NTNU
Norwegian University of
Science and Technology

Integrating CBR and BN for Decision Making with Imperfect Information

Exemplified by Texas Hold'em Poker

Sebastian Helstad Unger

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Agnar Aamodt, IDI
Co-supervisor: Tore Bruland, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Motivated by the TLCPC project in clinical decision support, in which a set of architectures have been outlined for combination of Case-Based Reasoning and Bayesian Networks, a combined architecture for improved decision making in the game of Texas Hold'em should be developed and partly tested. Poker Academy Pro is a relevant commercial poker game platform to start out from and compare results to.

Assignment given: January 18, 2011
Supervisor: Agnar Aamodt

Abstract

Texas Hold'em Poker provides an interesting test-bed for AI research with characteristics such as uncertainty and imperfect information, which can also be found in domains like medical decision making. Poker introduces these characteristics through its stochastic nature and limited information about other players strategy and hidden cards. This thesis presents the development of a Bayesian Case-based Reasoner for Poker (BayCaRP). BayCaRP uses a Bayesian network to model opponent behaviour and infer information about their most likely cards. The case-based reasoner uses this information to make an informed betting decision. Our results suggests that the two reasoning methodologies combined achieve a better performance than either could on its own.

Preface

This report constitutes my master thesis as part of the Master of Science studies at the Norwegian University of Science and Technology (NTNU). The thesis was carried out within the Division of Intelligent Systems (DIS) at the Department of Computer and Information Science (IDI) during the spring semester of 2011.

First, I would like to thank my supervisor, Agnar Aamodt, for his guidance, valuable feedback and suggestions during this research. I would also like to thank my bi-supervisor, Tore Bruland, whose advice lead me to the domain chosen for this thesis.

In addition, I would also like to thank my family and my significant other for moral support and for always believing in me.

Last but not least, I would like to thank the University of Alberta for their valuable research and tools in the domain of artificial intelligence for poker, and Jonathan Rubin and Ian Watson for their research on applying CBR for poker.

Trondheim, June 18, 2011

Sebastian Helstad Unger

Contents

List of Figures

List of Tables

Part I

Background & Related research

CHAPTER 1

Introduction

1.1 Background & Motivation

The field of knowledge based systems has over the years become a mature field with increasing focus on integrating various types of knowledge and reasoning methods [7][8]. The increasing focus on integration partly stems from the wish to better model human problem solving and learning, which involves representation and utilization of several types of knowledge and reasoning methods. Model Based Reasoning (MBR) and Case Based Reasoning (CBR) are two well established types of reasoning methods in the AI community.

MBR is an approach in which general knowledge is represented by formalizing the relationships present in a problem domain. These models typically capture causal relationships to diagnose problems or predict situation outcomes. A well known model of this kind is a probabilistic belief network, also referred to as Bayesian Networks (BN). Bayesian Networks have a strong statistical basis and maintains a symbolic representation in terms of a dependency network, where relations in the network may be given meaningful semantic interpretations, such as causality [9].

Case-based Reasoning is the most recent contribution in the history of knowledge based systems. CBR distinguishes itself from other AI approaches, by utilizing specific knowledge of previously experienced situations (cases) instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and solutions [2]. A new problem is solved by finding a similar past case, and reusing it in the new problem situation.

One of the strengths of CBR is that it uses situation specific knowledge to reason with, and relaxes the knowledge engineering phase of the user. Due to this reason many CBR researchers see the idea of reasoning with general knowledge as counterintuitive to the very idea of CBR. Still, problem solving and learning by combining general and case specific knowledge seems to be what people do [8]. Knowledge intensive CBR shifts the burden of the usual knowledge engineering phase to providing knowledge for indexing, relating and reusing cases [10].

There exist several possibilities for combining CBR with other reasoning modalities. This thesis will focus on the integration of Bayesian Networks for maintaining the general domain knowledge needed by the CBR system. The motivation for this thesis stems from both the ongoing TLCPC project [4], where members from the Department of Computer and Information Science (IDI) are working with medical professionals from the St. Olavs Hospital in Trondheim to create a computer based tool for assisting treatment in pallia-

tive care, and a more AI methodical motivation to combine CBR with another reasoning method to maintain its general domain knowledge. These motivational factors will be described further in the next section.

Decision making in the medical domain is to a large degree characterized by uncertain and incomplete information. Patients may reveal various symptoms for a underlying disease, and information acquired may be unreliable and uncertain. Treatment may progress over several stages, where new information is revealed at each stage, such as the results of applying a drug for relieving the pain of a patient in palliative care. There are also risks involved when applying various treatments.

Since medical professionals are busy people, and medical data is not easily shared, this thesis will focus on a domain which encompasses similar features as the ones detailed above and is of relevance to the clinical domain. One such domain is the game of poker, or more specifically the game of Texas Hold'em Poker. The opponent's cards are hidden, and may be predicted trough the opponents action at each stage of the game. This makes the domain suffer from incomplete or in other words imperfect information. The games stochastic nature combined with the possibility of the opponent deceiving you with their actions, makes the information available uncertain and unreliable. There are multiple stages in the game of Texas Hold'em, where each stage reveals new information trough actions and new cards available. Each betting action or strategy has its consequences or risks.

The rest of this chapter will detail the motivational factors in section 1.1.1 and 1.1.2, describe the goal and the method of investigation in section 1.2, and conclude with giving a structural overview of this report in section 1.3.

1.1.1 The TLCPC project

The TLCPC-project, Translational Research in Lung Cancer and Palliative Care , is one of the motivational factor for this research. The project is nationally funded and tightly linked to the larger EU project; European Palliative Care Research Collaborative (EPCRC)[11]. These projects share similar goals and the main objective is to improve the methods for management of cancer in the palliative care. The World Health Organization (WHO) defines palliative care as follows:

> the active, total care of patients whose disease is not responsive to curative treatment. Control of pain, of other symptoms, and of psychological, social and spiritual problems is paramount. The goal of palliative care is achievement of the best quality of life for patients and their families. Many aspects of palliative care are also applicable earlier in the course of the illness in conjunction with anti-cancer treatment. [12]

The TLCPC-project will achieve better palliative care through four collaborative research areas illustrated in figure 1.1. The genotyping track will involve identifying genes that may be associated with the development of lung cancer, and gene expressions as prognostic markers of disease devolvement, symptoms and treatment response. Developing genetic methods for predicting individuals' opioid responses is also a focus in the EPCRC project. The gene expression track will follow a similar approach but focus on genetic markers in peripheral blood cells (PBL) , which may result in the development of

a test for early prediction of cancer through blood-based gene-expression tests. The computer oriented part of the project will involve developing a decision support tool to guide the clinician in the assessment, classification, treatment of cancer related symptoms, such as pain, cachexia, dyspnoea, physical function, depression and fatigue. The two projects will together cover all of these symptoms. The results of these research areas will form the basis for an intelligent decision support system for clinical practice. This system will combine both general knowledge, such as clinical guidelines, with situation-specific knowledge from past experiences. This will be achieved by combining case-based reasoning and Bayesian networks, two well known artificial intelligence methods. The clinical objective for the system is to support the clinician in the management of pain and other effects of chronic cancer.

A group of researchers at NTNU is currently working on the intelligent decision support system. The first prototype they developed was a purely rule-based system based on clinical guidelines for the domain. The second prototype is currently under development and involves both Case-based reasoning and a Bayesian network to support the classification and treatment of pain. The Bayesian network will calculate probabilities based on its underlying model, while the Case-based reasoning component will retrieve similar past cases to support this decision making.

The combination of reasoning methods for clinical decision support, constituting track D of the TLCPC project, is a motivation factor for this thesis. The next section will detail the combination of CBR and BN further, but will move away from the medical domain and into the test-bed chosen for this thesis.

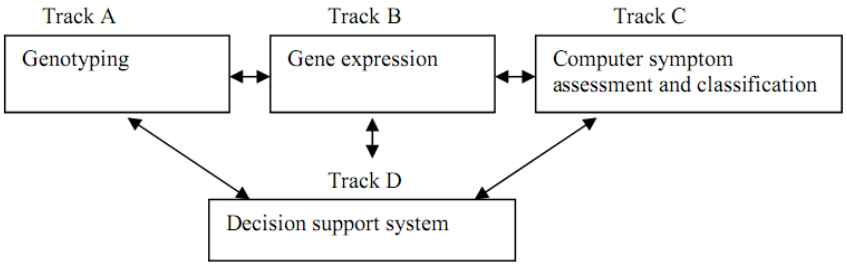


Figure 1.1: Model of the collaboration of the tracks in the TLCPC project [1]

1.1.2 Integrated Reasoning

The motivation for combining CBR and BN is that they both contribute to improved decision making under uncertainty and incomepleteness [4]. One of the main strengths of CBR is that it relies on situation specific knowledge to reason about newly presented events. This reduces one of the major concerns for knowledge based systems, the knowledge engineering phase, where the developer attempts to explicitly model the domain-dependent knowledge as computer software structures [13].

Although CBR relies on previously experienced situations to reason with, it does not mean that it is knowledge poor. Instead of having to explicitly model the whole domain, CBR relies on domain knowledge in different parts of its problem solving process[14]. By providing domain knowledge for indexing cases, the CBR system can better retrieve the cases relevant to its new problem situation. Domain knowledge in some form is required

if the retrieved case needs to be adapted to better fit the new situation, and can also be provided in the stage of comparing the similarity of cases.

Maintaining the domain knowledge needed by the CBR system in the form of a Bayesian network in the presence of uncertainty, plays on one of its major advantages, namely by representing the strengths of causal relationships in the domain with probabilities. Poker introduces uncertainty trough incomplete information about opponents cards, behavior and through randomization by the shuffling of cards. Some of the most important factors in poker decision making is experience and statistics[15], which makes the combination of CBR and BN a perfect candidate.

Experienced poker players will have participated in more scenarios than an amateur, and will have a better chance of having experienced a similar situation and its outcome of applying various strategies, when presented with a new situation, than players with less experience. This reasoning process is one of the fundamentals of case-based reasoning and motivates the desire to apply CBR to model this process. When presented with a new situation, there are various features that affect the poker players reasoning process. One of the more complex features used, is the opponents behavior and what cards he is most likely holding. Modeling this uncertainty motivates the use of Bayesian Networks. The BN can capture knowledge about relations in the domain, such as observed opponent behavior and the publicly available cards, to infer probabilities over the opponent's likely holdings.

1.2 Goal & Method of Investigation

The overall goal of this thesis is:

- to investigate the combination of CBR and BN in a non-deterministic imperfect information game through development of an experimental system in the domain of Texas Hold'em Poker.

Our approach to fulfilling the goal of this thesis is as follows. The research begins with an analytical approach to related research on the subject of combining the two reasoning methods, general AI applications in the domain and available software needed at a later stage. This background analysis will result in a proposed architecture, which will be implemented and evaluated using the selected tools. A more detailed view of our approach is given below, where the four first items constitute the analytical part of this research, and those following are of a more experimental nature.

- *Combining CBR and BN:* As one of the important factors in this project, the combination of CBR and BN in previous research will be studied and analyzed in the light of research performed at the Norwegian University of Science and Technology (NTNU).
- *AI in Texas Hold'em Poker:* Earlier attempts on applying AI methods in the domain of Texas Hold'em Poker will by studied, with the focus on attempts of applying CBR and BN.
- *Related software:* A short study of available software tools will be performed, in which some will be selected and investigated further. Only the selected tools will be described in this thesis.

- *Choosing environment:* Texas Hold'em Poker has a variety of different variants, game modes and sub-domains where AI could be applied. A suitable environment will be chosen from these.
- *Proposing an architecture:* Based on the analysis of the background study an architecture will be proposed and specialized in the chosen environment.
- *Implementing the architecture:* The proposed architecture will be implemented with the selected tools.
- *Testing the system:* The implemented system will be tested by playing against earlier attempts of applying AI in poker. The test will be performed by first using only one of the reasoning methods, and thereafter using the combination of both. If time allows it, the system will also be tested against human players.
- *Evaluating the system:* The system will be evaluated by the results obtained in the testing phase, based on known criteria for evaluating poker players performance. The architecture will also be evaluated in respect to the chosen way of combining CBR and BN.

1.3 Overview of Report

This thesis is divided into two main parts, the first concerning background information and related research, while the second describes the results achieved in this thesis.

Chapter 2 introduces the two main reasoning methodologies and the domain used in this research. Chapter 3 describe the tools necessary for the implementation of this system. Chapter 4 begins by describing the TLCPC framework, and uses this framework to categorize research related to the combination of the two reasoning methodologies. The second part of this chapter describes research related to applying these methods separately in the domain.

The part describing the results of this thesis begins by describing the design of the system in chapter 5 and the implementation of the system in chapter 6. Chapter 7 describes the results obtained when testing the system, and chapter 8 discusses these results. Finally, chapter 9 concludes this report and provides ideas for further work on improving the system.

CHAPTER 2

Methods & Domain

This chapter presents important background knowledge for this research. The chapter begins by presenting the two AI methodologies used, namely Case-based Reasoning and Bayesian Networks, and ends with an introduction to Texas Hold'em Poker. The essence of this chapter is to introduce and describe concepts that will be used throughout this thesis.

2.1 Case-Based Reasoning

Case-Based Reasoning (CBR) is an approach to problem solving and machine learning that uses the specific knowledge of previous encountered problem situations to solve new problems. A case is a set of features describing both the context of the problem situation, the solution applied to the given problem and the outcome of applying the solution. The most common way to describe the CBR problem solving process is by the four-step cycle illustrated in figure 2.1. How these steps are implemented may vary greatly between systems, and any given system may or may not incorporate the whole CBR cycle.

When a new case is presented to the system it retrieves one or more cases from the case-base. This selection process involves a similarity value which is used to describe the cases best fit to solve the new problem. The similarity calculations may vary between systems, but a common way to implement it is to use some kind of local similarity functions to compare different features in the cases and then use a global similarity measure to compute the overall similarity of the cases. This may be done by giving each feature a weight describing its importance in the matching process, and then calculating the weighted average of the relevant features to compute the global similarity. The retrieval step typically consists of several subtasks which involve identifying the relevant features, retrieving several matching cases and then selecting the most promising case.

The reuse step involves suggesting a solution to the input problem based on the output from the retrieval step. This is done by considering the differences among the past case and the current case and deciding what part can be transferred to the new case. This process may be divided into two subtasks: copy and adapt. The copy subtask is the most trivial form of reuse where the differences are abstracted away and the solution of the retrieved task is used as the solution of the new case. The adaption subtask is one of the more problematic areas in CBR and it involves choosing what parts of the case to reuse based on the differences amongst the cases. This may be done by either adapting the solution of the past case or by adapting the method that constructed the solution.

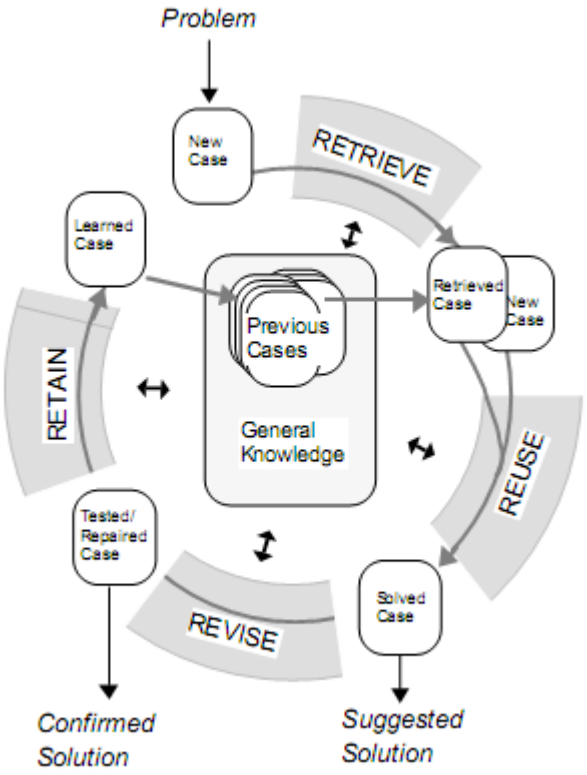


Figure 2.1: The CBR cycle [2]

The solution suggested by the reuse step is then evaluated by applying the solution in the real environment (asking a teacher or performing the task in the real world). If unsuccessful the solution should be repaired or revised using domain-specific knowledge or user input. In some domains, such as the medical domain the success or failure of a treatment may take some time to appear. In that case the case may still be learned but should be marked as non-evaluated.

Finally, the new knowledge acquired by the new problem-solving episode is retained into the case-base, which means learning from the success or failure of the proposed solution. This involves deciding the relevant parts of the case to retain, how to retain it, in what form, how to index the case for later retrieval from similar problems and how to integrate the new case into the case-base.

2.2 Bayesian Networks

Bayesian networks (BN), also referred to as belief networks or probabilistic causal networks, are directed acyclic graphs that represent conditional dependencies among a set of stochastic variables. The random variables are represented as nodes with connecting arcs to represent probabilistic dependence among them, which together constitute the topology of the network. Bayesian networks avoid specifying the complete probability distribution over a set of random variables by its built in independence assumption, represented by the arcs, where a node is conditionally independent of its non-descendants given its parents[16]. A node is considered a parent of another node if it has a direct link to that

node. Figure 2.2 illustrates a simple Bayesian Network, where the nodes *MarryCalls* and *JohnCalls* are conditional independent og *Burglary* and *Earthquake* given *Alarm*.

Each node contains a conditional probability table (CPT) containing a probability for each of its own values given a combination of the values in the parent nodes, as seen in figure 2.2. *P(A)* in the figure gives the posterior probability given a combination of the states(in this case boolean values) of its parent nodes *Burglary (B)* and *Earthquake (E)*. The product of these local probability distributions can then be used to calculate the full probability distribution of the domain, and is one of the advantages with Bayesian Networks. If we assume that all the variables are boolean, the BN reduces the numbers required for calculating the full probability distribution by 2^n to $n2^k$, where n is the number of variables in the network and k is the number of parents of each node in the network.

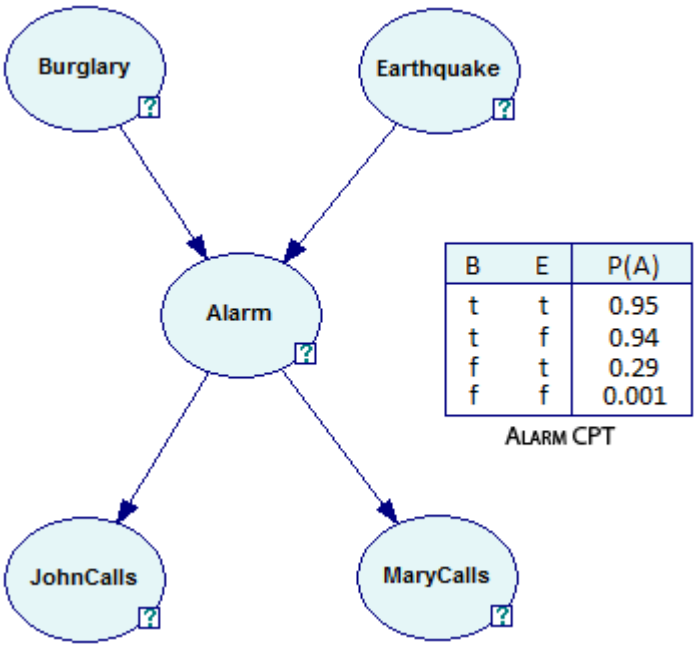


Figure 2.2: An example of a Bayesian network, showing both the topology and the CPT [3]

The full probability distribution can be used to answer any query about the domain, and as the BN is a representation of the full probability distribution, it too can be used to answer any query. This process of calculating a posterior probability for a query given some evidence is referred to as probabilistic inference. Probabilistic inference in Bayesian Networks is performed by inserting some observed evidence into it, which will result into the posterior probability being calculated over the rest of the nodes. There exists several algorithms for performing probabilistic inference in a Bayesian network, but they are all based on Bayes' theorem in some form.

Bayesian Networks also provide a way of integrating knowledge from different input sources. There exists a variety of algorithms for learning both the structure and the parameters in the network from observed data. The structure and parameters may also be elicited from an expert, or by using a combination of expert knowledge and data.

2.3 Texas Hold'em Poker

2.3.1 General

Poker is a non-deterministic imperfect information betting game that exists in numerous variations. The most famous of these is the game of Texas Hold'em, which is widely considered as the most strategically complex variant. Texas Hold'em is played with a standard 52-card deck with 2–10 players. Each round has a maximum of 4 stages, where the goal is to either out-bet the other players or possessing the strongest hand at the end of the round. The round ends when there is only one player left, or when the 4 stages of betting are complete and the players left reach *showdown* where their cards are revealed.

2.3.2 Stage 1: Preflop

The round begins with what is referred to as the *preflop* stage, where each player is dealt two private cards, called *hole cards*, which are only visible to the player himself. This is illustrated in figure 2.3, where we can observe the preflop stage from the perspective of a player called *Hero*. In this particular case, Hero's hole cards are ace of clubs and ten of clubs. These cards have the same *suit*, meaning that they are both clubs, and have a *rank* of 10 and 14 respectively. Hole cards of same suit are normally just referred to as *suited*, while the opposite is referred to as *offsuit*. Each player at the table started with 200$, and the snapshot illustrated the figure is eight rounds after the game began.

The two players to the left of the dealer, the *small-blind* and the *big-blind*, have to make a forced bet, which means that they have to place a predetermined amount of money in the pot before seeing their cards, ensuring that there is something to play for every round. The big-blind always doubles the amount of the small-blind.

The dealer is the one currently holding the dealer chip, illustrated by the *D*-chip in the right corner of the image, followed by the big-blind and the small-blind positioned to the left of the dealer. The dealer chip rotates for every round to ensure that each player will be able to play from every possible position at the table. The first player to act in theround is the person to the left of the big-blind, illustrated as the player with the visible cards. The betting then continues in a clockwise fashion until each player has acted, ending with the two *blinds*.

Since the small-blind is half the big-blind, this player is at least required to double his previously forced bet to continue playing, while the big-blind has the ability to *check* (see below), if no previous *bet* has been made.

There are three possible actions in poker; *fold*, *call* and *bet*:

- **Fold:** The player doesn't match the bet by a previous player and forfeits any chances of winning the pot. He will no longer participate in the upcoming stages of the round.
- **Call/Check:** The player puts money in the pot equal to the highest bet made by a previous player. If no bet has been made by a previous player, then the call action goes under the name of check.
- **Bet/Raise:** The player willingly puts new money into the pot even though it is not required. When a bet has been made, the other players have to match this bet to

Figure 2.3: The preflop stage

continue playing. If a previous player has already bet when a new player performs the bet action, this is referred to as a *raise*.

Although these are the most basic actions required to play poker, they may be referred to by a variety of names when used in combination and in various situations, which will be detailed further in section 2.3.7.

2.3.3 Stage 2: Flop

After the first round of betting is complete, the game continues to the *flop* stage where 3 cards are revealed face up on the board, as illustrated in figure 2.4. In this case Hero has a pair of tens and a decent chance of making it into a better *hand type*, if one of the next cards reveal a queen. If the queen comes, then Hero would have a *straight*, meaning five cards whose rank form a sequence.

The cards at the table can be used by all the players when attempting to make the best possible 5 card combination, and are referred to as *community cards*. After each new stage a new sequence of betting occurs, and while there are 2 players still left in the game, the round moves to the next stage. The two last stages in the game are referred to as the *turn* and the *river*, where 1 new card is revealed at the table in each of these stages.

2.3.4 Stage 3: Turn

The third stage in the game is referred to as the turn, where one new card is revealed at the table before the betting begins with the person to the left of the dealer. This is the same for all other rounds than the preflop, where the person to the left of the big-blind begins.

Figure 2.4: The flop stage

Revisiting the previous example, figure 2.5 shows that the turn card revealed another ten, making Hero's hand type into *three of a kind*.

Figure 2.5: The turn stage

2.3.5 Stage 4: River

After the last card is revealed at the *river*, the final sequence of betting commences, and if there are still players left in the game, they face each other in what is referred to as the *showdown*.

At showdown, seen in figure 2.6, the player with the strongest 5 card combination out of the 5 community cards combined with the players hole cards wins the pot. In this case, the ecstatic winner is Hero, with three of a kind in tens. For a ranked list of the possible 5 card combinations see the Appendix C.

Figure 2.6: The river stage (showdown)

In Texas Hold'em players can tie at *showdown* because there doesn't exist any suit rankings, which means that a *Ace* of *Spades* is equal to a *Ace* of *hearts*, which will result in the tied players splitting the pot evenly.

2.3.6 Betting restrictions

There are various ways to restrict the betting amounts in poker, which can lead to different styles of play. One option is the *No-Limit* variant where players bets are only limited by the money they currently possess. Other variants include *Pot-Limit* and *Spread-Limit* where the former sets a minimum and maximum amount to raise or bet depending on what is currently in the pot, while the latter provides a fixed minimum and maximum range for the allowed bets.

The focus of this thesis will be the *Fixed-Limit* format, referred to as Limit Texas Hold'em, where the allowed betting increments are of a predetermined size. For example a $1/2 limit game denotes that the betting increment for the *preflop* and *flop* are 1 dollar, while being 2 dollars for the *turn* and *river*. Fixed-limit also has a restriction on

a maximum number of allowed raises at a given stage of the game, meaning that if this maximum is exceeded the players aren't allowed to raise the pot any further.

2.3.7 Actions

The game of poker only allows for the actions described in Section 2.3.2, but when used in various situations they may reveal different things about an opponent. The poker community has therefore adopted a set of names for referring to the actions being used in different scenarios or combinations, which are illustrated below.

- **Three-Bet:** is the second raise or the third bet at a specific stage. At the preflop stage the big-blind constitutes one bet, which means that if a player raises after a previous raise, this would count as a three-bet.
- **Cap:** is the raise that ends or caps the allowed number of raises. This term is only used in the limit variants.
- **Continuation-Bet:** is a bet made from the previous rounds's aggressor at the flop stage. The term aggressor refers to the player which performs the last raise in the previous round. This action is often used to continue showing strength, even though the community cards may not have made the players hand any stronger.
- **Donk-Bet:** is a bet from a player that acts before the last stage's aggressor.
- **Check-Raise:** is an attempt to lure the opponent to bet by showing weakness with a check, and then raising his bet, which is usually a sign og strength.

CHAPTER 3

Tools

This chapter introduces the various software tools and technologies used for this research. Section 3.1 describes the CBR development tool, jColibri[17], which will be used to implement the CBR part of the architecture proposed in chapter 5. Among the reasons for choosing this particular CBR software is that it incorporates the whole CBR cycle, compared to myCBR[18] which only supports the retrieval step. jColibri is simple to use and provides a lot of information about its variety of functionality, and is easy to integrate with various third party software, such as myCBR. This way the user can take advantage of e.g. both myCBR's and jColibri's advantages when developing a CBR system. This makes jColibri the tempting choice over another well known object-oriented CBR framework such as CBR*Tools[19], and the less known exit*CBR[20] which is primarily intended for classification in the medical domain.

Section 3.2 describes the BN tool, GeNIe[21], used for implementing the BN part of the architecture proposed in chapter 5. GeNIe is one of several similar software tools for creating Bayesian Networks applicable for this thesis. The main difference between GeNIe and other software such as Hugin[22] and Netica is that it's full version is available to the public free of charge. Another difference between GeNIe and Netica, is that Netica doesn't support learning the structure of the BN from data. GeNIe also distinguishes itself from these two by providing multiple inference algorithms.

The chapter ends with some software tools related to the domain of Texas Hold'em Poker. Section 3.3 introduces Poker Academy Pro, which will later be used as a test platform for the implemented system, and the Meerkat API with some of its functionality. Open Hold'em in section 3.4 is used for interfacing with various online casinos, and is intended to be used for testing the system versus human competition.

3.1 jColibri

jCOLIBRI is an object-oriented framework in Java for developing CBR systems, with a major focus on reusing existing CBR knowledge [17]. The framework currently has two major releases; jColibri version 1 and 2. The First version includes a Graphical User Interface (GUI) that guides the user in the design of a CBR system, and was mainly intended for non-developers that want to create a CBR system without programming any code. The second version separates the architecture into two layers, one oriented to developers and the other to designers. Figure 3.1 illustrates the two layer architecture of jColbri2. The bottom layer contains the basic components of the framework with well defined and clear interfaces, and does not contain any GUI guide the user. The top layer is currently under

development and is meant to provide the user with semantic descriptions of the components and several tools for aiding the user in a user friendly GUI. The newest version of jColibri contains a lot more functionality and extensions, such as Hibernate for database interaction, new ontology-based similarity measures and advanced textual CBR functionality. For the rest of this project the term jColibri will be used for the second version. The following paragraphs will explain the jColibri components in more detail.

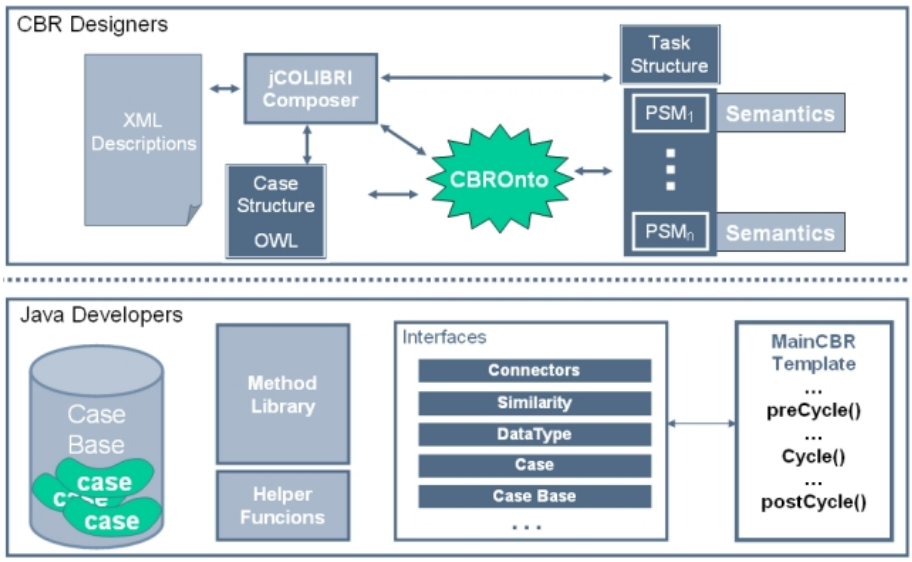


Figure 3.1: The jColibri two layered architecture

jColibri makes a clear distinction between the knowledge needed for the problem solving process, and the knowledge needed for the specific domains. To achieve this jColibri is built around a task/method ontology, which guides the design process. The task/method ontology is described using terms from a CBR ontology to provide a domain independent framework for CBR system development.

The CBR ontology[23] is an extensive ontology for defining and organizing the terminology used in the CBR process, resulting in a common language for defining CBR systems independent of the domain the system is intended for. The CBR ontology is built into the jColibri framework trough abstract classes and inheritance. This provides an abstract and flexible interface for developing a CBR system. The user can define his own components by inheriting the features of the abstract classes. CBROnto also includes knowledge about the tasks and methods, such as which tasks they are able to solve and other criteria related to a specific component.

These Problem solving methods (PSMs) are application and domain independent, reusable reasoning components. A PSM contains the problem solving behavior needed to achieve a specific task's goal. A PSM is decomposed into a number of subtasks. jColibri supports this behavior by providing methods to either decompose the task or solve the task directly. At the highest level of generality, they describe the CBR cycle in the four tasks described earlier, which again contains several subtasks. When a task is decomposed jColibri provides new PSMs to solve or decompose the task further[17]. There may be several ways to reach a task's goal, and jColibri provides an extensive library of well known PSMs to solve a variety of tasks.

jColibri splits its case base management into a persistency mechanism and in-memory organizations[24]. Persistence is built around the connectors, which represents the first

layer on top of the physical layer of the case-base. Connectors are objects that know how to access and retrieve cases from storage in a uniform way. jColibri provides the user with three different types of connectors to access cases represented as XML, in a database or in ontologies. The connectors are configured trough XML schema, such as specifying the database connector with the location and information needed to access the database. It is also possible for the developer to create his own connector by using the connector interface. The in-memory organization is used for organizing the cases once read into memory, such as using a simple linear list or a more advanced organization structure. jColbri currently defines only tree simple organizational structures, but provides the developer with the ability to create his own. In the same way as the connectors, the case-base also implements a common interface for the CBR methods to access the cases. This way the implementation of the methods are independent of what ways the developer chooses to organize and index the case-base.

jColbri represents a case in a very general way, a case is just an individual that can have any number of relationships with other individuals. The framework represents the cases using Java Beans, which is any case that uses *get()* and *set()* methods for their public attributes. The cases are divided into components with a restriction that they need to define an id that identifies them in the database. jColibri divides the features of a case into four case components: a case *description*, *solution* of the case, *result* of applying the solution and a *justification* for choosing the solution. This flexible way of representing the cases gives the developer the choice of which components to use in his application.

3.2 GeNIe & SMILE

GeNIe (Graphical Network Interface) is a software package for developing decision theoretic models, such as Bayesian Networks. GeNIe can be viewed as an outer shell for SMILE (Structural Modeling, Inference, and Learning Engine), providing a development environment for Bayesian Networks, while SMILE is the core reasoning component in the system. SMILE[21] is a fully portable library of C classes for implementing Bayesian networks, with wrappers provided for both Java and .NET. Dividing the software package in these to components, gives the user the ability to first create a model graphically with the GeNIe interface, and then implement and reason with it in the developers own system using the SMILE library. To facilitate sharing of research, both GeNIe and SMILE can also read and write a variety of different popular Bayesian network formats, such as e.g. the Hugin and Netica formats.

GeNIe and SMILE[1] provides a variety of inference algorithms, and the ability to learn both network structure and parameters from data. Data for this purpose can be inputted to the system trough a simple text file or a database. There exist several functions for cleaning the data, such as replacing missing values or discretization of numeric intervals, or viewing different statistic properties about the data.

If learning the structure of a network from data is required, the user is provided with different algorithms for doing so, including the ability to manually set important background knowledge for the network, such as existing or not existing relationships between variables.

If learning the parameters of a network is required, GeNIe provides the well known

[1]From this point on GeNIe will denote both GeNIe and SMILE

expectation-maximization algorithm (EM) for this task. The user is also provided with the ability to set a degree of confidence of the newly inputted data, which will affect how much the new data will influence the existing parameters.

GeNIe, in contrast to many other Bayesian Network software tools, includes a variety of probabilistic inference algorithms. These are divided into two categories, exact inference algorithms and approximation algorithms detailed further in GeNIe's user manual[2].

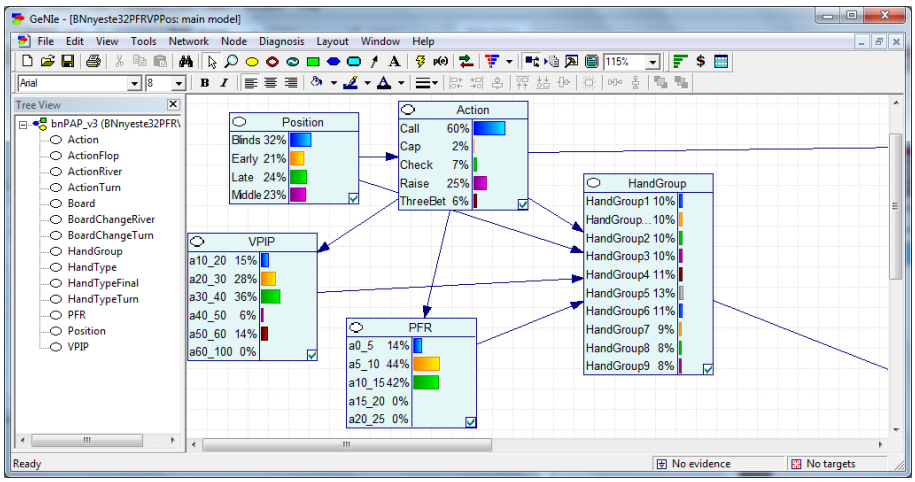


Figure 3.2: Screenshot of GeNIe

GeNIe also supports multiple node types, which can be used to extend the standard Bayesian Network capabilities into decision networks. The nodes include utility nodes, decision nodes, deterministic nodes and usual chance nodes among others. Figure 3.2 illustrates a Bayesian Network created with the GeNIe graphical interface. The nodes are viewed as bar charts, which provides a simple way of observing the effect of observed evidence on the states of the network.

3.3 Poker Academy Pro & Meerkat API

Poker Academy Pro (PAP) is a commercially available program aiming to provide a learning platform for its users. It provides a variety of features for analyzing different aspects of the game. The part of this software that makes it so attractive to AI researchers is that it includes a set of poker bots developed by the University of Alberta. These poker bots have been well explained in published research papers, which make them well suited for the purpose of testing the users own poker AI.

Meerkat is a java based API that provides users with the ability to plug in their own bots into Poker Academy Pro, by a uniform interface for performing poker actions and receiving information about the current state of the game. Another feature used by a lot of poker AI researcher is the hand evaluator provided in Meerkat. It takes a set of hole cards and community cards as input and calculates a numeric measure of how strong the current hand is compared to others, and a numeric measure for how likely the hand is to improve, the hands potential.

The hand strength evaluator calculates a numeric value in the range 0-1, based on the number of hole card combination that our current hand can beat, plus half the number of

[2]http://genie.sis.pitt.edu/wiki/Main_Page

draws, divided by the total. This can be seen as a measure of how likely it is that we beat an opponent with random cards, given the current board cards. If there is more than one opponent in the hand, the hand strength is reduced by a factor n, describing the number of players still left.

The potential of a hand can be either negative or positive, depending on if its probable that you are currently holding the best or the worse hand. If the former is the case, then the negative potential gives a numeric value of the probability of the hand not being the best after the new cards are dealt, while the latter measures the probability of improving to a winning hand.

3.4 Open Holdem

Open Holdem is an open source framework with a programmable logic engine for Texas Hold'em Poker. This framework is not used in the current implementation of the system, but is included for the sake of completeness, in that it provides an example of an alternative interfacing mechanism. Open Holdem provides the ability to test a poker bot against real human competition, by providing a way of interfacing with the poker clients from various online casinos where the game is played. It determines the game state at any given time by interpreting pixels presented on the screen by the poker client. This process of interpreting visual data from a source is often referred to as *screen scraping*.

To be able to successfully interpret the pixels at an online casino, Open Holdem needs to be provided with what is refered to as a *table map*. A table map is a text file with all the information required to interpret the current game state, such as the player's hole cards, actions performed by other players etc. This will be in the form of where on the screen the different components can be found.

Open Holdem also provides a way for implementing the actions chosen by the users logic, by clicking the buttons at the positions provided by the table map. Open Holdem can therefore be viewed as providing a poker bot with the *eyes* and *arms* required to play at a online poker casino.

CHAPTER 4

Related Research & The TLCPC framework

This chapter gives an overview of related research and describes the TLCPC framework, which has guided our approach to CBR-BN integration. The chapter begins by introducing the TLCPC framework, which is a framework for reasoning under uncertainty by combining CBR and BN. The framework is aimed at the medical domain, but will be used here to interpret and categorize related research combining CBR and BN in several domains. Section 4.2 will then describe related research in combining CBR and BN, and use this research as examples for the four architectures proposed by Bruland in the TLCPC framework[4] in section 4.2.1 through 4.2.4.

The next part of this chapter, section 4.3, begins by introducing some central AI research in the domain of Texas Hold'em Poker, before it proceeds with a more detailed description of research related to applying the two reasoning methodologies in the domain. Section 4.3.1 summarizes some relevant application of Bayesian Networks in poker, while section 4.3.2 introduces Case-based reasoning research in poker.

Each of the sections describing research related to the combination of CBR and BN, BN in poker and CBR in poker, will end with a comparison to our proposed architecture. These sections will describe how the proposed architecture relates to research concerning the combination of CBR and BN, and the application of each of them separately in the given domain. There does currently not exist any attempts to combine these two reasoning methods in the domain of poker.

4.1 The TLCPC framework

Bruland et al.[4] proposes four architectures that combine Bayesian networks and case-based reasoning for the medical domain, illustrated figure 4.1. These architectures combine CBR and BN in a sequential manner, in contrast to the parallel way of combining the two, where both methods use all the input variables and produce a classification independently. The parallel integrations rely on some kind of algorithm for selecting the better of the two classifications.

In the sequential combination of the two, one of the methods computes something that the other needs. These integrations can again be classified as either *tightly* or *loosely* coupled, where the latter describes architectures where the information used by one of the methods is hidden from the other. The variable types proposed by Bruland et al. in this problem domain is as follows:

- I_i is the input variable used by the systems. The CBR and BN parts of the system

take different inputs, but may also share some common inputs.

- A_j is a mediating variable representing concepts in the domain model, which can be set as evidence by a domain expert if one is used during the classification process, or be infered by the system.
- D is a variable that is derived by inference from domain knowledge, either as an output by the BN in the BN-CBR sequence or as a intermediate case solution in the CBR-BN sequence.
- C is the final classification variable acquired by the final system in the sequential combination of the two.

BN-CBR-1 illustrated in figure 4.1 is a loosely coupled architecture where the BN is used as a preprocessing step in the retrieve stage of the CBR-cycle. The BN includes cases in the network as binary variables with the values on/off, which decides if a case is active or not. The output from the BN results in a set of activated cases to be used by the CBR retrieval stage. Hence, the BN works as a filtering mechanism in this architecture.

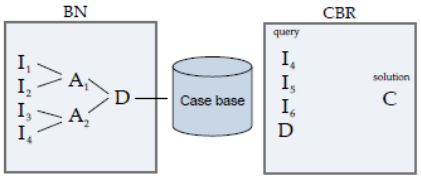


Figure 1. BN-CBR-1 Architecture

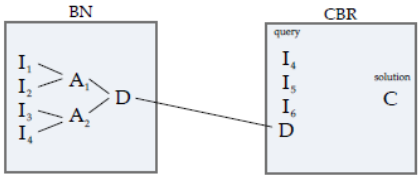


Figure 2. BN-CBR-2 Architecture

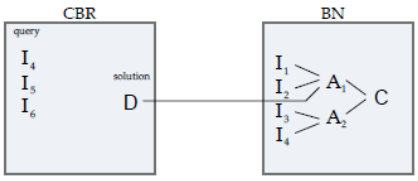


Figure 3. CBR-BN-1 Architecture

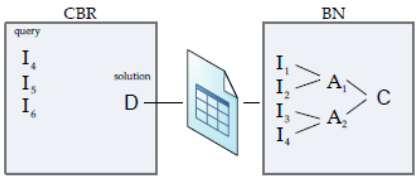


Figure 4. CBR-BN-2 Architecture

Figure 4.1: The four architectures outlined by Bruland et al. [4]

The BN-CBR-2 architecture is tightly coupled, where the resulting calculations performed by the BN are directly used as input in the case description. Viewed in a general way, the BN computes something that the CBR system needs in one of its reasoning processes.

The CBR-BN-1 architecture can be interpreted in two ways. The CBR module can work as a preprocessing stage, and use its retrieved case solutions to update part of the BN model that is unknown, or it can be used in the *reuse* phase of the CBR cycle. If the latter is the case, then the BN is used to adapt the solution of the most similar cases retrieved by the CBR system.

Similar to the first architecture, the CBR-BN-2 is loosely coupled and can be viewed as a indexing mechanism into a set of BNs. The CBR system uses its inputs to retrieve a case containing the most suitable Bayesian Network, which then calculates the final classification variable. The different BN models have common evidence and classification

nodes, but other nodes, causal links and the conditional probability tables can be different for each model.

These four architectures provide a way of analyzing the sequential combination of CBR and BN in the manners described above. Aamodt[8] proposes two general categories for interpreting the way general knowledge can be combined with case-based reasoning, through *horizontal* or *vertical* integration.

The term *horizontal* integration refers to cases where general knowledge, e.g. represented in a Baysian network is combined with CBR as an alternate problem solving method. This PSM can either be used concurrently with the CBR system on the same task to make the system more robust, or it can be used to perform some sub-task that will be used by the CBR system.

In the case of *vertical* integration, the general knowledge will be used within the CBR method itself. In a *vertically* integrated system, with a BN for representing the general knowledge, the BN would in some form be used to modify or enhance the traditional steps in the CBR-cycle to perform some interrelated task[25].

The four architectures described above mainly focuses on *horizontal* integration of the two reasoning methods, where the first system in the sequence performs some subtask that the second system relies upon. Still, the two tightly coupled architectures, BN-CBR-2 and CBR-BN-1, can illustrate both *hortizontaly* and *vertically* integrated systems, depending on how they are used. If the BN is used within one of the CBR steps, such as e.g. providing a probabilistic similarity measure in the BN-CBR-2 architecture, or used within the reuse step in the CBR-BN-1 architecture, it can be referred to as *vertically* intregrated.

Section 4.2.1 through section 4.2.4 will exemplify the four architectures illustrated in figure 4.1 with other related research.

4.2 CBR & BN

This section is divided into the four architectures proposed by Bruland in the TLCPC framework, and exemplifies these architectures through related research. This will end in a comparison of how our proposed architectures relates to the framework.

4.2.1 The BN-CBR-1 architecture

Aha and Chang[10] propose an architecture combining CBR and BN for solving multiagent planning tasks. They use robotic soccer as a test-bed for the architecture, where the BN is used to choose an action and the CBR module is used to implement the action. The Bayesian network can therefore be viewed as a preprocessing step for the CBR system, where the cases implementing the desired action are activated. Each possible action type has its own Bayesian network with sensor and action nodes, where the action with the highest probability given the current state is selected.

The cases are represented with a problem description, referred to as conditions in the Aha and Chang paper, containing information derived from its sensors. Each case also contains a solution in the form of an action sequence, and expected outcome from executing the action sequence. The Bayesian network activates the cases that contain the chosen action as its solution, and the *retrieval* phase of the CBR system then computes similarities between its current conditions and the activated cases conditions. The solution

of the most similar case is selected, but if the solution contains a non-primitive action it results in a new case *retrieval* for this action in a hierarchical fashion.

If the execution of the action sequence of the retrieved case fails, it signals the need for some kind of revision. This is performed by updating the Bayesian network with new feature nodes describing why the action sequence failed, which results in a new set of conditions as features and indices in the case-base. E.g. if an opponent intercepts the ball if a pass is being made, the Bayesian network will be updated with a node describing this threat, the CBR system will use this as a index to distinguish between different passing situations, and a new case is *retained* from this experience. The resulting state from executing a plan is also reported back to the Bayesian network to update its probabilities.

4.2.2 The BN-CBR-2 architecture

Aamodt and Langseth[9] propose an approach for integrating Bayesian Networks with knowledge intensive CBR. The Bayesian Network is combined with a semantic network, which assists the CBR system in both retrieval and reuse[4]. The combined network consists of a static part, the semantic network, which contains relationships that are assumed not to change over time and a dynamic part in the form of a Bayesian Network, which contains dependencies of a stochastic nature. The strength of the dependencies in the BN will continuously be updates as new data is observed.

Cases are represented as nodes in the network, linked into the rest of the network with its case features. Each case in the network is represented as binary variables with the values on/off, as illustrated in the BN-CBR-1 architecture. The bayesian case retrieval is then performed by entering the evidence observed in a new case into the network, and then calculating the conditional probability of a case beeing on, given the observed evidence. This works as the similarity metric in the system, and results in the retrieval of every case exceeding some threshold value.

The combined network is intended to be used in several steps in the CBR reasoning process, such as guiding case adaption in the reuse step, by calculating causal relations in the network[4]. It can also be used in several of the sub-tasks in the *retrieve* step, e.g. the steps from the explanation engine proposed by Aamodt[8], which would represent a vertical integration of the two reasoning methods.

Vadera et al. [26] proposed a probabilistic exemplar-based model for Case-based reasoning. The model utilizes two Bayesian networks, one for indexing and one for identifying exemplars within categories. The case-base is structured in a hierarchical fashion, where top level constitutes what is referred to as categories, which again is represented as a set of exemplars. These exemplars represent a set of similar cases indexed by their features. The cases themselves are not stored in this model, but instead contained through a summary representation in the exemplars.

The first Bayesian network in this model is used as an indexing structure to categories. The probability of the category a new case belongs to is calculated based on the features the new case and the categories have in common. The second BN defines the relationship between the features and the exemplars. The frequency a feature has been observed amongst the cases the exemplar has previously been used to classify is used to calculate the probability of the various exemplars based on the features of the new case. The cases themselves are not stored in this architecture, instead an exemplar encompasses a summary representation of the cases, containing the features and how many times the features

has been observed among the classified cases.

Learning in the model is incrementally conducted each time a new case is classified. This process involves adding the new case as an exemplar, or replacing the exemplar that was used to classify the case with the classified case. In order to decide if the new case is a better exemplar than the old one, the model uses the concept of prototypicality. A prototypical exemplar will be one that best represents its similar cases and has least similarity with the cases represented by other exemplars. As before, the similarity is calculated based upon the features the two have in common. The model described here illustrates a vertical integration of CBR and BN. One of the models prominent restrictions is that it requires that all features are binary.

4.2.3 The CBR-BN-1 architecture

Tran and Schönwalder[27] present a distributed Case-based reasoning system applied in the communication system fault domain. Inspired by the reasoning of experts in the medical domain, they propose extending the Case-based system with a Bayesian reasoning component to facilitate both case *retrieval* and *reuse*. Each case contains a set of symptoms and a fault hypothesis. Earlier research focusing on applying probabilistic reasoning to this part of the CBR system can broadly be placed into two categories; The ones that create a probabilistic model based on the entire case-base[26], and the ones that rely on a chosen part of the case-base to perform the probabilistic reasoning on.

This system uses the cases selected in the retrieve phase to build the probabilistic model, and therefore falls in the latter category. The first step focuses on selecting a ranked list of cases obtained by using the k-Nearest-Neighbor algorithm with a similarity measure considering the symptoms the two cases have in common, and the significance of the symptoms in each of the cases. This set of cases is then used to build a Bayesian network by calculating probabilities of observing a hypothesis given a set of symptoms in the retrived cases. This BN is then used for proposing the most probable hypothesis in the reuse phase of the CBR cycle, by calculating a posterior probability over the available hypothesis given the symptoms observed in the problem description.

4.2.4 The CBR-BN-2 architecture

Pavon et al.[28] propose a case-base reasoning system that uses a Bayesian framework for algorithm parameter tuning. Each case consits of a problem description and a solution, where the description is a collection of features describing the problem domain and the solution is an induced Bayesian Network. The BN solution consists of the possible algorithm parameters as well as a set of performance parameters, and is used to estimate the best parameter configuration for the particular problem instance at hand. The probabilities in the BN is collected from multiple runs with different parameters of the algorithm in the given problem domain.

The *retrieval* phase works in a straight forward manner, and can be viewed as an indexing mechanism for finding the most suitable Bayesian Networks for solving the task at hand. If several suitable cases are retrieved, a reliability measure based on number of runs of the case is used to select the best one.

In the *reuse* step the BN is used to calculate the best parameter configuration by entering the variables for measuring the performance of the algorithm as evidence, and calcu-

lating the posterior probability over the different parameter states. If the BN selected from the case retrieval phase has a low reliability, then the BN can be used to suggest multiple parameter configurations. This is done because a low reliability measure means that the BN is based on little data and may be very unreliable.

The *review* step is just the results of executing the algorithm with the parameter configurations proposed in the *reuse* stage, which is stored in a database in the system. When a sufficient number of runs is available in the database a Bayesian Network is induced and *retained* together with its problem description as a case in the case-base.

4.2.5 Comparison

The different ways of integrating CBR and BN outlined in this section was used as a starting point when designing the architecture proposed in chapter 5. The most focus was given the two tightly coupled architectures, CBR-BN-2 and BN-CBR-1, because they were believed to better combine the strengths of both reasoning methodologies in the domain. Through an analytical approach to related research in the domain we got the idea of splitting the reasoning process into two tasks that we believed would play on the strengths of both reasoning methodologies. The focus of the first task was modeling an opponent's playing behavior to predict what kind of cards the player is holding, while the second task would use this information combined with other important features in the domain to make the best possible betting decision from the information available at the time. This reasoning process also mimics good poker players, where the more complex task of reasoning about the opponents likely cards is a major factor when making a betting decision.

The representation of the features that was considered important when making the final betting decision was another factor influencing the design of the architecture. Most of the features that were considered important for making a betting decision were numeric. This is not optimal when considering the BN to be the final decision maker in the sequential execution of the two tasks, because most of the work in the domain of Bayesian Networks rely upon discrete variables, and discretizising these variables was considered less than optimal.

Since modeling and predicting opponents cards and behavior involves alot of uncertainty, this task was considered to play on the strengths of Bayesian networks, which are excellent for reasoning under uncertainty through casual relationships and probabilistic inference.

Based on the reasons outlined here, the architecture proposed in chapter 5 is a specialization of the general architectural design referred to as BN-CBR-2. The reasoning process is divided into two tasks, where the BN is used to infer important information from the opponent modeling task, while the CBR is used to combine this information with other important features in the domain to make the final betting decision. The CBR component takes the solution from the sub-task solved by the BN as input into the case description when retrieving similar cases. This architecture was also believed to represent ways in which poker player reason, by using a set of important features together with a prediction of the opponent's likely cards when consulting past experiences to find a good betting decision. The proposed architecture is a horizontal integration of the CBR and BN, where the two components are tightly coupled and used in a sequential order.

4.3 Poker & AI

There have been various attempts to apply a broad range of AI methodologies to the domain of poker. Figure 4.2 summarizes several reasons why poker may be viewed as a more attractive testbed for AI than deterministic perfect information games such as chess and checkers. Poker is *non-deterministic*, actions can never guarantee the same outcome, and suffers from imperfect information in the form of other players hidden cards.

General Application Problem	*Problem Realization in Poker*
imperfect knowledge	opponents' hands are hidden
multiple competing agents	many competing players
risk management	betting strategies and their consequences
agent modeling	identifying patterns in opponent's play and exploiting them
deception	bluffing and varying style of play
unreliable information	taking into account your opponents' deceptive plays

Figure 4.2: Characterization of the poker domain for AI research [5]

Over the years there have been several attempts to applying AI in simplified versions of poker, sub-sets of the game, and the full variant. In the recent years the University of Alberta has evolved as the leading force in AI poker research with their *Poki* system, which will be given a brief overview in this section.

Poki uses a combination of rules, statistics and simulations to play the game of poker. At the core of the system lies the simulation-based betting strategy, which can be viewed as an analogy to selective game-tree searching, where certain nodes are expanded with a higher probability, rather than expanding all nodes with an equal probability. How this is performed effectively will be explained further after Poki's main components have been outlined.

The first betting round where Poki has little information available other than its own holdings, it uses a rule-based approach to decide what cards to play and in what manner to play them. To do this it contains a ranked list of the different hand's expected income rate. This income rate is acquired by an offline simulation of a large number of hands, where the specific hole cards are simulated against virtual minimum betting opponent's with randomized cards.

In the later stages of the game, Poki constructs an opponent model of each of the players currently in the game. This model contains a weighted table of each possible hole cards the opponent can hold and a probability triplet for each of these entries. The table is weighted based on the opponents actions, and the probability triplet containing probabilities for folding, betting and raising is calculated based on a set of rules that Poki itself uses to decide what actions to take. Among the important factors that Poki uses to construct this triplet is a numeric measure for hand strength and the positive potential of the hand. The probability triplet for Poki's current situation, and the probability triplets for the opponents most likely holdings are then used to run a number of simulations for the rest of the game to estimate an expected value for the different actions Poki can make.

There are two versions of the Poki system included in Poker Academy Pro. The two versions differ in the strategy used for making the final betting decision. One of the versions uses a formula-based approach and the other uses a simulation-based approach for its betting strategy. The formula-based approach uses rules defined by an expert to

make the final betting decisions while the simulation-based approach uses the simulation strategy described above. The former was nicknamed *Pokibot*, while the latter goes under the name of *Simbot*.

4.3.1 Bayesian Poker

There have been several attempts to apply Bayesian Reasoning to various aspects of the poker domain. Baker and Cowling use a simplified version of poker to investigate the effect of bayesian opponent modeling[29], while Terry and Mihok[30] use a Bayesian network to predict opponents hole cards. Korb et al.[31] developed a Bayesian Network for playing 5-card stud poker and later continued their BN work in *heads-up* (two-player) *limit* Texas Hold'em[32]. Other attempts of applying BN for this particular poker variant include [33] and [34]. A common factor that all of the mentioned articles share, is that they all include opponent modeling in the Bayesian Network.

The subsequent sections describe the two attempts of applying BN in poker that are most relevant, and concludes with comparison to the BN component in the architecture proposed in this thesis.

Bayesian Network for playing poker

Korb et al.[31] developed a Bayesian Poker Program (BPP) which uses a Bayesian Network to model the program's poker hand, the opponent's hand and playing behaviour to determine its probability of winning. BPP started out by playing five-card stud poker, but was later improved and adapted to the domain of limit heads-up Texas Hold'em [32]. One of these improvements was extending the BN with Decision Network capabilities, to make it able to explicitly represent decisions and utilities of these decisions. The initial version of BPP used a separate network for each betting round. Another important change was adding the ability to not only predict when the opponent has a flush, but also when he possesses a flush draw. This is a important factor in Texas Hold'em compared to five-card stud, because of the increased probability of flushes appearing.

BPP attempts to model the opponents behavior by mapping a situation in the game to a probability distribution over a set of actions represented by the *OPP_Action* node. This action node has eight possible values; *fold*, *call*, *check*, *raise*, *bet*, *paysmallblind*, *paybigblind* and *pass*, which represent a finer granularity than the earlier three categories. The opponent action node is influenced by the observed board, round, pot, BPP's action and the opponent's own cards. The *round* node is used to combine the previously four separate networks used for each betting round into one, which provides continuity between the various phases of the hand.

The probability distribution of the opponent's actions is then used to calculate probabilities for the opponent's possible hands, and is together with the observable node for representing BPP's current hand used to predict both the opponent's and BPP's final hands. These final hand predictions are used to calculate a probability of winning, which is combined with a utility node to represent the expected utility of the possible next actions for BPP.

Since being predictable makes for a lousy poker player, BPP uses betting curves to add some randomization to the choice of actions. The betting curves gives a probability to two actions based on their expected utility, which means that if two actions have equal expected utility, they are both selected with a probability of 50%.

To reduce the complexity of the BN the large number of different possible hand combinations are bucketed into 25 different categories. This introduces some inaccuracy into the model, and Korb et. al[32] illustrated this by playing it against a combinatorial model, which makes its decision by considering 100.000 random decks of cards from any given position in the game. This resulted in a slight loss for the BN, which can be directly credited to hand abstraction, as this is the only difference between the two.

Another obstacle with this BN is the large amount of entries in the conditional probability table of the node used for modeling the opponents behaviour. This makes it very hard to acquire significant information throughtout a single game to update it to better represent a specific opponent. Korb et al.korbBnHoldem suggest using either a static opponent model which models a typical opponent and does not change throughout a game, or a dynamic opponent model. To effectively be able to perform dynamic opponent modeling, the previously mentioned CPT had to be reduced in size. This was accomplished by segmenting the CPT into groups where the opponent may behave equally.

Bayesian Network for predicting opponents cards

Terry and Mihok[30] combined expert knowledge and machine learning of a large collection of previously played hands to create a Bayesian network for predicting opponent hole cards. The nodes and links in the network were defined through expert knowledge, while the parameters in the network were learned statistically from a large collection of parsed hand histories. The network relies upon hand histories of rounds that went all the way to showdown, because the opponent's cards are only visible at showdown. Terry and Mihok point out that a critical aspect of the Bayesian network's ability to predict opponent's hole cards is the way the game state is represented in the network. To fully utilize the information from the hand histories, the state of the game may require a more specific representation than the ones used in the hand history. An example of this is when using only the regular poker actions to represent the opponent's actions the network misses a lot of crucial information. E.g. if a raise is beeing made in a early position at the table with a lot of players acting after that player, this should be considered an action of more strength than a raise beeing made in a late position where almost everyone else has folded.

The final BN proposed by Terry and Mihok uses pot odds, position, number of players and the opponent's action to influence nodes representing the opponent's probable hand strength and hand potential, which again is used to infer the opponent's most likely hole cards. Most of the nodes in the network are discretizised numeric intervals, excluding the node for representing the opponents hole cards, which has a state for each of the possible hole cards combinations.

Due to time limitations, some of the proposed ideas where not implemented in the final BN design. The final BN uses only one action, in contrast to using a full sequence of actions to predict the opponent's hole cards, and lacks the specialization of the action node into more specific action types.

BN Comparison

The Bayesian network proposed in this thesis is inspired by the research described in this section, although there are many differences the basic ideas behind the research are still similar. All of the attempts of applying BN to poker described here include some form of predicting the opponents cards, which is the main task of the BN proposed in this thesis.

The proposed BN share some common nodes with the Bayesian poker player, such as a node for representing the opponents action, a node for describing the board, and the separation of the opponents hand in a current and a final hand type. Although the work done by Korb et al. is in a similar domain, their focus was on using the BN to play Texas Hold'em in a two player environment, which describes a lot of the differences to our proposed network. An example of this is the states in the opponents hand type nodes, which are focused on a two player environment rather than the states used in our BN to achieve better performance in an environment with more players. Korb et. al's BN also lack a node for describing position, which is more important in an environment with more players.

Terry and Mihok highlighted the need of further specializing the regular poker actions into states that describe what kind of situations the actions where used. The BN proposed in this thesis takes this idea further, and implements a broader range of possible actions than the regular poker actions, based on the actions described in section 2.3.7.

In contrast to the research proposed here, our proposed BN has more focus on using a sequence of actions and other information at the different stages of the game to predict the opponent's cards. Korb et. al uses a node for representing what stage the game is currently in, to provide some continuity between the rounds. The BN described in section 5.3 is modeled with greater focus on the change of the game environment at the different stages of the game, both in the sequence of actions from an opponent as well as the change of the community cards. Observing the opponent's betting behavior in relation to the changes in the community cards is believed to be a good way of inferring information about the opponent's cards.

4.3.2 Case-Based Poker

There have been relatively few attempts to apply CBR to the game of poker. CASEY [35] and CASPER [36] are Case-Based systems for making betting decisions at table with multiple players, while SARTRE[37] is intended for heads-up play. Salim and Rohwer[38] applied CBR solely to the purpose of modeling opponents betting behavior to attempt to determine their hand strength. Compared to the Bayesian based poker research most of the CBR research in this field doesn't include any form of opponent modeling.

The following sections will describe the attempts of applying CBR in poker that are relevant for this research.

CASEY

CASEY[1] [35] is a Case-based learner applied in the domain of multiplayer limit Texas Hold'em. CASEY divides its case-base into a preflop and a postflop base, and each case contains an indexed contextual part, a solution and an outcome of applying the solution. The contextual part of the case contains features such as hand strength, relative position, bets to call and potential of the hand, where the latter is only used for postflop play. The postflop cases also contain a stage feature for describing in which stage of the round the case was recorded. The solution offered by a case is a strategy, which consists of an initial action and a follow up response if applicable. One such strategy could be to check

[1]CASEY seems to be a beloved name in the CBR community. This CASEY should not be confused with Koton's and Aha's CASEY.

and then raise if the opponent bets, referred to as a check-raise in poker and is usually a sign of strength. The outcome of a case consists of features describing the risk/reward on each separate stage of the round, and a net result for the whole round. These features are numeric measures of how much was won in the given case.

CASEY's retrieval phase consists of first retrieving the most similar cases, and then selecting up to a maximum of 100 of the retrieved cases above a specific similarity treshold. The selected cases are then used to create a combined summary, which consists of a sum of the outcomes of applying a strategy in a given situation. CASEY then reuses the strategy with the most successful outcome.

Sandven and Tessem tested CASEY by playing against instances of *RuleBot*, which is a rule-based poker bot included in the commercially available Poker Academy Pro software. CASEY begins with an empty case-base and plays every hand with a random selected action until it has acquired an adequate number of cases. Sandven and Tessem report that after the initial case aquisition is complete, CASEY plays on par with RuleBot.

CASPER

The CASPER system by Watson et. al[36], is an attempt to improve on the results of Sandven and Tessem's[35] application of CBR for making betting decisions in limit Texas Hold'em. When it is CASPER's turn to act, it creates a target case with similar features as the CASEY system, and uses k-nearest neighbor algorithm to search the case-base for relevant cases. In contrast to CASEY, the CASPER system separates its case-base into four different case-bases, one for each stage of the round. CASPER does not store any outcome of applying a specific solution, and instead of using more complex strategies as case solutions, it relies only on the atomic actions allowed in poker. For example if CASPER gets re-raised after the first CBR cycle made it bet, it will perform a new cycle for deciding the next action to take.

When CASPER is required to make a betting decision, its retrieval phase computes the similarity of the cases in the case-base, and selects to top 20 cases with a similarity over a threshold of 97%. It then computes a probability triplet by summing up the betting decisions suggested by these cases, and dividing them by all the selected cases. This probability triplet is then used to generate a betting decision. The similarity is computed by a weighted linear combination of the local similarity where features like hand strength and hand potential are given higher weights. Watson et. al experimented with both handpicked weights and weights derived by an evolutionary algorithm.

CASPER was like CASEY implemented with the Meerkat API to interface with Poker Academy Pro. Instances of Pokibot and Simbot where used to generate CASPER's training data by playing up to 13.000 rounds where CASPER retained every context and betting decision made into its case-base. CASPER was tested against the adaptive bots, in the form of Pokibot and Simbot, where it retained its case-base, and was shown to play evenly against these. Watson et. al also tested CASPER's performance against the non adaptive bots included in Poker Academy Pro, and by playing real people for play money. CASPER was shown profitably in both these settings.

SARTRE

The SARTRE system by Rubin and Watson[37] uses case-based reasoning for playing two-player limit Texas Hold'em. The SARTRE system represents a move from quan-

titative to more qualitative features for representing a case, compared to their previous research on CASPER. SARTRE uses three indexed features to determine the solution of a case. These features represent; the previous betting for the current round, the strength of SARTRE's current hand and information about the state of the community cards.

The previous betting in the round feature represents a sequence of actions made up to the given point of time. E.g. if SARTRE raises (r), the opponent re-raises and SARTRE chooses to call (c), this would be represented as *rrc*. SARTRE is designed for two-player poker, which means that the size of these betting patterns will be drastically reduced compared to a table with ten players. These betting patterns are represented by what they refer to as a betting tree, which is a tree structure representing each action in a hierarchical fashion. The similarity for this feature is calculated by comparing the betting path in the tree. If two betting paths are equal they will receive a similarity of 1.0.

The feature SARTRE uses to represent the strength of his own hand is a categorical feature, which at the preflop stage maps SARTRE's hole cards into a category, while using the best five card combination of the hole cards and the community cards at the flop stage. A simple rule-based reasoner is used to map the cards into the correct category. The categories represent the standard poker hand types, such as one pair, two pair, three of a kind, but also hands with alot of potential to improve, such as straight and flush draws (explained in section 5.3.3).

The final indexed feature, describing the community cards, is also represented through categories. These categories attempt to mimic the information a human player would extract from the current community cards. For example would the category *Is-Straight-Possible* and *Is-Straight-Highly-Possible* be used to represents community cards where there are three and four consecutive card values showing. Both of the categorical features were compared with a similarity measure assigning 1.0 to exact matches and 0.0 otherwise. SARTRE's case-base was generated by analyzing the game logs of previous AAAI Computer Poker Competitions, and uses approximately 250,000 cases for each stage of the game.

SARTRE presents the solution of a case as an action triplet, representing a probability distribution over the betting decisions the system should select in the given situation. The outcome of a case is presented in the same manner, with a triplet describing the average quantitative loss or profit that has been observed in the past given the three betting decisions. Rubin and Watson also experimented with different re-use policies, but concluded that the most profitable one was the majority-voting re-use policy, which selects the solution that is present amongst the majority of the retrieved cases.

CBR Comparison

Both CASPER and CASEY rely on similiar indexed features for descibing the state of the game at a particular time. These features are recognized as important for making betting decisions by poker experts, and also made more available for AI research through the Meerkat library developed by the UoA. The CBR part of the architecture proposed in this thesis will also base its reasoning on most of these features. The SARTRE system is intended for two-player Texas Hold'em and is therefore the most different to our proposed architecture. Although SARTRE shares few commonalities with the CBR component proposed in this thesis, some of its ideas are similar to the information represented in our proposed BN.

The case-based reasoner in this thesis incorporates ideas from all of the research described in this section. Both CASPER and CASEY relie on similiar indexed features for descibing the state of the game at a particular time. These features are recognized as important for making betting decisions by poker experts, and also made more available for AI research through the Meerkat library developed by the UoA. The CBR system proposed in chapter 5 has alot of similarities in the features used for representing the cases in the system. The retrieval phase, including local similarity measures and weighting of the various features in the case are heavily inspired by the research done with the CASPER system.

The proposed case-based reasoner can be viewed as a integration of of what is believed to be the strengths of each of the CASEY and CASPER systems. It shares the idea of separating between cases at the different stages of the game, and storing them into separate case-bases, although CASEY uses two, while CASPER uses four case-bases, one for each stage. CASPER's reuse policy is probabilistic, with a higher percentage of selecting the action represented in most of the retrieved similar cases, while CASEY attempts to use the outcome of a solution to choose the best solution. Both CASEY and SARTRE report problems with using outcome to select an action, because of the luck element involved in poker.

Our CBR part of this system attempts to overcome the problems introduced through the luck element when using outcome to choose actions, because choosing the most profitable action is the way good poker players decide on what actions to take. Our CBR component attempts to overcome this problem by introducing domain knowledge in the revise step of the CBR cycle, to prohibit cases representing bad solutions getting retained with a positive outcome due to luck, or the other way around. This is discussed further in section 5.4.5.

The architecture proposed in this thesis also distinguishes itself from related research of applying CBR to poker by including some form of opponent modeling. This is introduced by the Bayesian network in this architecture, which involves some new features in the case representation, and a better estimator of features involving representing the strength and potential improvment of a set of cards.

Part II

Results

CHAPTER 5

Design

This chapter describes the design of our proposed architecture. The three sections describing the comparison of how this architecture relates to other research in; combining CBR and BN, BN in poker and CBR in poker describes the foundation on which this architecture was built.

Section 5.1 describes the domain and the general structure and interaction in the architecture. The subsequent three sections describe each of the three components in the architecture.

5.1 System overview

This section gives an overview of our proposed architecture illustrated in figure 5.1. The purpose of this section is to illustrate the interaction between the various components in the architecture without going into to much detail. The following sections (5.2 through 5.4) will then give a detailed description on how each of these components contribute to the overall architecture outlined here. The system developed through this research was nicknamed BayCaRP (Bayesian Case-based Reasoner for Poker).

The domain chosen for this project is Texas Hold'em Poker with fixed-limit betting. Fixed-limit betting is easier for an automated poker bot than no-limit because there is a fixed betting amount, which means that the bot doesn't have to focus on the right amount of money to bet at a certain situation. Fixed-limit is also easier because all-in bets are not allowed, which prevents the bot from losing all its money in one single action. Fixed-limit also makes bluffing harder because it isn't possible to throw a large amount of money into the pot to scare opponents into folding. These reasons make the fixed-limit betting more attractive as a test domain for this project than no-limit, because the restriction on betting size simplifies some parts of the architecture. The architecture can be extended for no-limit play, but that would require more work on extending the architecture for handling various sized bets.

The architecture consists of two well known reasoning methods, namely Case-based reasoning (CBR) and Bayesian Networks (BN), and an interface component for extracting information from the game environment. In addition, the architecture also contains a component containing rules for transforming information extracted by the interface into states used by the BN and functions used to calculate various features for the CBR component. This component will for the rest of this research be referred to as the Rule-based reasoner (RBR).

The domain knowledge required by the system is achieved through the BN and the transformation rules and functions in the RBR component. Figure 5.1 illustrates the interaction between these components. The variables used to describe the architecture are based on the framework proposed by Bruland[4] and have the following meaning in our architecture:

- I_i is an input variable used by the system. The inputs are extracted from the poker client through the interface component as illustrated in figure 5.1.
- A_j is the result of applying some transformation to an input variable in a way that is required by the recieving component. This architecture performs the transformation by applying RBR to determine what the given input corresponds to in the two other components.
- D is a variable derived by the Bayesian network. In this architecture it represents the solution to a sub-task inferred by the Bayesian network, which is required by the case-based reasoner. The variable marked $D*$ in figure 5.1 refers to situations where the variable derived from the BN is used for some kind of calculations before its used as input to the CBR component.
- C is the final output variable acquired by the execution of the CBR system. In this architecture the CBR system is the master, which uses the inputs from the two other components in the process of achieving the most suitable final output of the system.

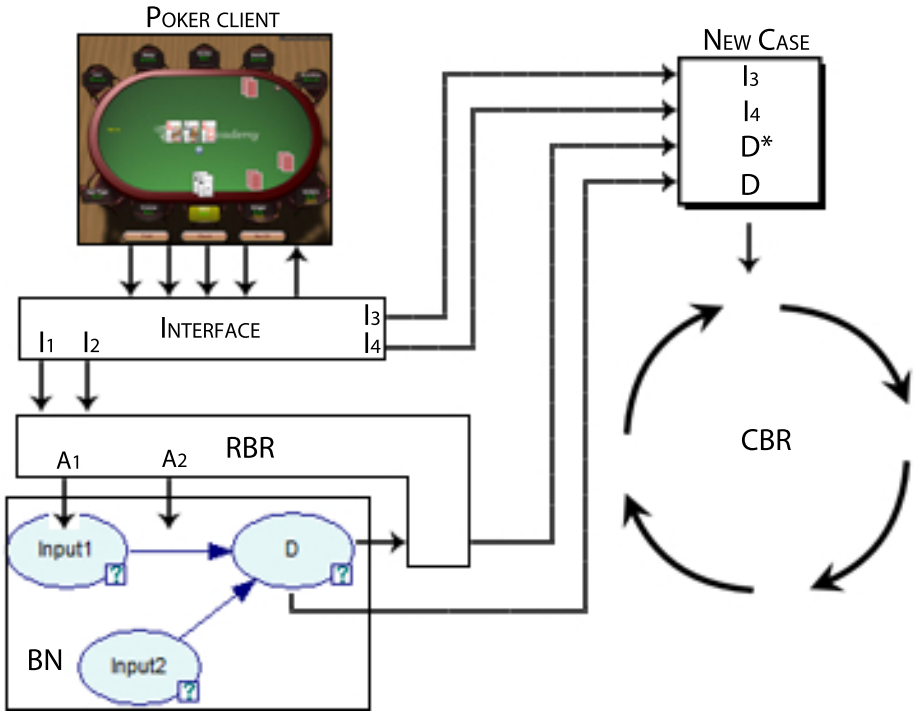


Figure 5.1: Overall system architecture

The interface component has a sole purpose of acting as a middle layer between the poker client, where the game is being played, and the rest of the system. Playing poker

at different locations requires different software, poker clients, and therefore also different interfacing mechanisms. The interfacing mechanism used in this research, described in section 3.3, will be further detailed in the implementation chapter 6. The interface component extracts information required by our system from the poker software, and executes the actions proposed by our system when acting as a fully automated poker playing software.

The rule-based reasoner acts as a middle layer between the interface and the rest of the system, but also between the BN and the CBR system itself. It takes some input from one component and applies rules to it, to determine the correct output that should be given to the receiving component. When used between the interface and the BN, it is mainly used to simplify the information extracted from the game environment into categories used by the Bayesian network. As illustrated in figure 5.1 the RBR component also takes a variable derived from the BN and processes it in a way that is required by the CBR system.

The Bayesian network in this architecture is mainly concerned with a sub-task of the reasoning process required by a good poker player. The focus of this sub-task is to infer information about the opponent's based on the observable state of the game, such as the opponents actions and the *community cards*. More specifically, the BN can be used to predict the current cards the opponent is holding and future actions the opponent is likely to take. This prediction, constituting the D or $D*$ variable described above, is then used either directly as a input by the CBR system, or indirectly through the RBR component.

The case-based reasoner is the main component in this system. It uses the input from all the other components to make an informed betting decision, the final output variable C. The case-based reasoner may also use the RBR and BN component in other parts of its reasoning process. Due to the fact that poker involves an element of luck, a person can make bad poker decisions and still win short-term. To prohibit the case-base of filling up with cases representing bad poker decisions that still would represent as a positive case in the case-base, some form of domain knowledge should be applied in the revise and retain step of the CBR cycle. Rounds that have gone to showdown can be analyzed through domain knowledge from a hindsight perspective, to determine if the success or failure of the applied solutions was due to luck or skill. The hindsight perspective of when a hand has continued to showdown involves knowledge about the opponents hand at any state of the game, and can therefore be used in some degree to determine the skill or luck involved. In this architecture we suggest an approach that overcomes the limitation of perfect knowledge obtained when the players go to showdown to assess the realistic outcome of an action. The roles of the two main reasoning methodologies are summarized below.

The BN

1. The Bayesian network uses information observed by the opponent's playing behavior relating to the state of the game to predict the opponent's cards. This involves predicting the opponent's hand group and hand type which is explained further in section 5.3.

The CBR

1. The case-based reasoner uses the prediction from the BN combined with other features describing the state of the game to make the final betting decision which is applied to the game environment. The case-based reasoner is described in further detail in section 5.4.

5.2 Rules & Functions

The rule-based reasoner contains its domain knowledge through a set of defined rules and functions, which are used to calculate or transform the input it receives into useful information that the other components need. The rule-based reasoner is also used to analyze the outcome of an applied solution in the revise step of the CBR component. Figure 5.2 illustrates the former function, where the RBR component is used between the other components.

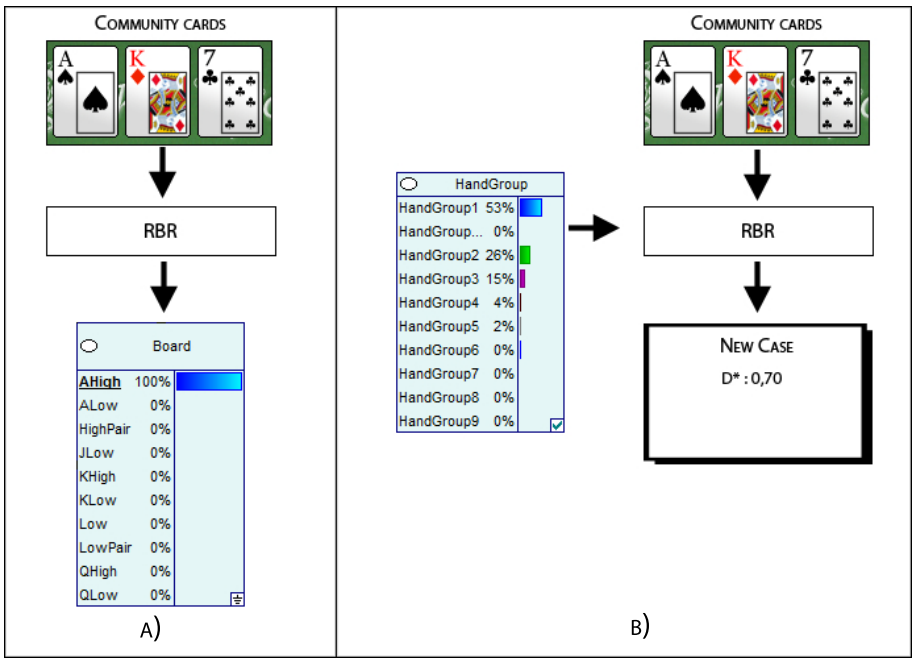


Figure 5.2: RBR interacting with: (a) The BN (b) Between the BN and CBR

Figure 5.2(a) shows an example of how rule-based reasoning is used to turn some input from the interface into the correct state used by the Bayesian network. The Bayesian network uses the RBR component to a large extent, since none of the states in the network can be directly inputted from the interface. The RBR is among other things used to classify a sequence of actions, or a action under a specific circumstance as one of the actions described in section 2.3.7. Figure 5.2(a) illustrates how the RBR component is used to simplify or categorize the community cards into the correct state of the board node.

Figure 5.2(b) shows how the RBR takes part of the information derived from the BN and computes something that the CBR system needs. This example takes the opponents predicted *hand groups*, a grouping of possible hole cards the opponent may hold together with the community cards as input to the RBR component. The RBR then applies a

function to compute a numeric measure representing the players *relative hand strength* (described further in section 5.4.1) relative to the opponents predicted hand groups.

5.3 The Bayesian Network

5.3.1 General

The Bayesian network is the second source of general domain knowledge in the system. It models the variables and relationships between these variables with the goal of inferring the opponents *hand type* through observable information such as the opponent's actions. The nodes and states in the BN are used to represent information that is considered important for inferring an opponent's hand type. The choice of what variables the network should represent was modeled through expert knowledge to mimic the knowledge that poker players use to infer information about an opponent's most likely cards.

The topology of the network was also defined through expert knowledge, although there exist several ways to learn the structure of the network automatically from data, so this would also have been an option. Still, after the variables of interest were defined, the dependencies between these were pretty straight forward. The dependencies in the network represent how one variable affects our belief of another. For example observing an action from an opponent changes our belief of what cards he might have. The node representing the opponent's action will therefore be a parent node for the one that is used to represent the opponent's cards.

The parameters of the network were learned automatically from data, by observing the bots in Poker Academy Pro and extracting the variables of interest. The data used to train the network consists exclusively of poker rounds that go all the way to showdown. The process of collecting this data is explained in further detail in section 6.1.1. The next section describes the structure and nodes of the BN.

5.3.2 Network structure & nodes

For simplicity, the BN can be viewed as consisting of four parts, each part representing the state of the game at a current stage. This is done to achieve some continuity between the four stages of the game, and to use as much information as possible to predict the opponent's hand. When modeled in this way the whole available action sequence an opponent has performed will influence the hand type prediction. Representing the opponent's hand at any stage of the game is also for easy integration with the CBR system.

The preflop stage is modeled through what is called the *PFR*, *VP*, *Position*, *Action* and *HandGroup* nodes, which are explained in more detail below. As shown in figure 5.3 each of the four nodes influence the *HandGroup* node, which represent well known information for inferring something about an opponent's likely holdings in the *preflop* stage.

PFR is an acronym for preflop raises and is a numeric measure for describing how many percent of the times a player performs a raise in the preflop stage. VPIP is equally an acronym for voluntarily put money in pot and is used to describe the percentage of times the player performs a call or raise. Both of these measures are used to get some idea of how many cards the opponent plays, and how he plays them. For example if a player has a PFR ratio of 10%, we can expect him to have one of the 10% best possible

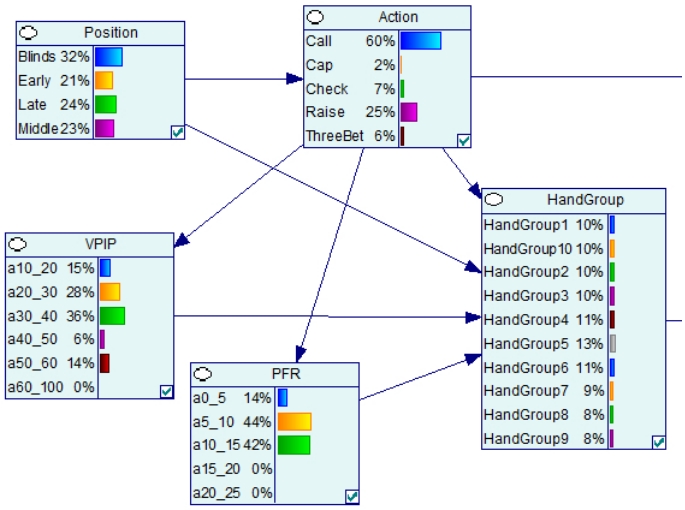


Figure 5.3: The preflop part of the BN

hole cards when he performs a raise. The states of these nodes in the network represent a discretization of these intervals, which means that the state *a5_10* in the *PFR* node represents a player that raises 5–10% preflop.

The *Position* node describes where on the table the acting player sits in relation to the dealer. Position is an important factor for deciding how to play a certain set of cards, because in a favorable position you get to observe what many players do before you. E.g. if you're in a late position and everyone folds you can play a broader range of cards than you would in an early position, because in the early position there is a greater chance that someone raises the pot after you have acted. For these reasons position is an important factor for predicting an opponent's hole cards. Poker terminology has different terms for referring to each position at the table, while the Bayesian network illustrated here categorizes these positions into four states. Each of these four states represent positions where players are most likely to play their cards in a similar fashion.

The action a player performs in a specific situation is one of the most important factors for predicting an opponent's cards. This is because when a player performs an action, he chooses the action that he thinks is ideal in relation to the information he has available. Looked at in this way, we can view an opponent's action as a window into what he believes about his current cards. For this reason the action nodes in the BN represent actions on a more specific level than the regular poker actions. These actions, which constitute the states in the action nodes, where described in section 2.3.7 and represent actions being used in specific situations. Revisiting the earlier thought about an action being a window into the opponent's belief, we can infer more information from this action when the action states are more specific. E.g when the player *caps* the betting preflop, we can expect him to hold one of the very high ranking hole cards. If we would simply represent this as a raise in the BN, we could not make this assumption. Another example would be the variations of the simple bet action. When a player performs a *donkbet*, he might have a decent hand at the current stage, but believes it is likely to be beat at a later stage. So in fear of allowing the other players to check, and observe a new card for free, he bets and forces them to pay to continue playing. If this was represented as a bet in the BN, we could not distinguish this kind of behavior from another type of bet. The BN uses the

most aggressive action performed by the player in the particular round. This means that if a player performs a threebet, and then calls the cap action of another player, this would be represented as a *ThreeBet*.

The *HandGroup* node is an extension of the eight hand groupings proposed by Sklansky and Malmuth's[39]. The idea behind these groups of hands, is that each set of cards in one of these groups are normally played similarly. Assuming this is correct, we can infer which group of hands the opponent is most likely holding by observing his style of play. This is modeled through the previously described nodes that influence the *HandGroup* node. The extension to Sklansky and Malmuth's hand groupings is an additonal group of hands, based on personal experience. The state referring to the tenth hand group is just a collection of all the other hands not represented by the other hand groups.

The stages after the preflop, referred to as postflop, are modeled through an action node, a hand type node and a node for representing the community cards(the board). The postflop part of the BN is illustrated in figure 5.6. The four action nodes in the network represent the four stages of the game, where each action node is dependent on the previous action taken. The preflop action node distinguishes itself from the others with fewer possible states, because some of the states are only possible in the postflop stages.

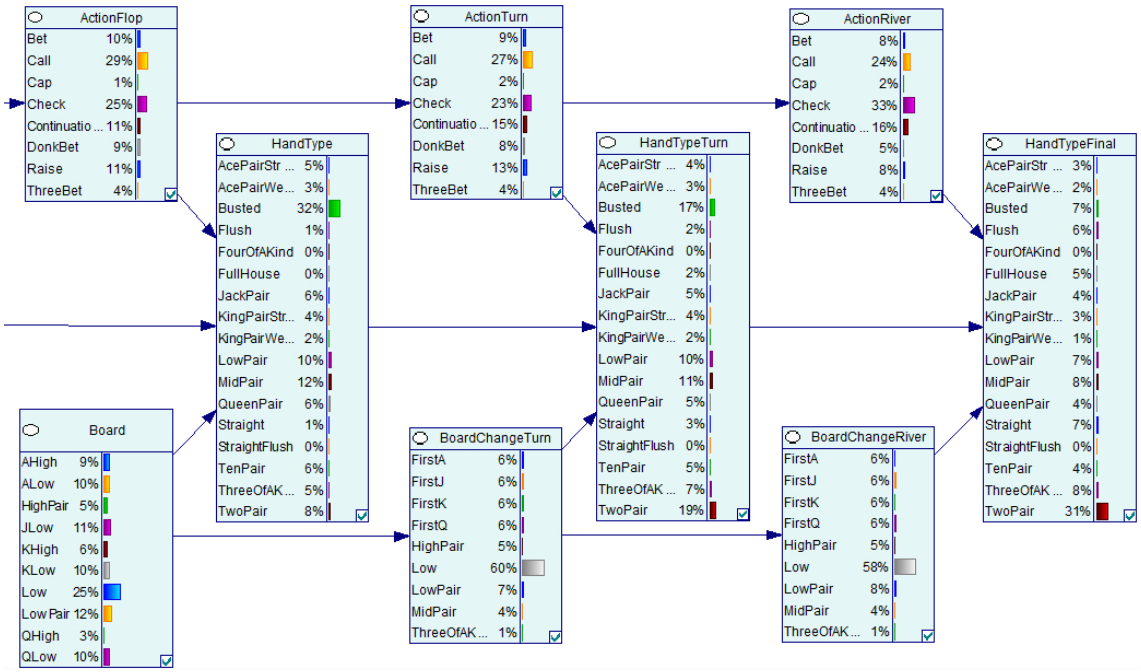


Figure 5.4: The postflop part of the BN

The three *HandType* nodes represent the opponent's hand type at a specific stage of the game, such as two pair or three of a kind. In contrast to the *HandGroup* node, they represent types of hands, which means the two hole cards combined with the community cards, whereas the states in the *HandGroup* node represent groupings of possible hole cards. The *HandType* node at a given stage is conditionally independent of the nodes in the previous stage, given the previous hand node(may be *HandGroup* or *HandType*), the action at the current stage, and the node representing the cards on the board.

Table 5.1 shows some of the less intuitive states of the hand type nodes together with examples of hole cards and community cards that would be represented as this state. Remember that the hole cards combined with the community cards form the hand type, which means that for the sake of the hand type classification, it is irrelevant if the cards that form the hand type are among the community cards or the hole cards. The term *kicker*, used in the description of the first two states in the table, refers to the leftover cards that were not directly used in the hand type. This means that if the player has a pair of aces, he has three leftover cards considering that Texas hold'em uses the best five cards to declare a winner. These three cards are then referred to as the player's kickers. If one of these cards has a rank of ten or above, this would count as a strong kicker, while the opposite would be a weak kicker. If two opposing players have the same hand type, then the kickers (if there are any) would determine the winner. E.g. a hand type that consists of five cards such as flush or straights do not have any kickers. The reason for only including the kicker on the two highest pairs, kings (K) and aces (A), is that these are the ones that are most frequently settled based on the value of the kicker. Kickers could also have been included on pairs of jacks (J) and queens (Q), but this was omitted to reduce the number of states in the hand type nodes and was considered of less importance.

Hand type	**Hole cards**	**Community cards**	**Description**
AcePairStrong	A♥ K♣	A♠ 8♣ 9♥	Pair of aces with a strong kicker
KingPairWeak	K♠ 5♠	K♥ 8♥ 6♣	Pair of kings with a weak kicker
QueenPair	Q♠ Q♥	10♥ 2♠ 6♣	Pair of queens
JackPair	J♠ 10♠	J♥ K♣ 7♠	Pair of jacks
MidPair	9♠ 9♣	A♠ Q♥ 8♥	Pair in the range of 88-TT(Tens)
LowPair	6♠ 6♣	A♠ Q♥ 8♥	Pair in the range of 22-77

Table 5.1: Example states of the Hand type node(s)

There are three nodes for representing the community cards for each of the three stages after the preflop. The node named *Board* in figure 5.6 describes the community cards dealt at the flop stage. The states in this node represent an abstraction of some of the important features of the community cards, such as the state *ALow* refers to a board where an ace is present, but the rest of the cards are considered to be of low rank. The two other nodes, named *BoardChange*, describe how the community cards change when the turn and river cads appear. An example of this would be if the turn card is an ace, and the community cards previously didn't contain an ace, this would be labeled as the state *FirstA*, which refers to the scenario where the new card introduced the first ace to the board. If the ace on the turn is the second ace on the board, then the board change would be *HighPair*, representing a pair with high rank on the board. This way of representing the change of the community cards combined with observing a sequence of actions (one for each stage), is a good predictor for an opponent's hand type. E.g. in a case where the probability is high that an opponent is holding cards from a hand group with high ranking cards, such as aces, and starts betting aggressively when the first ace appears, there is a high likelihood that he is holding a pair of aces.

Table 5.1 continues with some of the community cards used as examples in table 5.2, and shows how these would be classified in the board node. The table also shows how the node for representing the change in the community cards at the turn stage would classify the new card based on the cards that were present at the flop stage. The structure of the full implemented BN can be found in Appendix D.

Flop cards	Board	Turn card	Board change Turn
A♠ 8♣ 9♥	*ALow*	K♠	*FirstK*
K♥ 8♥ 6♣	*KLow*	K♠	*HighPair*
10♥ 2♠ 6♣	*Low*	Q♣	*FirstQ*
J♥ K♣ 7♠	*KHigh*	7♣	*LowPair*

Table 5.2: Example states in the board & board change node

Section 5.3.3 describes how the BN could be extended to account for the opponent's ability of having a *partial hand*. The term partial hands will be used to describe hands that are currently incomplete and need further cards to evolve into a real hand type. For example if the players hole cards combined with the community cards currently constitute four cards of *spades*, then he only needs one more spade to evolve into a *flush*. Another example would be if the player's hole cards combined with the community cards form a sequence of length four, then he only needs one more card in the sequence to evolve into a *straight*. Partial hands, or *drawing hands*, represent incomplete hands with potential of becoming very strong hand types if they are completed, which is an important aspect of Texas Hold'em Poker.

5.3.3 BN extension for partial hand prediction

Figure 5.5 illustrates the extended part of the BN with the goal of inferring information about the opponent's possible partial hands. This part of the BN was designed in the same manner as the original BN, with a focus on representing how the game state changes from one stage to another.

The first thing to note is that similar to the original network's way of representing the community cards, and change in these, the extension models the same, only in respect to how the community card relates to a possible straight or flush. The reason for modeling this in a separate set of nodes, is that a set of community cards can contain both a flush draw, a straight draw, and one of the states in the *Board* node in figure 5.6 simultaneously. And as states in the BN are mutually exclusive, we needed a separate set of nodes for modelling this aspect of the game.

The *FlushBoardFlop*, *FlushChangeTurn* and *FlushChangeRiver* nodes, represent how the community cards relate to a possible flush. The simplest of these, the FlushBoardFlop node contains three possible states; *None*, *Draw* and *Possible* as illustrated in the figure. To explain these states, remember the term suit described in section 2.3.2. Each of these states represent how many cards of the same suit is currently displayed among the community cards. The *None* state, representing no draw, refers to a set of community cards where there is no more than one card of each suite. The draw states means that the opponent can have a partial or drawing flush hand with these community cards, which would

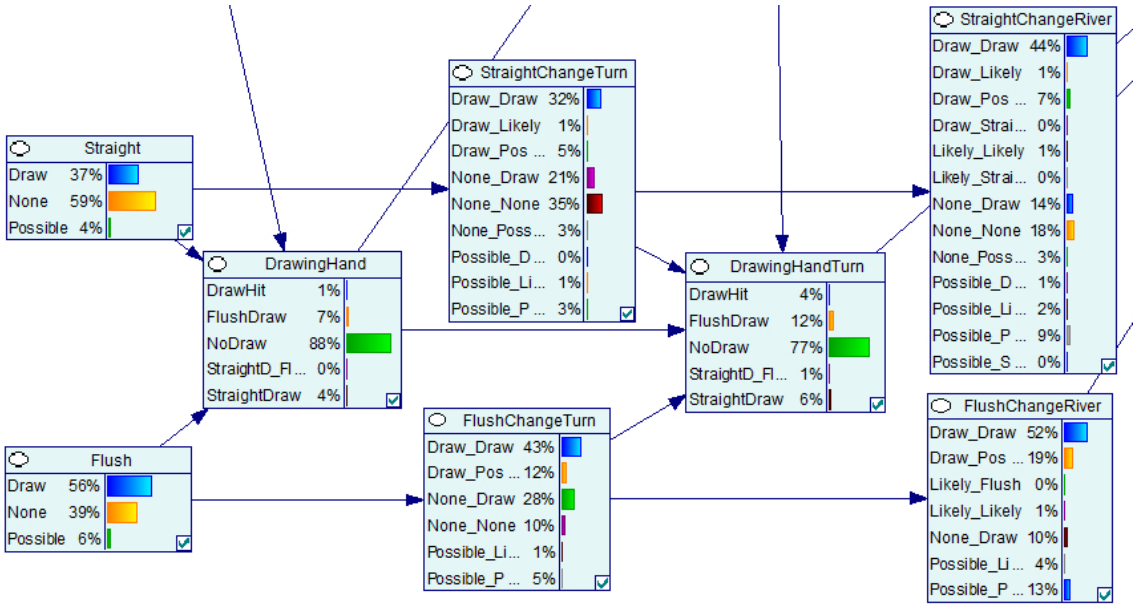


Figure 5.5: The extension to the BN

be the case if there exist two cards of the same suite among the community cards. The state *Possible* is true, if there are three cards of the same suit among the cards on the board. This means that the opponent's hand can possibly have evolved from a partial flush to a completed flush with these board cards.

The two other nodes represent the change of the board in relation to the case that was true at the previous stage. E.g. if the board was at the *None* state at the flop, and the turn card revealed a card with an equal suit as one of the cards at the flop, then the state *None_Draw* would be true for the *FlushChangeTurn* node, meaning that a flush draw is now possible given the new card. These two nodes also have an additional state, the *Likely* state, representing four cards of the same suit. The *FlushChangeRiver* node additionally contains a state called *Flush*, meaning that a full flush is presented by the community cards. This state is only possible at the river stage, because a flush is five cards of the same suit, and the 5th card is displayed at the river stage. Similar to the *Likely* state, which requires four community cards, and is therefore not possible at the flop stage.

The straight nodes work in the exact same manner, only that they use the rank of the cards to determine the straight potential of the board. Meaning that a board with a jack, queen and king would represent the *Possible* state in the *StraightBoardFlop* node, because it contains three cards in a sequence. The difference between flush and straight draws is that straight draws are a bit harder to determine, because a player may have an straight even if there is two holes in the sequence. To exemplify this consider a board with a ten, queen and ace. A player could still have a straight here, because he could have a jack and a king as his hole cards. Therefore, the data used to train this network, consisted only of straight draws in its simplest form, meaning a sequence without holes.

The last two nodes in figure 5.5, *PartialHandFlop* and *PartialHandTurn*, represents the opponent's partial hands. These are as the hand type nodes inferred from the observed evidence. If the opponent has a flush draw, four cards of the same suit, this would be represented as the *FlushDraw* in the partial hand nodes, while an opponent with both a

straight- and a flush-draw would be represented through the *FlushD_StraightD* state. The reason for not including a node for representing a partial hand at the river state, is that if the partial hand is not completed at that time, the hand will not further improve, and just be represented as a busted hand in the *HandTypeFinal* node.

This network is integrated with the original BN through the partial hand nodes. The partial hand node depends on the action at the particular state, in the same manner as the hand type nodes. The partial node also influences the hand type node at the next stage, meaning that if there is a high probability that the opponent is holding a flush draw, then its more likely that he would have a complete flush at the next stage.

This part of the BN was not implemented due to reasons discussed in section 8.1.2.

5.3.4 The BN-CBR interaction

The BN component's main task, is as mentioned previously, inferring opponent's hand types. This is done by inserting evidence into the network, applying an algorithm for probabilistic inference and observing the posterior probabilities for the target nodes. The three hand type nodes together with the hand group node are the target nodes in this network. Each of the hand type nodes are used at different stages of the game with the information observed up to that point of time. Figure 5.6 shows how the hand type probabilities are used in the acquisition of a new case in the CBR system. As illustrated in the figure, the CBR system takes the three states of the hand type node with highes probabilities and adds them to its case description. If there are several opponents still active in the game, the process of updating the Bayesian network is performed ones for each opponent, and the top three states of the combined runs of the BN are used as inputs in the case.

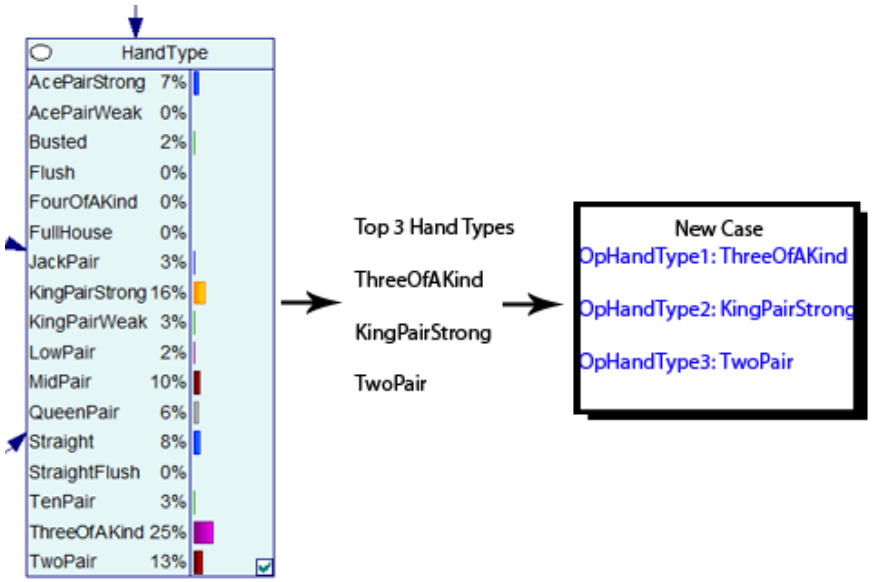


Figure 5.6: Top three opponent hand types prediction to case description

The second task of the BN is to infer a probable range of cards the opponent or opponents may currently possess. This is achieved through the *HandGroup* node, in the same manner as the *HandType* nodes. In contrast to the *HandType* nodes, the states in the

HandGroup node represent groupings of possible hole cards instead of hand types. The four most probable hand groups are used to calculate a numeric measure for representing the strength of the players cards in relation to the most probable opponent cards, referred to as *relative hand strength* in section 5.4.1 and described in more detail in section 6.2.1. Another possibility would be to use a threshold value to decide the number of hand groups to use for further calculation. This will be discussed further in section 8.1.1.

5.4 The Case-based reasoner

The case-based reasoner is the main component responsible for the decision making in the system. It generates a new case composed of features extracted directly through the interface and features inferred by the Bayesian network to retrieve cases similar to the target case, which is a description of the current problem that needs solving. The target case, or problem the system is currently facing, is a set of features describing the poker game at a time where the system is required to make a betting decision. Each case in the case-base has a solution, which is the betting decision made in that specific situation, and an outcome, which describes the outcome of applying the solution to the given situation. Each of the three components of a case, the problem description, solution and outcome are detailed further in section 5.4.1.

The CBR component incorporates all of the four sub-tasks outlined by Aamodt[2]; retrieve, reuse, revise and retain. How the proposed design applies these tasks is explained in section 5.4.3 through 5.4.6. Since the cases require an observed outcome before they are retained into the case-base, they are retained into a temporary storage until the outcome is available. The outcome for a sequence of cases (betting decisions) will first become available when the player folds, or when the round goes to showdown.

5.4.1 Case representation

The cases represent the situation of the game at the time the system is required to make a betting decision. In contrast, the cases could have represented a sequence of situations, or a more general representation of a whole stage, or even a whole round. The reason for choosing a case as a representation of the required information for making a single betting decision, compared to a case representing information required to lay out a strategy for a sequence of actions is the stochastic element poker introduces by its community cards. This makes it hard to plan to far ahead, because a decent hand at one stage may not be so good at a later stage, depending on what cards the board reveals.

The cases are stored into four separate case-bases, one for each stage of the game. This is possible based on the fact that a case represents information needed to make a single betting decision, and that cases retained from the various stages shouldn't be treated as similar.

Figure 5.7 illustrates a case, with its three components, a problem description, solution and outcome. The first component of the case, the problem description, contains a set of features that are considered important when making a betting decision. The features marked with colors are affected by, or directly inputted from the BN. The red features illustrate the former, where the BN output is used to perform calculations, which are then used as input in the problem description, while the blue features are directly inputted from

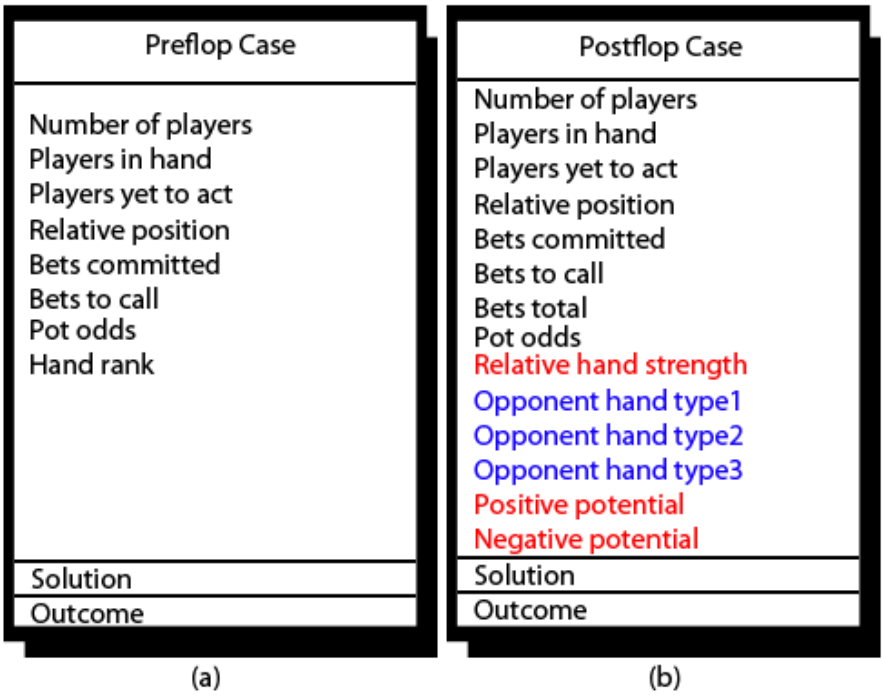


Figure 5.7: The case representation of: (a) Preflop cases (b) Postflop cases

the BN. The two other components of a case are the solution of the case and the outcome of applying the solution to the new case. These features are described further here:

- **Number of players:** is a numeric measure describing the number of players that are still playing at the beginning of each stage. This affects the betting decision in a way that you would play a lesser number of cards, than you would against fewer opponents, because the higher number of players the more likely that someone has better cards than you.

- **Players in hand:** is a numeric measure describing how many players have committed the minimum amount of money to still be active in the round. If many of the already acted players have committed money to the pot, this would mean that there are several players still believing that they can win the pot, in contrast to if one player bet and the rest folded.

- **Players to act:** is a numeric measure describing how many players are left to act after you have acted. The more players that act after you, the higher the likelihood is that someone raises your bet.

- **Relative position:** is a numeric measure describing in what order the player acts compared to the other players. Position is an important feature when making a betting decision, since you get to observe what other players do. If you have a good position to a player, which means you act after that player, you will also act after that player in future stages, which is obviously profitable.

- **Bets committed:** is a numeric measure describing the number of bets the player has committed to the pot at the current stage. This feature will provide some continuity

between cases, because if the player already has committed a lot of money to the pot, he should be less likely to fold to an opponent bet. If this feature was not present in the case, the player would treat every situation as a situation where he has committed no money to the pot, and would be more likely to fold.

- **Bets to call:** is a numeric measure describing the number of bets the player has to commit to the pot to still be able to participate in the round.

- **Pot odds:** is a numeric measure describing the odds you get at the time of the betting decision. If the bet required to continue playing is 1$ and the total pot is 5$ you get a 5-to-1 pot odds. This is a major factor when faced with a betting decision, because if you believe that you have a greater chance than 20% to win, you will have a positive expected value in the long run.

- **Bets total:** is a numeric measure describing the total number of bets the player has committed up to the time of the betting decision. As with the bets committed feature, this feature provides some continuity between cases from different stages of the game.

- **Hand rank/Relative hand strength:** is a numeric measure describing how strong the hand is. The hand rank feature is only used for the preflop cases, and ranks the 169 possible hole cards based on their strength. The relative hand strength is used for the postflop stages, and bases its calculations on the hole cards combined with the community cards and compares them to a set of other possible cards. The set of cards that are used to compare with is a set of handgroups inputted from the Bayesian network, which constitute hands the opponent has a high probability of holding. This provides a much more realistic hand strength measure than computing it in relation to all possible cards, because a lot of the cards are probably already folded.

- **Opponent hand type:** is a hand type describing the opponent's most likely hand type at the current stage. A case contains three of these features, which represent the top three predictions by the Bayesian network.

- **Positive/Negative potential:** is a numeric measure describing the potential the hand has for improving. The positive potential represents the probability of currently having a worse hand which improves to a winning hand after new community cards are dealt, while the negative potential represents the opposite scenario. These measures are also affected by the input from the BN, since they compute the measure in relation to a set of cards, namely the possible cards in the hand groups inputted form the BN.

- **Solution:** The solution of a case is a single betting action. The solution space is farily limited, since there are only five allowed actions in poker; fold, check, call, raise and bet.

- **Outcome:** The outcome of a case represents the money lost or earned by applying the solution to the problem. In other words, the cost of the action applied in the given situation.

There exist a few minor differences between the cases from the different stages of the poker game as illustrated in figure 5.7. The biggest of these is that the preflop cases do not use any of the features inputted from the BN. The reason for this is that there is little information available in the preflop stage compared to the other stages. The cases in the CBR system already encompass the important features for making a preflop betting decision. The preflop cases do not contain the hand potential feature, since there are no available community cards, and the hand rank feature is used to represent the strength of the hand. Since there has been no previous round at this stage, the total bets feature is not required either. The only difference between the postflop cases, is that the cases from the river stage does not contain any positive or negative potential, because there are no future community cards. Section 6.3 shows an example of an instantiated case.

5.4.2 Case-base

The initial case-base was acquired by observing instances of Pokibot's and Simbot's playing amongst themselves. This was possible through the Poker Academy Pro software, and the process of this data acquisition is described further in section 6.1.2. Approximately 10.000 rounds of play was observed and used to generate the initial case-base. Each action performed by one of the bots, in a certain context and with a certain outcome, was used as a case in the case-base. The stage the action was performed in was used to decide what case-base it should be stored in. Table 5.3 shows the number of cases that was retained into the case-base used for a given stage of the game. The preflop case-base was initialy larger, but was reduced in size to reduce the time needed in the retrieval step. Since the preflop cases are the simplest of the four case-types, this could be done without affecting the system performance.

Stage	**Total cases**
Preflop	34.350
Flop	30.297
Turn	19.305
River	14.816

Table 5.3: Size of the Case-base(s)

Both Pokibot and Simbot have been proven to be profitable against human competition for play money[6], and are easily accessible through the Poker Academy Pro software. The quality of the knowledge acquired by observing these play, is believed to be of greater quality than e.g. observing humans play for play money. Also human players may vary greatly in skill level which could potentially reduce the quality of the case-base. These reasons lead us to using Pokibot's and Simbot's for the case-base generation, which was easily accessible and guaranteed good quality for the cases in the case-base.

5.4.3 Retrieve

The case retrieval is performed when the system is faced with a betting decision. This is done by acquiring all the required information illustrated in the problem description in

figure 5.7 and creating a new case which is used for the retrieval process. Once the new case is constructed, the retrieval of similar cases begins by computing the similarity for all the cases in the case-base with the k-nearest neighbor algorithm. The local similarity metrics are described in section 5.4.3, while the global similarity measure is a weighted sum over all of the local similarities. The features considered the most important when making a betting decision are weighted higher than he ones that are less important. The weights used for this research is described in section 5.4.3.

After the initial similarity measure has been calculated for the whole case-base, the cases which have a similarity above a specified threshold are selected for further processing in the reuse step of the cycle.

Weights

When developing the CASPER system, Rubin[40] researched how different weights affected the system performance, and attempted to find an optimal set of weights. The weights in this research are based on the weights with the best performance in the CASPER system for the features that are common to our system. Figure 5.8 illustrates the weights for the cases used in each stage of the game. The default weight value is 5, and the features that are believed to be more important at the particular stage of the game are given a higher weight value. The weights may differ from stage to stage, which represents the belief that the given feature is of greater importance in the stage where it is given a higher value.

	Stage of the game			
Feature	Preflop	Flop	Turn	River
Number of players	5	5	5	5
Players in hand	10	5	5	5
Players yet to act	5	5	5	5
Relative position	10	5	10	5
Bets committed	5	20	10	10
Bets to call	10	20	10	10
Bets total	-	5	5	15
Pot odds	5	5	5	40
Hand-Rank/Strength	50	80	80	80
Opponent hand type1	-	5	5	5
Opponent hand type2	-	5	5	5
Opponent hand type3	-	5	5	5
Positive potential	-	80	80	-
Negative potential	-	80	80	-

Figure 5.8: Feature weights for each case type

If the given feature is not present for the case-type, this is marked as a - in the figure. The opponent hand type features inputted from the BN were given a default weight value of 5, since they encompass three features in a case. Further work should be done on this area, which is discussed in section 9.1.

Similarity Metrics

This section describes the similarity metrics used to compare cases in the case-base. As the majority of the features in the cases are numeric, this section will mainly focus on variations of numeric similarity measures.

Euclidian distance similarity This similarity measure computes the similarity of two features based on their distance inside an interval. This similarity acts as a linearly decreasing function as shown in figure 5.9, where the similarity value (y-axis) decreases as the distance between the two values (x-axis) increases. The mathematical representation of this similarity measure is:

$$S_{ij} = 1 - \frac{|x_1 - x_2|}{MAX\_DIFF}$$

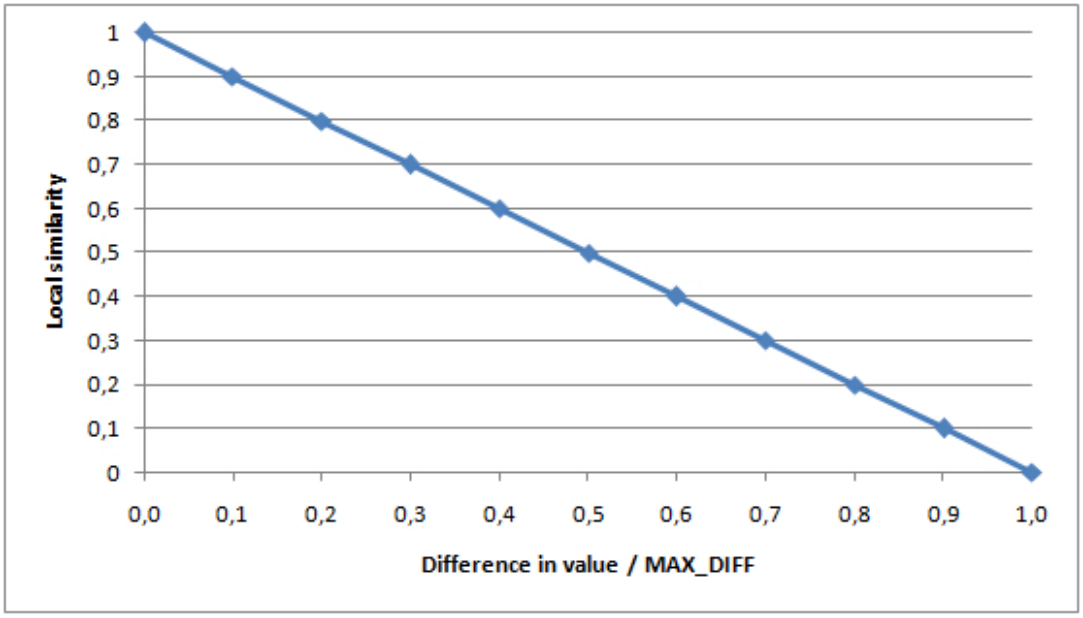


Figure 5.9: The Euclidian distance similarity metric

x_1 represents the feature in the new case, while x_2 is the feature of a given case in the case-base. MAX_DIFF is the length of the interval, meaning the maximum difference the two values can have. This similarity metric was used for the following attributes:

- Number of players
- Players in hand
- Players yet to act
- Relative position
- Bets total
- Pot odds

Expontential decay similarity The exponential decay similarity metric was provided for features where small differences in values should provide less similarities than they would have received by using the Euclidian distance similarity metric. This was used for features where small differences in values have a big effect on how similar two situations are. The hand rank feature of the preflop cases is one such example, where small differences in the rank may affect the way this type of cards should be played. The mathematical representation of this similarity measure is as follows:

$$S_{ij} = e^{-k(\frac{|x_1 - x_2|}{MAX\_DIFF})}$$

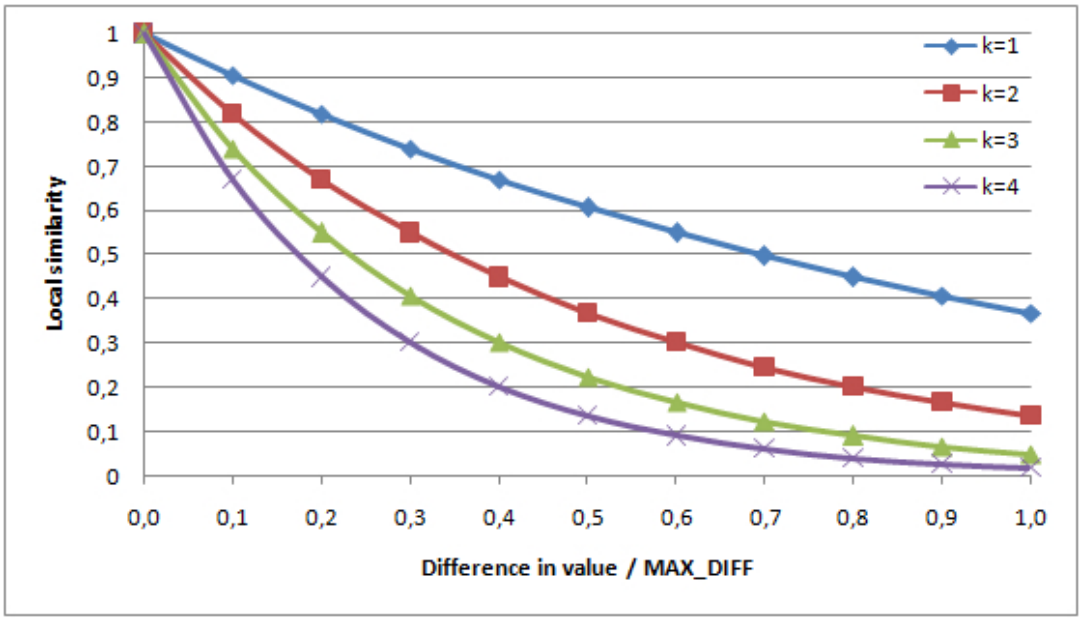


Figure 5.10: The exponential decay similarity metric

k is a constant which affects the rate of decay in the similarity. The greater the value k the faster the similarity will decay. Figure 5.10 shows the exponential decay similarity metric with different k-values. To better illustrate the need for the exponential decay metric, consider Sklansky and Malmuth's hand groups described in the design of the BN. These hand groups represent hole cards that are often played in a similar fashion. Take for example AA from hand group 1, which has a rand rank of 1, and compare that to JTs from hand group 3 with a hand rank of 18. The hand rank feature has a maximum difference of 168, which would lead these two to have a distance of approximately 0.1 in this interval. By observing figure 5.9 this would have a similarity of approximately 0.9, while with an exponential decay constant of k=4, the resulting similarity would be 0.67. As highlighted by this example the exponential decay similarity metric is required to drastically reduce the similarity between hole cards that should not be considered too similar. This metric was also used for other features with the same characteristics:

- Relative hand strength with k=3
- Positive potential with k=2
- Negative potential with k=2

- Bets to call with k=2
- Bets committed with k=4

Bayesian table similarity The Bayesian table similarity is a local similarity measure used for the three opponend hand type features. The similarity between the top three hand types is based on the difference in the probability outputted from the BN for each hand type. The hand types which are not among the three hand types in a given situation will be given a similarity of 0. The mathematical representation of this similarity measure is:

$$S_{ij} = 1 - |Prob(x) - Prob(y)|$$

This similarity measure results in a decreasing similarity when the probability for a certain hand type is high compared to the others, while an increase in similarity when the difference in probability for two hand types is low. Table 5.4 shows an example of how this similarity measure would look like if the BN inferred the top three opponent hand types to be : *FullHouse* (39%), *ThreeOfAKind* (27%) and *TwoPair* (24%) for a new case (query).

	Case-base hand type		
Query hand type	**FullHouse**	**ThreeOfAKind**	**TwoPair**
FullHouse	1.00	0.88	0.85
ThreeOfAKind	0.88	1.00	0.97
TwoPair	0.85	0.97	1.00

Table 5.4: Example of the Bayesian similarity measure

This similarity measure is used for each of the three features used to represent the top three opponent hand types in a case. This example would give a similarity of 1.00 for each case in the case-base that have *FullHouse* as its most likely hand group (represented in the opponent hand type 1 feature), while cases that have *ThreeOfAKind* or *TwoPair* in the same position would be given a similarity of 0.88 and 0.85 respectively. If, in contrast, a given case in the case-base has *ThreeOfAKind* as its most likely and *FullHouse* as its second most likely, this would result in a similarity of 0.88 for both features.

5.4.4 Reuse

The reuse task uses the cases selected in the retrieve task to calculate the sum of outcomes of each of the case solutions present amongst the selected cases, as illustrated in figure 5.11. The solution that has the most profitable outcome from the selected cases will be reused as a solution for the new case, which is marked with the color blue in the figure. The idea behind applying the solution with the best outcome to the new case is to mimic the general idea in poker of maximizing the profit over the long run. The selected cases represent previous similar experiences where different solutions have been applied. The action which has maximized the winnings in the past should be a good predictor of what action maximizes the profit in the future.

The outcome measure could be the cost of an action, being positive when the action lead to a win and negative otherwise, or it could be more complex like the concept described in the revise step.

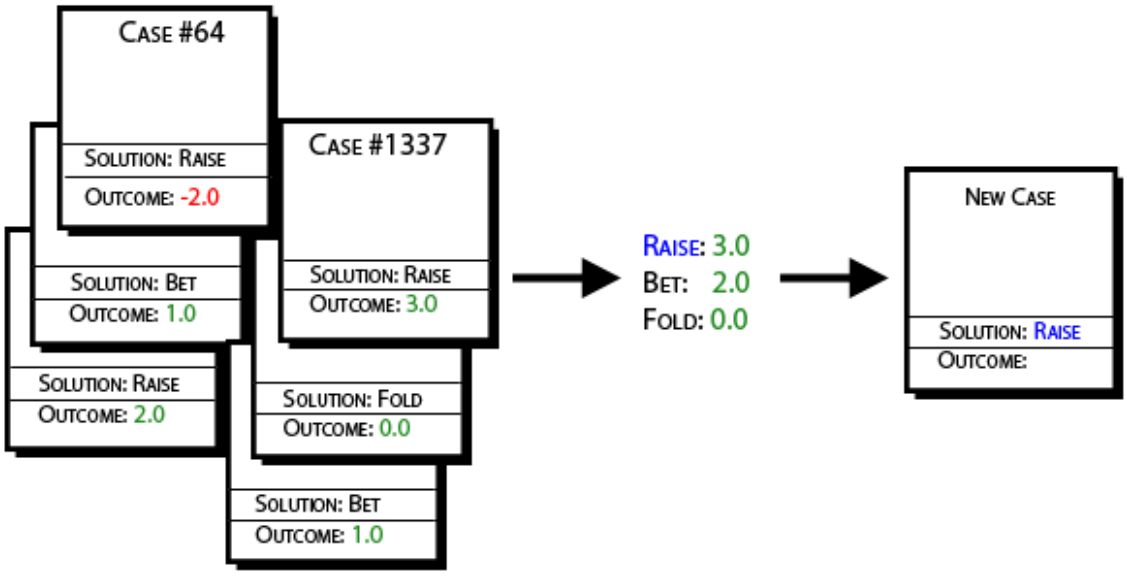


Figure 5.11: The reuse step

5.4.5 Revise

The revise step consists of two sub-tasks; evaluating the case solution generated by reuse and repairing the solution if it was unsuccessful. When using the system as a fully automated poker bot, testing of the solution is performed when the bot applies the action to the game environment. Since the outcome of applying the solution first becomes available when the bot folds, or when the round goes to showdown, the evaluation of the solution has to wait until the outcome becomes observable. One round of poker contains several actions, which leads to several runs of the CBR cycle before the outcome becomes available. This results on several applied case solutions being dependent on the same outcome. The evaluation part of the revise step therefore evaluates a sequence of case solutions that lead to the observed outcome.

The game of poker involves a big element of luck, which results in bad players being able to win short-term. This makes the evaluation of case solutions more difficult, because the outcome of the solution, the win or loss of money, may be the result of luck. By only evaluating the success of the solution based on a positive outcome, may result in the case-base getting filled up with cases that do not represent good solutions, but are rather an element of luck. If a good action results in a bad outcome, this would also count as a negative case. For this reason we have to extend the evaluation of the solutions with other criteria to evaluate. Billings and Kan proposes one approach for this purpose, by analyzing actions through perfect knowledge achieved by hindsight analysis when the players go to showdown[41]. A limitation of this approach is that it requires perfect knowledge achieved only when the players go to showdown. To overcome this limitation

we suggest the approach explained below, which also has the advantage of using the domain knowledge introduced by the BN to provide a good evaluation for the outcome of a case.

For assesing the outcome of an action in our architecture we propose using a concept introduced by Galfond[42]. Galfond introduced the term Galdons dollars (G-bucks), which is a tool for calculating the expected value of an action. To do this, we first have to obtain the *equity* we have against a specific hand, which means the percentage of times our hand beats the other after all community cards are dealt. The equity is usually calculated through simulation, by simulating a large number of scenarios and recoding how many percent of the times a specific hand wins against another at showdown.

The second thing required to calculate G-bucks, is a likely range of opponent hands, which in our architecture is provided through the BN's hand group prediction. To obtain the G-bucks of a specific action, we then calculate the combined equity our cards have against all of the opponent's most likely cards. This, results in the percentage of times we are expected to win in the given situation. To obtain the money we are expected to win long run, we then multiply the win percentage with the current pot size. The result of this calculation is the amount of money we are expected to win long run in the given situation. By comparing this amount with the money the given action cost us, we obtain the G-bucks of the action, which tells us if the given action would result in a negative or positive income over the long run.

This way of evaluating the outcome of a solution removes the issues concerning the luck element of poker, has the advantage of involving the BN's hand group prediction to achieve a better assessment of outcomes, and does not require the perfect knowledge obtained at showdown. Although the concept is intended against one player and for the no-limit variant of Texas Hold'em, a modification of this concept should also provide useful for this domain.

5.4.6 Retain

The retain step integrates the newly solved case into the case-base. If domain knowledge is used to evaluate the outcome of the case in the revise step, then the revised case is retained into the case-base.

When there is no outcome available the retain step will store the case with an outcome of zero or in temporary storage. This means that with the reuse strategy proposed in this architecture, the case will not have any effect on future case retrievals before it is provided with an outcome.

CHAPTER 6

Implementation

This chapter describes the process of implementing the architecture proposed in chapter 5. The chapter is structured in a way that describes the implementation required for each of the components in the system, starting with the data collection and the interface (section 6.1). This is followed by section 6.2 describing the implementation of the overall system, where section 6.2.1 describes the CBR implementation and section 6.2.2 describes the BN implementation. Finally section 6.3 describes how to run the system, and provides the reader with a run time example.

6.1 The interface & Data collection

The interface was implemented by using the Meerkat API(section 3.3), which contains a set of methods for observing and extracting information from the game environment, as well as inputting betting decisions to it. The Meerkat API divides this into two interfaces, the *GameObserver* and the *Player* interface. The *GameObserver* interface is used to *listen* to general events at the table, such as actions performed by other players or rounds that went to showdown. Each type of events results in the firing of a method for listening to that event. The *GameObserver* interface was mainly used to collect data to the reasoning methods, while the *Player* interface was used to alert the reasoning methods when its time to perform an action, and then implement the chosen action. The next two sections, section 6.1.1 and 6.1.2, describe the acquisition of data required by the CBR and BN.

6.1.1 BN data collection

The Bayesian network was trained by observing the bots in Poker Academy pro play. This was done by creating a bot with the necessary event listeners, with the sole purpose of observing the other bots play. The BN does only learn from rounds that have gone all the way to showdown, because it then gets to observe the players cards together with the states that lead there. Each time a player performs an action it collects the relevant set of primitive data from the interface, and calls the RBR to classify it appropriately, e.g. a raise would be classified as a *ThreeBet* or *Cap* based on the number of observed raises before that action. This data is then stored in temporary storage, and will be used depending on the round going to showdown or not.

If the round goes to showdown, the data for the players currently in showdown is retrieved from temporary storage and combined with the newly observed information, such as the revealed hands. This data is then written to a text file, which is used to train

VPIP	PFR	Position	Action	HandGroup	ActionFlop	Board	HandType
a50_60	a0_5	Early	Call	HandGroup10	Check	JLow	TenPair
a20_30	a5_10	Late	Raise	HandGroup9	Check	JLow	Busted
a10_20	a5_10	Early	Raise	HandGroup3	Check	JLow	JackPair
a30_40	a10_15	Middle	Call	HandGroup5	Check	JLow	Busted

ActionTurn	BoardChangeTurn	HandTypeTurn	ActionRiver	BoardChangeRiver	HandTypeFinal
Check	FirstK	TenPair	Check	MidPair	ThreeOfAKind
Check	FirstK	Busted	Check	MidPair	TenPair
Bet	Low	JackPair	Check	HighPair	ThreeOfAKind
Call	Low	Busted	Check	HighPair	JackPair

Figure 6.1: The data used to train the BN. Top(preflop-flop) & bottom(turn-river)

the BN through GeNie's learn parameter function. Figure 6.1 shows an excerpt of the text file when loaded into GeNie. Each row in this text file represents the information extracted from the perspective of one player when he reaches showdown.

6.1.2 CBR data collection

The CBR component acquired its inital case-base much in the same way as the BN was trained, by observing the bots in Poker Academy Pro play amongst themselves. In contrast to the BN, the CBR was required to learn from each of the actions performed by observing the bots, and not only the ones that went to showdown. As the CBR system also needs to know when to fold it has to observe situations where the bots fold. A problem with the approach of using the interface to retain cases through live play, was that the Meerkat API only allows for observing the bots hole cards when and if they reach showdown. This gave us two options for gathering the initial case-base, either through parsing the game logs generated by Poker Academy Pro, or by using a combination of the interface and the game logs.

The first option, parsing the game logs, was evaluated to be an extensive and cumbersome task, because we needed to extract data required for the reasoning of both CBR and the BN, which would result in the parsing of a lot of different data. Since the Meerkat API already contained methods for extracting these features by observing live play of the bots, the full parsing of the game logs was considered unnecessary and redundant.

These reasons lead to an approach combining the extraction of most of the features through the Meerkat API by observing live play, and then parsing the game logs to acquire the player's hole cards. This was possible by using the Meerkat *gameID* feature, which provides a unique id for every round of play. Combining this id with the name of the player the case was extracted from, the appropriate hole cards could be extracted from the text file containing the game log.

The first part of this data collection was performed in the same way as the BN, by creating an observer bot and plugging it into Poker Academy Pro. It then observes and collects the required features that it is able to extract through the interface and stores the cases in a database[1]. After the bot has observed and retained an appropriate number of cases, the parser is used to extract the various hole cards from the game log, and calculates and inserts the case features depending on these hole cards into the correct cases in the database.

[1]The database was implemented with Apache Derby network database server

6.2 The system

The first attempt at extending the data collector described in section 6.1 with reasoning capabilities proved more difficult than expected. The reason for this was the lack of information on how to successfully use third party software packages inside the Poker Academy Pro environment. This was a serious problem for the implementation of the proposed architecture, because it is heavily dependent on using a set of third party software packages for implementing the reasoning capabilities in the system, such as jColibri and SMILE.

The thing that made this problem particularly difficult was that the system executed normally when tested outside Poker Academy Pro, but did not execute correctly when plugged into it. To overcome this problem we attempted to use a program called Fat-Jar, which creates an executable jar file from an eclipse project by extracting and adding all referenced third party libraries to one single jar file. This process fixed some of the issues caused by the intital problem, but still had problems with e.g. the driver for connecting to the database used to store the cases for the case-based reasoner.

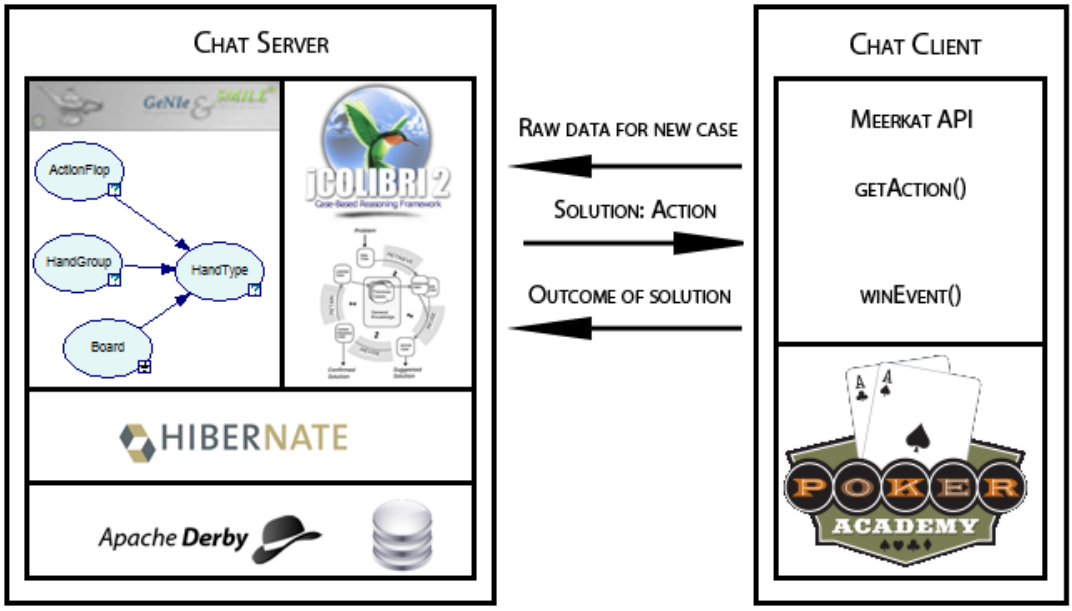


Figure 6.2: Interaction between the server (brain) and the client (arms and eyes)

As this had already been a time consuming process and the system did still not run properly, we were forced to think of another way to implement the system. This resulted in the system illustrated in figure 6.2, where the components in the system were separated into two programs. The component concerned with interacting with the poker client, the interface, was divided into a separate jar file, whereas the rest of the components required for the reasoning of the system were included in another jar file. This can be viewed as the interface being the arms and eyes of the system, while the jar file containing the reasoning methods constitutes the brain. The communication between these two is done through java transmission control protocol (TCP) sockets, where the jar file with the reasoning components work as a chat server, while the interface works as the chat client. The arms and eyes send the required arguments to the brain, which performs the reasoning in the system resulting in an action being sent back for the arms to execute. As seen in the figure,

the outcome of applying the solution is also reported back to the brain (server) when it becomes available.

By dividing the system as described here, no external third party software had to be included in the part that is plugged into Poker Academy Pro, which solved the problems with the third party software integration. This way of implementing the system also has the advantage of making the reasoning part easier to integrate with interfaces required by other poker clients, such as Open Holdem, because only the interface part of the program has to be changed.

The next sections describe the process of implementing the reasoning methodologies.

6.2.1 The CBR component

The case-based reasoner in this architecture was implemented with the jColibri framework. jColibri divides a CBR application into four parts; *configure*, *precycle*, *cycle* and *postcycle*. The process of implementing the CBR application involved specifying these methods with the required code. Section 6.2.1 through 6.2.1 described this process. The implementation of the *cycle* method is divded into the 4R's of the CBR cycle. The precycle and postcycle methods were not needed for this implementation.

Case representation

A case in this architecture is represented by three case components, a problem description, solution and outcome of applying the solution. Each of these case components are represented as java objects, including fields for representing the features of that particular component combined with a set of set and get methods for each of the fields. To map these components to the correct database tables and columns a hibernate mapping file was created, which provides information on how hibernate should map a object to the database.

The cases also include a set of unindexed features that are not used in the retrieval of cases. These features are used for other tasks, such as the initial acquisition of the casebase and to identify the cases uniquely in the database. The unindexed features of a case are :

- Case id
- Game id
- Player name
- Board cards
- Hole cards

The most of the case features presented in section 5.4.1 are easily extractable through the interface by predefined methods or straightforward calculations, or inputted directly from the BN. The rest of this section will detail the features that encompass more complex calculations, such as the strength and potential of a hand.

Hand Strength The process of calculating the hand strength feature is illustrated in figure 6.3. The right side of this figure illustrates the work done inside the RBR component, which was illustrated superficially in figure 5.2. The four most likely hand groups inferred by the BN is inputted to the RBR component. These hand groups are then used to extract the hole cards contained in each group through a table containing all of the hand groups and what cards they represent. The figure illustrates only the relevant hand groups, and some of the cards contained in these. For a full list of what cards are represented in each hand group see Appendix B.

Since the look-up table contains hole cards on a more general level, e.g. in the form of *AA* and *AKs*, a method for transforming these into real hole cards is applied (the transform method in the figure). The *s* in this situation refers to hole cards that are suited, while offsuited hole cards are not marked by any letter. The term AA represents having a pair of aces as hole cards. There exists four different suits in poker (hearts (h), diamonds (d), spades (s) and clubs (c)), which means that the term AA encompasses six unique combinations of aces in different suits, namely; A♠ A♥, A♠ A♣, A♠ A♦, A♥ A♣, A♥ A♦ and A♦ A♣. The transform method in the figure performs this process for each of the cards represented in the hand groups. Another example of this process would be the AKs, which represents the notion of having a king and an ace with the same suit as hole cards. Considering the four possible suits, this example would have four unique combinations. The transform method also removes the cards that the opponent can't have, because they are already observed among the community cards or the players own hole cards. The transformed cards are then used as input in the function for computing the hand strength, which is described below.

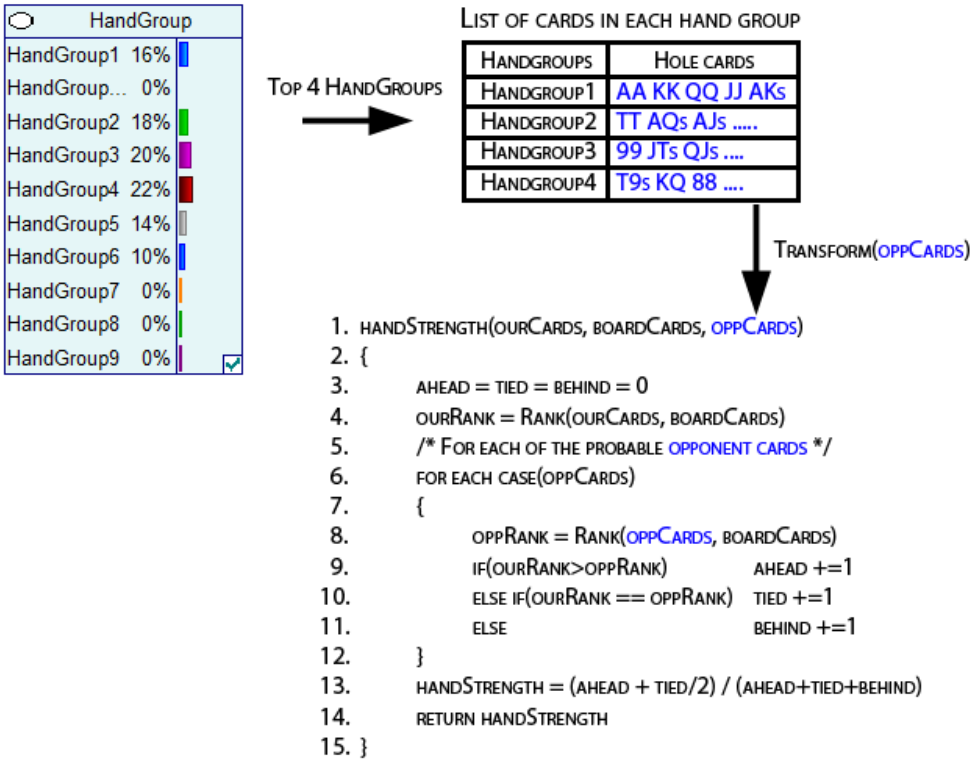


Figure 6.3: The process of calculating the hand strength

The algorithm for calculating the hand strength feature is given in figure 6.3. The

algorithm is performed in the same way as proposed by Billings et. al[6], but instead of calculating this measure based on all possible card combinations, the BN is used to provide the most likely cards the opponent is holding. This provides a much more accurate strength measure of the hand if used correctly.

Line number 4 and 8 in the algorithm uses a method refered to as *Rank* to calculate the rank of the players hand and a given opponent hand. This method is provided through the Meerkat API and calculates an integer value representing how strong the hand is. The higher the integer value the stronger the hand, which means that a hand with a given integer value beats all hands with smaller values at the current stage of the game.

Line number 5 and 6 in the algorithms is where the input from the BN is used to provide a set of the opponent's likely cards. This step uses the list of the opponent's most likely cards resulting from the process illustrated in the figure, rather than iterating over all possible card combinations. Line 9 through 11 compares the rank of each set of opponent cards with the rank of the players hole cards, and increments ahead, tied and behind depending on the result of the comparison.

The final return statement returns a numeric measure in the range of [0,1] representing the percentile chance that the players cards is better than the opponent's cards at the current stage of the game. This is calculated by taking the number of cards the player beats plus half of the draws divided by the total. If there are more than one opponent still left in the game this process is executed for each opponent. The hand strength from each of these executions are then multiplied to represent the probability of beating all of the opponent's cards, which results in a decrease in hand strength compared to a situation where there is only one opponent.

Hand potential Figure 6.4 illustrates the general algorithm for computing the potential of a hand. The potential of a hand is a measure for describing how the player's current hand strength changes when future community cards appear. The implemented system uses both positive (*PPot*) and negative (*NPot*) potential as separate features in a case. Positive potential is the chance that a hand which is currently worse than another hand ends up beating that hand at showdown, while negative potential is the chance that a leading hand looses at showdown.

The algorithm illustrated in the figure calculates this for all of the opponent's possible hole cards. The first part of the algorithm is similar to the hand strength algorithm, which calculates whet ever we are currently ahead, behind or tied with the given opponent hole cards. The next part of the algorithm iterates over all possible future community cards, and checks if the new cards had any effect on how the two hole cards relate to each other. E.g. if the players hand was worse than a particular opponent hand, but improved to beat the opponent hand after the new community cards, the counter for this scenario (*HP[behind][ahead]*) would be incremented. The final positive and negative potential is calculated as shown in figure 6.4.

The algorithm shown in the figure calculates the potential of a hand in relation to all possible opponent cards. The function for calculating the hand potential implemented in this system uses the most likely opponent cards inferred by the BN, as was shown in the figure which illustrated the calculation of the hand strength feature (figure 6.3). This means that the hand potential function also takes the opponent's most likely hole cards as input, and uses these as the basis for the rest of the steps in the algorithm. Apart from this the hand potential is calculated in the same manner as in the figure.

```
HandPotential(ourcards,boardcards)
{
   /* Hand potential array, each index represents ahead, tied,
      and behind. */
   integer array HP[3][3]    /* initialize to 0 */
   integer array HPTotal[3] /* initialize to 0 */

   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards
      for the opponent.*/
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank > opprank)       index = ahead
      else if(ourrank = opprank) index = tied
      else /* < */                index = behind
      HPTotal[index] += 1

      /* All possible board cards to come. */
      for each case(turn)
      {
         for each case(river)
         {  /* Final 5-card board */
            board = [boardcards,turn,river]
            ourbest = Rank(ourcards,board)
            oppbest = Rank(oppcards,board)
            if(ourbest > oppbest)        HP[index][ahead ] += 1
            else if(ourbest == oppbest) HP[index][tied  ] += 1
            else /* < */                 HP[index][behind] += 1
         }
      }
   }

   /* PPot: were behind but moved ahead. */
   PPot = (HP[behind][ahead] + HP[behind][tied]/2
            + HP[tied][ahead]/2)
          / (HPTotal[behind] + HPTotal[tied]/2)
   /* NPot: were ahead but fell behind. */
   NPot = (HP[ahead][behind] + HP[tied][behind]/2
            + HP[ahead][tied]/2)
          / (HPTotal[ahead] + HPTotal[tied]/2)
   return(PPot,NPot)
}
```

Figure 6.4: The process of calculating the potential of a hand. Image from Billings et. al [6]

Configure

The configure method initializes and configures the application. It creates a set of database connectors provided by jColibri and initializes them by loading a set of XML files. The connector relies on Hibernate to manage the persistence of the cases by accessing and retrieving them in a uniform way by the use of mapping files. The XML files that initialize the connectors contain information about the location of where the required mapping files can be found. These include a hibernate mapping file, containing the database specific details such as what driver to use and the database connection URL, and mapping files for each of the case components represented as java objects.

Since our architecture uses four separate case-bases, the configure method initializes a connector for each of these. Each case-base is mapped to its own database table through its own mapping files. The configure method is run when the server is executed, and will keep the case-base in memory until the server is stopped.

Retrieve

Implementation of the retrieval step involved specifying the global similarity function and a set of local similarity functions for the all of the features of the case. Both the global similarity and the local similarity function referred to as Euclidian distance similarity in section 5.4.3 were already included in the jColibri framework. Using these only required specifying what attributes they should be used on, and providing the required arguments, such as the length of the interval. The Bayesian and the exponential decay similarity functions needed to be implemented separately. The exponential decay similarity function was implemented to take the decay constant and the length of the interval as arguments, while the Bayesian similarity function is initialized with the top three hand types from the BN combined with their probabilities.

The last part of the retrieval step was the implementation of the method that selected all the cases with a similarity above a certain threshold value. In order to account for a decreasing case-base size for the four stages and a more complex case structure, the threshold value was set differently for the four case-bases. The threshold was set to 95%, 93%, 91% and 90% for the preflop, flop, turn and river case-bases. This was not directly supported in jColibri, but since jColibri supported functionality for selecting the top k most similar cases, this was easily extendable to using a threshold rather than a specified number of cases.

Reuse

The strategy for deciding what solution to reuse from the cases selected in the retrieve step, was not included in the jColibri framework. The reuse step was implemented by first extracting the solution and outcome from the selected cases. Each of the solutions present in the selected cases are candidates for being used as the solution to the new case. The decision between these solutions is made by taking the sum of the outcomes for each action, and selecting the solution with the best outcome. The selected solution is then copied as a solution to the new case. The chat server then takes the selected solution and sends it to the client for execution.

The approach we proposed for effectively evaluating the outcome of a solution, to reduce the affect of luck, described in section 5.4.5, was not implemented in this system

due to time limitations. For this reason, we chose to implement a second strategy for deciding which solution to reuse from the most similar cases. This was implemented to measure the affect the outcome based reuse strategy had, when not applied with domain knowledge to effectively assess the outcome of a solution. This alternative reuse strategy is referred to as *majority voting* in the case-base community, and involves selecting the solution which is present amongst the majority of the selected similar cases. This resulted in two different versions of the BayCaRP system, which only differ in the reuse step of the CBR-cycle:

- **BayCaRP1**: which reuses the solution with the best outcome, described in section 5.4.4, without the application of domain knowledge in the revise step to evaluate the outcome of a solution.
- **BayCaRP2**: which uses majority voting to decide what solution to reuse.

Revise

The revise step encompasses applying the solution selected in the reuse step, which is performed when the client receives the action and applies it to the poker environment through its interfacing mechanism. As explained in section 5.4.5 the result of applying the solution is not available before the player folds, or all the other players fold, or a showdown occurs. The implemented revise step therefore sets the cost of applying the solution as a temporary outcome for the new case. E.g. if the solution selected is the call action, and the cost of this action is 2$, then this is set as the temporary outcome for the new case.

The second task in the revise step is activated either when the selected solution represents a fold action, or when the client reports that the player has lost or won. When activated this task takes the cases retained in temporary storage (explained below) and revises the outcome of these cases to represent the real result of applying the solution. For example, if the player folds or loses, all the cases leading up to this event will be revised to a negative outcome, because each of the outcomes of these solutions represent money lost. In contrast, if the player wins, the outcomes are kept positive.

Retain

The retain step retains the new case into storage. If the revise step has not yet revised the cases to represent the real outcome of applying the solution, then the new case is stored into temporary storage which is separated from the case-base. When the revise step is finished with revising the outcome of the cases in temporary storage, the revised cases in temporary storage are retained into the correct case-base, depending on what stage of the game the case was created.

6.2.2 The BN component

The BN described in section 5.3 was modeled with the GeNie software, and trained with the data extracted from the data collection phase (section 6.1.1 through GeNie's functionality for learning the parameters of a network. This network was then implemented with SMILE's java wrapper, which provided the functionality for using the network for

inference in a code-based environment. Like the CBR-component, the BN was also implemented with a configure method, which initializes the code-based network by reading the network that was created graphically with GeNie.

The inference functionality of the BN was implemented by creating a method, which takes the currently available information (evidence) as input and returns a list of *BNobjects* sorted based on the probability of each state. The *BNobjects* were created as java objects, with the purpose of containing both the name of the state and its probability.

The list of *BNobjects* contains the states and posterior probabilities of the nodes considered target nodes after the evidence is inserted. The method decides what nodes to use as target nodes, depending on what stage the game is currently in. Since the top four hand groups are used for the relative hand strength feature of a new case in every stage of the game, this node is considered a target node every time the method is executed. The correct hand type node is then selected as a target node depending on the stage of the game.

The top four hand groups are then used as input to the function for calculating the hand strength, where the result of this calculation is used as input to the new case. The top three hand types are used directly as input to the new case. The probabilities for these states, contained in the *BNobjects*, are used to initialize the Bayesian similarity function which is applied in the retrieval of cases that are similar to the new case.

6.3 System example run

Since most of the work done by the system is hidden from the user, this example is included to give an illustration of the reasoning performed behind the scenes when the system makes a betting decision. The example is taken from a specific round from BayCaRP's testing run and the images illustrate important aspects of BayCaRP's reasoning in relation to the current game state.

Figure 6.5: The game state at the flop stage

Figure 6.5 shows this scenario at the flop stage, but before going into more details a

description of the preflop betting has to be given. *Jake*, a Pokbibot, began by raising from an early position, which resulted in a threebet from BayCaRP and finally a call from Jake which ended the preflop betting. BayCaRP has over the previous rounds collected Jake's VPIP and PFR, which at the time of this event is in the categories 20_30% and 5_10% respectively. The area marked with yellow in the figure represents the probability each player has to win in the current setting. This is calculated by Poker Academy Pro through simulation based on knowledge of both players' cards.

Jake was the first to act at the flop stage and started with checking to BayCaRP. BayCaRP answered this with a bet which resulted in a call from Jake. Figure 6.6 shows the state of the BN at the time BayCaRP decides to bet. Based on the evidence observed at the preflop stage, the BN infers the opponent's four most likely hand groups to be as illustrated in the figure. Jakes is holding AK offsuit from hand group 2, which is the BN's most likely hand group with a probability of 31%. The community cards are classified as *JLow* and Jake performed a check. Combined, this evidence results in the state of the BN observed in figure 6.6.

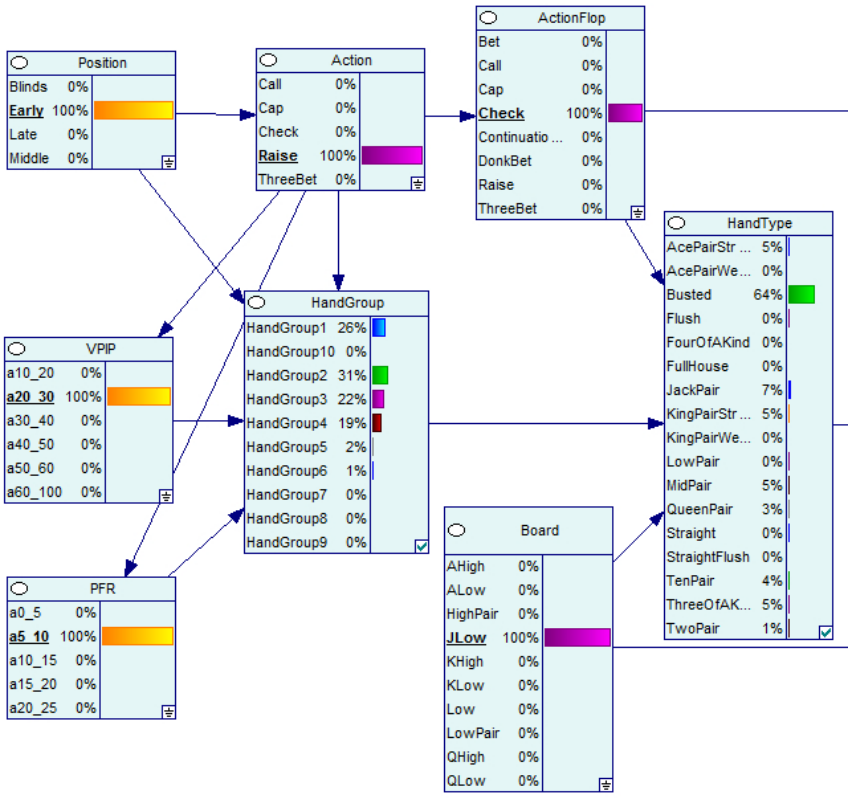


Figure 6.6: The BN at the flop stage

The top three hand types are inserted in the new case, and the probabilities are used to generate the Bayesian similarity measure. The case generated in this situation is shown in figure 6.7. In this case, the opponent's most likely hand types are *Busted* (64%), *JackPair* (7%) and *AcePairStrong* (5%). This high difference in probability results in a low similarity for the cases where *Busted* is not the number one most likely hand type. Jake's hole cards do not form any hand type when combined when the board, referred to as a *Busted* hand type, which is accurately predicted by the BN with a probability of 64%.

The opponent's most likely hand groups are used to calculate the three features marked

Flop Case		
Number of players	:	2
Players in hand	:	1
Players yet to act	:	0
Relative position	:	1.0
Bets committed	:	0.0
Bets to call	:	0.0
Bets total	:	6.0
Pot Odds	:	0.118
Relative hand strength	:	0.565
Opponent hand type1	:	BUSTED
Opponent hand type2	:	JACKPAIR
Opponent hand type3	:	ACEPAIRSTRONG
Positive potential	:	0.087
Negative potential	:	0.158
Solution		
Outcome		

Figure 6.7: The new flop case

with red through the process explained earlier. The CBR component uses this case to retrieve similar cases, and selects the cases above the specified threshold. The selected cases are then used to calculate the most profitable action in the reuse step based on the outcome observed by applying these solutions in the past. The bet action is shown to be the most profitable action, and is copied to the new case and applied in the game environment.

Figure 6.8: The game state at the turn stage

The turn stage unfolds very similar as the flop stage. Figure 6.8 shows the game

state at the turn stage, and as illustrated here, BayCaRP still holds the best hand with a 84,1% probability of winning, and Jake performs a check. Figure 6.9 shows how the new game state affects the BN. The new community card at the turn stage is classified as *Low* in the *BoardChangeTurn* node and Jake was again observed checking to BayCaRP. Additionally, the last most aggressive action performed by Jake in the flop stage was a call, which means that this is added as evidence in the BN. As seen in the figure the BN still predicts Jake's most likely hand type to be *Busted* with a high probability. This is used in a new CBR cycle, which results in BayCaRP performing a new bet.

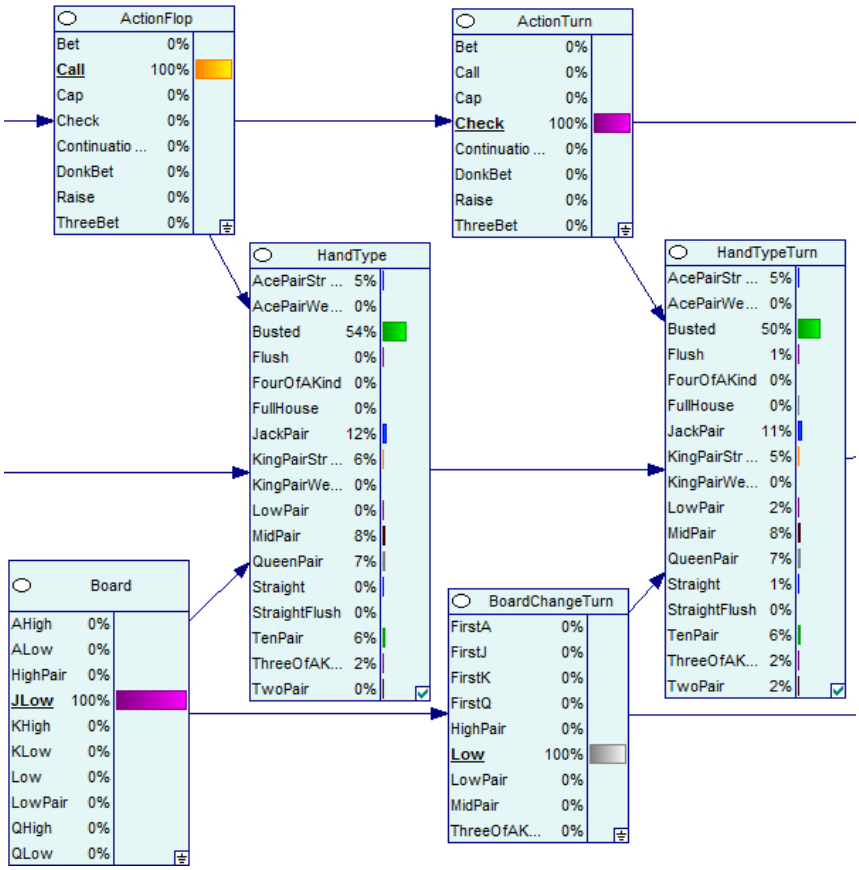


Figure 6.9: The BN at the turn stage

As the final community card appears at the river stage, Jake's busted hand evolves into a straight. Since there are no further community cards to come Jake has the winning hand, which is illustrated in figure 6.10. Jake chooses to bet before its BayCaRP's turn to act, which is classified as a *DonkBet* by the RBR and used as evidence in the BN. The bet is classified as a *DonkBet* since Jake chose to bet before BayCaRP, which was the previous round's aggressor, as explained in section 2.3.7. Additionaly, the river card is a queen and is classified as *FirstQ* in the BN, and the last most aggressive action from Jake at the turn stage was a call. Figure 6.11 shows the BN with the new information added as evidence. As observable from the figure, the BN now infers Jake's hand types to be *Straight* (31%), *QueenPair* (15%) and *TwoPair*(14%).

Figure 6.12 shows the new case generated for the river stage. The river case does not contain any hand potential features, because all community cards are already on the board. The reader should also observe that the relative hand strength feature has decreased

Figure 6.10: The game state at the river stage

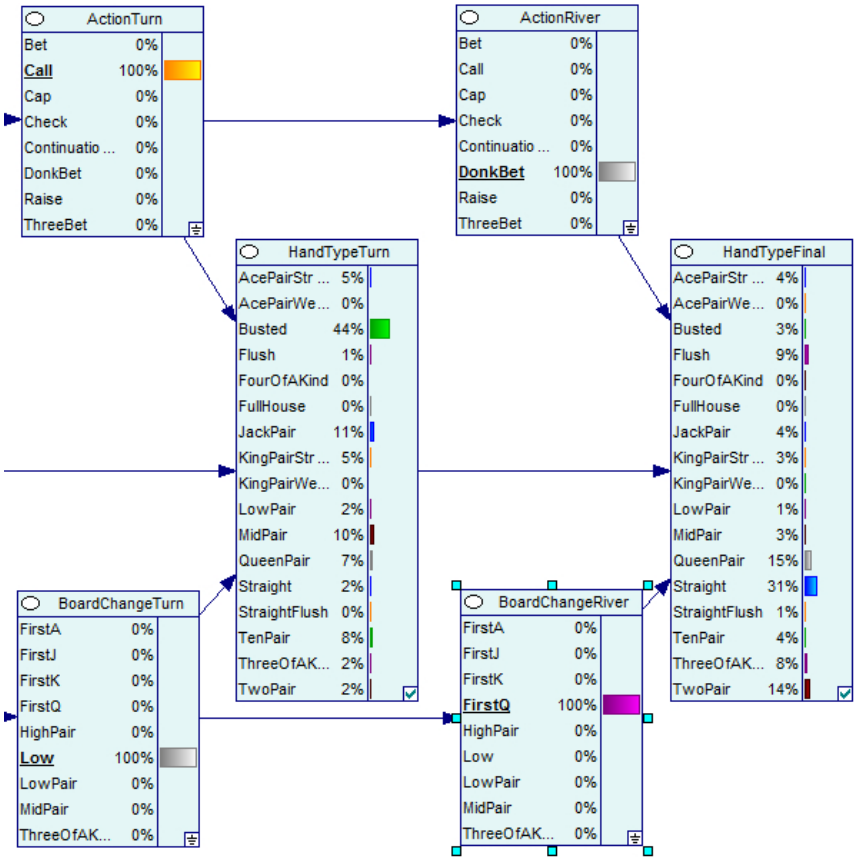


Figure 6.11: The BN at the river stage

significantly as a result of the queen appearing on the river stage. This decrease in hand strength reflects that with a queen on the table most of Jake's most likely hole cards, inferred through the hand group prediction, now beats BayCaRP's hand.

Figure 6.12: The new river case

The newly constructed case is used in the CBR cycle which leads BayCaRP to fold, as this is the most profitable solution present in the selected cases. This leads the revise step to assign negative outcomes to all the cases leading up to the given point in time. Finally the cases previously in temporary storage are now retained to the case-base.

This example gives the reader an example of BayCaRP's reasoning process and will also be used for discussion in section 8.2. This example shows the strength of the two reasoning methods combined. The BN correctly predicts the opponent's cards at each stage of the game, which results in the retrieval and selection of good cases in the CBR component. Combined the two reasoning methods make BayCaRP play this round brilliantly. BayCaRP attempts to maximize its profit when it possesses the best hand, and folds to minimize its losses when the opponent has a better hand.

CHAPTER 7

Testing & Results

This chapter describes the results obtained by testing the system. The chapter begins be describing the testing environment, followed by the results obtained by testing the BN separately from the system, and ends by describing the results of testing the whole system. The results described in this chapter will form the basis of the discussion in chapter 8.

7.1 Testing environment

The system was tested at two important steps of our proposed architecture. The first tests were designed to measure the accuracy of the sub-task solution derived from the BN, namely predicting an opponent's cards. This test is important because the overall performance of the system depends on good results from the BN in order to produce good results itself. The second tests where aimed at testing the final output from the system, the betting decision, produced by the CBR system.

The BN was tested by a separate data set consisting of approximately 1.000 rounds of play, collected by observing three types of bots included in PAP, playing amongst themselves. The real hand group and hand types were extracted from this session, and compared with the hand groups and hand types inferred by the BN, to produce the results described in section 7.2. The tests were designed to measure the accuracy of the BN predictions, but also to detect patterns when the BN predictions are wrong. This will be used as a starting point for further discussion in chapter 8.

The combined system was tested through live play in the Poker Academy Pro software by creating a poker table consisting of 8 players. These 8 players were 5 instances of Pokibot's, 2 instances of Simbot's and the BayCaRP system. Each testing session was run over approximately 4.500 rounds. Both versions of the BayCaRP system described earlier were tested under equal circumstances with the same initial case-base.

7.2 Testing the BN

The results described below are measured at the end of each stage of the game after all the information at that stage is observed. The hand group prediction accuracy is measured at the end of the preflop stage, since all the information required to making this prediction is observed at that time, while the hand type prediction is tested at the end of the flop, turn and river stage respectively.

7.2.1 Hand group prediction

Table 7.1 shows the result of testing the hand group prediction accuracy at the end of the preflop stage. This test was performed with 2069 testing instances, representing the number of times a player is still active in the round after the preflop, in the data set used for testing. The displayed result is a measurement of how many percent of the times the BN correctly predicts the opponent's hand group inside the five categories shown in the table. Our system currently uses the top four hand group predictions for further reasoning, which constitutes a prediction accuracy of 77%. The last category in the table (Top5-10) constitutes a faulty hand group prediction in our system, which will negatively affect further reasoning in the system. As shown in table 7.1, the BN incorrectly predicts the opponents hand group 23% percent of the time, which will be further analyzed below.

Hand group	**Top 1**	**Top 2**	**Top 3**	**Top 4**	**Top5-10**
Accuracy (%)	32%	50%	65%	77%	23%

Table 7.1: Hand group prediction accuracy

To detect any patterns when the BN fails to predict the correct hand group among its top 4 predictions, the 23% of the instances it failed to predict was used for further analysis. Figure 7.1 shows the percentage of the times the position node is at a given state, when the BN fails to predict the hand group. The figure shows that 44% of the times it fails, is when the opponent's position is either the small-blind or the big-blind, shown as *Blinds* in the figure. Another thing to note from this figure, is that the number of times the BN fails has a positive correlation with position of the opponent (excluding the blinds) . The later the position of the opponent, the more times the BN fails to predict the hand group.

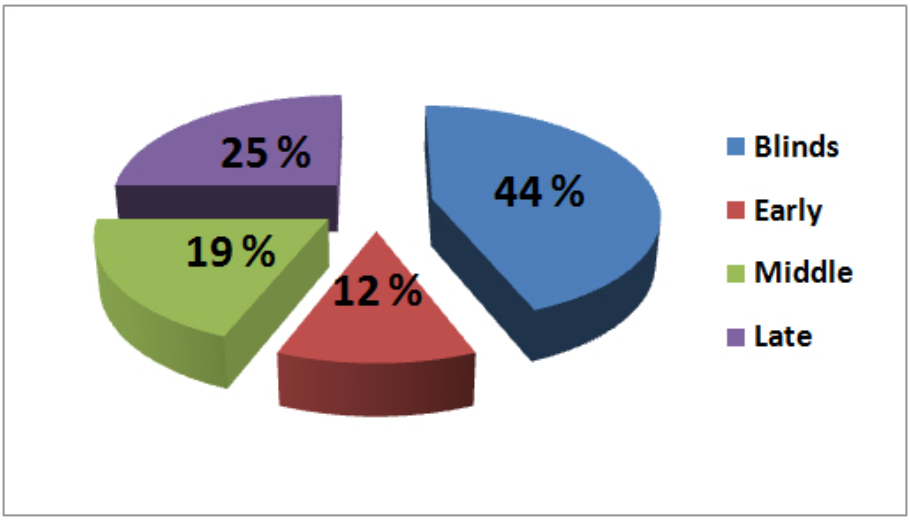


Figure 7.1: Fault (%) distributed over the states of the Position node

The same process was also used for the action of the opponent, to detect if there is a trend on which actions the BN fails the most. The result from this test, as shown in figure 7.2, indicates that the *Call* action is a major influencing factor of the faulty BN predictions, with 67% of the faults happening when the opponent performs the call action. The *Raise*

action constitutes 24% of the faults, while the two most aggressive actions, *ThreeBet* and *Cap*, have a very low fault rate.

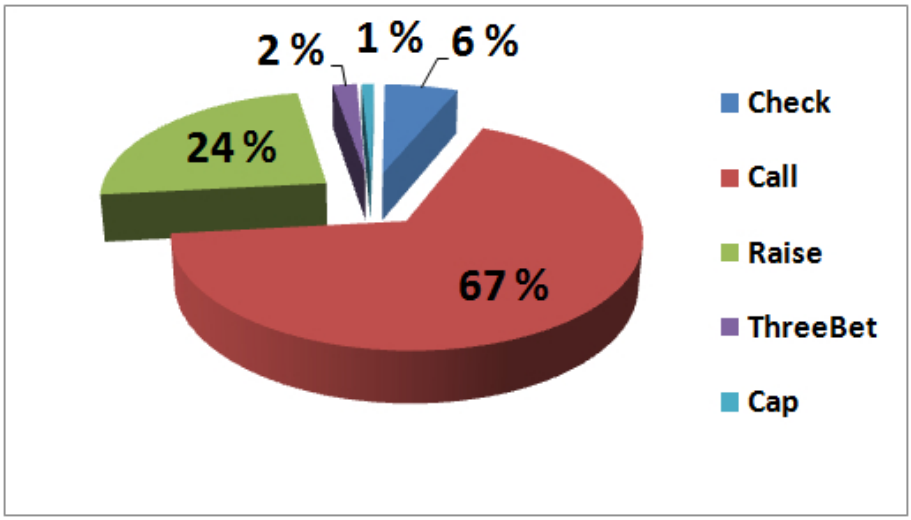


Figure 7.2: Fault (%) distributed over the states of the Action node

Table 7.2 shows at what combinations of states of the action and position node, where the highest percentage of errors occurs. The *ThreeBet* and *Cap* actions are omitted from the table because they have a very low error rate irrelevant of what position the action is performed. The *Check* action can only be performed by the big-blind, and has a error rate of approximately 6%. Table 7.2 shows the error rate of the *Call* and *Raise* action at different positions. The table shows that 32% of the errors occur when the opponent is positioned at one of the blinds and performs a call action. This table also shows the positive correlation effect between the BN's fault rate and the position, as was described earlier.

	Position			
Action	**Blinds**	**Early**	**Middle**	**Late**
Call	32%	6%	12%	16%
Raise	4%	5%	6%	8%

Table 7.2: Fault (%) at different state combinations of the Action & Position nodes

The results described in this section will be used for discussion in section 8.1.1. The next section is dedicated to the purpose of measuring the accuracy of the inference of an opponent's hand type.

7.2.2 Hand type prediction

Table 7.3 shows the percentage of times the BN classified the opponent's hand type as one of its top 1, top 2, top 3 or top 4-17 most probable opponent hand types. The last category would represent a misclassification in our system, since the rest of the reasoning in the system relies on the top 3 most probable opponent hand types. These percentages

are calculated from the testing data set based on the players still active in the round at the end of each stage, resulting in 1453, 1115, 943 instances of data used for testing the flop, turn and river hand type prediction accuracy. Based on this data, the BN falsely predicts the opponents hand type 22% at the flop stage and 28% of the times at the turn and river stage. Another thing to note is that the BN correctly predicts the opponent's hand type as the most likely hand type 50% of the times at the flop stage. As with the hand group testing, the data that resulted in a faulty prediction was used for further analysis.

	HandType			
Stage	**Top 1**	**Top 2**	**Top 3**	**Top 4-17**
Flop (%)	50%	67%	78%	22%
Turn (%)	44%	61%	72%	28%
River (%)	42%	59%	72%	28%

Table 7.3: Hand type prediction accuracy

The most prominent trend witnessed while running a series of tests designed to extract information relating to the faulty prediction, was that 40% of the times that the BN failed to predict the opponent's hand type among its top 3, the most likely opponent hand type inferred from the BN was the *Busted* hand type. A player is holding a busted hand when the player's hole cards combined with the community cards do not form another hand type. A busted hand is the weakest possible hand in poker, and is beaten by any pair.

To further analyze this error we observed how the probability distribution over the hand type node for the flop stage changed when only inserting a single action as evidence. The two most important probabilities related to this test, is the probability of the opponent having a busted hand given a bet or a raise action as evidence:

- P(Handtype=*Busted*|ActionFlop=*Bet*) = 18%
- P(Handtype=*Busted*|ActionFlop=*Raise*) = 13%

This means that when only observing the action at the flop stage, the BN would infer the busted hand type as the most likely opponent's hand type given a bet action, and as its top 3 most likely hand types when observing a raise. This effect is also observed to a lesser degree at the turn stage. These results will form the basis of the discussion in section 8.1.2.

7.3 Testing BayCaRP

The results described below tests the system from a poker perspective. Each of the two main sections are dedicated to testing the two different versions of the system. The first is BayCaRP1 with a outcome based reuse strategy, and the second is BayCaRP2 with majority voting based reuse strategy. The term big-blinds per hand (*bb/h*) was used to measure how profitable BayCaRP was during its approximately 4.500 rounds of play. Big-blinds per hand refers to how many big-blinds the player wins or loses for each round played. The reason for using big-blinds or small-blinds per hand as a measure of profitability

compared to money won per hand is that it provides a measure that can be used for comparison between games with different limits. Consider e.g. a player winning 1$ per round at the limits 1$/2$ and a player winning the same amount at a 10$/20$ limit. These two players would win the same amount in money, but would have a bb/h win rate of 1.0 and 0.1 respectively, which provides a more accurate measure on how profitable each of the players are at each of their limits.

All of the following tests were played as a 1$/2$ limit game, which means that 1 bb/h equals 1$ per hand.

7.3.1 BayCaRP1 outcome reuse

Figure 7.3 illustrates a plot of BayCaRP1's winrate (bb/h), while figure 7.4 shows some additional statistics for each player based on the whole session.

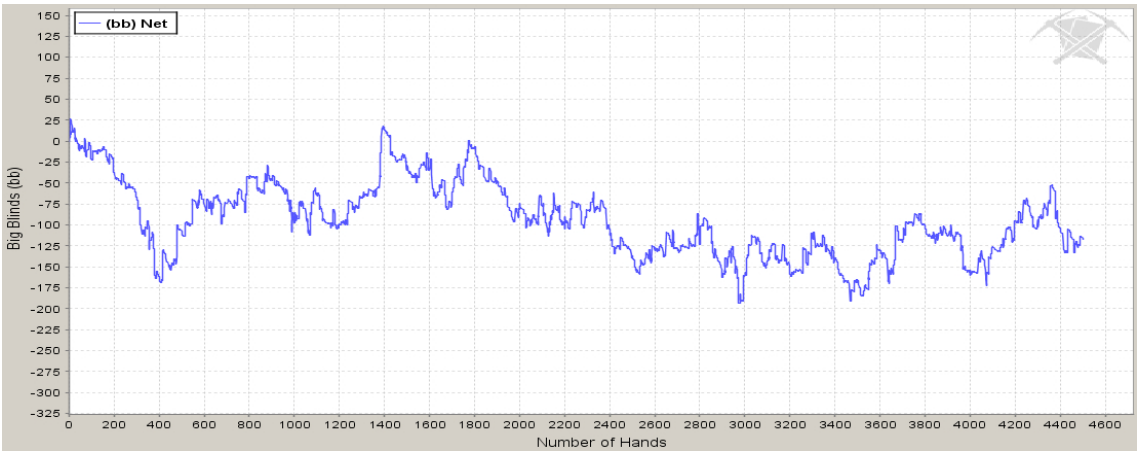


Figure 7.3: BayCaRP's bb/h with outcome based reuse

The highs and lows in the plot can be attributed to the variance in the domain, resulting from poker's statistic nature. Still, by disregarding the variance in the plot, we can see that the graph is decreasing slightly. As seen in figure 7.4 BayCaRP1 loses an average of -0.026 bb/h, which in this case represents 0.026$ per hand. For comparison, the most profitable player at the table won an average of 0.111 bb/h, and the least profitable player lost an average of -0.158 bb/h. The most profitable player at the table was an instance of the PokbiBot (Karma), while the least profitable was an instance of SimBot (Absinthe).

Player	Hands	Net $	Played %	Win %	Uncontested Wins	Showdowns Seen	Showdowns Won
Absinthe	4501	-$709,00	29%	14%	9%	10%	48%
BayCaRP	4501	-$116,75	26%	14%	10%	6%	63%
Doyle	4501	$169,00	23%	12%	7%	10%	46%
Erasmus	4501	-$384,00	18%	10%	6%	8%	41%
Frida	4501	$213,92	29%	15%	9%	11%	48%
Hari	4501	-$79,09	17%	9%	5%	7%	53%
Jake	4501	$406,92	43%	20%	13%	14%	51%
Karma	4501	$499,00	18%	10%	6%	7%	49%

Figure 7.4: BayCaRP outcome based reuse: session statistics

Another thing to note from figure 7.4 is that two instances of Pokibot, Karma and Erasmus perform very differently. Some of this can be attributed to the variance in the domain, while some should be attributed to the way the different instances are configured. The various Pokibot instances are configured to play slightly different, such as Karma's configuration makes it more inclinded to raise after the flop than e.g. Erasmus[1].

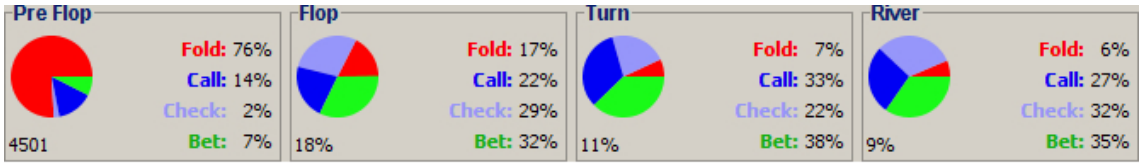
Although more rounds should be played to reduce the variance to a minimum, the results indicates that BayCaRP1 loses slightly over time at the given table. Still, this loss is quite small, which means that BayCaRP almost plays evenly with the Pokibot's and Simbot's.

Figure 7.4 also shows us that BayCaRP plays 26 % of its hand, which is close to the average play percentage of 25% at the table. BayCaRP1's percentage of uncontested win is 10%, which is above the average of 8%. The high percentage of uncontested wins indicate that BayCaRP1 has an aggressive playing style, which makes the other players fold. Another important thing to note from the figure is that BayCaRP1 wins 63% of the times when it goes to showdown, which is the highest at the table. Although a high win percentage at showdown is good, BayCaRP only chooses to go to showdown 6% of the times he plays a hand. This low showdown percentage may indicate that BayCaRP1 has learned to fold too many of the hands it chooses to play from preflop.

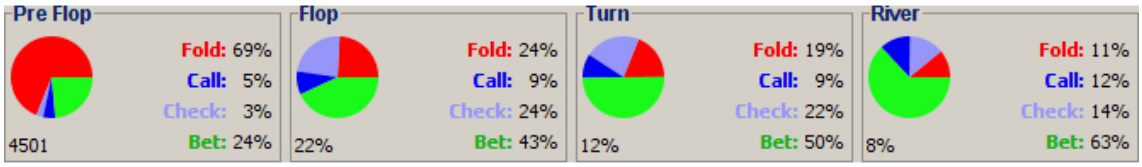
The subsequent section attempts to discover what parts of BayCaRP1's playing style are less than optimal by comparing BayCaRP to the most profitable player at the table.

BayCaRP1 vs Karma comparison

Figure 7.5 shows the action frequency at each stage of the game for Karma and BayCaRP1. As shown here BayCaRP1 plays his cards more aggressively than Karma, observed by the high percentage of the bet action at each stage. The reader should also note the large difference in the fold and call frequency between the two players. This becomes especially clear at the turn stage, where BayCaRP1 folds 19 % of his hand compared to Karma's 7%, and calls 9% compared to Karma's 33%.



(A) KARMA



(B) BAYCARP

Figure 7.5: Action at each stage for: (a) Karma (b) BayCaRP1

These results may indicate that BayCaRP1 folds to often in cases where its hand may have had a decent chance of winning. The results described here were used as a basis for

[1]The configuration of the various instances can be observed in the .pd file in the /logs/players folder

analyzing which hand types BayCaRP1 loses the most money on at each stage compared to Karma. The difference of how much money the two players win or lose with each hand type was largest at the flop stage. Figure 7.6 and 7.7 illustrates this graphically for each player. The y-axis shows the amount of big-blinds won or lost per hand type (x-axis).

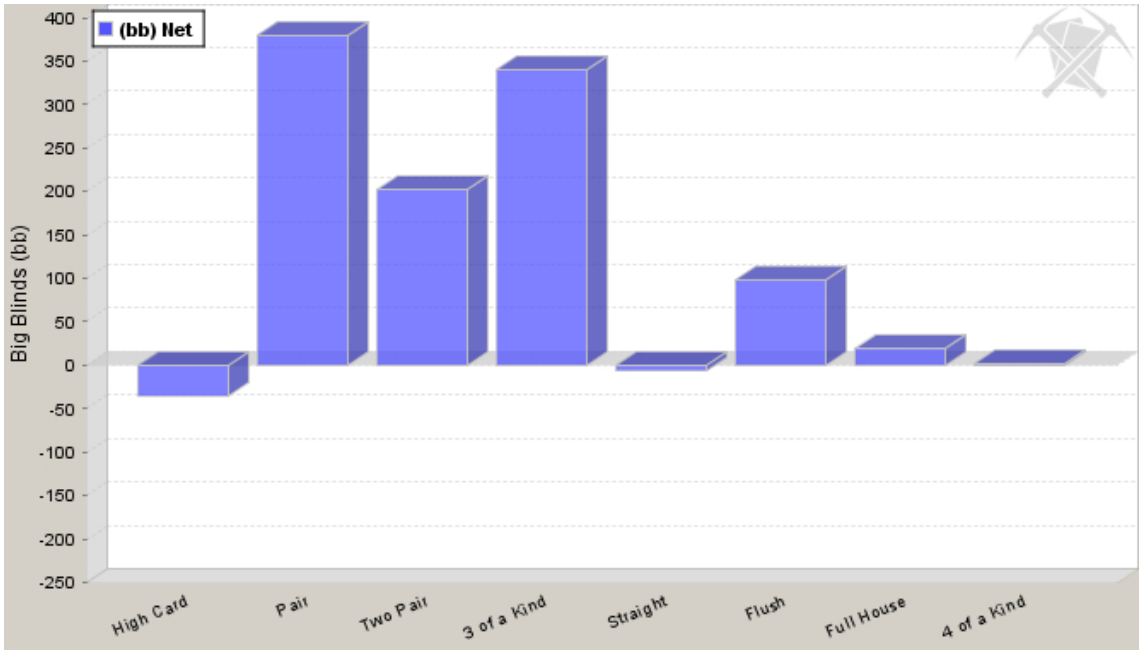


Figure 7.6: Karma's winrate(bb) per hand type

Figure 7.6 shows Karma's winrate with various hand types at the flop stage. The hand type referred to as *High Card* is essentially the same as what has been referred to as a *Busted* hand type in this thesis. The cards that typically fall into this category when played by the University of Alberta bots, is either having atleast one hole card that is of greater rank than the community cards, or a drawing hand, such as straight or flush draw. This hand type is the only one that Karma has a negative winrate with at the flop stage. The reader should also note the high winrate Karma has with any *Pair* at the flop stage.

When comparing this to BayCaRP1's winrate with the different hand types (figure 7.7) the first striking difference is the winrate with the *High Card* and pair hand types. Although both players have a negative winrate for the high card hand type, BayCaRP1 loses over 6 times as much as Karma. When holding a pair, BayCaRP1 loses approximately -35 big-blinds over the session, while Karma gains 380. Also notable is the fact that Karma has a higher winrate with strong hands at the flop stage, especially prominent with the three of a kind hand type.

The combined results described in this section provides us with some pointers where BayCaRP1's playing style may be flawed when using the outcome based reuse policy. These results will be used for discussion in section 8.2.1.

7.3.2 BayCaRP2 majority reuse

BayCaRP2's performance with the majority vote reuse policy is shown in figure 7.8. This version of BayCaRP2 has a winrate of 0.056 bb/h, which is a vast improvement over the

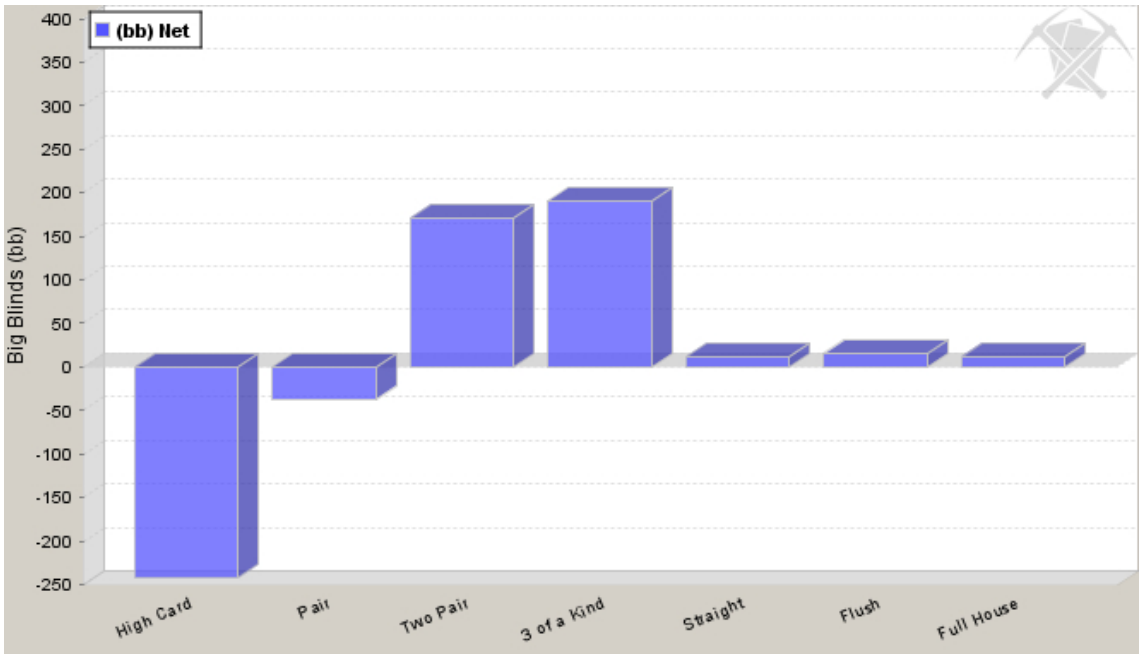


Figure 7.7: BayCaRP1's winrate(bb) per hand type

previous version. BayCaRP2's results indicate that it is indeed a winning player at a table with Pokibot's and Simbot's, in fact it is the third most profitable player at the table.

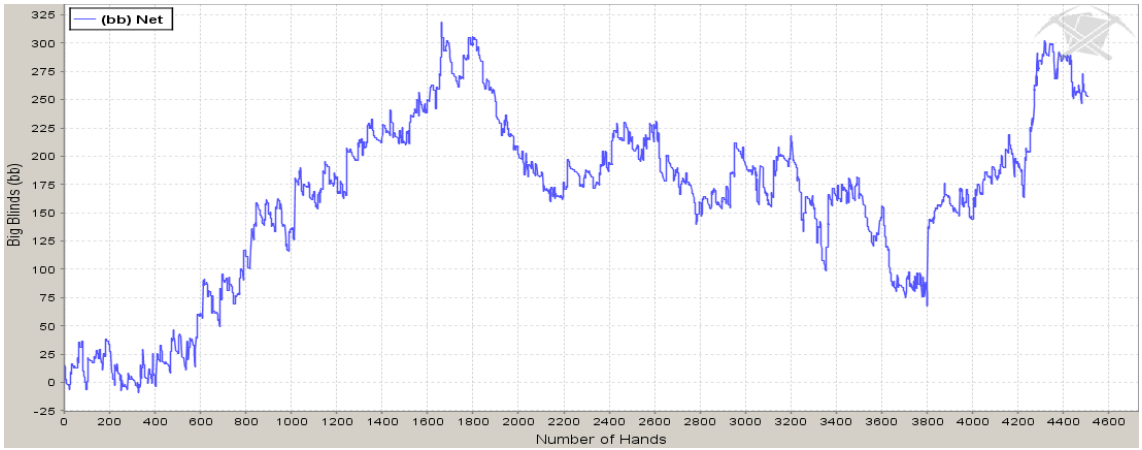


Figure 7.8: BayCaRP2's bb/h with majority vote reuse

Figure 7.9 shows some general statistics for BayCaRP2's testing session. By comparing this to the previous version, we see that BayCaRP2 has a lower percentage of uncontested wins indicating that he plays less aggressively. The second thing to note is that he wins less showdowns, but reaches showdown more frequently.

By observing the action frequency of BayCaRP2 in figure 7.10, we see a smaller fold percentage, especially at the turn and river stage, and a higher call percentage. This action distribution has a close resemblance to the Pokibot in figure 7.5.

Figure 7.11 shows BayCaRP2's winrate with the various hand types. Although this

Player	Hands	Net $	Played %	Win %	Uncontested Wins	Showdowns Seen	Showdowns Won
Absinthe	4506	-$461,70	28%	12%	6%	11%	50%
Anders	4506	$92,88	29%	13%	7%	12%	48%
BayCaRP	4506	$252,71	25%	11%	7%	9%	50%
Doyle	4506	-$341,75	21%	9%	5%	10%	41%
Erasmus	4506	$226,00	19%	9%	5%	8%	50%
Frida	4506	$618,91	29%	13%	7%	12%	51%
Hari	4506	-$447,25	19%	8%	4%	8%	51%
Jake	4506	-$236,80	42%	17%	10%	15%	46%
Karma	4506	$297,00	19%	9%	5%	9%	49%

Figure 7.9: BayCaRP2 majority vote reuse: session statistics

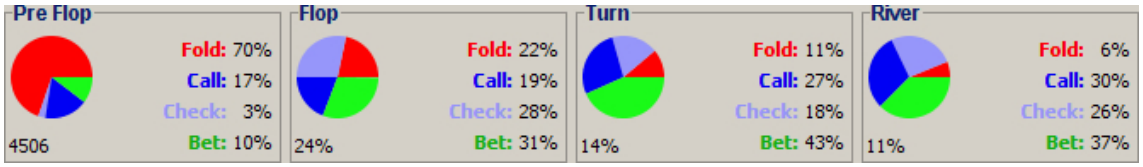


Figure 7.10: BayCaRP2's action frequency at each stage

version actually loses almost double the amount of the previous version when holding the hand type *High Card*, it also wins more with strong flop hand types such as two pair and three of a kind. The most notable difference is the winrate when holding a pair at the flop. This has become one of BayCaRP's big earners, in contrast to the previous version, where pair had a negative winrate.

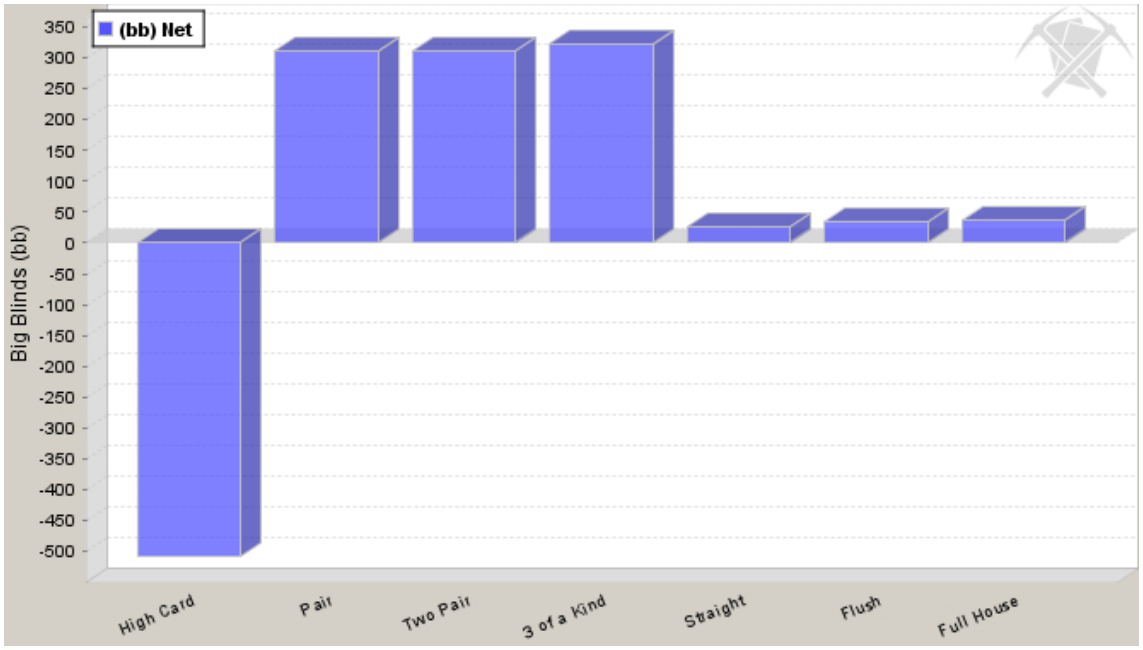


Figure 7.11: BayCaRP2's winrate(bb) per hand type

The results described here will be used for discussion in section 8.2.2.

CHAPTER 8

Discussion

This chapter presents an evaluation of the results obtained in the previous chapter, and discusses weaknesses, strengths and potential improvements. The chapter begins by discussing the strength of the BN, section 8.1, before it moves on to discussing the overall system in section 8.2. Section 8.3 discusses how the proposed architecture can be utilized in the TLCPC project.

8.1 The strength of the BN

The topology of the BN was modeled with expert knowledge acquired by a mixture of prior experience and textbook knowledge. This section will discuss the strengths and limitations of the implemented Bayesian network. The section begins by discussing the results obtained by testing the network in section 7.2 from the perspective of predicting the opponent's hand group (section 7.2.1) and hand type (section 7.2.2). Section 8.1.3 discusses the abstraction the BN introduces by discretizising game states and opponent's cards.

8.1.1 Hand group prediction

The testing of the Bayesian network's hand group prediction accuracy showed that the BN correctly predicted the opponent's hand group among the ones that are used for further reasoning 77% of the time. This is not bad considering the uncertainty introduces in poker in the form of players intentionally underplaying or overplaying their cards, referred to as *bluffing*. This means that a player with a very strong preflop hand can decide to play the hand more passively than expected, or a player with a weak hand can play a bad hand more aggressively than expected. Still, by analyzing the times the BN failed, we discovered areas where the BN could be improved to provide a higher prediction accuracy.

The most notable error relating to the players position that appeared in the testing phase, was that 44% of the faulty predictions were made when the player's position was one of the blinds. This high error percentage at the blinds made sense when relating this to the action with the most errors, the call action. Together, these results highlighted the need for differentiating the call action more than was done in the currently implemented BN. To fully grasp this thought consider the player at the small-blind performing the call action. If no player has previously raised the pot, then he only needs to double the amount he currently has committed to the pot to continue playing in the round, because he already has paid the small-blind, which constitutes half of the big-blind. Compare this to

a player in another position performing the same action, which would cost him the double of what the player in the small-blind voluntarily put in the pot, or if the pot has previously been raised, then the player calling this would have to commit even more to the pot. When training and using the BN all of these actions would be considered equally as a call action. As described in the design of the BN, the BN uses the most aggressive action of player, but as discussed here this does currently not work optimal for the call action. The reason for this is that when the player only performs a call action, the difference between the amounts of how much the various call actions costs a player is abstracted away in the BN.

The other effect observed while testing the BN was the positive correlation between the number of times the BN fails to predict the opponents hand group and the position of the opponent. This effect is as expected in the eyes of a poker player, because at an earlier position a player will play a smaller range of cards than he would in a late position, thus making the prediction easier when the player performs an action from an early position. A player at the late position has the luxury of observing what the majority of the players do before he acts. This means that if most of them fold, he could decide to e.g. raise a hand he would not normally play to attempt to scare the rest of the players into folding, thus successfully stealing the money that was forced onto the table in the form of the blinds.

The results also show us that the more aggressive an action is, the less faulty predictions are produced by the BN. This effect is also as expected because the more aggressive the action, the more money you put in the pot, the better cards you need to have to win in the long run. This means that an aggressive action narrows the possible range of cards for the opponent making it easier to predict.

A possible solution to the high fault percentage related to the call action, is to further specialize the action into several categories, such as the raise action was specialized into *Raise*, *ThreeBet* and *Cap*. The call action would then be represented as different states in the action node, depending on the situation it was performed in, such as a state for; calling an amount equal to the small-blind, calling the regular amount, or calling a raise.

An alternative to this solution would be introducing a node for representing the pot odds the player receives at the time of the action. Since pot odds is a numeric measure that takes into account how much money the player needs to put into the pot to continue playing, it would help in the process of differentiating various call actions, based on the cost required to perform them.

The effect regarding the higher error rate when inferring an opponent's hand group from passive actions and at late positions, may lead us to the belief that the number of hand groups used as output from the BN should be based on a threshold probability value rather than a static number than is currently used. Optimally this would result in using only one or two hand groups for further reasoning, when the BN predicts the opponent's hand group with a high enough probability, e.g. in the case of a aggressive enough action, while using a higher amount of hand groups when the probability is lower. This could e.g. be done by setting a specific threshold probability, and use the number of hand groups whose combined probability sum exceeds this threshold. For example if the probability of the most likely hand group given the evidence exceeds the threshold alone, then we only use this hand group for further reasoning. If, in contrast, the top 5 most likely hand groups are needed to exceed the threshold value, then we use all five. Using a threshold value would better represent the uncertainty in the Bayesian network's hand group prediction by allowing for an increase in number of hand groups if the prediction is unreliable, and a decrease when the prediction seems reliable.

8.1.2 Hand type prediction

The results of the Bayesian network's hand type prediction test showed good results, considering that the implemented BN omitted an important aspect of Texas Hold'em Poker, namely opponent's possibility of having partial hands. A possible extension to the implemented BN for representing this aspect of poker was described and illustrated in section 5.3.3. There were three major reasons for not implementing this extension to the BN. The first reason was that this project has been time consuming, and there simply wasnt enough time to take this extension to the level that it should be in the time frame that was available for this project. The second reason was that the data used to train the network was not optimal for partial hand type prediction. The reason for this was that we used rounds that went to showdown to train the network, and since partial hands that do not evolve into complete hands in that time often get folded before showdown, the resulting BN was biased towards partial hands evolving into complete hands more times than it realistically would. Since a majority of the partial hands get folded before showdown, the resulting BN would also be trained to infer a lesser likelihood for partial hands than it should. The third and final reason was that the additional complexity introduced more time spent when using the BN for inference in the implemented system, and since the results of this extension was less than optimal, the implementation of this part was not given a priority.

Although this extension had its limitation, it was still included in the design of this system as a suggestion of how Bayesian Networks could be used to predict partial hands. We believe that with some more work, this design could successfully be used to predict opponent's partial hands. The most apparent area for future work on this particular problem, is to supplement the BN with additional training data, so that the network's partial hand probabilties better reflect reality at a specifig stage of the game. Why and how the lack of representing the partial hand aspect of the game affects the implemented BN will be discussed in the next paragraphs.

The testing in section 7.2.2 highlighted how the lack of partial hand prediction degraded the prediction accuracy of our implemented BN. The results showed that 40% of its erroneous predictions at the flop stage appeared when the busted hand type was inferred as the most likely from the BN, and the same effect was observed at the turn stage. To further analyze this trend we observed the probabilities of the busted hand type, given an aggressive action from the opponent as evidence, which showed a probability of 18% for the bet action and 13 % for a raise.

These results confirmed our initial assumptions, that these errors were related to the lack of partial hand prediction. The high probability for the busted hand type even when aggressive actions are observed, tells us that the training data contained a lot of these cases. Bluffing can account for some of these cases, but as bluffing is less used in fixed-limit Texas Hold'em, because of the restriction on how much money you can bet, there has to exist another reason for this trend.

The main reason for this trend, is that since the currently implemented BN has no way of modeling partial hands, it will attribute e.g. a bet or a raise to a busted hand type, when in reality the opponent is possessing a strong partial hand. From the perspective of the BN, this will lead to a higher probability for a busted hand than it should be, since the BN was trained with data that did not contain information about partial hands.

This section highlighted the need for including partial hand prediction in the BN, to further improve its accuracy. This can be achieved by supplementing the BN with training

data from another source, to achieve more realistic probabilities of an opponent holding a partial hand at a particular stage of the game.

8.1.3 Abstraction in the BN

A limitation of Bayesian network's in poker discussed in both Korb et. al's and Terry and Mihok's research, is the decrease in accuracy of the model introduced by abstracting the game state into various categories. This abstraction of the game state is performed to reduce the complexity of the network, which is important because poker betting decisions often have a relatively short time restriction.

Although this issue is also present in the proposed network, the combination of the components in this architecture reduces this problem to some degree. First of all, since the Bayesian network is not concered with the betting decision itself, a lot of the features that would typically need to be abstracted into categories is now present as features in the cases of the CBR component. Still, the BN abstracts the state of the community cards, the opponent's hole cards and hand types, which will result in the loss of some information.

Consider the abstraction of the opponent's hole cards into the hand groups described earlier. By using the RBR component as an intermediary between the BN and the CBR component, no loss in information occurs, because the hand groups are transformed into the hole cards they represent before being used for further reasoning. The hand types could be used in a similar manner, e.g. by using the BN's hand type predictions to refine the inital hand group prediction.

For simplicity, consider a scenario where the BN inferred the opponent's most likely hand group to be hand group 2, which would result in the hole cards; TT, AQs, AJs, KQs and AK being used for the calculation of the relative strength feature of a case. If the BN then inferred that the opponent's most likely hand type was TwoPair, and the current community cards where AJK, we could remove the hole cards that do not form this hand type from the hand strength calculation, which in this case would be TT, AQs and KQs.

The current system uses the most likely hand types directly as input to the CBR component, which results in the loss of some information, since TwoPair could represent a variety of combinations that would form two pairs. Future work should encompass more testing on how this type of abstraction affects the overall system performance, and how to reduce this to a minimum.

8.2 The strength of the system

This section describes the results obtained by testing the whole system. Section 8.2.1 discusses the results with the outcome based reuse policy, while section 8.2.2 dicusses the majority voting results. Finally, section 8.2.3 discusses the overall results obtained in the testing phase.

8.2.1 BayCaRP1 outcome reuse

The results obtained by testing BayCaRP1 with the outcome based reuse policy described in the design chapter showed that BayCaRP1 loses -0.026 bb/h in the given testing session. Although this makes BayCaRP a losing play, it is not that far from playing evenly against the Pokibot's and Simbot's.

To further analyse this results consider the CASPER system described in the chapter concerning related research (section 4.3.2). CASPER was tested with different weighted features of a case, and with two different sized case-bases. CASPER with the most profitable weights will be used for further comparison. The CBR-component in the BayCaRP system shares a lot of similarities with the CASPER system, which makes it a good candidate for comparison. BayCaRP uses the weights obtained through the CASPER research, and has a similar case representation and retrieval step.

CASPER1, with a slightly smaller case-base than BayCaRP, obtained a result of -0,009 bb/h, while CASPER2, with a larger case-base, won 0,004 bb/h[1]. Comparing this to the results obtained by testing BayCaRP1 with the outcome based reuse policy we see that BayCaRP1 performs worse than both versions of CASPER. At first sight this results may seem confusing, since BayCaRP1 incorporates opponent modeling through the Bayesian network, while CASPER does not. But, as shown by the results in section 8.2.2, BayCaRP's performance increases a lot when provided with a different reuse policy. The decrease in performance can therefore be attributed to the outcome based reuse policy. Why this is the case will be described below.

Consider the run time example from section 6.3. In this example BayCaRP1 played perfectly, by betting as much as possible when it had the best hand, and minimizing its losses by folding, when the river card against the odds turned the hand into a losing hand. With the outcome based reuse policy described earlier this would count as a negative case in the case-base, because the fold at the river would result in all of the previously used cases being revised to a negative outcome. This example illustrates how this reuse strategy affects the performance in a negative way, by potentially labeling good cases with a negative outcome, due to an unlucky end result. This could also happen the other way around, by bad cases receiving positive outcomes due to luck.

The results obtained by comparing BayCaRP1 to the most profitable Pokibot attempts to highlight how this reuse strategy affects BayCaRP's playing behavior. The results showed that BayCaRP wins 63% of the showdown it takes part in, which is a good attribute for a poker player. Still, the fact that BayCaRP loses over the long run combined with the knowledge that it only goes to showdown 6% of the times it chooses to play a hand, indicates that BayCaRP folds too many hands that may potentially evolve into a winning hand. The comparison to the most profitable Pokibot (Karma) made this clearer. As observed here BayCaRP1 folds a lot more hands than Karma at the postflop stages of the game. When comparing the two players winrate with various hand types, we also see a major difference in the winrate for the high card and pair hand types. Combined with the high win percentage at showdown, and the low percentage of seen showdowns, this tells us that BayCaRP folds a lot of his hands when his hole cards do not form a good hand type when combined with the community cards at the flop, but also that BayCaRP1 folds to many of its hands when it has a pair at the flop. In other words, this means that BayCaRP1 folds to many hands with potential of improving into a winning hand, such as two hole cards of greater rank than the community cards, or straight and flush draws. BayCaRP also folds to much when he has a pair that is not the highest possible ranking pair. This could e.g. be when BayCaRP1 is holding a pair of nines on a board where a jack is present among the community cards.

Although this reuse policy makes BayCaRP1 into the strongest player at the table when it comes to winning at showdown, it also degrades the performance by making

[1] bb/h is referred to as small-bets per hand(sb/h in the CASPER research

BayCaRP1 fold to many hands at the postflop stages. Since BayCaRP1 has already invested money at the preflop stages, the fact that it folds frequently when it does not evolve into a strong hand type at the flop stage, makes him a losing player over the long run. These results make it clear that judging the outcome of a solution purely on the money earned or lost by that action in the poker environment is not optimal. Still, the author believes that the idea of reusing the most profitable action is the ideal strategy for this domain. But to achieve this optimally domain knowledge is required to provide better evaluation criterion for how good a given solution is.

8.2.2 BayCaRP2 majority reuse

The results obtained by testing BayCaRP2 with a majority voting reuse strategy showed BayCaRP2 to be a profitable player in the testing environment, with a winrate of 0,056 bb/h. For comparison's sake, consider again the results achieved by the CASPER system in a similar environment. The CASPER system is similar to the CBR-component in our system, but it employs no form of opponent modeling, and relies on a probabilistic reuse policy. This reuse policy assigns a probability to each solution based on the solutions present amongst the cases selected in the retrieval step, which is not that different from the majority voting reuse policy. BayCaRP2 achieves a 14 times better winrate than the best CASPER version. Although the CASPER system was tested more extensively by playing almost 4 times more rounds than the BayCaRP system, the results indicate that BayCaRP achieves better performance with its combination of two reasoning methods, than any of the two could achieve on its own. The results obtained by the BayCaRP2 system shows the strength of the BN and CBR combined, when not using a reuse policy that degrades system performance.

When comparing the action frequency of BayCaRP2 with the previous version and Karma, we see that BayCaRP2's playing style resembles Karma's, while BayCaRP1 seems to have learned its own playing style. The reason for this is that when using a majority voting reuse policy, the reused solution will resemble the solutions used by the bots that were used to acquire the case-base, which in this case was Pokibot's and Simbot's. In contrast, the idea of the outcome based reuse policy was to surpass its teachers by detecting what solutions have the best outcome in a given situation. Although the majority voting reuse policy is profitable at the table with Pokibot's and Simbot's, it is still somewhat limited by how good the solutions of its teachers are.

8.2.3 BayCaRP overall

Although the system performance depends on the reuse policy chosen, the results indicate that the integration of Bayesian Network's and Case-based reasoning provide a good combination for decision making in poker. The results obtained by BayCaRP2 seems to outperform the CASPER system, which is solely based on Case-based reasoning. Since there to the authors knowledge do not exists any purely Bayesian network based attempts in a Poker Academy Pro multiplayer environment, it is hard to compare BayCaRP to such an attempt. Still, the limitations discussed in research related to applying Bayesian Network's to poker, such as the abstraction limitation discussed in section 8.1.3, makes the author believe that a combined reasoner will outperform such a system.

The run time example described in section 6.3 provides a good indication of the

strength of the combined system. The Bayesian network provides the case-based reasoner with a more extensive contextual information than it would manage on its own, making the system play this round close to perfection. With the information inferred by the BN, BayCaRP bets when it has the best hand, and folds when it realizes that the opponent has a better hand, thus attempting to maximize its profit when it has the best hand, and finally minimizing its loss when the hand is beat by the opponent's hand.

Despite the fact that BayCaRP achieves a good performance, there are still areas that should be improved. Firstly, BayCaRP plays his cards very truthfully, meaning that he bets with strong hands and folds with weak ones. Since being to predictable in poker is not a very good trait, BayCaRP should be extended to incorporate more deceptive plays. The check-raise action is one such deceptive play that is more heavily used by the Pokibot's compared to BayCaRP. The current implementations of BayCaRP has really no way of learning how to perform a check-raise profitably, which results in a negative winrate the few times it is used, compared to a positive winrate when used by the Pokibot's. By incorporating knowledge about deceptive plays such as check-raises somewhere in the system, may increase BayCaRP's winrate even more in situations where it has a really strong hand.

Secondly the time it takes for BayCaRP to make a decision is rather high compared to e.g. the Pokibot's. The time it takes to make a decision also increases when the complexity of the Bayesian network increases, and especially when the case-base increases. One approach for reducing the time spent on retrieval and similarity computation of previous cases, is to provide the retrieval step with some form og inital matching process. This process should quickly remove cases that would result in a low similarity score with the new case. This could be done by e.g. analyzing the case features with high weights, and removing the cases that have a large distance compared to the these features in the new case. Another such approach would be structure the case-base hierarchically with generalized cases. These generalized cases should contain a form of a summary of cases in the case-base with a high enough similarity. The generalized cases could e.g. contain information on how many times each solution is present among the cases it represents and the combined outcome for each such solution.

8.3 TLCPC and our design

TLCPC aims to develop a decision support system for classification and treatment of pain. Decision making in the medical domain is to a large degree characterized by uncertain and incomplete information. These characteristics are also shared by the domain selected for this thesis. Texas Hold'em encompasses both aleatory uncertainty, by the stochastic nature of poker and epistemic uncertainty and incomplete information by the unknown values of hidden cards. Both of these types of uncertainty are also a focus area in Bruland et. al's research.

The design in this thesis uses the Bayesian network to provide the case-based reaoner with additional contextual information, relating to the opponent's most likely cards. A similar approach may also prove to be efficient in the medical domain, where e.g. the Bayesian network infers information about the patient's pain classification, which is used by the case-based reasoner to retrieve more relevant past experiences containing suggested treatments. Although our system is fully automated, it might as well work as a decision support system, by providing information to the user, rather than making the decision

itself.

Our system has succesfully shown a way to integrate Bayesian Networks and Case-based reasoning in a domain with similar characteristics. The good results obtained by this integrated reasoner support the idea that our design can bring the TLCPC system added value.

CHAPTER 9

Conclusion

The field of knowledge based system has experienced an increasing trend towards integrating various knowledge and reasoning methodologies to achieve better performance than either could achieve on its own. The belief that integrated reasoning can achieve higher performance combined with the ongoing TLCPC project constitute the motivational factors for this research. In this thesis we presented an integrated reasoner for imperfect information games. The presented system uses the combination of Bayesian Networks and Case-based reasoning to make informed betting decisions in the game of Texas Hold'em Poker. There exists some attempts to applying the two reasoning methodologies separately in the domain of Texas Hold'em poker, where the research performed by Rubin and Watson[36] on the CASPER system was the most prominent inspirational factor for our designed system. Our research attempts to show the combined strength of integrated reasoning through Bayesian Networks and Case-based reasoning in a domain such as Texas Hold'em poker, and is to the authors knowledge the first system combining these two effectively for decision making in poker.

The result of this research was a system nicknamed BayCaRP, which employs Bayesian Networks and Case-based reasoning to play poker at a table with multiple players. The Bayesian network is concerned with the opponent modeling aspect of poker, more specifically predicting the opponent's most likely cards at each stage of the game. The result of the inference performed by the BN is used to provide additional contextual information to the case-based reasoner, which is responsible for the final betting decision.

BayCaRP's initial case-base is acquired by observing bots developed at the University of Alberta play amongst themselves. When BayCaRP is required to make a betting decision it acquires information about the state of the game, including information relating to its opponent's, such as their actions. The game state information required by the BN is abstracted into different categories through a rule-based component. This information is used as evidence in the network to infer an opponent's most likely cards, which is used as input to the CBR system. The CBR-component uses a part of this information directly from the BN, while the other part is used to calculate the strength and future potential of BayCaRP's hand in relation to what the BN predicts the opponent's most likely cards to be.

Our research suggests that solutions of similar cases are reused based on previous outcomes of applying a given betting decision, but experiments with different strategies for reusing solutions from the most similar retrieved cases. This part of the research highlighted the need for domain knowledge to better evaluate the outcome of a solution. Without this, the outcome based reuse policy degrades system performance.

The results represted in this thesis show the effectiveness of combining Bayesian networks and case-based reasoning for a domain with characteristics such as uncertainty and imperfect information. The results obtained by BayCaRP indicates that it plays profitably against University of Alberta's Pokibot's and Simbot's, and achieves better performance than the CASPER system, which is a purely case-based reasoning approach.

9.1 Future work

Although the overall system achieved great performance, there are still areas of the system that could be improved with further work. Since some of these issues have already been discussed earlier, this section will provide a summary of these including some new improvements that can be made to further enhance the system performance. We first provide a description of improvements that can be made related to the Bayesian network, followed by improvements to the case-based reasoner before we describe further work for the overall system.

The Bayesian network

- Recall that the majority of the faulty hand group predictions from the BN was when the opponent performed a call action. The reason for this was that the BN did not differentiate the call action enough, which means that it represents a 0,5$ and 2$ call action as the same state in the network. This resulted in a higher percentage of faulty predictions related to this action. Future work should be done to resolve this issue, by e.g. differentiating the call action more or adding nodes for representing the size of the pot. This issue could also be present at other states in the network leading to decreased prediction accuracy. More work should also be done to further test and resolve this potential issue at other states in the network.

- The extension to the BN for predicting opponent's partial hands should be implemented and tested. This requires another source of data for training this part of the network, since the showdown data is not applicable for this task. The reason for this is that partial hands are mostly folded before showdown if they do not evolve into a real hand type. This results in the BN infering wrong probabilities relating to partial hands.

- Although the BN achieves a decent prediction rate for different types of bots from the University of Alberta, more work should be done to achieve better performance for different types of opponent's, which would be especially relevant when playing against human competition. This could be done by e.g. extending the network with more features such as VPIP and PFR relating to the other stages of the game, such as a numeric measure describing the opponent's aggression factor for each stage of the game. A different strategy could be to use these kinds of features to classify an opponent's playing style into a set of groups. The idea should be that players inside such a group would have similar playing behaviors. Each of these player groups could then have its own BN specialized to fit that type of players. Another possibility would be to have a network for each new player, starting with data to fit an average player, and then using the evidence observed by that player to heavily influence the current parameters in the network.

The case-based reasoner

- Future work for the case-based reasoner should encompass work on reducing the time spent on retrieving similar cases without decreasing performance. This could be achieved by organizing the case-based hierarchically with generalized cases representing a set of cases which are sufficiently similar. In this architecture the generalized cases should contain information about the frequency of each of the actions represented by the similar cases, and the combined outcome of applying these actions. A different method would be to provide the retrieval step with the necessary logic to remove cases that would result in a low similarity with the new case. This could be done by e.g. removing cases which have a big distance in the heavily weighted features compared to the new case. If done efficiently this would reduce decision making time, by only calculating the full similarity measures for a subset of the case-base.

- As discussed in this thesis, the reuse policy chosen and the way outcome is represented greatly affects system performance. Further work should be done on this area, especially on providing the system with domain knowledge to efficiently judge the outcome of a solution. The approach described in the section 5.4.5 should be implemented and tested to provide cases with a better outcome measure.

The overall system

- Further work on the overall system should experiment and test different ways of BN and CBR interaction. The current design uses hand group prediction to calculate relative hand strength, and the hand type prediction as separate features in a case. Another way of doing this would be to use hand type predictions to further refine the opponent's most likely hole cards. For example if the opponent's most likely hand type is two pair, the hole cards resulting from the hand group prediction that do not form this hand type when combined with the community cards could be removed to further enhance the relative hand strength feature. Which of these approaches provide the best performance should be further tested.

- The system should be tested more extensively against different types of opponent's, including human players. The Open Holdem software provides the required interfacing mechanism for doing so. The results acquired by these tests should be used as a basis for further improvements of the system.

- The current implementation of the system plays its cards very truthfully, which may make it easier exploitable by opponent's. Work should be performed to make BayCaRP more deceptive in its playing behavior, e.g. by including actions such as the check-raise. This could be done by representing combined actions as a solution of a case, rather than only atomic actions.

References

[1] NTNU, "Translational research in lung cancer and palliative care - from genomics to symptom control (tlcpc)," tech. rep., 2008.

[2] A. Aamodt and E. Plaza, "Case-based reasoning; foundational issues, methodological variations, and system approaches," *AI COMMUNICATIONS*, vol. 7, no. 1, pp. 39–59, 1994.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[4] T. Bruland, A. Aamodt, and H. Langseth, "Architectures integrating case-based reasoning and bayesian networks for clinical decision support," in *Intelligent Information Processing V* (Z. Shi, S. Vadera, A. Aamodt, and D. Leake, eds.), vol. 340 of *IFIP Advances in Information and Communication Technology*, pp. 82–91, Springer Boston, 2010.

[5] D. Billings, J. Schaeffer, and D. Szafron, "Poker as a testbed for machine intelligence research," in *Advances in Artificial Intelligence*, pp. 1–15, Springer-Verlag, 1998.

[6] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artificial Intelligence*, vol. 134, p. 2002, 2001.

[7] C. Marling, M. Sqalli, E. Rissland, H. Muñoz-Avila, and D. Aha, "Case-based reasoning integrations," *AI Magazine*, vol. 23, no. 1, 2002.

[8] A. Aamodt, "Explanation-driven case-based reasoning," in *Topics in Case-Based Reasoning* (S. Wess, K.-D. Althoff, and M. Richter, eds.), vol. 837 of *Lecture Notes in Computer Science*, pp. 274–288, Springer Berlin / Heidelberg, 1994.

[9] A. A. Helge and H. Langseth, "Integrating bayesian networks into knowledge-intensive cbr," in *In American Association for Artificial Intelligence, Case-based reasoning integrations; Papers from the AAAI workshop. Technical Report WS-98-15. AAAI Press*, pp. 1–6, 1998.

[10] D. Aha and L. W. Chang, "Cooperative bayesian and case-based reasoning for solving multiagent planning tasks," tech. rep., Navy Center for Applied Research in AI, 1996.

[11] S. Kaasa, J. H. Loge, P. Fayers, A. Caraceni, F. Strasser, M. J. Hjermstad, I. Higginson, L. Radbruch, and D. F. Haugen, "Symptom assessment in palliative care: A need for international collaboration," *Journal of Clinical Oncology*, vol. 26, no. 23, pp. 3867–3873, 2008.

[12] S. Kaasa and F. D. Conno, "Palliative care research," *European Journal of Cancer*, vol. 37, no. Supplement 8, pp. 153 – 159, 2001.

[13] A. Aamodt, "A case-based answer to some problems of knowledge-based systems," in *in Proceedings of SCAI, 1993) , Fourth Scandinavian Conference on Artificial Intelligence*, pp. 168–182, IOS Press, 1993.

[14] M. M. Richter, "Knowledge containers," tech. rep., TU Kaiserslautern, 2003.

[15] A.-C. Olsson, N. di Zazzo, and J. Tjäderborn, "Social decision making strategies in internet poker playing." url: http://csjarchive.cogsci.rpi.edu/proceedings/2007/docs/p1831.pdf, mar 2011.

[16] E. Charniak, "Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated," *AI Magazine*, vol. 12, pp. 50–63, November 1991.

[17] J. J. Bello-Tomás, P. A. González-Calero, and B. Díaz-Agudo, "Jcolibri: An object-oriented framework for building cbr systems," in *Advances in Case-Based Reasoning* (P. Funk and P. A. González Calero, eds.), vol. 3155 of *Lecture Notes in Computer Science*, pp. 29–39, Springer Berlin / Heidelberg, 2004.

[18] A. Stahl and T. Roth-Berghofer, "Rapid prototyping of cbr applications with the open source tool mycbr," in *Advances in Case-Based Reasoning* (K.-D. Althoff, R. Bergmann, M. Minor, and A. Hanft, eds.), vol. 5239 of *Lecture Notes in Computer Science*, pp. 615–629, Springer Berlin / Heidelberg, 2008.

[19] M. Jaczynski and B. Trousse, "An object-oriented framework for the design and the implementation of case-based reasoners," 1998.

[20] B. López, C. Pous, P. Gay, A. Pla, J. Sanz, and J. Brunet, "exit*cbr: A framework for case-based medical diagnosis development and experimentation," *Artificial Intelligence in Medicine*, vol. 51, no. 2, pp. 81 – 91, 2011. Advances in Case-Based Reasoning in the Health Sciences.

[21] M. J. Druzdzel, "Smile: Structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models," in *In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI–99)*, pp. 902–903, 1999.

[22] A. Madsen, M. Lang, U. Kjærulff, and F. Jensen, "The hugin tool for learning bayesian networks," in *Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (T. Nielsen and N. Zhang, eds.), vol. 2711 of *Lecture Notes in Computer Science*, pp. 594–605, Springer Berlin / Heidelberg, 2003.

[23] J. A. Recio-garcía, B. Díaz-agudo, P. González-calero, A. S. ruiz granados, and D. S. Informáticos, "Ontology based cbr with jcolibri," in *Applications and Innovations in Intelligent Systems XIV*, pp. 149–162, Springer-Verlag London, 2006.

[24] B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, and A. A. Sánchez-Ruiz-Granados, "Building cbr systems with jcolibri," *Sci. Comput. Program.*, vol. 69, pp. 68–75, December 2007.

[25] Y. Reich and A. Kapeliuk, "Case-based reasoning with subjective influence knowledge," *Applied Artificial Intelligence*, vol. 18, no. 8, pp. 735–760, 2004.

[26] A. F. Rodriguez, S. Vadera, and L. E. Sucar, "A probabilistic exemplar-based model for case-based reasoning," *Springer-Verlag*, pp. 40–51, 2000.

[27] H. Tran and J. Schönwälder, "Fault resolution in case-based reasoning," in *PRICAI 2008: Trends in Artificial Intelligence* (T.-B. Ho and Z.-H. Zhou, eds.), vol. 5351 of *Lecture Notes in Computer Science*, pp. 417–429, Springer Berlin / Heidelberg, 2008.

[28] R. Pavón, F. Díaz, R. Laza, and V. Luzón, "Automatic parameter tuning with a bayesian case-based reasoning system. a case of study," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, pp. 3407–3420, 2009.

[29] R. Baker and P. Cowling, "Bayesian opponent modeling in a simple poker environment," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 125–131, april 2007.

[30] M. A. Terry and B. E. Mihok, "A bayesian net inference tool for hidden state in texas hold'em poker." url: http://ocw.mak.ac.ug/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/26C3A790-77CA-460A-B97B-CE26E3BFCCF4/0/mihokterry.pdf, mar 2011.

[31] K. B. Korb, A. E. Nicholson, and N. Jitnah, "Bayesian poker," *UAI99–Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 343–350, 1999.

[32] K. B. Korb, A. E. Nicholson, and D. Boulton, "Using bayesian decision networks to play texas hold'em poker," *ICGA Journal*, 2006.

[33] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' bluff: Opponent modelling in poker," in *In Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 550–558, 2005.

[34] K. Tretyako and L. Kamm, "Modeling texas hold'em poker strategies with bayesian networks." url: http://ats.cs.ut.ee/u/kt/hw/bayesnets-poker/bayesnets-poker.pdf, mar 2011.

[35] A. S and B. Tessem, "A case-based learner for poker. the," in *Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, 2006.

[36] I. Watson and J. Rubin, "Casper: A case-based poker-bot," in *AI 2008: Advances in Artificial Intelligence*, vol. 5360, pp. 594–600, Springer Berlin / Heidelberg, 2008.

[37] J. Rubin and I. Watson, "Similarity-based retrieval and solution re-use policies in the game of texas hold'em," in *Case-Based Reasoning. Research and Development* (I. Bichindaritz and S. Montani, eds.), vol. 6176 of *Lecture Notes in Computer Science*, pp. 465–479, Springer Berlin / Heidelberg, 2010.

[38] M. Salim and P. Rohwer, "Poker opponent modeling," 2005.

[39] D. Sklansky and M. Malmuth, *Hold 'em Poker for Advanced Players*. Two Plus Two Publications, 1999.

[40] J. Rubin, "Casper: Design and development of a case-based poker player," Master's thesis, University of Auckland, 2007.

[41] D. Billings and M. Kan, "A tool for the direct assessment of poker decisions," tech. rep., International Computer Games Association Journal, 2006.

[42] P. Galfond, "'g bucks' conceptualizing money matters," *Bluff Magazine*, 2007.

[43] H. Tirri, P. Kontkanen, and P. Myllymäki, "A bayesian framework for case-based reasoning," *Springer-Verlag*, pp. 413–427, 1996.

[44] K. B. Korb, A. E. Nicholson, and N. Jitnah, "Bayesian poker," in *In Uncertainty in Artificial Intelligence*, pp. 343–350, Morgan Kaufman, 1999.

[45] K. Tretyakov and L. Kamm, "Modeling texas hold'em poker strategies with bayesian networks," tech. rep., 2009.

APPENDIX A

Running the system

When deploying BayCaRP we experienced an increase in the time needed by BayCaRP to make its betting decisions. This increase in decision making time was not very user friendly, so we decided to attach a version of BayCaRP with a smaller case-base. This reduced the time issue into a more acceptable range, but was still slower than the undeployed version. Still, we hope the reader can enjoy this version of BayCaRP although it is a little bit slower than it should.

The execution of the system involves performing the following steps:

1. Extract the *.zip* file. It is important that *jsmile.dll*, *BNnyeste32PFRVPPos.xdsl* and *BayCaRP_Server.exe* are in the same folder.

2. Execute the *BayCaRP_Server.exe*. When the text *BayCaRP is configured and ready to use!* appears the server is ready to handle requests.

3. Copy *BayCaRP_Client.jar* and *BayCaRP_Client.pd* to the data/bots path in your Poker Academy Pro folder.

4. Run Poker Academy Pro. Open Window->Opponent Manager in your Poker Academy Pro application, and select import an opponent.

5. Find and select the *BayCaRP_Client.pd* file and press ok.

6. Finally go to Ring Games and either create your own limit Texas Hold'em table, or import the attached table (*BayCaRP-test.tbl*) through the import table button on the left side.

If you do not want to participate in the game yourself, and just observe BayCaRP play do the following steps:

1. Click on the player with your name.

2. Select *Set Bankroll* and set this to zero and click *OK*.

Additionaly, if you do not want to observe BayCaRP play a series of bad cards, you can select *Dealer -> Dealer Options* in the menu, and make BayCaRP only receive hole cards amongst e.g. the top 10% starting hands. And if you desire to see all of the players cards, then go to *Options* and click *Play All Face-Up*.

APPENDIX B

Hand groups

Figure B.1 shows the hole cards in each hand group. Suited cards are shown on the top side of the diagonal line, while offsuit cards are shown below the diagonal line. The diagonal line is represented as the pairs, pair of aces through pair of two's. The cells that are not marked with a number constitute hand group 10 in the Bayesian network. Handgroup 1-8 are the orginial Sklansky and Malmuth hand groups, while hand group 9 is the extension to this concept.

OFFSUIT CARDS	SUITED CARDS													
		A	K	Q	J	T	9	8	7	6	5	4	3	2
	A	1	1	2	2	3	5	5	5	5	5	5	5	5
	K	2	1	2	3	4	7	7	7	7	7	7	7	7
	Q	3	4	1	3	4	5	7						
	J	4	5	5	1	3	4	6	8					
	T	6	6	6	5	2	4	5	7					
	9	8	8	8	7	7	3	4	5	8				
	8	9			8	8	7	4	5	6	8			
	7	9						8	5	5	6	8		
	6	9							8	5	6	7		
	5	9								8	6	6	7	
	4	9									8	7	7	8
	3	9											7	8
	2	9												7

Figure B.1: The hand groups

APPENDIX C

Rank of hands in Texas Hold'em

Figure C.1 shows the ranking of the different hand types in Texas Hold'em poker, with the best hand beeing *Royal Flush* and the worst *High Card*. The *High Card* hand type also goes under the name *Busted*, which is used as a state in our BN.

Figure C.1: Texas Hold'em hand ranks

APPENDIX D

The implemented BN

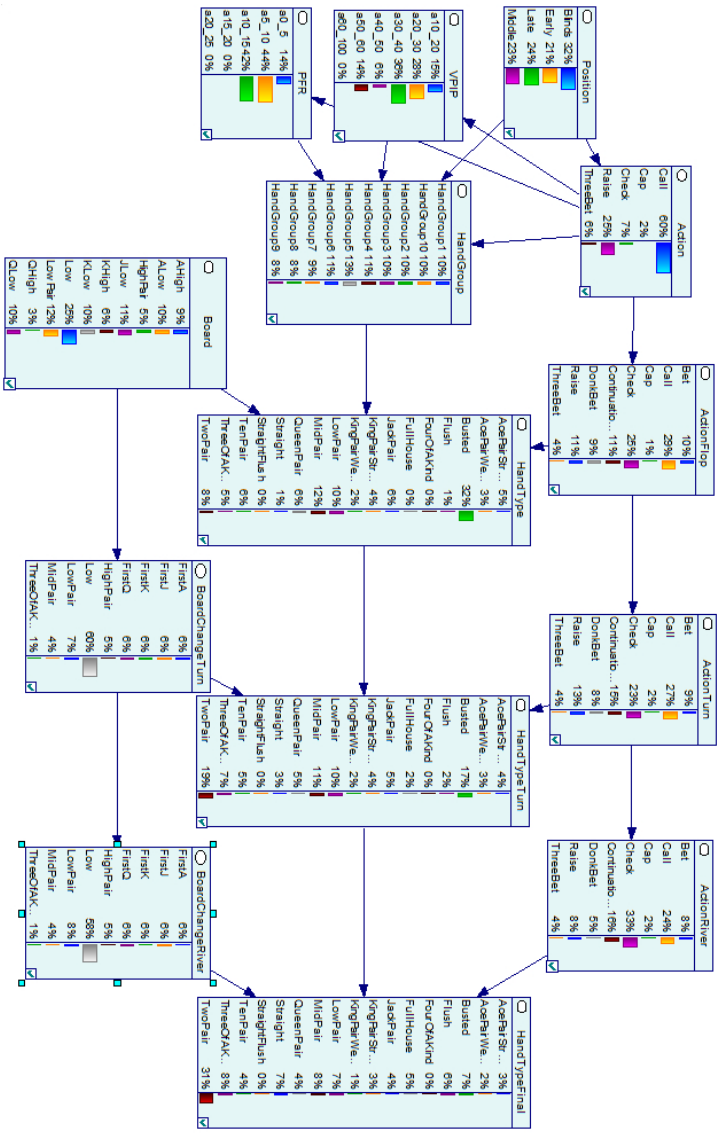


Figure D.1: The implemented BN