0

Opp_Hand_Strength_Prediction

Opponent	High														
Opponent_Action	Raise					Call					Fold				
Opponent_Type	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	-5	-5	-10	0	0	0	-5	-5	0	0	0	0	0	0

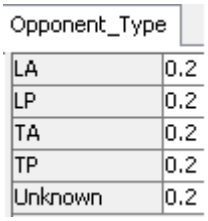
Opp_Hand_Strength_Prediction

Opponent	None													
Opponent_Action	Raise				Call					Fold				
Opponent_Type	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown	LA	LP	TA	TP	Unknown
Utility	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.11: Opp_Hand_Strength_prediction node.

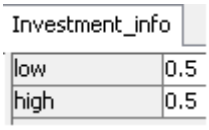
Opponent_Action	
Raise	0.333333
Call	0.333333
Fold	0.333333

Figure A.12: Opponent_Action.



Opponent_Type	
LA	0.2
LP	0.2
TA	0.2
TP	0.2
Unknown	0.2

Figure A.13: Opponent_Type.



Investment_info	
low	0.5
high	0.5

Figure A.14: Investment_info node.

Bibliography

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson, 3rd edition, 2010.

[2] http://www.pokernews.com/poker-rules/texas-holdem.htm.

[3] http://www.poker.dk/artikler.php.

[4] http://www.pokerlistings.com/poker-player-interviews/wpt/season6/interview-with-lapc-champ-phil-ivey.

[5] http://www.pokerlistings.com/poker-strategy-articles/tournament-nl-holdem.

[6] http://www.texas-holdem-net.com/types-of-poker-players.html.

[7] http://www.pokerlistings.com/strategy/general-poker/optimizing-play-for-your-stack-size.

[8] http://www.tightpoker.com/position.html.

[9] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007.

[10] http://hugin.com/developer/documentation.

[11] Ian H. Witten and Eibe Frank. *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier, 2nd edition, 2005.

[12] http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm.

[13] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Boston Academic Press, 3rd edition, 2006.

[14] Xiaojin Zhu and Andrew B. Goldberg. Introduction to semi-supervised learning, 2009.

[15] Dion Bak Christensen, Henrik Ossipoff Hansen, Anders Hesselager, Lasse Juul, Kasper Kastaniegaard, and Michael Skoett Madsen. Caroline: Analytical real-time object-oriented lazy influence network engine, 2009.