On the Construction, Maintenance and Analysis of Case-Based Strategies in Computer Poker

Jonathan Rubin

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science,

The University of Auckland, 2013.

Abstract

The state-of-the-art within Artificial Intelligence has directly benefited from research conducted within the computer poker domain. One such success has been the the advancement of *bottom up* equilibrium finding algorithms via computational game theory. On the other hand, alternative *top down* approaches, that attempt to generalise decisions observed within a collection of data, have not received as much attention. In this thesis we examine *top down* approaches that use Case-Based Reasoning in order to construct strategies within the domain of computer poker. Our analysis begins with the development of frameworks to produce static strategies that do not change during game play. We trace the evolution of our *case-based* architecture and evaluate the effect that modifications have on strategy performance. The end result of our experimentation is a coherent framework for producing strong *case-based strategies* based on the observation and generalisation of expert decisions. Next, we introduce three *augmentation procedures* that extend the initial frameworks in order to produce case-based strategies that are able to adapt to changing game conditions and exploit weaknesses of their opponents. Two of the augmentation procedures introduce different forms of opponent modelling into the case-based strategies produced. A further extension investigates the use of transfer learning in order to leverage information between separate poker sub-domains. For each poker domain investigated, we present results obtained from the Annual Computer Poker Competition, where the best poker agents in the world are challenged against each other. We also present results against a range of human opponents. The presented results indicate that the *top down* case-based strategies produced are competitive against both human opposition, as well as state-of-the-art, *bottom up* equilibrium finding algorithms. Furthermore, comparative evaluations between augmented and non-augmented frameworks show that strategies which have been augmented with either transfer learning or opponent modelling capabilities are typically able to outperform their non-augmented counterparts.

Acknowledgments

First, I would like to thank my supervisor, Ian Watson, without whom this thesis would not have been written. Thank you for the freedom that allowed me to devote approximately five years of my life to a true passion of mine. Thank you also to my co-supervisor, Jim Warren, for providing a much needed alternative perspective along the way. Thank you to all my fellow graduate students at the University of Auckland. I would also like to acknowledge both the academic and hobbyist members of the computer poker community. Thank you to the members and moderators of the pokerai forums - an invaluable resource for anyone interested in computer poker. Thank you also to the organisers of the computer poker competition. Without such an active and vibrant community much of this work would not have been possible. Finally, thank you so much to my friends and family who have encouraged and supported me along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many games have successfully (and enjoyably) been used as domains for Artificial Intelligence (AI) research over the years. Early efforts for games such as chess, checkers, backgammon, Connect-4, Othello, Lines of Action and Scrabble have produced strong computerised players and notable success stories [100]. More recently, games such as bridge, Hex, Go and poker have increasingly been the focus of AI research, so too have modern interactive computer (video) games such as first-person shooters, role-playing games and real-time strategy games. Jonathan Schaeffer states in his 2001 article, *A Gamut of Games* [101]:

> Successfully achieving high computer performance in a non-trivial game can be a stepping stone toward solving more challenging real-world problems.

The use of games as research domains has also played a significant role in the proliferation of competitions that guide modern AI related research. Beginning with the highly publicised human vs. machine chess matches between Kasparov and Deep Blue in the 1990s, today, competitions exist for many different games. The International Computer Games Association [51] presently organises regular Computer Olympiads, where competitions are held for board and card games such as chess, checkers, bridge and Go. RoboCup [94] is an annual competition for both simulated and robotic soccer matches. There are also annual competitions for poker [2], real-time strategy games [22] and general game playing [37]. For the above domains, competitions are successfully used to drive research and improve the state-of-the-art.

Each of the non-trivial games mentioned above share common beneficial properties such as well defined rules over possible actions and behaviours, the existence of clear goals and objectives, as well as embedded performance metrics. Each particular game also has its own unique, beneficial properties that make them good research areas. The game of poker has been identified as a beneficial domain for current AI research [14] due to properties of the game such as **imperfect information** – the inability to observe opponents' cards and **chance events** – the random private and public cards that are dealt each hand.

The popularity of poker experienced a massive boom in the early 2000s due to a combination of factors such as the emergence of online poker, the introduction of *hole card* cameras to televised events, as well as an amateur's unlikely win at the main event of the 2003 World Series of Poker. The use of poker as an academic research domain has undergone a similar increase in popularity. The identification of poker as a useful research domain has inevitably resulted in increased attention from academic researchers and has led to the investigation of a wide range of new algorithms and approaches. In particular, work done by the University of Alberta's Computer Poker Research Group[1] (CPRG) and the establishment of annual computer poker competitions at conferences such as AAAI and IJCAI since 2006, has resulted in the development of a large number of computerised poker agents. The vast majority of these poker agents focus on one particular variant of poker – Texas Hold'em. The Texas Hold'em variation of poker will be the sole focus of this thesis. Furthermore, previous computer poker research can roughly be split into two broad categories [14]. The first attempts to analyse smaller isolated subsets of the problem domain which typically leads to valuable insights about specialised aspects of the game. The second category, on the other hand, attempts to tackle the domain as a whole, resulting in the creation of an agent that is able to play the game autonomously. In this thesis we will restrict our attention to the second category, i.e. producing autonomous poker playing agents that tackle the full game of Texas Hold'em and can be evaluated by challenging human or computerised opposition.

We investigate a novel approach that uses Case-Based Reasoning (CBR) [91, 61, 1] to construct sophisticated strategies in the computer poker domain. The CBR methodology attempts to solve new problems or scenarios by locating similar past problems and re-using or adapting their solutions for the current situation. Common game scenarios, together with their playing decisions are captured as a collection of cases. Each case attempts to capture important game state information that is likely to have an impact on the final playing decision. We refer to the strategies produced as *case-based strategies*. Case-based strategies attempt to generalise game playing decisions recorded within data via the use of similarity metrics that determine whether two game playing scenarios are sufficiently similar to each other, such that their decisions can be re-used. Training data can be both real-world data, e.g. from online poker casinos, or artificially generated data, e.g. from other poker agents.

The stochastic, imperfect information world of Texas Hold'em poker is used as a test-bed to evaluate and analyse our case-based strategies. The game of Texas Hold'em presents tremendous challenges for CBR as the environment is stochastic, dynamic, sequential, partially ob-

[1]http://poker.cs.ualberta.ca/

servable and may involve multiple adversaries, whereas traditional applications of CBR tend to take place in static and episodic environments [24, 48, 50]. Texas Hold'em offers a rich environment that allows the opportunity to apply an abundance of strategies ranging from basic concepts to sophisticated strategies and counter-strategies. In this thesis, we will consider issues to do with opponent modelling, solution adaptation and dealing with very large, probabilistic solution spaces, which are seldom addressed in CBR research. Moreover, the rules of Texas Hold'em poker are incredibly simple. Contrast this with CBR related research into complex environments such as real-time strategy games [4, 81], which offer similar issues to deal with – uncertainty, chance, deception – but don't encapsulate this within a simple set of rules, boundaries and performance metrics. Successes and failures achieved by applying case-based strategies to the game of poker may provide valuable insights for CBR researchers using complex strategy games as their domain, where immediate success is harder to evaluate. Furthermore, it is hoped that results may also generalise to domains outside the range of games altogether to complex real world domains where hidden information, chance and deception are common place.

Before detailing the objectives and contributions of this thesis, we present a description of the rules of Texas Hold'em, followed by a brief overview of the various performance metrics and evaluators mentioned throughout the document.

1.1. Texas Hold'em

Here we briefly describe the game of Texas Hold'em, highlighting some of the common terms which are used throughout this work. For more detailed information on Texas Hold'em consult [111], or for further information on poker in general see [110].

Texas Hold'em can be played either as a two-player game or a **multi-player** game. When a game consists only of two players it is often referred to as a **heads-up** match. Game play consists of four stages – **preflop**, **flop**, **turn** and **river**. During each stage a round of betting occurs. The first round of play is the preflop where all players at the table are dealt two **hole cards**, which only they can see. Before any betting takes place, two forced bets are contributed to the pot, i.e. the **small blind** and the **big blind**. The big blind is typically double that of the small blind. In a heads up match, the dealer acts first preflop. In a multi-player match the player to the left of the big blind acts first preflop. In both heads up and multi-player matches, the dealer is the last to act on the post-flop betting rounds (i.e. the flop, turn and river). The legal betting actions are fold, check/call or bet/raise. These possible betting actions are common to all variations of poker and are described in more detail below:

Fold: When a player contributes no further chips to the pot and abandons their hand and any right to contest the chips that have been added to the pot.

Check/Call: When a player commits the minimum amount of chips possible in order to stay in the hand and continues to contest the pot. A check requires a commitment of zero further chips, whereas a call requires an amount greater than zero.

Bet/Raise: When a player commits greater than the minimum amount of chips necessary to stay in the hand. When the player could have checked, but decides to invest further chips in the pot, this is known as a bet. When the player could have called a bet, but decides to invest further chips in the pot, this is known as a raise.

In a **limit** game all bets are in increments of a certain amount. In a **no-limit** game a player may bet any amount up to the total value of chips that they possess. For example, assuming a player begins a match with 1000 chips, after paying a forced small blind of one chip they then have the option to either fold, call one more chip or raise by contributing anywhere between 3 and 999 extra chips[2]. In a standard game of heads-up, no-limit poker both players' chip stacks would fluctuate between hands, e.g. a win from a previous hand would ensure that one player had a larger chip stack to play with on the next hand. In order to reduce the variance that this structure imposes, a variation known as **Doyle's Game** is played where the starting stacks of both players are reset to a specified amount at the beginning of every hand.

Once the round betting is complete, as long as at least two players still remain in the hand, play continues on to the next stage. Each post-flop stage involves the drawing of **community cards** from the shuffled deck of cards as follows: **flop** – 3 community cards, **turn** – 1 community card, **river** – 1 community card. All players combine their hole cards with the public community cards to form their best five card poker hand. A **showdown** occurs after the river where the remaining players reveal their hole cards and the player with the best hand wins all the chips in the pot. If both players' hands are of equal value, the pot is split between them.

1.2. Performance Metrics & Evaluation

This section will briefly summarise the different types of performance measurements and agent evaluations that are mentioned in this work. Before evaluating the performance of a system, it is necessary to consider the different types of strategies that a poker agent may employ.

[2]The minimum raise would involve paying 1 more chip to match the big blind and then committing at least another 2 chips as the minimum legal raise.

1.2.1. Types of Strategies

A strategy in this context refers to a mapping between game states and the actions that an agent will take at that game state. Typically, an agent's strategy consists of specifying a **probability triple** at every game state. A probability triple, *(f,c,r)*, specifies the proportion of the time an agent will either fold, check/call or bet/raise at a particular point in the game. An agent's strategy is said to be **static** when the strategy does not change over the course of the game. A strategy that does evolve over time is said to be **adaptive**.

Throughout this work the concept of a **Nash equilibrium** [73] strategy will be referred to. A Nash equilibrium is a robust, static strategy that attempts to limit its exploitability against a worst-case opponent. In general, a set of strategies are said to be in *equilibrium* if the result of one player diverging from their equilibrium strategy (while all other players stick to their current strategy) results in a negative impact on the expected value for the player who modified their strategy [73]. Given the size of the poker game tree and the limitations of today's hardware, it is currently intractable to compute exact Nash equilibria for full-scale Texas Hold'em [55], but by applying simplifying abstractions to the game it is possible to derive ϵ-Nash equilibria, or simply *near-equilibrium* strategies, where, ϵ refers to the maximal amount a player can gain by deviating their strategy.

As an ϵ-Nash equilibrium strategy assumes an unknown, worst-case opponent it will limit its own exploitability at the expense of taking advantage of weaker opponents. Hence, while this sort of strategy may not lose, it will also not win by as much as it could against weaker opponents. On the other hand, a player that attempts to isolate the weaknesses of their opponent and capitalise on those weaknesses is said to employ an **exploitive** (or **maximal**) strategy. This is typically achieved by constructing a model of an opponent and using it to inform future actions. An exploitive strategy can be either static or adaptive, depending on how the opponent model is constructed. A consequence of an exploitive strategy is that it no longer plays near the equilibrium and hence is vulnerable to exploitation itself, especially if the model of the opponent is incorrect or no longer valid.

1.2.2. Performance Evaluators

Evaluating the performance of a computer poker agent can be a difficult task due to the inherent variance present in the game. Listed below are some of the various methods that have been used to evaluate the agents we make reference to throughout this work. Measurements are made in **small bets per hand** (sb/h), where the total number of small bets won or lost are divided by the total hands played. For example, assuming a \$10/\$20 hold'em game, where the small bet is \$10 and the big bet \$20, a value of $+0.1$ sb/h indicates an average profit of \$1 for each hand played.

Measuring exploitability The goal of computing ϵ-Nash equilibria is to produce robust poker agents that can perform well against unknown competition by limiting their own exploitability. Here, ϵ refers to how much an opponent can gain by deviating their strategy. ϵ can be analytically computed by deriving a best-response to an agent's strategy. Given a known, static strategy a best-response can be calculated by choosing the action that maximises the expected value against the agent's strategy, for every game state. Strategies with lower values of ϵ allow an opponent to gain less by deviating. Therefore, ϵ provides a convenient measure of exploitability that specifies a lower bound on the exploitability of an agent in the full-scale version of Texas Hold'em.

IRC Poker Server Before the arrival of real-money online poker sites, humans and computerised agents played poker online via an Internet Relay Chat (IRC) poker server. The IRC poker server allowed players to challenge each other to non-real money games of Texas Hold'em. Notable early poker agents such as Loki-1, Loki-2, Poki and r00lbot were all evaluated by their performance on the IRC server [17]. While all the games involved play money, the IRC poker server offered several levels where players were required to earn a certain amount in lower level games before being allowed access to higher levels. As such the calibre of opponent in the higher tiered games was usually strong. A consistent profit, over a large number of hands, in the higher tiered games was most likely an indication of a strong poker agent.

Poker Academy Poker Academy Pro 2.5 is a commercial software product[3] that allows humans and/or computerised agents to compete against each other by playing Texas Hold'em poker. Poker Academy provides an API that allows developers to create their own poker agents and plug them into the software to test their performance. Also provided with the software are strong poker agents developed by the University of Alberta CPRG including Poki, Sparbot and Vexbot. Both Sparbot and Vexbot specialise in playing heads-up, limit hold'em. Sparbot attempts to approach a Nash equilibrium strategy whereas, Vexbot plays an exploitive strategy. Results that are obtained using Poker Academy typically involve gauging a poker agent's performance by challenging Sparbot or Vexbot. As the variance of Texas Hold'em is large, tens of thousands of hands need to be played to have confidence in the results.

[3]Poker Academy Pro 2.5 is no longer available for purchase.

Annual Computer Poker Competition The Annual Computer Poker Competition[4] (ACPC) has been held each year at either AAAI or IJCAI conferences since 2006. The ACPC involves separate competitions for different varieties of Texas Hold'em, such as limit and no-limit competitions, as well as heads-up and multiple-opponent competitions. Since 2009, the ACPC has evaluated agents in the following variations of Texas Hold'em:

1. Two-player, Limit Hold'em.
2. Two-player, No-Limit Hold'em.
3. Three-player, Limit Hold'em.

Entrance into the competition is open to anyone and the agents submitted typically represent the current state of the art in computer poker. Agents are evaluated by playing against each other in a round-robin tournament structure. As poker is a stochastic game that consists of chance events, the variance can often be large especially between agents that are close in strength. This requires many hands to be played in order to arrive at statistically significant conclusions. Due to the large variance involved, the ACPC employs a **duplicate match** structure, whereby all players end up playing the same set of hands. For example, in a two-player match a set of N hands are played, after which the agents' memories are wiped. This is then followed by dealing the same set of N hands a second time, but having both players switch seats so that they receive the cards their opponent received previously. As both players are exposed to the same set of hands, this reduces the amount of variance involved in the game by ensuring one player does not receive a larger proportion of higher quality hands than the other. A two-player match involves two seat enumerations, whereas a three-player duplicate match involves six seat enumerations to ensure each player is exposed to the same scenario as their opponents. For three players (ABC) the following seat enumerations need to take place:

ABC ACB
CAB CBA
BCA BAC

The ACPC employs two winner determination procedures:

1. Total Bankroll. As its name implies the **total bankroll** winner determination simply records the overall profit or loss of each agent and uses this to rank competitors. In this division, agents that are able to achieve larger bankrolls are ranked higher than those with lower profits. This winner determination procedure does not take into

[4] http://www.computerpokercompetition.org/

account how an agent achieves its overall profit or loss, for instance it is possible that the winning agent could win a large amount against one competitor, but lose to all other competitors.

2. **Bankroll Instant Run-Off.** On the other hand, the **instant run-off** division uses a recursive winner determination algorithm that repeatedly removes the agents that performed the worst against a current pool of players. This way agents that achieve large profits by exploiting weak opponents are not favoured, as in the **total bankroll** division.

Man-Machine Poker Championship Another competition that has been cited in the results of some of the literature reviewed in this thesis is the Man-Machine Poker Championship. The Man-Machine Poker Championship is a limit Texas Hold'em competition played between an agent named Polaris and a selected group of professional human poker players. Polaris is a suite of state-of-the-art poker agents developed by the University of Alberta CPRG. Once again a duplicate match structure is used. As it is not possible to erase a human player's memory between sets of N hands, a pair of human professionals are used to challenge Polaris.

1.3. Variance Reduction

Duplicate matches offer a simple and effective procedure for reducing variance, however they cannot completely eliminate the effects of *luck*. A further variance reduction technique is the use of DIVAT analysis. **DIVAT** (ignorant value assessment tool) [12] is a perfect information variance reduction tool developed by the University of Alberta CPRG. The basic idea behind DIVAT is to evaluate a hand based on the expected value (EV) of a player's decisions, not the actual outcome of those decisions. DIVAT achieves this by comparing the EV of the player's decisions against the EV of some baseline strategy, see Equation 1.1.

$$EV(ActualActions) - EV(BaselineActions) \tag{1.1}$$

Equation 1.1 attempts to factor out the effects of luck as both the actual strategy and the baseline strategy experience the same *lucky* or *unlucky* sequence of events. For example, a player that benefits by a statistically unlikely event no longer makes a difference as the baseline strategy also benefits by the same improbable event. What actually matters is any difference in EV the player is able to achieve by varying the actions that they take. When the player is able to achieve a greater EV than the baseline, the player is rewarded. Alternatively, if the player's actions result in a lower EV then they are punished with a negative outcome. If the EV of both strategies is the same the outcome is 0.

Any type of strategy can be used as a baseline. Typically, in limit Hold'em a bet-for-value baseline strategy is adopted. A bet-for-value strategy makes no attempt to disguise the strength of its hand. By establishing game theoretic equilibrium-based thresholds, a sequence of baseline betting decisions can be constructed, based on hand strength alone. Strong hands will be bet and/or raised, whereas weak hands will be checked or folded. In no-limit Hold'em, trivial *check-call* or *bet-call* baseline strategies have mostly been used as they roughly result in the same variance reduction as more involved no-limit bet-for-value strategies [104].

The EV, from Equation 1.1, refers to the average amount a player can expect to win given the current game state. The current game state is described by things such as each player's hole cards, the current round, the community cards and the amount of money in the pot. However, EV can also be influenced by less obvious factors such as the type of player the opponent is and any previous history between the two players which affects the player's beliefs about each other. For these reasons EV is approximated by a concept known as *pot equity* or *all-in equity*. To determine *all-in equity* in a two-player Hold'em match, both player's hole cards need to be known. A player's all-in equity is given by multiplying the current amount in the pot by the probability that the player will win the hand. On the last game round the probability that the player will win is either 0 or 1. For intermediate rounds the probability that the player will win is determined by enumerating all possible combinations of the remaining community cards and counting the number of times the player's hand wins, loses and ties. All-in equity is computationally efficient as it makes the basic assumption that a player's equity is given by the *current* pot size and does not attempt to predict what the final pot size might have been had future betting occurred. On the other hand, *rollout equity* is a more computationally intensive calculation that does attempt to simulate betting on future rounds to more accurately measure the final pot size.

1.4. Thesis Objectives

The focus of this thesis is on the construction, maintenance and analysis of *case-based* strategies in the domain of computer poker. The case-based strategies produced are based on *expert* imitation, where the game playing decisions of a single expert are observed and recorded and the resulting traces are then used to construct an agent that attempts to imitate the original expert's style of play. We investigate *static* strategies, which remain unchanged once computed, as well as *adaptive* strategies that react to changing game conditions. Furthermore, we address the issue of exploitability by constructing and evaluating both non-exploitive and exploitive strategies, i.e. strategies that take the opponent into consideration and attempt to capitalise on any weaknesses they reveal.

The first objective of this research is to determine the requirements of using case-based reasoning to construct strategies in the computer poker domain. This involves investigating the appropriate architectural and design decisions necessary to appropriately handle a complex environment that consists of one or more competing adversaries, chance and imperfect information, as well as deception and bluffing. By identifying the appropriate architectural requirements our objective is to construct and integrate the necessary components to create a framework for producing strong computer poker players that use the case-based reasoning methodology. We wish to construct frameworks that will produce strategies that are able to challenge opponents in multiple poker domains. The agents produced should have the ability to handle both limit and no-limit betting structures, as well as single and multiple opponents.

We address questions such as *how closely the case-based strategies produced can approximate the strength of the original expert used to train the system.* Furthermore, we investigate framework alterations and approaches, which allow the strategies produced to not be restricted by the quality of the original experts' decisions. In so doing, we address whether case-based strategies have the ability to actually outperform the original expert used to train the system.

A further objective of this research is to apply the principles and techniques of case-based reasoning to derive exploitive behaviour. We wish to produce case-based strategies that are able to identify weaknesses in their opponents play and have the capability to capitalise on those weaknesses. In order to do this, opponent modelling procedures will need to be investigated and decisions made about how best to integrate opponent modelling capabilities into a case-based framework.

Throughout this thesis, the emphasis will be on constructing agents that are able to play the complete game of poker autonomously at an expert level. The final objective is to thoroughly evaluate the efficacy of all case-based strategies produced by challenging them against both computerised agents, as well as human opposition. This requires having to overcome the inherent variance associated with the domain of poker, so that accurate assessments can be made about the quality of the strategies produced.

1.5. Thesis Contributions

The four major contributions made by this thesis are as follows:

1. A **comprehensive review** of the major approaches previously investigated in the domain of computer poker, together with a survey of the world-class computerised agents these approaches have produced.

2. Development and analysis of **frameworks for producing non-exploitive, static case-based strategies via expert imitation** in three poker sub-domains.

3. The development and experimental evaluation of **two novel opponent modelling procedures**, which integrate with and extend the basic frameworks to allow the construction of exploitive case-based strategies:

    - The first opponent modelling procedure produces **static, exploitive case-based strategies via implicit opponent modelling and decision combination**.
    - The second opponent modelling procedure results in the construction of **dynamic, exploitive case-based strategies via explicit opponent modelling and solution adaptation**.

4. An investigation into the efficacy of **transfer learning via the mapping and re-use of case-bases** between separate sub-domains.

1.6. Thesis Outline

This thesis is structured as follows. Chapter 2 provides the necessary background required by this thesis together with a comprehensive review of the algorithms and approaches that have previously been investigated within the increasingly popular domain of computer poker. After a thorough examination of related approaches we introduce our novel approach within the computer poker domain in Chapter 3. We begin with the application of case-based reasoning to three poker sub-domains that each provide unique challenges:

1. Two-Player, Limit Texas Hold'em
2. Two-Player, No-Limit Texas Hold'em, and
3. Three-Player, Limit Texas Hold'em

Within each of the above domains a framework is established for producing strong, sophisticated case-based strategies. The final frameworks presented are the product of an iterative period of maintenance, where modifications in performance have been closely measured. We detail the maintenance that has taken place and provide empirical evidence to justify the design decisions of our final frameworks. A reasonable criticism of *imitation*-based strategies described in Chapter 3 is that they will never out-perform the original experts that were used to train the system in the first place. Chapters 4, 5 and 6 address this criticism by extending the basic frameworks introduced in order to improve performance. Three approaches for augmenting the initial frameworks are introduced and evaluated. In particular, Chapter 4 investigates the use of implicit opponent modelling to produce static, exploitive case-based strategies, which combine decisions retrieved from multiple case-bases trained on the actions of a collection of experts. Chapter 5 extends the underlying frameworks from Chapter 3 with explicit opponent modelling. Here, experimental results and analysis is provided for exploitive and adaptive case-based strategies produced via the use of solution adaptation. The final augmentation procedure, which employs transfer learning to re-use learning between poker domains is described and evaluated in Chapter 6. We provide conclusions and directions for future research in Chapter 7.

Chapter 2

Computer Poker: Agents and Approaches

[1]This chapter provides the necessary background required for this thesis and reviews related approaches that construct artificially intelligent computer poker players which have been previously investigated. Each approach presented within this chapter will roughly follow the same pattern. First, the approach is introduced, typically followed by an example. Next, a survey of the autonomous agents that have employed the above approach to play Texas Hold'em is presented, along with a discussion of the agent's performance. Section 2.1. begins with a review of early work on knowledge-based poker agents, followed by section 2.2. which introduces Monte-Carlo simulation. Section 2.3. focuses on ϵ-Nash equilibrium based agents produced via both linear programming and state-of-the-art iterative algorithms. Section 2.4. reviews exploitive agents that attempt to adapt to their opponents' playing style by constructing accurate opponent models and section 2.5. briefly reviews some alternative approaches that have received attention in the literature, such as Bayesian poker and evolutionary algorithms.

2.1. Knowledge-Based Systems

The first approach that we review is that of knowledge-based systems. Knowledge based systems require experts with *domain knowledge* to aid the design of the system. Knowledge-based applications for Texas Hold'em that have been implemented and investigated in the past typically fall into two broad categories: *rule-based expert systems* and more general *formula-based*

[1]The contents of this chapter originally appeared in the journal *Artificial Intelligence*. Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5-6):958-987 (2011) [95]

```
Action preflopAction(Hand hand, GameState state){
  if( state.betsToCall > 2 &&
    state.opponentsInHand > 1 &&
    state.relativePosition > 0.8 &&
    hand.AAo){
      return getAction(new Triple(0.0, 0.05, 0.95));
  } else if...
}
```

Figure 2.1: A hypothetical rule within a knowledge-based system for Texas Hold'em.

methods. Both are briefly summarised below, before reviewing the agents that apply these approaches.

2.1.1. Rule-Based Expert Systems

A simple *rule-based expert system* consists of creating a collection of **if-then** rules for various scenarios that are likely to occur. An example of a possible rule that might be used in a rule-based expert system for Texas Hold'em is depicted in Figure 2.1. In this example, a rule is created for the preflop round where the agent holds a pair of Aces (the best starting hand possible). The rule also specifies various other conditions that must be satisfied about the game state before it becomes activated, for instance there must be more than one opponent already involved in the hand *(state.opponentsInHand* $>$ *1)*, the agent must be in late position *(state.relativePosition* $>$ *0.8)* and must call more than two bets to stay in the hand *(state.betsToCall* $>$ *2)*. The result of satisfying all these conditions is the generation of a probability triple. The example in Figure 2.1 produces a triple that indicates the agent should never fold in this situation and they should call 5% of the time and raise the remaining 95%. A betting action is then probabilistically chosen based upon this distribution.

2.1.2. Formula-Based Strategies

A more generalised system, similar to a rule-based system, uses a formula-based approach. A formula-based approach can roughly be characterised as accepting a collection of (possibly weighted) inputs that describe salient information about the current game state. The formula then outputs a probability triple and a betting decision is made by randomly selecting an action based upon the values provided in the triple.

Typically the inputs accepted by a formula-based system revolve around a numerical representation of hand strength and pot odds [28, 17]. Various methods are available for computing hand strength [28, 12], some of which are mentioned below.

Hand Strength Computations

Enumeration techniques can be used to calculate the *immediate hand rank* (IHR) of a postflop hand [28]. Given a pair of hole cards and the public community cards, the IHR can be computed by ranking the hand relative to the entire set of all possible hands a random opponent may have. Combining all possible two card combinations of the remaining cards in the deck with the known community cards results in the set of all possible hands the opponent may have. Counting the number of times the hand wins, loses and ties against this set of all possible opponent holdings gives the IHR. Treating ties as one half, the formula for IHR is given as follows:

$$IHR = (wins + (ties/2))/(wins + ties + losses) \tag{2.1}$$

Calculating IHR is fast and can easily be computed in real time. On the flop there are $\binom{47}{2} = 1081$ possible opponent holdings to consider, $\binom{46}{2} = 1035$ on the turn and $\binom{45}{2} = 990$ on the river.

As an example, consider player A has the hole cards T♣ J♣ and the flop cards are 2♢ T♠ K♡. Out of the 1081 possible opponent holdings, player A will win 899 times, tie 6 times and lose the remaining 176 times, giving an IHR of: $(899 + 6/2)/(899 + 6 + 176) = 0.834413$.

The example above assumes a uniform distribution over the cards the opponent could be holding, however this assumption is usually not correct as players will typically fold weaker hands before the flop and will therefore hold stronger hands post flop. To improve the hand strength calculation the opponent's set of possible two-card combinations can be weighted to reflect the likelihood of holding a particular combination of cards at a specific point in the hand. This more informed *immediate hand strength* (IHS) metric allows beliefs about opponents to be directly incorporated into the hand strength calculation.

The immediate hand rank/strength described above provides a measure of hand strength at the current point in the hand, but makes no considerations for future community cards to be dealt. A separate measure, *hand potential* [28], can be used to compute the positive or negative effect of future community cards. *Positive potential* gives the probability of currently holding an inferior hand, but improving to the best hand with future community cards. Conversely, *negative potential* computes the probability of currently holding the better hand, but falling behind with future cards. *Effective hand strength* (EHS) [17, 28] combines the results of immediate hand strength and positive potential:

$$EHS = IHS + (1 - IHS) \times PositivePotential \tag{2.2}$$

Alternatively, potential can be directly factored into the hand strength calculation by first performing a *roll-out* of the remaining community cards followed by an enumeration of all possible opponent holdings after all community cards are known. Computing the average outcome for every possible *roll-out* combination gives the 7-card Hand Strength (7cHS) [56].

Pot Odds

A further input typically considered within a formula-based approach are the *pot odds*. The pot odds give a numerical measure that quantifies the return on investment by contrasting the investment a player is required to make to stay in the hand, given the possible future rewards. The equation to calculate pot odds is as follows:

$$PotOdds = \frac{c}{p + c} \tag{2.3}$$

where, c is the amount to call and p is the amount currently in the pot. The *pot odds* can be used to determine the correctness of committing further chips to a pot. For example, if there is currently \$60 in the pot and a player needs to call a bet of \$20 to stay in the hand, the pot odds are 0.25. This means the player needs to have better than a 25% chance to win the hand to correctly call the bet.[2]

2.1.3. Knowledge-Based Poker Agents

We now review a collection of agents, found in the literature, that have applied the knowledge-based approaches described above to produce systems that play Texas Hold'em poker.

One area where rule-based expert systems commonly appear is in preflop play. Due to the restricted problem size during preflop play, a rule-based expert system (of reasonable size) can be sufficient to handle the preflop stage of play independently. There have been several poker agents that separate the preflop and postflop implementation in this way, including [13, 17, 69]. Hobbyist, Greg Wohletz developed an agent named r00lbot that competed on the early IRC poker server. r00lbot's preflop play was determined by a rule-based system based upon detailed guidelines for preflop play published by notable poker authors David Sklansky and Mason Malmuth in [111], where they identify and rank various equivalence classes of starting hands. [17, 28] used preflop *roll-out* simulations to determine the income rate of various starting hands and built a preflop rule-based system around the results. They report that the results of the *roll-out* simulations were strongly correlated with the ranked equivalence classes specified by [111].

[2]Note this example uses *immediate pot odds*, a further concept in poker known as *implied pot odds* also considers the value of possible future bets in the situation where the player does improve, however this is not easily calculated.

Turbo Texas Hold'em [121] is a commercial software product that identifies a vast number of possible scenarios and encodes these decision points within a knowledge-based system. Even considering the large scope of the project, [11] still reports that the system is easily beaten by mediocre competition after initial exposure.

For his masters thesis project Follek [35] developed SoarBot, a rule-based expert system for multi-player, limit Texas Hold'em, built upon the SOAR rule-based architecture [63]. Overall, Follek reports that SoarBot's play was mediocre, stating [35]:

> It played much better than the worst human players, and much worse than the best human and software players.

Early agents produced by the University of Alberta CPRG that use knowledge-based approaches include the first version of Loki [85, 13] and the formula-based version of Poki [28, 17], which was a re-write of the original Loki system. Poki uses opponent modelling techniques and a formula-based betting strategy that accepts effective hand strength as its main input and produces a probability triple as output. The formula-based Poki achieved consistent profit in the higher tiered, limit hold'em games on the early IRC poker server [17]. Furthermore, a member of the Poki family of agents achieved first place at the 2008 ACPC 6-Player limit hold'em competition [2].

More recently [69] compared and contrasted a knowledge-based implementation *(Phase 1)*, with a more sophisticated *(Phase 2)* implementation based on *imperfect information game tree search* (see section 2.4.). McCurley [69] designed the system to play the no-limit, heads up variation of Texas Hold'em and tested it against human opponents on an online poker site. While testing of the system was limited, the results reported that the *Phase 1* implementation lost to the human opponents, whereas the *Phase 2* implementation was profitable.

While knowledge-based systems provide a simplistic framework for implementing computer poker agents, their various short-comings have been identified and highlighted in the literature cited above. To begin with, knowledge-based systems require a domain expert to encode their knowledge into the system which can itself be a challenging problem (i.e. the *knowledge elicitation bottleneck*). As the system evolves, with the addition of further information and improvements, it soon becomes difficult to maintain [15] and can easily lead to conflicting information and performance degradation. Furthermore, any kind of system defined on static expert knowledge is likely to produce a rigid strategy that is unable to adapt to changing game conditions and hence, will be exploitable by any observant opposition [16]. In general, there exist too many scenarios in the complete game of Texas Hold'em for a rule or formula-based system to identify and handle, which results in too many dissimilar situations being merged together and handled in a similar manner. Rather, what is preferable is a dynamic, adaptable system where advanced strategies and tactics are an emergent property of the approach itself.

2.2. Monte-Carlo and Enhanced Simulation

2.2.1. Introduction

Dynamic search of a state space usually offers an advantage over the static knowledge-based approaches mentioned in section 2.1.. A typical game tree search, such as minimax search with *alpha-beta* pruning [57], involves exhaustively searching the breadth of a tree until a certain cut-off depth is reached. Nodes on the frontier of the cut-off depth are then assigned values by some evaluation function and these values are backpropagated to the intermediate nodes of the tree. This type of game tree search works well for games of perfect information, but fails for games involving imperfect information as the missing information ensures that the exact state of the game cannot be known.

An alternative search procedure, known as Monte-Carlo simulation [70], involves drawing random samples for choice nodes in the game tree and simulating play until a leaf node is reached, at which point the payoff value is known. By repeating this procedure many times, the average of the expected values eventually converge to a robust evaluation value for intermediate nodes in the game tree. Therefore, instead of completely searching the game tree until a certain cut-off depth and estimating payoff values, a particular line of play is repeatedly selected and searched to the leafs of the tree, until the average of the known payoff values approximate the true value of intermediate nodes.

Before being applied to poker, Monte-Carlo simulation was successfully used in games that involve chance events, such as backgammon [114], as well as games that involve imperfect information such as Scrabble [108] and bridge [47].

2.2.2. Simulation Example

The following example describes the simulation process as applied to the game of Texas Hold'em poker. Figure 2.2 depicts three hypothetical trials within a simulation. The known information at the beginning of each trial is as follows:

1. There are two players in the hand after the preflop betting.
2. The small bet is $10 and the big bet is $20.
3. The community cards on the flop are: 2$\diamondsuit$ T$\heartsuit$ 5$\clubsuit$
4. Player A's personal cards are: A$\diamondsuit$ K$\spadesuit$
5. Player A is first to act and wishes to determine the expected value of either checking (k) or betting (b).

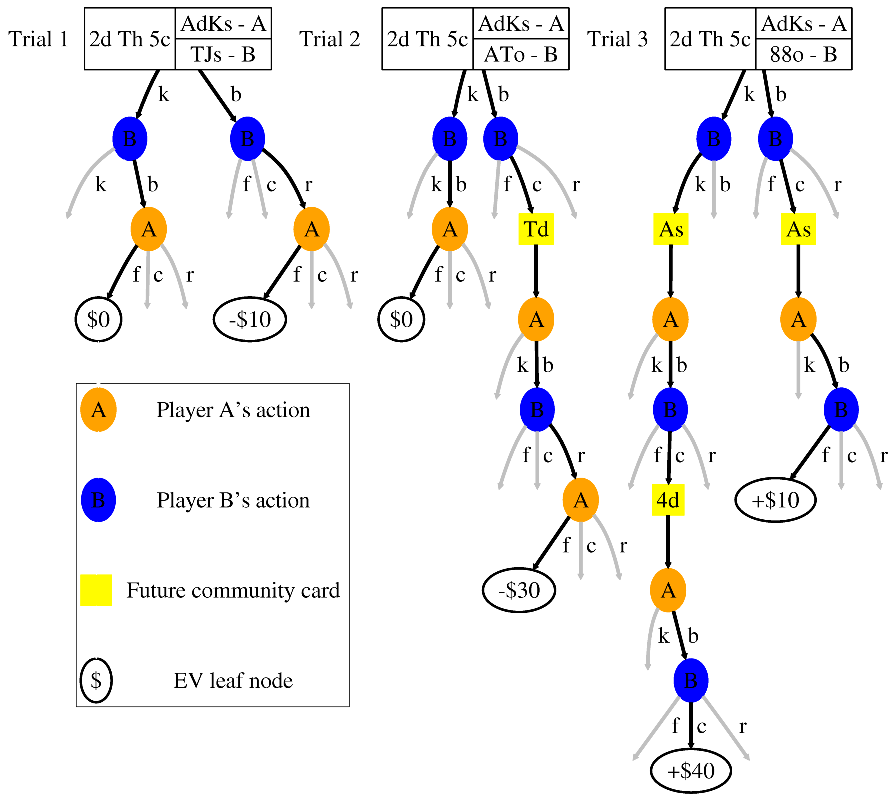

Figure 2.2: Three hypothetical trials within a simulation. Each trial contributes to the final estimation of the EV for checking or betting. The betting decisions are as follows: k - check, b - bet, f - fold, c - call, r - raise.

In order to estimate the expected values for either checking or betting on the flop, player A must simulate out the rest of the hand by making predictions about what hand the opponent may have, as well as their own future betting actions, the opponent's future betting actions and any future community cards yet to be dealt. Referring back to Figure 2.2 each trial continues as follows:

1. Player A predicts what hand player B may have at this point in the game and assigns this hand to player B. In Figure 2.2, player B's predicted hand is listed in the bottom right hand corner of the box located at the root of each tree, e.g. for trial 3 player A predicts B holds a pair of eights[3].

2. Separate simulations are conducted to estimate the EV for checking/calling and betting/raising. The expected value of a fold is always set to zero and player A contributes no future bets to the pot when they fold.

3. Player A then predicts future betting actions for itself and its opponent by generating a probability triple at each decision node and then randomly selecting a betting action based on this triple. In Figure 2.2, where a node is labelled 'A', player A predicts its own future action and where a node is labelled 'B' player A makes a prediction about the opponent's betting decision. The bold edges represent the predicted actions made during the simulation.

4. Where a further community card is required, player A randomly selects this card from the set of possible remaining cards. For example, in trial 2 player A has randomly selected the T♢ as the turn card and in trial 3, player A has randomly selected the A♠ as the turn card and the 4♢ as the river card.

5. Finally, given the above information, a numerical value for the trial can be determined at either the fold or showdown nodes depending on how much money player A has won or lost during the trial. For example, in trial 2 when player A decides to check they lose no money as they predict folding to player B's bet. However, in the same trial, the outcome of a betting decision on the flop is to lose $30 as player A has predicted they will bet on the turn and then fold to player B's raise.

As mentioned above, each trial contributes an estimate for the expected value of checking/calling as well as the expected value of betting/raising at a certain point in the match. As the number of trials increase the average of the EV will eventually converge to a stable set of values. This could involve hundreds or even thousands of trials. Furthermore, the more accurate

[3]In the diagram a player's hand is identified by first listing the ranks of both hole cards followed by either an *s*, if the cards are of the same suit, or an *o*, if the cards are offsuit.

the predictions that player A makes about the strength of player B's hand and the future betting decisions, the more accurate the final EVs will be. Typically, the action with the greatest EV is then chosen as the appropriate move for player A.

Finally, Figure 2.2 depicts the nature of the search conducted via Monte-Carlo simulation. Rather than exhaustively examining every node in the game tree to a certain depth, a specific line of play is investigated by selectively expanding certain nodes in the tree until the leafs are reached. This specific line of play is illustrated by following the bold edges through the game tree in each trial. A thorough search of the game tree is ensured by performing many trials within the simulation.

2.2.3. Simulation-Based Poker Agents

One of the earliest simulation-based "bots" that played on the IRC poker server was Greg Wohletz's r00lbot. During postflop play r00lbot typically conducted around 3000 trials by simulating the hand out to the showdown, against N random opponent holdings, to determine the likelihood of winning. This information was then used to select a betting action.

Another early poker agent that played over IRC was Loki, which was developed by the University of Alberta CPRG to play limit Texas Hold'em against multiple opponents. Loki-1 originally used a formula-based evaluation function to determine its betting strategy [85, 13]. Improvements to the Loki-1 system resulted in a new system, Loki-2, that augmented the static evaluation function with simulation to better determine the expected value of check/call and bet/raise decisions [15, 16, 23].

Billings et al. [15] label their approach *selective-sampling* simulation, as opposed to Monte-Carlo simulation, due to the selective bias introduced when choosing a random sample at a choice node. Rather than assigning an opponent random hole cards during a simulation, a weight table is maintained and updated for each opponent after every observable action. Each weight table contains an entry for every possible hole card combination that the opponent may hold e.g. AKs, 89s, QQo etc. The weight assigned to each entry represents the possibility of the opponent playing the hand in the observed way to a specific point in the game. These weights are then used to bias the distribution used for sampling during a simulation.

Self-play experiments between Loki-1 and Loki-2 highlighted a significant difference in earnings between the two agents, with Loki-2 typically earning 0.05 sb/h more, on average, than Loki-1 [15]. Loki-2 also appeared to do better than Loki-1 against human competition on the early Internet Relay Chat (IRC) poker server, where both humans and agents could compete for play money stakes [15]. The apparent success of the selective-sampling simulation-based approach led the University of Alberta CPRG to advocate the method as a general framework to be used for games that involve chance events and imperfect information [15, 16, 17].

Loki was later re-written and renamed Poki [28, 17]. Once again Poki consisted of a Formula-Based Strategy (FBS) and a Simulation-Based Strategy (SBS). The system was also improved with better opponent modelling capabilities and an updated re-weighting method [28, 29]. Analysis reported by Davidson [28] reiterated earlier findings [15] about the emergence of sophisticated plays, such as check-raising, that were witnessed using strategies produced by SBS-Poki. However, the overall results of comparison between FBS-Poki and SBS-Poki were less convincing. While SBS performed better at introductory levels on the IRC poker server, the same was not true at higher levels where the calibre of opponent was typically a lot stronger. Furthermore, an experiment conducted by [28] showed that SBS-Poki performed a lot worse than FBS-Poki when challenging human opponents at heads-up play. These results led Davidson to remark [28]:

> It was expected that the SBS would be far superior to the FBS system, but to date, this has not been demonstrated.

It was observed that SBS-Poki played too tight in heads-up play, folding many hands. Conversely, SBS-Poki was too aggressive in full, 10-player matches, raising and re-raising a lot. Davidson [28] identifies several possible scenarios that cause the expected values generated to be inaccurate due to small biases introduced during simulation. The behaviour observed during heads-up matches is roughly due to SBS-Poki falling into a cycle of folding behaviour. When simulating future actions SBS-Poki predicts itself folding too often – this causes the EVs generated to become negative. Once the EVs for calling or raising are less than 0, SBS-Poki simply decides to fold. Moreover, after making the decision to fold, SBS-Poki needs to update its own model of itself, which ensures that given similar future situations it will again predict itself folding during the simulations. On the other hand, during full table play SBS-Poki acts too aggressively. During simulations SBS-Poki predicts its opponents calling and raising too often, which increases the amount of value in the pot, this gives SBS-Poki the right odds to stay in the hand even with a moderate strength hand. So, SBS-Poki is observed calling and raising too often with mediocre hands. Davidson concludes that a better, more robust method of game tree search[4] is required that is not as sensitive to bias as selective-sampling simulations are. Billings [11] also questions whether *full-information* simulations are appropriate for decision making in an imperfect information environment, i.e. rather than making future decisions based purely on what cards it is believed an opponent may hold, what is actually required is a solid, objective decision due to the unknown information.

Despite the mixed results presented for simulation based betting strategies by [28] and [11], there have been other efforts, outside of the University of Alberta CPRG, to develop poker agents based on Monte-Carlo Simulation. AKI-RealBot [105] is a Monte-Carlo based exploitive poker agent that was the second place finisher in the 6-Player limit hold'em competition at the 2008

[4]The new method proposed is *miximax* and *miximix* search discussed in section 2.4.1..

ACPC. AKI-RealBot uses a similar simulation procedure as described above, but augments the result of the simulations with a post-processing decision engine that makes it more likely for AKI-RealBot to get involved in a pot with a player who is considered weak. A player is considered weak if they have lost money to AKI-RealBot over the course of the last 500 hands. The post-processing procedure used by [105] decreases the likelihood of AKI-RealBot folding to this player preflop, even if the EV for folding indicates this is the correct decision. An analysis of the 2008 AAAI competition results show that AKI-RealBot's success was largely due to its ability to exploit one weak opponent. So, while it lost to 3 out of the 6 agents, it made up for these losses by exploiting the weak agent by much more than any of the other competitors [2].

A recent algorithm that makes use of Monte-Carlo simulations is Monte-Carlo Tree Search (MCTS) [25]. MCTS combines the evaluation information returned via Monte-Carlo simulations with game tree search. An initial game tree, consisting only of the root, is built up through repeated simulations. Each node in the game tree records information such as expected value and the number of visits to the node. The consequence of each successive simulation is the addition of a node to the game tree, along with updates to its associated values. The effective result is the use of simulation to inform the construction of a game tree via iterative deepening that reduces the probability of exploring ineffective lines of play. MCTS requires a *selection policy* that controls the *exploration/exploitation* trade-off for nodes in the currently generated tree (e.g. the UCT heuristic [58]) as well as a *backpropagation policy* that controls the value updates for nodes in the tree (typically the average or maximum expected values of the node's children).

The use of MCTS resulted in performance breakthroughs for the the game of Go [25]. Standard *alpha-beta* search procedures failed in the domain of Go due to the game's massive branching factor, whereas the ability of MCTS to identify sub-optimal branches and concentrate efforts on sampling only promising branches is considered to be one of the main reasons for its success.

Van den Broeck et al. [115] have recently applied MCTS to the game of no-limit Hold'em. Van den Broeck et al. [115] construct an offline non-player specific opponent model by learning a regression tree from online poker data. During simulations the model is used to predict action and hand rank probabilities for specific opponents. Van den Broeck et al. [115] experiment with novel *selection* and *backpropagation* strategies that take into account information about the standard error over the sampling distribution. The result is an exploitive Hold'em agent constructed via MCTS that is able to challenge multiple opponents and handle a no-limit betting structure. Van den Broeck et al. [115] present results that indicate the MCTS agent was able to successfully exploit naive, non-adaptive rule-based players.

2.3. Game Theoretic Equilibrium Solutions

The next approach we review considers the computation of equilibrium solutions using game theoretic principles. We start by introducing the field of game theory. An example, using the game of Rock-Paper-Scissors is described and the concept of an equilibrium solution is explained. Approaches to computer poker that attempt to derive *near*-equilibrium solutions using game theory are then surveyed and evaluated.

2.3.1. Game Theory Preliminaries

The field of game theory provides analytical tools to study situations where multiple agents compete and interact within a particular environment [83, 73]. Agents have individual goals and objectives that they try to achieve. Typically, a game will involve the following components (replicated from [73]):

- A finite set of players.
- A finite set of moves each player can make at specific points in the game.
- A numerical *payoff* that is assigned to each player once the game is finished. A payoff could be either a negative value (a loss), positive (a gain), or 0 (a draw).

In addition to the above components, a game can also consist of:

- Chance events, such as the flip of a coin or the drawing of a card from the deck.
- Hidden information, i.e. information that is not available to a player at the time that they make their move.

2.3.2. Rock-Paper-Scissors Example

A game, as described above, can be represented either by rules, as a tree or as a matrix. Figure 2.3 illustrates the game tree for the simple game of rock-paper-scissors. When a game is represented as a tree it is said to be in *extensive form*.

Figure 2.3 depicts the game tree for a two-person game of rock-paper-scissors. The two players are referred to as player A and player B in the tree. Where a node is labelled with an A, this node is said to be owned by player A and where a node is labelled with a B, the node is owned by player B. Each player has a choice of three moves that they can choose, either **rock**, **paper** or **scissors**, represented as the edges in the tree labelled with R, P or S, respectively. The rules of the game state that rock beats scissors, paper beats rock and scissors beats paper. If both players make the same choice then the game is a tie. As a win for one player results in

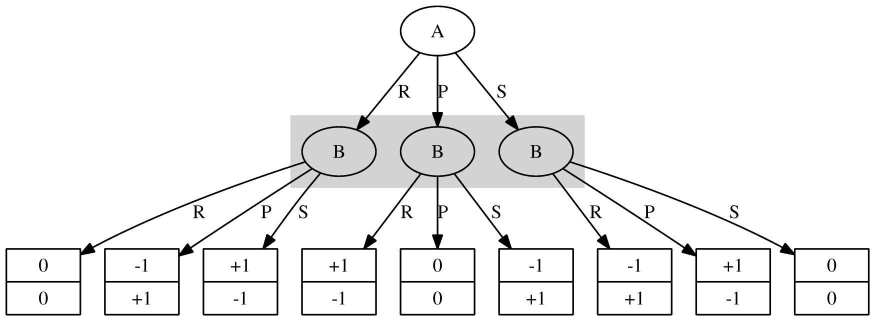

Figure 2.3: The game of Rock-Paper-Scissors represented as a tree.

a corresponding loss for the other player, this type of game falls under the category of a *two-person, zero-sum game.*

A and B choose their moves simultaneously. Once each player has made their decision they make their choice known to their opponent. To represent this in the game tree of Figure 2.3, you can imagine player A first makes their move and whispers it to an independent third party, then B makes their move and also makes their choice known to the third party. The third party then determines the outcome of the game. The fact that B does not know which move A has chosen is represented by the grey background that surrounds the nodes that belong to B. In game theory, this is known as an *information set.* At this point in the game, B knows that they are at one of the three nodes in the information set, but they don't know exactly which node it is. If they did then B would have *perfect information* of the game, however B doesn't know which move A made, hence they have *imperfect information* of the game, represented by the information set.

The leafs of the tree correspond to the conclusion of the game. Each leaf is labelled with two values that represent the payoff to each player. The upper payoff value belongs to player A and the bottom value belongs to player B. For example, if A chooses paper and B chooses paper the payoff is 0 for each player, i.e. it's a draw. If however, A chooses paper and B chooses rock the payoff is now +1 for A and -1 for B.

This payoff information can be represented in matrix form. Figure 2.4 shows the payoff matrix for player A. When a game is represented in this way it is said to be in *normal form.* In Figure 2.4, each row of the matrix has been labelled with a strategy that A could select and each column represents a strategy for B. A *strategy* can be thought of as specifying a move for every possible game state a player may encounter. Referring again to the game tree in Figure 2.3, there is only one node where player A is required to make a move, i.e. the root node. At this node A has the choice of three moves R, P, S. This gives a total of 3 strategies for player A. On the other hand, B owns 3 nodes in the game tree, however as these nodes all belong to the same

$$\begin{array}{c c} & \begin{array}{ccc} R & P & S \end{array} \\ \begin{array}{c} R \\ P \\ S \end{array} & \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 2.4: The game of Rock-Paper-Scissors represented as a matrix.

information set, the move that B makes at each node must be the same, e.g. B cannot choose P at the first node, S at the second node and R at the third node as this assumes B has access to hidden information. So, B also has only three strategies to play in total.

These 3 strategies are known as *pure strategies.* If A always played rock, B would eventually pick up on this and could play paper to win every time. On the other hand, a *mixed strategy* is one which assigns a probability value to each pure strategy, e.g. A may play rock 80% of the time, paper 10% of the time and scissors 10% of the time. Different combinations of probabilities will result in different *expected values* for each player.

A pair of mixed strategies is said to be in *equilibrium* if the result of deviating from one of the strategies, while holding the other constant, results in a loss in *expected value* for the player who changed strategies. In other words, given a pair of equilibrium strategies, no player can do any better by choosing a different strategy as long as the other player sticks to the equilibrium strategy. For two-person, zero-sum games, such as the rock-paper-scissors game we have described, an *equilibrium strategy*, also known as a *Nash equilibrium*, is generally considered a solution to the game [73]. Finding a Nash equilibrium is equivalent to determining the optimal combination of probability values that make up a mixed strategy. One approach to determine these probability values is to represent the problem as an optimisation problem and use *linear programming* algorithms (such as the simplex algorithm [73] or interior point methods [116]) to derive a solution. Doing so for the rock-paper-scissors example gives a Nash equilibrium vector of probabilities as follows: $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, i.e. the equilibrium solution is to just play rock, paper or scissors with equal probability.

2.3.3. Equilibrium Strategies for Texas Hold'em

The example presented in the previous section showed that by representing a game as a payoff matrix in normal form we can use linear programming to derive an equilibrium strategy. However, the scope of rock-paper-scissors is extremely small when compared to more strategically interesting games such as poker. Using the normal form to represent the game of rock-paper-scissors is satisfactory due to its size, however as the size of the game tree increases the corre-

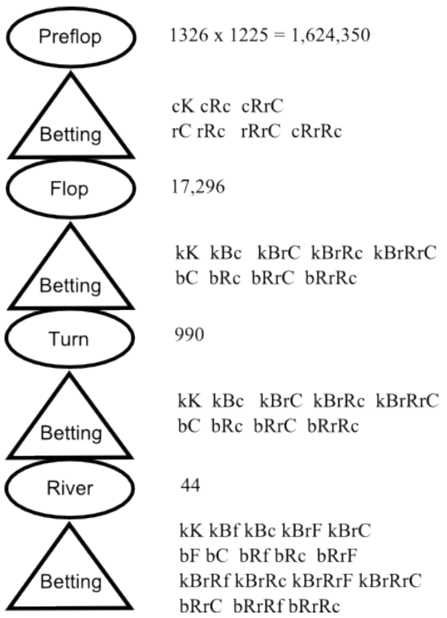

Figure 2.5: A high level view of the game tree for 2-player Texas Hold'em, which consists of approximately 10^{18} game states. Based on original image presented in [18]

sponding size of the normal form matrix increases exponentially. Using linear programming to solve the matrix game becomes infeasible as the matrix gets too large.

Another representation, known as the *sequence form* [60], addresses some of the shortcomings of the normal form. Rather than representing a strategy as a combination of probabilities over all possible pure strategies, the sequence form specifies the probabilities of taking certain actions at each of the game tree's information sets. This results in a representation that is only linear in the size of the game tree and therefore allows larger games to be solved via linear programming.

While the use of the sequence form allows a much larger class of games to be solved using game theoretic methods [59], it is still computationally infeasible to represent and solve for games that involve massive game trees, such as full-scale Texas Hold'em. Figure 2.5 depicts a high level view of the game tree for two-player, limit Texas Hold'em, which consists of approximately 10^{18} game states. The numbers next to each round represent the number of nodes in the game tree generated due to chance alone and the characters represent the possible betting sequences that lead to the next round. To put this in perspective consider a reduced version of Texas Hold'em, called Rhode Island Hold'em [109]. Rhode Island Hold'em was created because it retains the beneficial properties of Texas Hold'em, but reduces the number of game states from 10^{18} down to 10^{9}. Gilpin and Sandholm [41] report that applying the sequence form to

Rhode Island Hold'em produces a linear program with 91,224,226 rows and the same number of columns, which is much too large a matrix to solve. To overcome this limitation it is necessary to limit the size of the game tree by imposing simplifying *abstractions* (see section 2.3.4.) to the game itself. Applying abstractions to the game of Rhode Island Hold'em reduces this matrix to just 1,190,443 rows and 1,181,084 columns which was able to be solved via linear programming methods. The coupling of abstractions together with the sequence form representation has resulted in computationally tractable large-scale games that are able to be solved using linear programming methods.

2.3.4. Abstractions

The combination of game tree abstractions and the use of linear programming methods has resulted in the construction of strong poker programs that approximate equilibrium solutions [18, 39, 6]. Each of these programs requires the use of a combination of abstractions to reduce the size of the game tree.

In general, there are two categories of abstraction, those that respect the original strategic complexity of the game (*lossless* abstractions) and those that compromise it (*lossy* abstractions). The following example will highlight the differences between the two.

Consider the preflop stage of the game where each player is dealt two cards that only they can see. Out of a deck of 52 cards the first player is dealt 2 cards at random $\binom{52}{2}$ followed by the second player $\binom{50}{2}$ resulting in a total of 1,624,350 possible combinations (see Figure 2.5). Imagine for one of these combinations that player one is dealt the two cards A♡K♡ and player two is dealt T♣J♠. Now, during the preflop stage of the game there is no strategic difference between the hands A♡K♡, A♠K♠, A♣K♣ or A♢K♢. The only difference between these hands are the suits and at present the suit information has no bearing on the quality of the hand. All that matters is whether the hand is of the same suit or not. Hence, an abstraction can be used where the hands A♡K♡, A♠K♠, A♣K♣ and A♢K♢ could all be represented as the single hand AK*s* (where *s* stands for suited). The same holds for player two who was dealt T♣J♠, this player could have equivalently been dealt T♡J♠, T♢J♣, T♡J♢ etc. the only difference is that the two cards are not of the same suit, so they could all be represented as TJ*o* (where *o* stands for off-suit). In total there are 169 equivalence classes of this form for all two card combinations. Using this abstraction technique can drastically reduce the number of preflop card combinations in the game tree from 1,624,350 to 28,561. The important point is that no strategic information has been lost to achieve this reduction. The game tree has been reduced in size simply by reclassifying hands in a more concise manner.

Unfortunately, lossless abstractions alone do not always provide the reduction in size required to solve large scale games. This results in the need for lossy abstractions that will affect the original strategic integrity of the game to some degree. Furthermore, the example presented above only works for one stage of the poker game tree, i.e. the preflop. Once play transitions to the next stage (the flop) suit information is required and this information has now been lost. A better abstraction would be one that is able to be used throughout the entire game tree. *Bucketing* (also known as *binning*) is a powerful abstraction that has been used within many poker agents [109, 18, 39, 6]. Bucketing is a lossy abstraction that groups categories of hands into equivalence classes. One possible method of bucketing groups hands based on the probability of winning at showdown against a random hand. This is given by enumerating all possible combinations of community cards and determining the proportion of the time the hand wins. This is referred to as *roll-out hand strength* or *expected hand strength* ($E[HS]$). Hands can then be grouped by assigning them into a particular *bucket* which holds hands with the same $E[HS]$. For example, the use of five buckets would create the following categories:

[0.0 - 0.2] [0.2 - 0.4] [0.4 - 0.6] [0.6 - 0.8] [0.8 - 1.0]

and group all hands with the same winning probabilities into the same bucket. The abstract game model is now constructed using buckets rather than cards. To complete the abstract game model, the final requirement of bucketing is to determine the probability of transitioning between different buckets during each stage of play.

Abstraction via bucketing can either be performed manually, by the system designer, or the process can be automated. Creating a manual abstraction allows the designer control over bucket types and boundaries, but may be costly when re-creating an abstraction. On the other hand, automated abstraction, groups similar hands into buckets that are determined by the algorithm itself. Automated abstraction typically allows the specification of some granularity to determine how fine (or coarse) to make a particular abstraction. Finer abstractions lead to larger models that are required to be solved.

Expectation-Based Abstraction

An abstraction that is derived by bucketing hands based on expected hand strength is referred to as an *expectation-based* abstraction. Johanson [55] points out that squaring the expected hand strength ($E[HS^2]$) typically gives better results, as this assigns higher hand strength values to hands with greater *potential*. A hand's *potential* refers to its ability to improve in strength, given future community cards. Typically in poker, hands with similar strength values, but differences in potential, are required to be played in strategically different ways [111].

The example above presented one standard procedure for automatically grouping hands by placing them into buckets, based on a predefined partition of winning probabilities. One problem with this approach is that some buckets will contain many hands, while others may contain very few. Further bucketing procedures attempt to improve upon the standard bucketing approach presented above.

Nested bucketing attempts to further accommodate for the effects of hand potential. Nested bucketing begins like standard bucketing, by combining hands into buckets that have similar $E[HS^2]$ values, however after the first bucketing stage a second bucketing stage splits each of the first buckets based on standard $E[HS]$. This is done in order to better distinguish between hands with high hand strength and hands with high potential.

Percentile bucketing is an automated bucketing technique that creates buckets that hold roughly the same amount of hands. Percentile bucketing modifies the hand strength boundaries used for each bucket, such that each bucket contains the same percentage of hands. For example, given 5 buckets for a betting round, the bottom 20% of hands would be assigned into the first bucket, the next 20% of hands assigned into the next bucket, and so on. The top 20% of hands would be assigned into the last bucket.

History bucketing is a manual bucketing technique that allows finer control over bucket boundaries. In history bucketing child buckets are individually determined for each parent bucket in the previous round. History bucketing makes use of the fact that some buckets have a greater probability of transitioning to particular buckets in the next round. For example, it is more likely that a bucket that represents high hand strength in one round, will transition to a high hand strength bucket in the next round. Correspondingly, it is likely that a low hand strength bucket will remain in a low hand strength bucket between rounds, rather than transitioning into a very high hand strength bucket. By modifying bucket boundaries, history bucketing allows finer control over how many hands can be mapped into a child bucket for each parent, resulting in an improved quality of abstraction, without increasing the number of buckets.

Potential-Aware Automated Abstraction

A separate automated abstraction method is *potential-aware* automated abstraction, developed by Gilpin and Sandholm at Carnegie Mellon University [45]. *Expectation-based* abstractions require information about hand potential to be encapsulated within a one-dimensional hand strength value, whereas *potential-aware* abstractions make use of multi-dimensional histograms to assign similar hands into buckets.

Gilpin et al. [45] describe their procedure for generating potential-aware automated abstractions as follows. Firstly, given the desired size for the final linear program to be solved, the maximum number of buckets available for each round is decided upon. Buckets are populated for each individual playing round starting with the preflop. In order for the algorithm to capture information about hand potential, buckets for future rounds first need to be represented in order to derive buckets for the current round. This is achieved by performing several bottom-up parses of the game tree.

Consider the following example for computing preflop buckets. First, a bottom-up parse of the game tree is performed by bucketing all river hands into a much smaller set of river clusters. As there are no future cards to come after the river, there is no need to consider hand potential at this point, so river clusters are determined based on their final hand strength. The *k-means clustering* [66] algorithm is used to group river cards into their corresponding clusters. Once the set of river clusters has been determined, histograms for all possible hands on the turn are computed that represent the frequency of transitioning to each river cluster. For example, if there are 5 clusters on the river, each hand histogram on the turn will consist of 5 values, representing the possibility of ending up in each of the river clusters, given the final community card that is dealt for the river. The use of a histogram for each hand captures whether a hand has high potential or not, i.e. whether the hand has a high possibility of transitioning into a better cluster on the next round. Once histograms are determined for each hand on the turn, these are also clustered into a smaller set, again via k-means clustering. This process requires grouping together histograms that are similar to each other and hence requires a similarity function to be defined between histograms. Gilpin et al. [45] determine the similarity between two histograms by summing the Euclidean distance for each value in the histogram. This same process is repeated to create histograms for the flop and finally for the preflop. Once the initial bottom-up parse of the game tree has completed the result is a set of N buckets for the preflop stage of the game.

Establishing buckets for the flop and turn follow along similar lines, however instead of performing a complete bottom-up parse of the entire game tree, a series of smaller bottom-up parses are conducted. One bottom-up parse for each of the previously decided upon parent buckets is performed, where the combination of hands to consider is limited to only those hands that belong to the appropriate parent bucket.

A further requirement of deriving non-preflop buckets is to determine how many children each of the parent buckets should have, given the specified limit on the total number of buckets available for the current round. [45] achieve this by performing multiple k-means clusterings for each parent bucket, ranging from $k = 1$ up to some maximum value, such that the limit on the number of buckets per round constraint is satisfied. For each k, an error measure is calculated based on the distance of data points to their cluster centroids. Once these error measures are known, Gilpin et al. [45] solve an integer program that minimises the error, based on constraints that limit the number of buckets to the pre-specified amount. The result of solving this integer program determines the bucket structure for the non-preflop betting round.

Finally, as there is no potential to consider on the river, k-means clustering is performed on the final hand strength values to determine the river buckets. Once again an integer program is solved to determine how many children each of the turn buckets should have.

Gilpin and Sandholm [42] perform a comparative evaluation of automated *expectation-based* abstraction compared to *potential-aware* abstraction. They report that the *potential-aware* automated abstraction procedure outperformed *expectation-based* automated abstraction for finer-grained abstractions, whereas *expectation-based* abstraction was better for coarser abstractions. Furthermore, [42] show that *potential-aware* abstractions are *lossless* in the limit, i.e. given a fine enough abstraction and the computational power to solve the corresponding model, a *potential-aware* abstraction could be used to find an exact Nash-equilibrium strategy.

Imperfect Recall Abstractions

The abstractions mentioned above all assumed *perfect recall*. Perfect recall ensures a player is not required to forget previous observations and actions they have made at a particular point in the game. On the other hand, *imperfect recall* forces a player to forget previous observations. In human beings, imperfect recall may not be enforceable, but for artificial agents this type of forgetting is possible.

Imperfect recall has been investigated in the computer poker domain [120], due to the fact that creating an abstraction that relaxes the perfect recall assumption can allow more emphasis to be placed on the most recent information a player has at decision time, at a cost of forgetting (or blurring) the information about previous observations. In equilibrium finding approaches for computer poker, imperfect recall has the beneficial quality of reducing computational costs and allowing more expressive abstractions to be solved.

An imperfect recall abstraction allows more computational resources to be assigned to the current betting round by reducing the granularity or completely eliminating information from previous betting rounds. For example, an increased number of buckets could be assigned on the turn betting round by merging buckets from previous rounds together and hence forcing the player to consider fewer histories from previous rounds. An even more extreme abstraction could force a player to completely forget all previous round buckets, therefore allowing a finer distinction to be placed on the current round buckets. On the other hand, a *perfect recall* abstraction would require that the number of buckets used per round be reduced, as these need to be remembered and distinguished between during rounds.

Since the current state of the player's hand is more important than how it came to be, a trade-off can be made by allocating more computational resources for information that is important right now, in exchange for forgetting past information. However, by relaxing the perfect recall assumption, many theoretical guarantees are now lost and, in some cases, equilibrium finding algorithms are no longer well defined. Despite the loss of these guarantees, Waugh et al. [120] present results that suggest this is not an issue in practise. Their results show imperfect recall abstractions are superior to their perfect recall counterparts.

Other Abstractions

Overall, bucketing is a powerful abstraction technique that can reduce the number of nodes in the abstract game tree dramatically, however the consequence of this is that now multiple hands, which may not be strategically similar, will be mapped into the same category. In addition to bucketing two further lossy abstractions for Texas Hold'em are described below.

Betting round reduction In a typical game of limit hold'em each player is allowed to commit a maximum of 4 bets each to the pot during each betting round. By reducing the allowed number of bets the branching factor in the abstract game tree can also be reduced.

Elimination of betting rounds A more drastic abstraction involves the elimination of entire betting rounds. As an example, consider a game of poker where play ends at the turn. This reduces the size of the game tree by eliminating the last community card and the final round of betting. The effect of this abstraction can be softened by performing a roll-out of all 44 possible river cards and computing the expected value of the outcome, rather than simply truncating the game tree and awarding the pot to the player with the winning hand [18].

As lossy abstractions are required to produce full-scale poker agents they cannot be said to be true Nash equilibrium solutions for the original game, as solving the abstract game model cannot guarantee Nash equilibrium solutions will be preserved. Instead, this approach is said to produce approximate equilibrium solutions, which may come close to a true equilibrium strategy, but are not true equilibria, hence allowing the strategy to be exploitable to some degree. We will use the terms near-equilibrium and ϵ-Nash equilibrium interchangeably when referring to these strategies.

2.3.5. Near-Equilibrium Poker Agents

This section will survey a collection of poker agents that were produced using the techniques described above, i.e. by representing and solving large abstract game models as linear programs (LP). Most of the agents presented play 2-player Texas Hold'em and are split into limit and no-limit agents.

Heads-up Limit

Shi and Littman [109] produced a game theoretic player for the two player game of Rhode Island Hold'em. In Rhode Island Hold'em each player receives one personal card followed by a round of betting. There are only two community cards dealt separately between betting rounds. Shi and Littman's game theoretic player was able to beat a range of rule-based players.

Selby [106] focused on the preflop stage of Texas Hold'em to produce equilibrium strategies for the first round of play only. By performing a *roll-out* of the remaining community cards for future betting rounds, but not considering any betting during these rounds, payoff values were computed and an LP created that involved all 169 equivalence classes of preflop hands. Selby then solved this LP using the simplex algorithm to create an equilibrium strategy for preflop play.

Billings et al. [18] were the first to derive *near*-equilibrium solutions for full-scale two-player, limit Texas Hold'em by constructing and solving abstract game models that used a manually constructed, *expectation-based* abstraction that eliminated betting rounds. The collection of poker agents produced by Billings and colleagues using this approach are known as PsOpti, one version of which is publicly available as Sparbot within the commercial application Poker Academy Pro 2.5. Billings et al. [18] report that they were able to reduce the original size 10^{18} poker game tree down, by a factor of 100 billion, to separate models each of approximately 10^7 game states. In addition to the use of bucketing and betting round reduction, Billings et al. [18] compute and solve separately a preflop model and a number of postflop models which they attempt to tie together using preflop betting sequences to inform which postflop model to employ to effectively solve for the entire game tree. A combination of the PsOpti suite of agents, named Hyperborean, was able to defeat all competitors at the first AAAI Computer Poker Competition in 2006 [65].

Teppo Salonen's limit version of BluffBot [99] finished second in the same 2006 competition. The limit version of BluffBot is a near-equilibrium agent based on expert defined abstractions similar to PsOpti [99].

GS1 [39], developed by Gilpin and Sandholm from Carnegie Mellon University, also attempts to derive *near*-equilibrium solutions for limit hold'em. GS1 relies on very little poker domain knowledge to construct the abstract model. Instead, a system known as *GameShrink* [41] is used to automatically abstract the game tree. Given a description of a game, the *GameShrink* algorithm is able to automatically generate lossless or lossy abstractions to reduce the size of the game tree. For GS1 lossy abstractions were required to reduce the game tree to a manageable size, that was then able to be solved via linear programming. However, rather than solving for the entire game offline, as PsOpti does, GS1 uses a combination of offline and real-time equilibrium finding. For the first two stages of the game, i.e. the preflop and the flop, an abstracted model of the truncated game is generated and solved. Whereas, on the turn and river equilibrium solutions are computed in real-time during game play.

Gilpin and Sandholm later identified several weaknesses that resulted from using the automated, lossy abstractions produced by the *GameShrink* algorithm when applied to Texas Hold'em [40]. The next agent they developed, GS2 [40], used a better automated abstraction routine that relied on *k-means clustering* [66] to identify groups of strategically similar hands. Using these hand groups a truncated, abstract game tree (consisting of the preflop, flop and turn betting rounds) was constructed. This truncated model was then solved to derive a near-equilibrium strategy for GS2 to use during the preflop and flop betting rounds. Rather than rely on uniform roll-outs that assume no future betting to determine payoff values at the truncated leaf nodes (an assumption used by both GS1 and PsOpti), a further improvement introduced by GS2 is the use of simulated betting actions on the river to determine more accurate payoff values for the leafs of the tree [40]. As with GS1, GS2 again determines its betting strategy on the turn and river in real-time using the game state information. GS2 placed third in the limit equilibrium division of the 2006 AAAI competition [65].

Heads-up No-Limit

The agents mentioned in this section, so far, have solely focused on the limit variation of 2 player Texas Hold'em. In limit poker there are only three betting actions possible at every decision point in the game, i.e. fold, check/call or bet/raise. However, in no-limit a player may bet any amount up to the total value of chips that they possess. In a standard game of heads-up, no-limit poker both player's chip stacks would fluctuate between hands, e.g. a win from a previous hand would ensure that one player had a larger chip stack to play with on the next hand. In order to reduce the variance that this structure imposes, a variation known as *Doyle's Game* is played where the starting stacks of both players are reset to a specified amount at the beginning of every hand.

Recall that with limited betting the Texas Hold'em game tree consists of approximately 10^{18} nodes, whereas for a no-limit game, where each player begins with 1000 chips, the size of the game tree increases to roughly 10^{71} nodes [46]. Furthermore, increasing the starting stack sizes of the players has the effect of exponentially increasing the size of the game tree.

Aggrobot [6] is a poker playing agent similar to PsOpti, but modified to play the no-limit variation of Texas Hold'em. Aggrobot uses the same abstractions that were employed to create PsOpti with the addition of a further betting abstraction to handle no-limit betting. Aggrobot generates separate preflop and postflop models that are then combined to produce a *near*-equilibrium strategy for the game of no-limit hold'em. As the use of no-limit betting causes an exponential blow-up of the game tree, a further abstraction that discretises betting amounts is required to produce a tractable model. Aggrobot creates betting abstractions based on the value of chips that are currently in the pot. In total Aggrobot defines 4 separate betting abstractions which are: half the pot, the full amount in the pot, $2 \times$ the pot and all-in (all the player's remaining chips). All bets in the abstract model can now only belong to one of the above four categories. A consequence of this abstraction is that a reverse mapping for betting amounts is required when the actual game-play begins. If Aggrobot's opponent bets an amount that isn't represented by the above abstractions, such as $\frac{3}{2} \times$ the pot, this then needs to be classified as either a pot sized bet or a bet of double the pot. Simply choosing the betting abstraction that is closest to the bet amount results in an easily exploitable model [6, 46].

Due to processor and memory limitations [6] was only able to produce near-equilibrium strategies for a starting chip stack of at most 40 chips (assuming a 2 chip big blind).

Teppo Solonen has also produced no-limit versions of his agent BluffBot [99], starting with BluffBot2.0 which placed first in the heads-up, no-limit competition at the 2007 ACPC [2]. While exact details about the no-limit versions of BluffBot are not available, they are described as using a game theoretic *near*-equilibrium strategy.

Rather than focusing on hand for hand *cash game* play, as the above agents have, [71] computed game theoretic equilibrium *jam/fold* strategies for the end-game phase of a one table tournament where there are only two players remaining with a total of 8000 chips in play and a small blind and big blind of 300 and 600 respectively. By restricting the player's betting actions to *jam* (*all-in*) or fold, Miltersen and Sørensen [71] were able to compute exact Nash equilibrium solutions using standard LP procedures and showed that these restricted strategies also closely approximated a Nash equilibrium for the unrestricted tournament where other betting actions are allowed. A separate analysis by [36] used an extension of *fictitious play* (see section 2.3.6.) to compute *near*-equilibrium *jam/fold* strategies for the end-game of a one table tournament consisting of 3 players.

Procedures such as the simplex method and interior point methods for solving linear programs are severely restricted by the size of the model they can accommodate. As such [18] were forced to combine separate preflop and postflop models to approximate an equilibrium strategy. [39, 40] handled the problem by creating offline equilibrium solutions for the preflop and flop rounds and computing equilibrium strategies in real-time for the turn and river rounds. Producing *near*-equilibrium strategies in this way was later identified as problematic [11] due to the fact that simply gluing disjoint strategies together is not guaranteed to produce a complete and coherent equilibrium strategy.

Waugh et al. [118] in their paper on *strategy grafting* later showed that piecing together separately solved sub-games, which they refer to as *grafts*, was able to produce a coherent overall strategy as long as the *grafts* were tied together via a common base strategy. *Strategy grafting* allows finer abstractions to be solved, as each independent sub-game can be made as large as is tractably possible to solve. After separately solving each sub-game these are then *grafted* onto the *base strategy* to produce a completely unified strategy.

Next, we review a range of iterative algorithms for computing ϵ-Nash equilibria that are able to solve larger models due to having fewer memory requirements than standard LP procedures. Even with the *sequence form* representation, solving a model that considered all 4 betting rounds using simplex or interior point methods would likely require too coarse an abstraction (e.g. too few buckets) to produce reasonable quality strategies. The iterative algorithms, introduced in the next section, are able to solve fine abstractions involving all 4 betting rounds at once.

2.3.6. Iterative Algorithms for finding ϵ-Nash Equilibria

The previous section described an approach for finding equilibria in large scale games, where the sequence form representation was used to construct matrices that acted as constraints within an optimisation problem. By solving the problem using linear programming algorithms, equilibrium strategies were found for an abstract game. These were then used within the original game as ϵ-Nash equilibrium strategies. A major drawback of this approach is that representing the game in this way requires memory linear in the size of the game tree. There are however, other ways of finding equilibria in extensive form games that make use of iterative algorithms. A selection of these approaches are introduced below along with the poker agents they produce. We begin with a discussion of algorithms such as *fictitious play* and *range of skill*, which have been used to produce solid limit hold'em agents. Following this, two algorithms that represent the current state-of-the-art for computing ϵ-Nash equilibria are presented. The first is the *excessive gap technique*, which has been specialised for zero sum, extensive form games by Andrew Gilpin and colleagues at Carnegie Mellon University. The second state-of-the-art algorithm we review is counterfactual regret minimisation developed by the University of Alberta CPRG.

Fictitious Play

An alternative method for computing Nash equilibrium is via fictitious play [21]. Fictitious play revolves around the idea that as two players repeatedly play a game against each other, their strategies will steadily adapt and improve over time. The algorithm begins with each player adopting some arbitrary strategy, where a strategy is simply a probability distribution over actions at every information set. Both players are aware of the details of their opponent's strategy. A training phase occurs where random game situations are presented to the players. Each player can then determine the correct move in the situation, given the known strategy of their opponent. Players then update their average strategies based on this information. As the number of iterations increases the updated strategies typically approach a Nash equilibrium.

Dudziak [31] produced a *near*-equilibrium agent, called Adam, for 2-player, limit Texas Hold'em by abstracting the poker game tree and executing the fictitious play algorithm until it converged on a stable equilibrium strategy. Adam, was able to achieve a consistent win rate when challenging Sparbot and Vexbot from Poker Academy Pro 2.5, over a period of 20,000 and 50,000 hands respectively.

Two successful agents developed using fictitious play are INOT and Fell Omen 2 – both developed by Ian Fellows at the University of California, San Diego [32]. In 2007 INOT placed 2nd out of 15 competitors in the heads-up, limit instant run-off competition at the AAAI Computer Poker Competition. The next year, in 2008, INOT's successor, Fell Omen 2, placed 2nd equal out of a total of 9 competitors, in the same event.

Range of Skill

The *Range of Skill* algorithm is an iterative procedure that was developed by Zinkevich et al. [122] at the University of Alberta CPRG. The *Range of Skill* algorithm considers creating a sequence of agents, where the next agent created employs a strategy that can beat the previously created agent by at least a certain amount, ϵ. This sequence has a finite length. As the number of agents in the sequence approaches the maximum length, the agents' strategies approach an ϵ-Nash equilibrium, i.e. any further agents are not able to exploit the agent's strategy by more than ϵ.

The *Range of Skill* algorithm makes use of a separate algorithm known as *generalised best response*. The *generalised best response* algorithm computes a best response to a restricted set of allowed strategies, known as a *restricted game*. A best response is a strategy that maximises its utility against a specific strategy. A best-response can be considered the worst case opponent of a specific strategy.

The *Range of Skill* algorithm repeatedly calls the *generalised best response* algorithm. On each subsequent iteration the best response from the previous invocation is included in the set of allowed strategies of the restricted game and once again a best response is derived. The algorithm works by returning equilibrium strategies for restricted games of increasing size. As the number of iterations increases, the strategies returned approach an ϵ-Nash equilibrium.

This algorithm was used to produce a family of ϵ-Nash equilibrium agents known as SmallBot and BigBot. [122] presents SmallBot2298 and BigBot2298. SmallBot2298 uses a reduced betting abstraction where a maximum of 3 bets are allowed per round, whereas BigBot2298 considers the full 4 bets per round. Both SmallBot2298 and BigBot2298 are some of the first *near*-equilibrium strategies that consider a full four round betting model (preflop, flop, turn and river). Zinkevich et al. [122] shows that this offers a significant improvement upon previous approaches which rely on connecting together separate smaller models such as [18] and [39].

Excessive Gap Technique

Gilpin et al. [44] present an adapted version of Nesterov's *excessive gap technique* [75] that has been specialised for two-person, zero-sum imperfect information games. EGT refers to a procedure that allows smoothing to take place within an optimisation problem.

Previous linear programming based approaches made use of the sequence form representation and used procedures such as interior-point methods to solve the corresponding LP. The computational and memory costs associated with this approach resulted in having to severely restrict the size of the LP's that were able to be solved and required the combination of several distinct LP's in order to cover decision points for the entire game of 2-player limit Texas Hold'em. The use of EGT was one of the first major advances away from this approach, which allowed a non-truncated model to be solved that consisted of all four rounds of play.

The EGT algorithm presented by [44] is an iterative, anytime algorithm that requires $O(1/\epsilon)$ iterations to approach an ϵ-Nash equilibrium. Allowing more iterations results in lowering the ϵ value of the resulting ϵ-Nash equilibria produced. The algorithm makes use of the sequence form representation and directly handles the saddle-point formulation of the Nash equilibrium problem, given by the following equation:

$$\max_x \min_y x^T Ay = \min_y \max_x x^T Ay \tag{2.4}$$

where A is the payoff matrix for player 1, x is player 1's strategy and y is a strategy for player 2. Eq. 2.4 represents the situation in a two-player, zero-sum game where the first player is attempting to maximise their payoff and the second player is attempting to minimise the payoff given to the first player.

As some of the terms in the equation are non-smooth, i.e. not differentiable, first smoothing takes place. This results in smoothing the saddle point problem into differentiable, convex functions that can then be minimised. Once this is achieved *gradient-based* methods are used to solve the corresponding minimisation problem. A major benefit of the EGT algorithm is the low computational and memory costs associated with each iteration (the most costly operation being the performance of a matrix vector product). Using EGT, Gilpin et al. [44] were able to construct and solve abstractions that consisted of roughly 10^{10} game states.

The use of EGT resulted in one of the first agents to solve a model that consisted of all four rounds of play [45]. The result of solving a non-truncated game, together with the use of potential-aware automated abstraction (see section 2.3.4.) was a solid opponent, GS3, that was able to beat the best agents from the 2006 ACPC, as well as Sparbot and Vexbot with statistical significance [45].

Gilpin, Sandholm & Sørensen have also used EGT to create ϵ-Nash equilibrium strategies for no-limit poker. Tartanian [46], is able to build models for starting chip stacks of up to 1000 chips (cf. [6] which was only able to produce strategies for starting stacks of 40 chips). Tartanian solves a complete abstract model of the entire game using automated potential-aware card abstractions and discretisation of betting amounts. The actions allowed within the betting model chosen by [46] were similar to those chosen by [6] with the removal of the $2\times$pot bet. Gilpin et al. [46] also removed the possibility of a player becoming *pot committed* within the model, i.e. when a player commits a large proportion of chips compared to their chip stack such that they become committed to the hand and there is no point in folding. Out of a total of 10 competitors, Tartanian placed second in the no-limit hold'em competition at the 2007 AAAI Computer Poker Competition [46, 2].

Recently, Gilpin and Sandholm have presented two speed up improvements for EGT algorithms [43]. The first improvement is based on randomised sampling, whereas the second specialises the algorithm to make use of modern *ccNUMA* (*cache-coherent non-uniform memory access*) architecture, used in high-performance computing. The improvements can be applied together or separately and are able to offer significant reductions in computation time [43].

EGT remains at present, one of the state-of-the-art algorithms for constructing ϵ-Nash equilibria in two-player zero sum games.

Counterfactual Regret Minimisation

Another state-of-the-art algorithm for computing ϵ-Nash equilibria in two-player zero sum games is *Counterfactual Regret Minimisation* (CFR). CFR is an iterative algorithm for finding equilibria in extensive form games [123, 55]. The algorithm relies on iteratively minimising a counterfactual regret value. A major advantage of this algorithm is that it only requires memory linear in the size of a games *information sets*, rather than game states. The algorithm is only required to traverse the extensive form game tree, rather than store it in memory. A separate information set tree is stored for each player. A single node in an information set tree corresponds to an entire information set within the extensive form game tree.

Consider the example presented in Figure 2.6. The tree at the top represents a portion of the extensive form game tree for 2-player Texas Hold'em. Non-terminal nodes are labelled with either an A or a B to indicate which player they belong to. Information sets are highlighted by the dashed lines between nodes. The bottom tree is the corresponding information set tree. Each of the 3 nodes that belong to B's information set (in the top tree) will be mapped to 1 node in the information set tree. Correspondingly, each of the 3 nodes that belong to the information set shown for player A will also be mapped to a single node in the information set tree. Each node in the information set tree is required to also store values for the accumulated regret so far, at that information set. As only the information set tree is required to be stored in memory and not the extensive form game tree, this is what allows CFR to solve larger models than alternative algorithms, such as standard LP solvers. The strategies developed are therefore better approximations to the true equilibria of the original game [55].

The CFR algorithm is a regret minimising algorithm. An algorithm is regret minimising if, as the number of iterations goes to infinity, the overall average regret approaches 0. The concept of *regret* is similar to *opportunity cost* in economics. At each information set a player has a range of actions, a, to choose from, each with their own utility $u(a)$. If the maximum utility possible is given by taking action, a^*, a player's regret for choosing action a is then:

$$u(a^*) - u(a)$$

The CFR algorithm does not minimise one overall regret value, but instead decomposes regret values into separate information sets and minimises the individual regret values. The *counterfactual* aspect of the algorithm refers to the fact that calculations are weighted by the probability of reaching a particular information set, given that a player had tried to reach it. The summation of counterfactual regret from each information set provides a bound on overall regret, which when minimised gives a strategy that approaches a Nash equilibrium.

Counterfactual regret minimisation is one of the most promising algorithms in current computer poker research and has been used to create a range of successful hold'em agents in both heads-up limit [55] and no-limit variations [104], as well as multi-player games [92]. Next, we present an example of how counterfactual regret values are calculated and how these values help to determine an action probability distribution at each information set.

CFR Example

The algorithm begins by assigning arbitrary strategies to two players and having them play repeated perfect information games against each other. After every game the players' strategies are modified in a way that attempts to minimise their regret against the current strategy of their opponent. As the number of iterations increases, the combination of both players' strategies approaches an equilibrium strategy.

The following example will illustrate how CFR values are calculated at a single node in the extensive form game tree. These values are then added to accumulated CFR values within the information set tree, which are then used to update a player's strategy. The game being played is two-player, limit Texas Hold'em. The example is intended to give a general feel for the way

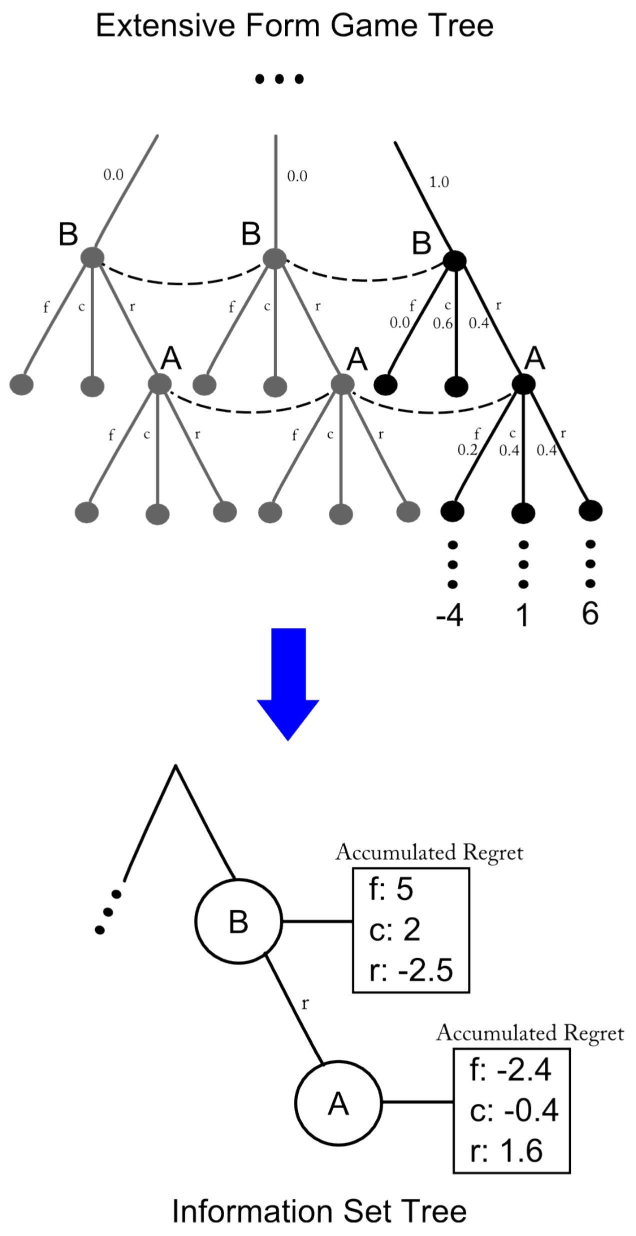

Figure 2.6: Depicts a partial extensive form game tree for two-player Texas Hold'em (top) and the corresponding portion of the information set tree (bottom). The extensive form game tree is required to be traversed, but not stored in memory. The information set tree is required to be stored in memory, it records accumulated regret values at each information set.

the algorithm proceeds. For a more detailed discussion on the theoretical background of the algorithm consult [123].

We refer again to Figure 2.6, which depicts a small portion of the extensive form game tree and the corresponding information set tree. The greyed out nodes indicate portions of the game tree that will not be reached, given the players' current strategies. An ellipsis signifies missing game information due to space considerations. We have chosen to calculate CFR values for a node belonging to player A, that is reached after player B makes a raise. The actions available to player A at their choice node are either fold, call or raise.

Let's assume that B's strategy indicates a 40% chance of raising in this situation and that A's current strategy specifies the following action probabilities at this node:

$$(f, c, r) = (0.2, 0.4, 0.4)$$

- The expected value for player A, given their current strategy is:

$$0.2 \times (-4) + 0.4 \times 1 + 0.4 \times 6 = 2$$

- For each action, *a*, at this choice node. Player A's *regret* for not taking action *a* is the difference between the expected value for taking that action, and the expected value given A's current strategy.

$$f : -4 - 2 = -6$$
$$c : 1 - 2 = -1$$
$$r : 6 - 2 = 4$$

- The values above indicate that A regrets not raising more than it regrets not calling or folding. These values then need to be weighted by the probability of actually reaching this node, which gives the following immediate counterfactual regret values.

$$f : -6 \times 0.4 = -2.4$$
$$c : -1 \times 0.4 = -0.4$$
$$r : 4 \times 0.4 = 1.6$$

- Recall that each node in the extensive form tree maps to a corresponding node in the information set tree. The information set tree can be thought of as storing the sum of the accumulated CFR values for all nodes in the same information set. After traversing all nodes in the extensive form tree, the information set tree is traversed and the accumulated CFR values are used to calculate the player's strategy for the next iteration.

- In this example, only 1 node in A's information set contributes to the final accumulated counterfactual regret values, because the probability of reaching the other nodes in the same information set is 0. Therefore, the information set tree stores the following accumulated CFR values:

$$f : -2.4$$
$$c : -0.4$$
$$r : 1.6$$

- The updated strategy for player A, at this information set, can be calculated as follows:

$$Let, D = max\{0, f\} + max\{0, c\} + max\{0, r\}$$

if $D > 0$ then

$$(f, c, r) = (\frac{max\{0, f\}}{D}, \frac{max\{0, c\}}{D}, \frac{max\{0, r\}}{D})$$

otherwise, a default strategy is chosen, e.g. $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

- Using the formulas above updates player A's strategy at this choice node to

$$(f, c, r) = (0, 0, 1)$$

- Hence, if player A reaches this node on a future iteration the updated strategy advises them to raise with 100% probability.

Recall, that previous approaches solved separate smaller models with approximately 10^7 states each [18, 39]. These approaches were forced to alter the game's structure by eliminating betting rounds, reducing the number of raises that were allowed and tying together separate preflop and postflop models. The CFR algorithm has been used to produce and solve models with up to 10^{12} states, that preserve the full Texas Hold'em betting structure and only rely on bucketing abstractions [123]. Generally, increases in the size of the model (the less abstraction that is used) correlate with improvements in performance (see Waugh and colleagues discussion on *abstraction pathologies* [119] for why this is not always the case). Zinkevich et al. [123]

and Johanson [55] show that CFR agents, produced with anywhere between 5 and 10 buckets per round, consistently perform better than one of the strongest agents from the PsOpti family.

The example above presented CFR in its vanilla form where each game state within an information set contributes to the overall accumulated counterfactual regret values in the information set tree. Further research efforts have shown that sampling game states, rather than travering every game state can offer significant improvements on convergence time. Monte Carlo CFR (MCCFR) modifies the original CFR algorithm by only updating information sets along one particular history during each iteration. MCCFR has been shown to converge to a Nash equilibrium much faster than standard CFR in several game domains [64].

CFR-based Poker Agents

The University of Alberta CPRG have used CFR to produce ϵ-Nash equilibrium strategies for both Hyperborean-Eqm and Polaris [20] since 2007[5]. Both Hyperborean-Eqm and Polaris are based on similar techniques, but refer to competitors in two separate events.

Polaris refers to the *“Machine”* entry in the Man-Machine Poker Championship in which some of the best human players challenge computerised opponents in limit Texas Hold’em. In 2008, Polaris defeated its human opponents by a statistically significant margin.

Hyperborean-Eqm refers to the group’s entry for the ACPC, where only computerised agents compete. From 2006 to 2008 Hyperborean-Eqm won every limit hold’em instant run-off competition at the ACPC. Both the limit and no-limit versions of the 2008 Hyperborean-Eqm agent were constructed using an *imperfect recall* abstraction [120]. In 2009, the instant run-off competition of the limit hold’em ACPC was won by GGValuta. GGValuta was developed by a team of students from the University of Bucharest based on the CFR minimisation algorithm [2]. GGValuta differs from Hyperborean-Eqm in the abstractions that are used to produce the model. In particular, GGValuta used a modified *k-means clustering* algorithm to assign postflop hands into groups, based on hand strength and potential [2]. In the same competition Hyperborean-Eqm placed 2nd. The 2009 version of Hyperborean-Eqm was constructed by solving separate independent sub-games (*grafts*) via the CFR algorithm and using *strategy grafting* [118] to combine the grafts via a base strategy.

Recently the CFR algorithm was applied to the 3-player game of limit hold’em [93]. The addition of the extra opponent produced a massive increase in the number of game states and hence a more coarse-grained abstraction was required to compute the ϵ-Nash equilibrium strategy [92]. Nevertheless, the resulting player, HyperboreanRing-Eqm, was able to beat all 6 opponents at the 2009 ACPC 3-player limit competition [2].

[5]Polaris actually consists of a range of strategies, at least one of which is an ϵ-Nash equilibrium strategy constructed via CFR.

CFR has also been used to create no-limit agents [104]. In particular, the agent HyperboreanNL-IRO won the instant run-off division of the 2010 heads-up, no-limit competition. The no-limit variation of the game requires abstract game trees that are much larger than the corresponding trees in the limit variation. Furthermore, no-limit involves a more complicated translation phase in order to have the strategies produced play appropriately in the full version of the game [103].

2.4. Exploitive Counter-Strategies

Nash equilibrium solutions are static, robust strategies that limit their own exploitability by maximising a minimum outcome against a perfect opponent. In a *two-person, zero-sum* game where the guaranteed minimum outcome is zero, a true Nash equilibrium strategy will never lose in the long run. However, while the strategy may never lose, it will also never identify and exploit weaknesses in its opponents' play. To exploit a weak opponent requires *opponent modelling*. An *exploitive counter-strategy* will determine its actions based on how it believes the opponent plays, as represented by the opponent model. The end result is that the exploitive counter-strategy will play off the equilibrium, therefore allowing itself to be exploitable, however it will also exploit weak opposition for more value than an equilibrium strategy.

In this section, we review agents that attempt to exploit their opponents by constructing opponent models. First we discuss *adaptive, exploitive* counter-strategies produced via imperfect information game tree search. This is followed by a review of *static, exploitive* agents that use game theoretic principles to create strategies that make compromises between limiting their own exploitability versus the level at which they exploit their opponents.

2.4.1. Imperfect Information Game Tree Search

In *two-player, zero-sum* games of *perfect information* the minimax search algorithm considers a Min and a Max opponent. The algorithm proceeds by assuming that when it is Max's turn to act it will perform an action that maximises its own utility. Correspondingly, when it is Min's turn to act they will choose an action to minimise the utility given to the Max player. At the leaf nodes of the game tree the exact outcome of the game is known. These values are then recursively backed up to intermediate nodes using the minimax procedure. In the case where the game tree can be fully searched to the leaf nodes the minimax algorithm corresponds to a Nash equilibrium [102]. Typically, for games with large search spaces it is not possible to search all the way down to the leaf nodes in the tree. Instead, an evaluation function is used to assign utility values to nodes at some cut-off depth.

The minimax algorithm is useful for games such as chess and checkers. Expectimax is an extension of the minimax algorithm that enables the procedure to also handle chance events, such as the roll of the dice or the dealing of cards. The expected value at a chance node is given by weighting each outcome's utility by its probability of occurrence and summing over all the possible outcomes [9]. This addition allows the expectimax algorithm to handle games that involve chance, e.g. backgammon.

Chess, checkers, backgammon etc. are all examples of perfect information games. In these domains the exact outcome of the game is known at the leaf nodes, or, if an evaluation function is being used, the public game information can provide a reasonable approximation for these values. A player's action at any node in the tree can then easily be determined by backing these values up from the leafs assuming Max will choose the action with the maximum value and Min will choose the action with the minimum value. Unfortunately, when a game involves hidden information (such as the hidden cards in poker) the exact expected value at the leaf nodes may no longer be known and the back up procedure can no longer provide a reasonable assumption of the opponent's actions.

Texas Hold'em is a game of imperfect information, as the hole cards of an opponent are hidden. Moreover, the cards a player holds usually determines that player's strategy. Without this information the likelihood of the opponent taking certain actions is unknown, as is the probability of winning at a showdown. For these reasons the expectimax algorithm cannot be directly applied to the game of Texas Hold'em. However, it is possible to augment the expectimax algorithm with an opponent model to "fill in" the missing information. The effectiveness of the algorithm then depends on how accurate the opponent model is. Two pieces of information required by the opponent model would be:

1. A distribution over an opponent's actions at each opponent choice node in the game tree. And,

2. The probability of winning at showdown leaf nodes.

Using the information from the opponent model allows expected values to be assigned to opponent nodes in the game tree by weighting each action's outcome by the likelihood of the opponent taking that action at a particular point in the game. Leaf nodes can also be assigned expected values by using the outcome probabilities estimated by the opponent model.

The University of Alberta CPRG have developed two algorithms based on the above ideas, **miximax** and **miximix** [28, 19]. The major difference between the two is that the miximax algorithm dictates that when choosing a player's action within the game tree the action that results in the maximum expected value should be chosen. On the other hand, the miximix proce-

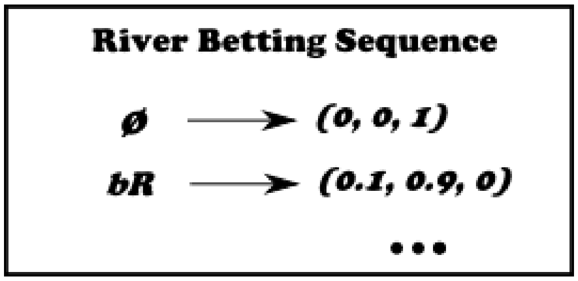

Figure 2.7: Part of Player A's opponent model of player B.

dure avoids always choosing the maximum action so as to balance the exploration/exploitation trade-off and avoid predictable behaviour of the agents produced.

Incomplete Information Game Tree Search Example

We now present an example to illustrate the calculation of leaf node expected values and how these values are backed-up to intermediate nodes in the imperfect information game tree using the miximax algorithm. Our example takes place on the last betting round in a game of heads-up, limit Texas Hold'em and involves two players, player A and player B. At the start of the last betting round there are 10 chips already in the pot, all further bets are in increments of 2 chips and player B acts first.

Player A's current beliefs about player B are represented by the opponent model in Figures 2.7 and 2.8. Figure 2.7 depicts the action frequencies player A has observed player B make at this point in the game in the past, i.e. A expects B to always bet when B is first to act and if raised A expects B to fold 10% of the time, call 90% of the time and never raise.

To simplify the calculations a player's hand rank is assigned into 1 of 5 separate buckets. Bucket 1 represents the bottom 20% of hand ranks, bucket 2 represents hand ranks in the range of 20% - 40% and so forth, all the way up to bucket 5 which indicates a hand rank in the top 20%. Figure 2.8 depicts two showdown histograms based on A's opponent model for player B. The top histogram corresponds to hand ranks that player B has showed down in the past after making a bet on the river. This tells us that player A has observed player B bet once with a hand in bucket 1, once with a hand in bucket 2 and 3 times with a hand in bucket 4. The bottom histogram represents the hand ranks that player B has showed down after being raised on the river. In this case B has called the raise once with a hand in bucket 2 and 3 times with a hand in bucket 4. For this example we assume that A has a hand rank in bucket 3.

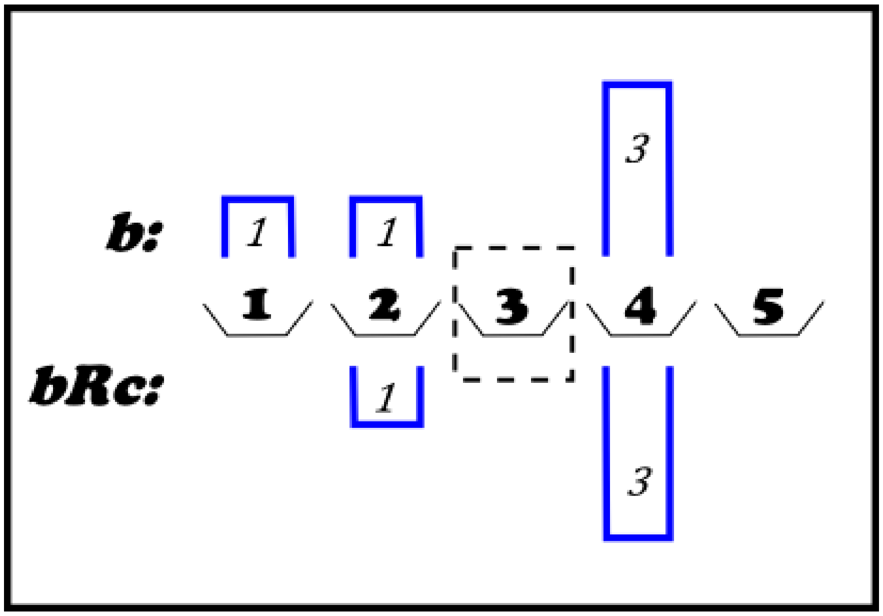

Figure 2.8: Two showdown histograms for betting sequences on the river. Player A's hand rank is in bucket 3.

We can now combine the information contained in the opponent model with a depth first search on the imperfect information game tree (see Figure 2.9). A simple formula is used to determine the leaf node values [19]:

$$EV(x) = Pr(Win) \times TotalPot - PlayerInvestment \tag{2.5}$$

Where, $Pr(win)$ is the probability that player A wins and x refers to a particular node in the game tree. In Figure 2.9 all nodes have been labelled with numbers to clarify the calculation. A depth first search on the game tree in Figure 2.9 proceeds as follows. As A's opponent model indicates that player B will never check in this situation we ignore that entire sub-tree, therefore the first leaf node we encounter (node 2) represents the situation when A folds to player B's initial bet. Applying formula (2.5) to the leaf node gives:

$$EV(2) = 0 \times 12 - 5 = -5$$

The probability of winning when player A folds is trivial to calculate, i.e. it is always 0. There are a total of 12 chips in the pot (10 plus the 2 chip bet by player B) and A has invested 5 of these chips, which gives an EV of -5. However, if A calls the initial bet made by B (node 3) instead of folding, now calculating the probability of winning requires the use of A's opponent model.

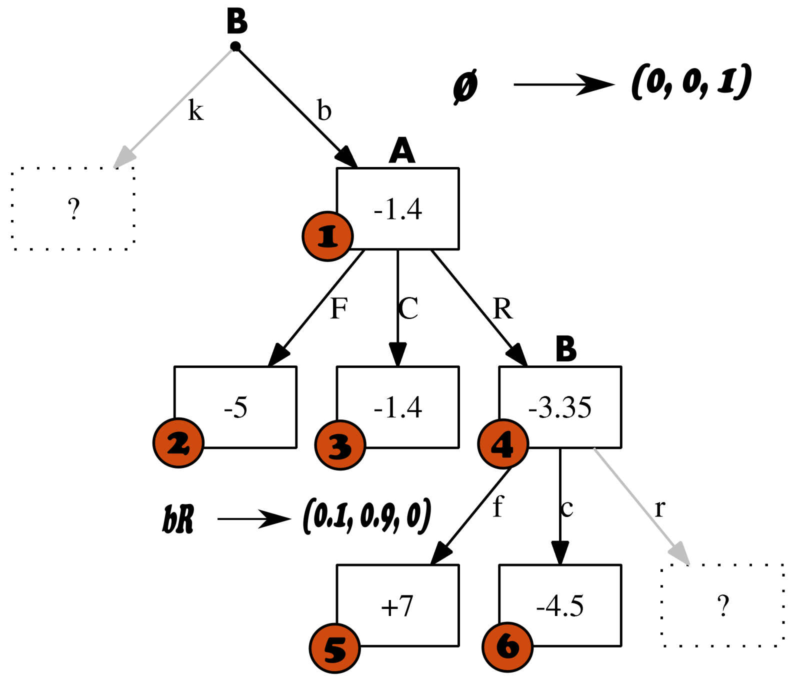

Figure 2.9: Expected value calculation using the miximax algorithm.

Recall that, based on A's opponent model in Figure 2.8, player A can expect to win a showdown 2 out of 5 times if A calls B's bet. Inputting this value into formula (2.5) gives:

$$EV(3) = \frac{2}{5} \times 14 - 7 = -1.4$$

The last action A has available at this point is to raise. Choosing this action results in a sub-tree in which the expected values of the leaf nodes will have to be calculated as above. As A's opponent model predicts B will never raise in this situation (i.e. $bR \rightarrow$ (0.1, 0.9, 0.0) in Figure 2.7), the sub-tree that would have been generated at this point can be ignored as its value will only end up being multiplied by 0. This leaves EV calculations assuming that B folds (node 5) or calls (node 6):

$$EV(5) = 1.0 \times 16 - 9 = +7$$
$$EV(6) = \frac{1}{4} \times 18 - 9 = -4.5$$

Given the expected values for the leaf nodes we can now back-up these values to intermediate nodes using the miximax procedure. Recall, that at opponent nodes the expected value is backed-up by multiplying the distribution over opponent actions (given by the opponent model) by the expected values calculated at the child nodes. For node 4, this is given by:

$$EV(4) = 0.1 \times (+7) + 0.9 \times (-4.5) = -3.35$$

The backed-up expected value at node 1 is then just the maximum of the expected values of the child nodes, i.e. $EV(1) = -1.4$.

The outcome of the imperfect information tree search indicates that, given A's current beliefs about player B, A should simply call B's initial bet as this gives the maximum expected value.

Adaptive Game Tree Search Agents

Using the above ideas the University of Alberta CPRG produced two heads-up, limit Texas Hold'em agents that are both adaptive and exploitive. Vexbot [19] is an imperfect information game tree search based agent that is available as part of the commercial software product Poker Academy Pro 2.5. A separate, but similar agent is BRPlayer [102]. Both Vexbot and BRPlayer use *context tree* data structures within their opponent models, which disregard the chance events and instead only record betting sequences. Each betting sequence in the context tree records opponent action frequencies for intermediate nodes and observed hand rank histograms at showdown leaf nodes. As few observations have been made during early stages of play, both Vexbot and BRPlayer perform generalisations at showdown leaf nodes by defining various metrics to identify similar betting sequences which can then be used to bolster observations at a particular leaf node. For example, betting sequences that exactly match would correspond to the highest level of similarity. The next level of similarity would consider a more coarse generalisation, such as counting the total number of bets and raises that were made by both the player and the opponent during each betting round. In the case that there are not enough observations available in the first similarity level, observations from the next level can be included in the model. This process would continue for further levels of similarity until a required number of observations was attained, with greater levels of similarity being assigned greater weight than observations from lower levels.

The main difference between Vexbot and BRPlayer is that BRPlayer considers more levels of (fine grained) similarity compared to Vexbot. In total BRPlayer considers 5 separate similarity levels whereas Vexbot considers 3 levels [102]. Results reported for Vexbot show that it is able to exploit a wide range of computerised opponents [19]. Both Vexbot and BRPlayer were able to discover and exploit weaknesses in Sparbot's *near*-equilibrium strategy. Furthermore, Vexbot was able to exploit the static benchmark strategies of Always-Call and Always-Raise at a level that closely approximates the theoretically computed maximum exploitability of those strategies [19].

McCurley [69] applied the miximax algorithm to heads-up, no-limit hold'em. Rather than recording action frequencies to predict opponent actions, McCurley [69] trained artificial neural networks (ANN) with hand history data obtained from online poker sites. The hand history data was clustered to create opponent models based on different playing styles. During game tree search the ANN is used at opponent choice nodes to predict action frequencies. A weighted hand range is also maintained. Hand ranks in the range, as well as other game state information, are used as inputs into the ANN. The weighted hand range is also used to determine EV at showdown leaf nodes. Due to computational costs, the imperfect information game tree search was only conducted for the current betting round, with future betting rounds being ignored. However, the results reported by [69] indicate that the agent was still profitable against human opposition on an online poker site.

Maîtrepierre et al. [67] created an adaptive heads-up, limit hold'em agent called Brennus. Rather than using miximax search to explore an imperfect information game tree, Brennus constructs a forest of game trees. Within each separate game tree the opponent is assigned a single weighted pair of hole cards, where the weight represents Brennus's belief that the opponent actually has the assigned hand. The weighted expected values from each tree are then combined to determine a betting action.

Brennus maintains a *belief table* during each hand, which consists of all possible hole cards and a weight specifying the probability that the opponent has the corresponding hole cards. At the beginning of a hand all pairs of hole cards are assigned uniform probability. As the hand continues the probabilities are updated (via *Bayes' Theorem*) based on the actions that Brennus witnesses the opponent make. To update the probability values Maîtrepierre et al. [67] rely on a set of basis opponent models that define particular styles of play (e.g. tight/aggressive).

Using the opponent models to guide the agent's beliefs combined with the forest of game tree computations, gives Brennus five basic strategies that it must dynamically select during play. To do so, Brennus uses the UCB1 [7] policy selection procedure to appropriately handle the **exploration/exploitation** trade-off. Using the approach just described Brennus placed 8th out of 17 competitors at the 2007 AAAI Computer Poker Competition [67, 2].

2.4.2. Game-Theoretic Counter-Strategies

Imperfect information game tree search creates exploitive, adaptable agents by generating and searching the game tree in real-time. This section reviews static, counter-strategies that are derived using offline, game theoretic algorithms.

Frequentist Best Response

Frequentist Best Response (FBR) [54, 55] is an exploitive strategy that builds an offline opponent model for a chosen opponent by observing training games involving that opponent. FBR assumes a static opponent strategy. This results in FBR counter-strategies that are exploitive, but not adaptive. The following items briefly summarise the FBR procedure:

Preliminaries:

- Determine the abstraction to use (recall section 2.3.4.). FBR can be used within any abstraction.
- Determine a default policy. Given that FBR relies on observations to build its opponent model, it is likely that gaps in the strategy will surface where no observations have been made in the training phase. A typical default policy is always call.

Training:

- To develop an accurate and complete model of the opponent it is important to observe many training games consisting of full information i.e. hole cards must be known. The training phase involves mapping observations of played hands in the real game to frequency counts in the abstraction chosen above. The more training games observed, the better the opponent model will become. As observations are being made about how the opponent plays in the real game, the choice of the opponent's opponent will also impact on the resulting model. For example, an opponent who mostly folds is going to reduce the number of observations in the model. Johanson [55] recommends a simple "Probe" opponent whose strategy involves never folding, but instead always calls and raises with equal probability. The use of this simple opponent ensures that areas in the state space will be explored that otherwise may not have been, given a more sophisticated opponent.

Best-response:

- Once the frequentist opponent model has been constructed a best-response is computed in the chosen abstraction. Computing a best-response roughly involves visiting every information set in the abstraction and determining the action that will maximise the expected value against the opponent's strategy. The frequentist opponent model derived in the training phase is used as an approximation of the opponent's actual strategy in the given abstraction.

Johanson [55] reports that a typical FBR counter-strategy requires approximately 1 million training games against the "probe" opponent to generate an appropriate opponent model in a 5 bucket abstraction. Using this model and an always-call default policy, Johanson presents results that indicate FBR counter-strategies are able to significantly exploit the opponents they were designed to exploit. However, using these counter-strategies to challenge other opponents, who they weren't designed for, results in significant losses for the FBR strategies, highlighting their extremely brittle nature. Nevertheless, the FBR algorithm remains a powerful tool for determining the exploitability of an opponent.

Restricted Nash Response

The Restricted Nash Response (RNR) [54, 55] algorithm attempts to bridge the gap between FBR and ϵ-Nash equilibria. While FBR strategies are able to exploit opponents they were designed to defeat by large margins, the losses they incur against other opponents cancel out these winnings and degrade their overall effectiveness. RNR, on the other hand attempts to produce counter-strategies that are still able to exploit specific opponents while limiting their own exploitability.

The RNR algorithm involves solving a modified game where the opponent plays a known fixed strategy with probability p, and is free to play an unrestricted game theoretic strategy with probability $1-p$. Any method that computes a Nash equilibrium can be used to solve the game, such as solving a linear program or using the counterfactual regret minimisation algorithm. The result of finding an ϵ-Nash equilibrium in this modified game produces counter-strategies that will attempt to exploit the known fixed strategy, while still protecting the counter-strategies own exploitability. To determine the opponent's fixed strategy FBR is used. Intuitively, when $p = 0$ the strategies produced are ϵ-Nash equilibrium strategies, whereas when $p = 1$ the counter-strategy is a best-response to the fixed strategy. Where p varies between 0 and 1, a trade-off is made between the counter-strategies exploitation of its opponent and its own exploitability.

While the counter-strategies produced using RNR are not able to exploit their intended opponents by as much as FBR, they also are not beaten by nearly as much as the FBR counter-strategies when facing opponents they were not designed for [55]. As such, they have been labelled robust counter-strategies.

Just as with the vanilla implementation of the CFR algorithm, RNR algorithms have also been extended by introducing selective sampling in replacement of full tree traversal. Monte-Carlo RNR (MCRNR) [87] is one such algorithm that allows a faster rate of convergence than its non-sampled counterpart.

Data Biased Response

The last approach that we discuss in this line of research that produces non-adaptive, exploitive counter-strategies is Data Biased Response (DBR) [53]. DBR produces robust counter-strategies in a manner similar to RNR, described above.

As with RNR, DBR computes a Nash equilibrium in a modified game where the opponent is forced to play a fixed strategy a certain proportion of the time. However, instead of constraining the opponent's entire strategy with one p value, DBR provides individual p_{conf} values for each information set in the game tree. The opponent is then forced to play a fixed strategy at that particular location in the game tree with probability p_{conf} and is free to play an unrestricted strategy with probability $1 - p_{conf}$. The use of only one p value in the RNR algorithm imposes an unrealistic expectation on the opponent model. It is more likely that the opponent model will have more observations at some areas in the game tree and fewer observations in other areas. In this way p_{conf} acts as a confidence measure of the accuracy of the opponent model. [53] experimented with various functions to calculate values for p_{conf} which vary between 0 and p_{max}, where p_{max} refers to the final trade-off between exploitation and exploitability, similar to p in the RNR approach.

An advantage of DBR is that it eliminates the need for a default policy. For information sets where no observations have been recorded for the opponent model, a p_{conf} value of 0 can be returned, allowing the opponent to play an unrestricted game theoretic strategy at this point. As the number of observations increase p_{conf} will return greater values making it more likely that the opponent will be constrained to take the action dictated by the opponent model at a particular information set. Johanson and Bowling [53] present results that show that DBR does not overfit the opponent model, like RNR tends to. Furthermore, with fewer observations DBR is still an effective strategy for constructing robust counter-strategies. The DBR procedure was used to construct the 2009 version of the agent Hyperborean-BR [52], which placed second in the 2009 ACPC 2-player limit bankroll competition.

Teams of Agents

The descriptions above suggest that a single counter-strategy will successfully exploit the intended opponent. However, when challenging other competitors, dissimilar to the opponents they were designed to exploit, then the counter-strategies will not fare as well. A brittle counter-strategy, such as FBR, is likely to lose a lot in this situation, whereas a robust counter-strategy, such as RNR or DBR, will limit their own exploitability, but will be unable to exploit the new opponent.

To enable the exploitation of a range of dissimilar opponents, a team of counter-strategies is required, where each member of the team is able to exploit a different style of opponent. During game play it is then required to select the strategy that achieves the greatest profit against the current opponent. Johanson [55] describes the use of a *coach*, that determines the appropriate strategy to employ against an opponent using the UCB1 algorithm [7]. The UCB1 algorithm defines a policy selection procedure that achieves an effective trade-off between exploration and exploitation. In the case of a team of poker agents, strategy selection must consider the benefits of re-using a strategy that is known to successfully exploit an opponent for a particular amount (exploitation), compared to selecting a different strategy that could potentially exploit the opponent for more (exploration). Johanson et al. [54] present results which show that a team of robust counter-strategies are able to achieve a greater profit against a set of opponents, compared to the use of a single ϵ-Nash equilibrium strategy against the same opponents.

2.5. Alternative Approaches

This final section briefly introduces some alternative approaches for constructing poker strategies that have not been mentioned in the previous sections.

2.5.1. Evolutionary Algorithms and Neural Networks

The use of evolutionary algorithms, to automatically evolve strong poker agents, has also been the subject of investigation [77, 78, 79, 10, 89]. Evolutionary procedures evolve a population over successive generations. Members of the population are typically evaluated via the use of a *fitness function*. Members of the population that achieve higher fitness are more likely to proceed to the next generation, in which they produce offspring via a *crossover* procedure. *Coevolution* maintains separate genetic populations, where members from one population are able to compete against members from another, but evolution is restricted to within their separate populations.

A notable effort from this line of research is presented by Jason Noble in [78]. Noble used *Pareto coevolution* (PC) to evolve limit hold'em agents that compete in full ring games with up to 10 players. The members of each population were artificial neural networks (ANN) that compete against each other over numerous generations. Separate ANNs were used for the preflop and postflop stages, where the inputs of each network mostly consisted of hand evaluation, opponent modelling information and other game state parameters. The output nodes of the networks correspond to fold, call and raise betting actions. A previous effort, from the same author, used a less sophisticated rule-based representation [79].

Noble identifies the potential problem of using a simplistic one-dimensional *fitness function* that rewards ANNs that achieve a greater average bankroll compared to other ANNs. The problem is due to the intransitive nature of poker strategies, i.e. while strategy A may beat strategy B and strategy B may beat strategy C, it does not necessarily follow that A beats C. Rather, Noble attempts to evolve robust strategies by selecting members of a population based on multi-dimensional optimisation. To do so Noble [78] bases selection on *Pareto dominance*, where a player, A, *Pareto-dominates* another player, B, if A performs at least as well as B against all opponents (including against B itself).

Noble [78] presents experiments that compare PC with standard coevolutionary algorithms where selection is based on average bankroll balance. 5000 generations of evolution take place where the members of two populations compete against a *reference group* of strategies. The results show that PC produces a steady improvement against the reference group, whereas the performance of the standard coevolutionary procedure, against the same reference group, varies wildly. The results indicate that knowledge was able to be maintained over successive generations using PC.

Nicolai and Hilderman [77] use a similar evolutionary procedure to evolve agents that play no-limit Texas Hold'em at a full table consisting of 9 players. Once again ANNs are used as members of the population. A static network structure is defined that consists of 30 input nodes, 20 hidden nodes and 5 output nodes. Evolution takes place by refining the weights used within the networks over successive generations.

Nicolai and Hilderman [77] use a tournament structure during the evolutionary process to assess a player's performance. In the tournament structure all players begin a match with an equal number of playing chips and play continues until one player possesses all the chips. Players are then ranked based on their tournament placement, e.g. at a 9 player table the first player eliminated receives a rank of 9, the next 8, then 7 etc. The winner of the tournament is assigned a rank of 1. *Selection* is based on the average rank obtained over multiple tournaments. The *selected* players then act as parents within the next generation. Nicolai and Hilderman [77] define a crossover procedure that produces offspring using a biased sum over the weights of the parent networks. The evolved ANNs were evaluated by challenging a range of static benchmark strategies, such as Always-Call, Always-Raise and Always-Fold together with a group of evolved agents from previous work [10]. The results show that improvements were observed over a baseline evolutionary algorithm by introducing separate coevolved populations (*coevolution*), as well as the introduction of a *hall of fame* heuristic, where members of the population compete against a distinct set of *high fitness* past players, rather than competing against members from other populations.

A further neural network based agent worthy of mention is MANZANA. MANZANA is a limit hold'em agent, created by Marv Andersen, that won the bankroll division of the limit hold'em competition at the 2009 and 2010 ACPC [2]. While relatively few details of MANZANA's design and implementation are available, it is known that separate ANNs are used for preflop and post-

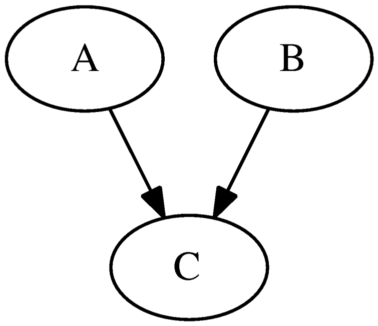

Figure 2.10: A directed acyclic graph. A, B and C are random variables. Nodes A and B are the parents of node C and therefore they influence node C, represented by the edges.

flop play [5]. Furthermore, rather than tuning the weights of the neural network via evolution (as the agents above did) MANZANA was trained on the hand histories of the top agent from the previous year's ACPC [2].

2.5.2. Bayesian Poker

The last alternative approach we review focuses on the design and application of *Bayesian networks* to produce poker playing agents. A *Bayesian network* is a directed acyclic graph where each node in the graph represents a random variable [86]. The edges between nodes in the graph represent dependency relationships (see Figure 2.10).

Each node within the network has an associated *conditional probability table* (CPT), which allows conditional probability values to be calculated based on the values of the parent nodes. By assigning nodes within the network different values, probabilities can be propagated throughout the network, resulting in a probability distribution over the random variables.

An *influence diagram* (or *decision network*) augments a Bayesian network with information about the utility of outcomes within the network. In this way Bayesian networks can be used to inform decisions based on maximising expected utility.

An example Bayesian network, adapted from [62] is illustrated in Figure 2.11. Here the network identifies variables for a player and an opponent. Looking at the direction of the edges in the graph, the network indicates that the final hand of a player and their opponent determines whether the player wins the hand or not. The current hand is influenced by the final hand type and the opponent's action is influenced by their current hand

AI researchers Nicholson and Korb from Monash University have spent many years investigating the use of Bayesian networks within the domain of poker, beginning with five-card stud

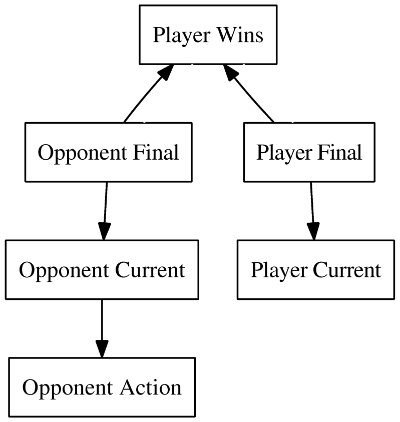

Figure 2.11: A Bayesian network for poker, adapted from original image presented in [62]

[62] and later adapting this approach to Texas Hold'em [76]. Korb, Nicholson *et. al.* have designed and applied Bayesian decision networks for heads-up, limit hold'em resulting in a poker agent called BPP (Bayesian Poker Program). The networks used by BPP consist of variables such as the amount of chips in the pot, the opponent's action, BPP's current and final hand type, the opponent's current and final hand type and whether BPP will win the hand given the final hand types [76]. The estimated probability that BPP will win the hand is then used to make a betting decision that attempts to maximise BPP's total winnings.

While BPP has undergone several stages of development, results against other computerised players indicate there is still further room for improvement. BPP competed in both the 2006 and 2007 AAAI Computer Poker Competitions. In 2006 BPP placed 3rd out of 4 competitors in the limit hold'em bankroll competition and 4th out of 4 in the instant run-off competition [65]. In 2007, BPP lost to all, but one competitor placing 16th out of 17 in the bankroll competition and 14th out of 15 in the instant run-off competition [2].

2.6. Summary

We have presented a review of recent algorithms and approaches in the area of computer poker, along with a survey of some of the autonomous poker agents produced using the algorithms described. Each approach presented has highlighted its own specific opportunities and challenges for AI research within the poker domain. For example, research into ϵ-Nash equilibrium strategies has mostly been driven by solving larger game models. As a result, solving models that require fewer simplifying abstractions has typically resulted in performance improvements and better automated poker agents. On the other hand, a major challenge for research into exploitive agents involves the construction of accurate opponent models in real-time with limited data, that have the ability to adjust quickly to changing opponents. For Monte-Carlo simulations, problems were addressed such as how best to bias the sampling distribution to accurately reflect the game conditions and hence perform a better informed search of the state space. Alternative research efforts within the poker domain have also highlighted challenging problems such as how to best select strategies within an evolutionary procedure given the intransitive nature of poker strategies.

We now turn our attention to the construction and analysis of case-based strategies for computer poker. In Chapter 3 we introduce a framework for the construction of case-based poker agents via expert imitation in three separate poker domains. The frameworks introduced are the result of an iterative period of maintenance and empirical evaluation. Chapter 3 also addresses issues such as whether strong, sophisticated strategies can successfully be produced via observation and generalisation and how closely an original strategy can be approximated via imitation. In later chapters, we discuss how the frameworks introduced in Chapter 3 can be extended to improve overall performance and we seek to determine whether strategies based on expert imitation can actually outperform the original experts that were used to train the system.

Chapter 3

Case-Based Strategies in Computer Poker

3.1. Introduction

[1]The state-of-the-art within artificial intelligence research has directly benefited from research conducted within the computer poker domain. Perhaps its most notable achievement has been the advancement of equilibrium finding algorithms via computational game theory. State-of-the-art equilibrium finding algorithms are now able to solve mathematical models that were once prohibitively large. Furthermore, empirical results tend to support the intuition that solving larger models results in better quality strategies (see [119] for a discussion of why this is not always the case). However, equilibrium finding algorithms are only one of many approaches available within the computer poker test-bed. Alternative approaches such as imperfect information game tree search [19] and, more recently, Monte-Carlo tree search [115] have also received attention from researchers in order to handle challenges within the computer poker domain that cannot be suitably addressed by equilibrium finding algorithms, such as dynamic adaptation to changing game conditions.

The algorithms mentioned above take a *bottom up* approach to constructing sophisticated strategies within the computer poker domain. While the details of each algorithm differ, they roughly achieve their goal by enumerating (or sampling) a state space together with its pay-off values in order to identify a distribution over actions that achieves the greatest *expected value*. An alternative *top down* procedure attempts to construct sophisticated strategies by generalising decisions observed within a collection of data. This *lazier* top down approach offers its own

[1]The contents of this chapter originally appeared in the journal *AI Communications*. Jonathan Rubin and Ian Watson. Case-Based Strategies in Computer Poker. *AI Communications*, 25(1):19–48 (2012)[97]

set of problems in the domain of computer poker. In particular, any top down approach is a slave to its data, so quality data is a necessity. While massive amounts of data from online poker sites is available [90], the quality of the decisions contained within this data is usually questionable. The imperfect information world of the poker domain can often mean that valuable information may be missing from this data. Moreover, the stochastic nature of the poker domain ensures that it is not enough to simply rely on outcome information in order to determine decision quality.

In this thesis we investigate a *top down* approach that uses case-based reasoning to construct sophisticated strategies within the computer poker domain. We address the problems introduced by *top-down* strategies, mentioned above, and evaluate the performance of poker agents produced using a *top-down* approach compared to their *bottom-up* counterparts. Our *case-based approach* can be used to produce strategies for a range of sub-domains within the computer poker environment, including both limit and no-limit betting structures as well as two-player and multi-player matches. We have applied and evaluated case-based strategies within the game of Texas Hold'em. Texas Hold'em is currently the most popular poker variation. To achieve strong performance, players must be able to successfully deal with imperfect information (i.e. they cannot see their opponents' hidden cards) and chance events (the random distribution of playing cards).

In this chapter, we present case-based strategies in three poker domains:

1. Two-player, Limit Hold'em.
2. Two-player, No-Limit Hold'em.
3. Three-player, Limit Hold'em.

Our analysis begins within the simplest variation of Texas Hold'em, i.e. two-player, limit hold'em. Here we trace the evolution of our case-based architecture and evaluate the effect that modifications have on strategy performance. The end result of our experimentation in the two-player, limit hold'em domain is a coherent framework for producing strong case-based strategies, based on the observation and generalisation of expert decisions. The lessons learned within this domain offer valuable insights, which we use to apply the framework to the more complicated domains of two-player, no-limit hold'em and multi-player, limit hold'em. We describe the difficulties that these more complicated domains impose and how our framework deals with these issues. For each of the three poker sub-domains mentioned above we produce strategies that have been extensively evaluated. In particular, we present results from Annual Computer Poker Competitions for the years 2009 – 2012 and illustrate the performance trajectory of our case-based strategies against the best available opposition.

3.2. Case-Based Reasoning Motivation

Our approach makes use of the Case-Based Reasoning methodology [91, 61, 1]. The CBR methodology encodes problems, and their solutions, as cases. CBR attempts to solve new problems or scenarios by locating similar past problems and re-using or adapting their solutions for the current situation. Case-based strategies are top down strategies, in that they are constructed by processing and analysing a set of training data. Common game scenarios, together with their playing decisions are captured as a collection of cases, referred to as the *case-base*. Each case attempts to capture important game state information that is likely to have an impact on the final playing decision. The training data can be both real-world data, e.g. from online poker casinos, or artificially generated data, for instance from hand history logs generated by the ACPC. Case-based strategies attempt to generalise the game playing decisions recorded within the data via the use of similarity metrics that determine whether two game playing scenarios are sufficiently similar to each other, such that their decisions can be re-used.

Case-based strategies can be created by training on data generated from a range of expert players or by isolating the decisions of a single expert player. Where a case-based strategy is produced by training on and generalising the decisions of a single expert player, we refer to the agent produced as an *expert imitator*. In this way, case-based strategies can be produced that attempt to imitate different styles of play simply by training on separate datasets generated by observing the decisions of *expert* players, each with their own style. The lazy learning [3] of case-based reasoning is particularly suited to expert imitation where observations of expert play can be recorded and stored for use at decision time.

Case-based approaches have been applied and evaluated in a variety of gaming environments. CHEBR [88] was a case-based checkers player that acquired experience by simply playing games of checkers in real-time. In the RoboCup soccer domain, [34] used case-based reasoning to construct a team of agents that observes and imitates the behaviour of other agents. Case-based planning [49] has been investigated and evaluated in the domain of real-time strategy games [4, 81, 80, 112]. Case-based tactician (CaT) described in [4] selects tactics based on a state lattice and the outcome of performing the chosen tactic. The CaT system was shown to successfully learn over time. The Darmok architecture described by [81, 80] pieces together fragments of plans in order to produce an overall playing strategy. Performance of the strategies produced by the Darmok architecture were improved by first classifying the situation it found itself in and having this affect plan retrieval [72]. Combining CBR with other AI approaches has also produced successful results. In [107] transfer learning was investigated in a real time strategy game environment by merging CBR with reinforcement learning. Also, [8] combined CBR with reinforcement learning to produce an agent that could respond rapidly to changes in conditions of a domination game.

One of the major benefits of using case-based strategies within the domain of computer poker is the simplicity of the approach. Top down case-based strategies don't require the construction of massive, complex mathematical models, that some other approaches rely on [44, 103, 93]. Instead, an autonomous agent can be created simply via the observation of expert play and the encoding of observed actions into cases. Below we outline some further reasons why case-based strategies are suited to the domain of computer poker and hence worthy of investigation. The reasons listed are loosely based on Sycara's [113] identification of characteristics of a domain where case-based reasoning is most applicable (these were later adjusted by [117]).

1. **A case is easily defined in the domain.**

   A case is easily identified as a previous scenario an (expert) player has encountered in the past and the action (solution) associated with that scenario such as whether to fold, call or raise. Each case can also record a final outcome from the hand, i.e. how many chips a player won or lost.

2. **Expert human poker players compare current problems to past cases.**

   It makes sense that poker experts make their decisions based on experience. An expert poker player will normally have played many games and encountered many different scenarios; they can then draw on this experience to determine what action to take for a current problem.

3. **Cases are available as training data.**

   While many cases are available to train a case-based strategy, the quality of their solutions can vary considerably. The context of the past problem needs to be taken into account and applied to similar contexts in the future. As the system gathers more experience it can also record its own cases, together with their observed outcomes.

4. **Case comparisons can be done effectively.**

   Cases are compared by determining the similarity of their local features. There are many features that can be chosen to represent a case. Many of the salient features in the poker domain (e.g. *hand strength*) are easily comparable via standard metrics. Others features, such as *betting history*, require more involved similarity metrics, but are still directly comparable.

5. **Solutions can be generalised.**

   For case-based strategies to be successful, the re-use or adaptation of similar cases' solutions should produce a solution that is (reasonably) similar to the actual, known solution (if one exists) of the target case in question. This underpins one of CBR's main assumptions: that similar cases have similar solutions. We present empirical evidence that suggests the above assumption is reasonable in the computer poker domain.

3.3. Two-Player, Limit Texas Hold'em

We begin with the application of case-based strategies within the domain of two-player, limit Texas Hold'em. Two-player, limit hold'em offers a beneficial starting point for the experimentation and evaluation of case-based strategies, within computer poker. Play is limited to two players and a restricted betting structure is imposed, whereby all bets and raises are limited to pre-specified amounts. The above restrictions limit the size of the state space, compared to hold'em variations that allow no-limit betting and multiple opponents. However, while the size of the domain is reduced, compared to more complex poker domains, the two-player limit hold'em domain is still very large. The game tree consists of approximately 10^{18} nodes and, given the standards of current hardware, it is intractable to derive a true Nash equilibrium for the game. In fact, it proves impossible to reasonably store this strategy by today's hardware standards [55]. For these reasons alternative approaches, such as case-based strategies, can prove useful given their ability for generalisation.

We have conducted an extensive amount of experimentation on the use of case-based strategies, using two-player, limit hold'em as our test-bed. In particular we have investigated and measured the effect that changes have on areas such as *feature and solution representation*, *similarity metrics*, *system training* and the use of different *decision making policies*. Modifications have ranged from the very minor, e.g. training on different sets of data to the more dramatic, e.g. the development of custom betting sequence similarity metrics. For each modification and addition to the architecture we have extensively evaluated the strategies produced via self-play experiments, as well as by challenging a range of third-party, artificial agents and human opposition. In this chapter, we restrict our attention to the changes that had the greatest affect on the system architecture and its performance. We have named our system Sartre (Similarity Assessment Reasoning for Texas hold'em via Recall of Experience) and we trace the evolution of its architecture below.

3.3.1. Overview

In order to generalise betting decisions from a set of (artificial or real-world) training data, first it is required to construct and store a collection of cases. A case's feature and solution representation must be decided upon, such as the identification of salient attribute-value pairs that describe the environment at the time a case was recorded. Each case should attempt to capture important information about the current environment that is likely to have an impact on the final solution. After a collection of cases has been established, decisions can be made by searching the case-base and locating similar scenarios for which solutions have been recorded in the past. This requires the use of local similarity metrics for each feature.

Given a target case, t, that describes the immediate game environment, a source case, $s \in S$, where S is the entire collection of previously recorded cases and a set of features, F, global similarity is computed by summing each feature's local similarity contribution, sim_f, and dividing by the total number of features:

$$G(t, s) = \sum_{f \in F} \frac{sim_f(t_f, s_f)}{|F|} \tag{3.1}$$

Fig. 3.1 provides a pictorial representation of the architecture we have used to produce case-based strategies. The six areas that have been labelled in Fig. 3.1 identify six key areas within the architecture where *maintenance* has had the most impact and led to positive affects on system performance. They are:

1. Feature Representation
2. Similarity Metrics
3. Solution Representation
4. Case Retrieval
5. Solution Re-Use Policies, and
6. System Training

3.3.2. Architecture Evolution

Here we describe some of the changes that have taken place within the six key areas of our case-based architecture, identified above. Where possible, we provide a comparative evaluation for the maintenance performed, in order to measure the impact that changes had on the performance of the case-based strategies produced.

Feature Representation

The first area of the system architecture that we discuss is the feature representation used within a case (see Fig. 3.1, Point 1). We highlight results that have influenced changes to the representation over time. In order to construct a case-based strategy a case representation is required

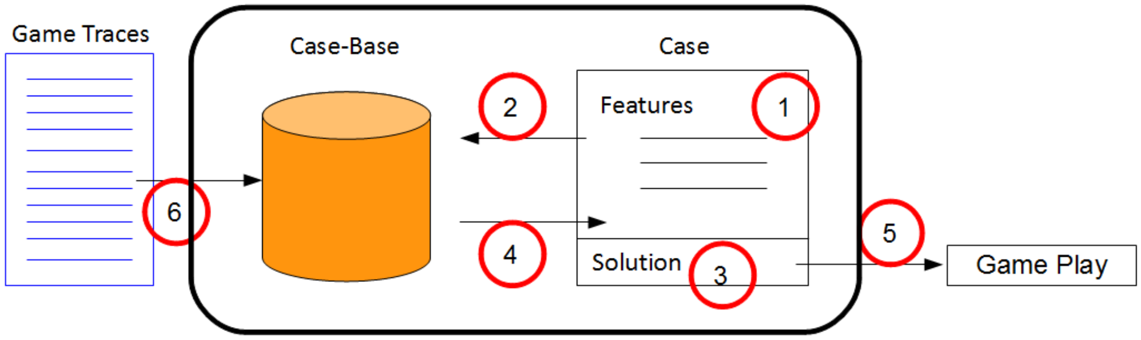

Figure 3.1: Overview of the architecture used to produce case-based strategies. The numbers identify the six key areas within the architecture where the affects of maintenance have been evaluated.

that establishes the type of information that will be recorded for each game scenario. Our case-based strategies use a simple attribute-value representation to describe a set of case features. Table 3.1 lists the features used within our case representation. A separate representation is used for preflop and postflop cases, given the differences between these two stages of the game. The features listed in Table 3.1 were chosen by the author as they concisely capture all the necessary public game information, as well as the player's personal, hidden information.

Each feature is explained in more detail below:

Preflop

1. Hole Cards: the personal hidden cards of the player, represented by 1 out of 169 equivalence classes.

2. Betting Sequence: a sequence of characters that represent the betting actions witnessed until the current decision point, where actions can be selected from the set, $A_{limit} = \{f, c, r\}$.

Postflop

1. Hand Strength: a description of the player's hand strength given a combination of their personal cards and the public community cards.

2. Betting Sequence: identical to the preflop sequence, however with the addition of round delimiters to distinguish betting from previous rounds, $A_{limit} \cup \{-\}$.

3. Board Texture: a description of the public community cards that are revealed during the postflop rounds

Table 3.1: Preflop and postflop case feature representation.

	Preflop	**Postflop**
1.	*Hole Cards*	*Hand Strength*
2.	*Betting Sequence*	*Betting Sequence*
3.		*Board Texture*

While the case features themselves have remained relatively unchanged throughout the architecture's evolution, the actual values that each feature records has been experimented with to determine the affect on final performance. For example, we have compared and evaluated the use of different metrics for the **hand strength** feature from Table 3.1. Fig. 3.2 depicts the result of a comparison between three **hand strength** feature values. In this experiment, the feature values for *betting sequence* and *board texture* were held constant, while the *hand strength* value was varied. The values used to represent **hand strength** were as follows:

CATEGORIES: Uses expert defined categories to classify hand strength. Hands are assigned into categories by mapping a player's personal cards and the available board cards into one of a number of predefined categories. Each category represents the type of hand the player currently has, together with information about the *drawing* potential of the hand, i.e. whether the hand has the ability to improve with future community cards. In total 284 categories were defined[2].

E[HS]: *Expected hand strength* is a one-dimensional, numerical metric. The E[HS] metric computes the probability of winning at showdown against a random hand. This is given by enumerating all possible combinations of community cards and determining the proportion of the time the player's hand wins against the set of all possible opponent holdings. Given the large variety of values that can be produced by the E[HS] metric, *bucketing* takes place where similar values are mapped into a discrete set of buckets that contain hands of similar strength. Here we use a total of 20 buckets for each postflop round.

E[HS^2]: The final metric evaluated involves squaring the *expected hand strength*. Johanson [55] points out that squaring the expected hand strength (E[HS^2]) typically gives better results, as this assigns higher hand strength values to hands with greater potential. Typically in poker, hands with similar strength values, but differences in potential, are required to be played in strategically different ways [111]. Once again bucketing is used where the derived E[HS^2] values are mapped into 1 of 20 unique buckets for each postflop round.

[2]A listing of all 284 categories can be found at the following website: http://www.cs.auckland.ac.nz/research/gameai/sartreinfo.html

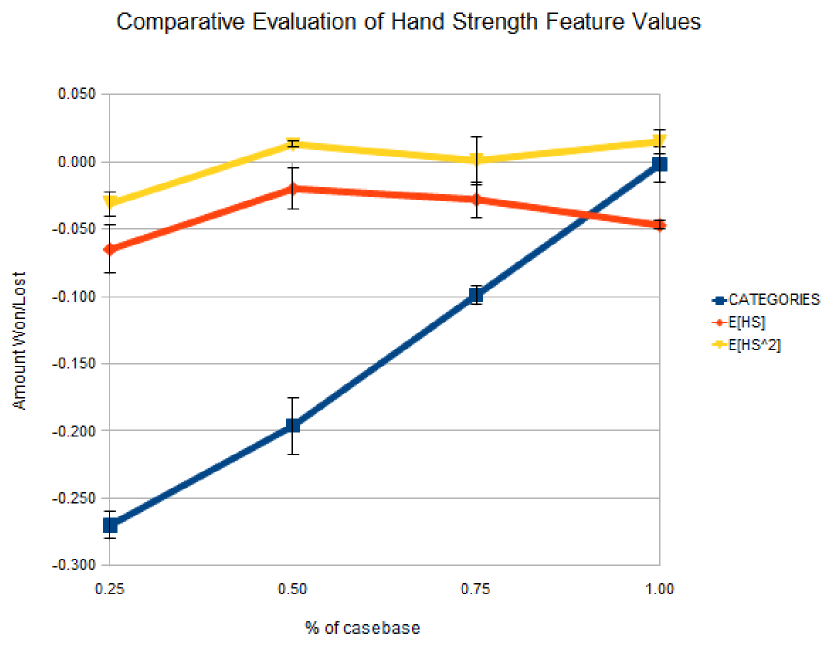

Figure 3.2: The performance of three separate case-based strategies produced by altering the value used to represent hand strength. Results are measured in sb/h and were obtained by challenging Fell Omen 2.

The resulting case-based strategies were evaluated by challenging the computerised opponent Fell Omen 2 [32]. Fell Omen 2 is a solid two-player limit hold'em agent that plays an ϵ-Nash equilibrium type strategy. Fell Omen 2 was made publicly available by its creator Ian Fellows and has become a standard benchmark agent for strategy evaluation. The results depicted in Fig. 3.2 are measured in small bets per hand (sb/h), i.e. where the total number of small bets won or lost are divided by the total number of hands played. Each data point records the outcome of three matches, where 3000 duplicate hands were played.

Results were recorded for various levels of case-base usage to get an idea of how well the system is able to generalise decisions. The results in Fig. 3.2 show that (when using a full case-base) the use of E[HS^2] for the hand strength feature produces the strongest strategies, followed by the use of CATEGORIES and finally E[HS]. The poor performance of E[HS] is likely due to the fact that this metric does not fully capture the importance of a hands future *potential*. When only a partial proportion of the case-base is used it becomes more important for the system to be able to recognise similar attribute values in order to make appropriate decisions. Both E[HS] and E[HS^2] are able to generalise well. However, the results show that decision generalisation begins to break down when using CATEGORIES. This has to do with the similarity metrics used.

Next we discuss the area of *similarity assessment* within the system architecture, which is intimately tied to the feature representation chosen.

Similarity Assessment

For each feature that is used to represent a case, a corresponding local similarity metric, $sim_f(f_1, f_2)$, is required that determines how similar two feature values, f_1 and f_2, are to each other. The use of different representations for the hand strength feature in Fig. 3.2 also requires the use of separate similarity metrics. The CATEGORIES strategy in Fig. 3.2. employs a trivial *all-or-nothing* similarity metric for each of its features. If the value of one feature has the same value of another feature, a similarity score of 1 is assigned. On the other hand, if the two feature values differ at all, a similarity value of 0 is assigned. This was done to get an initial idea of how the system performed using the most basic of similarity retrieval measures. The performance of this baseline system could then be used to determine how improvements to local similarity metrics affected overall performance.

The degradation of performance observed in Fig. 3.2 for the CATEGORIES strategy (as the proportion of case-base usage decreases) is due to the use of *all-or-nothing* similarity assessment. The use of the overly simplified *all-or-nothing* similarity metric meant that the system's ability to retrieve similar cases could often fail, leaving the system without a solution for the current game state. When this occurred a *default-policy* was relied upon to provide the system with an action. The default-policy used by the system was an *always-call* policy, whereby the system would first attempt to check if possible, otherwise it would call an opponent's bet. This default-policy was selected by the author as it was believed to be preferable to other trivial default policies, such as *always-fold*, which would always result in a loss for the system.

The other two strategies in Fig. 3.2 (E[HS] and E[HS^2]) do not use trivial *all-or-nothing* similarity. Instead the *hand strength* features use a similarity metric based on Euclidean distance. Both the E[HS] and E[HS^2] strategies also employ informed similarity metrics for their *betting sequence* and *board texture* features, as well. Recall that the *betting sequence* feature is represented as a sequence of characters that lists the playing decisions that have been witnessed so far for the current hand. This requires the use of a non-trivial metric to determine similarity between two non-identical sequences. Here we developed a custom similarity metric that involves the identification of *stepped* levels of similarity, based on the number of bets/raises made by each player. The exact details of this metric are presented in Section 3.3.3.. Finally, for completeness, we determine similarity between different *board texture* classes via the use of hand picked similarity values.

Fig. 3.2 shows that, compared to the CATEGORIES strategy, the E[HS] and E[HS^2] strategies do a much better job of decision generalisation as the usable portion of the case-base is reduced. The eventual strategies produced do not suffer the dramatic performance degradation that occurs with the use of *all-or-nothing* similarity.

Solution Representation

As well as recording feature values, each case also needs to specify a solution. The most obvious solution representation is a single betting action, $a \in A_{limit}$. As well as a betting action, the solution can also record the actual outcome, i.e. the numerical result, $o \in \Re$, of having taken action $a \in A_{limit}$, for a particular hand. Using this representation allows a set of training data to be parsed where each action/outcome observed maps directly to a case, which is then stored in the case-base. Case retention during game play can also be handled in the same way, where the case-base is updated with new cases at runtime. To make a playing decision at runtime the case-base is searched for similar cases and their solutions are combined to give a probabilistic vector (f, c, r) that specifies the proportion of the time our strategy should take each particular action.

A major drawback of the single action/outcome solution representation is that the case-base becomes filled with redundant cases, i.e. cases that record the same feature values, which also may or may not record the same betting action. An alternative solution representation, would involve directly storing the action and outcome vectors, i.e. effectively merging the redundant cases into one case by shifting combined information into the case solution. In order to achieve this solution representation, it now becomes necessary to pre-process the training data. As there is no longer a one-to-one mapping between hands and cases, the case-base must first be searched for an existing case during case-base construction. If one exists, its solution is updated, if one doesn't exist, it is added to the case-base. Case retention at runtime can also still take place, however, this requires that case solution vectors not be normalised after case-base construction as this would result in a loss of probabilistic information. Instead, it is sufficient to store the raw action/outcome frequency count information as this can later be normalised at runtime to make a probabilistic decision.

A solution representation based on action and outcome vectors results in a much more compact case-base. Table 3.2 depicts how much we were able to decrease the number of cases required to be stored simply by modifying the solution representation. Table 3.2 indicates that a single valued solution representation requires over 10 times the number of cases to be stored compared to a vector valued representation. Moreover, no information is lost when switching from a single valued representation to a vector valued representation. This modification has a follow on effect that improves the quality of the case-based strategies produced. As the number of cases required to be stored is so large – given the single action/outcome representation, training of the system had to be cut short due to the costs involved in actually storing the cases.

Table 3.2: Total cases stored for each playing round using single value solution representation compared to vector valued solutions.

Round	Total Cases - Single	Total Cases - Vector
Preflop	201,335	857
Flop	300,577	6,743
Turn	281,529	35,464
River	216,597	52,088
Total	1,000,038	95,152

The compact case-base produced, by representing solutions directly as vectors, bypasses this problem and allows the system to be trained on a larger set of data. By not prematurely restricting the training phase, the case-based strategies produced are able to increase the amount of scenarios they are able to encounter and encode cases for during training. This leads to a more comprehensive and competent case-base.

Case Retrieval

Our architecture for producing case-based strategies has consistently used the k-nearest neighbour algorithm (k-NN) [26] for case retrieval. Given a target case, t, and a collection of cases, S, the k-NN algorithm retrieves the k most similar cases by positioning t in the n-dimensional search space of S. Each dimension in the space records a value for one of the case features. Equation 3.1 (from Section 3.3.1.) is used to determine the global similarity between two cases, t and $s \in S$.

While the use of the k-NN algorithm for case retrieval has not changed within our architecture, maintenance to other components of the architecture has resulted in modifications regarding the exact details used by the k-NN algorithm. One such modification was required given the transition from a single action solution representation to a vector valued solution representation (as described in Section 3.3.2.). Initially, a variable value of k was allowed, whereby the total number of similar cases retrieved varied with each search of the case-base. Recall, that a case representation that encodes solutions as single actions results in a redundant case-base that contains multiple cases with the exact same feature values. The solution of those cases may or may not advocate different playing decisions. Given this representation, a final probability vector was required to be created on-the-fly at runtime by retrieving all identical cases and merging their solutions. Hence, the number of retrieved cases, k, could vary between 0 and N. When $k > 0$, the normalised entries of the probability vector were used to make a final playing decision. However, if $k = 0$, the *always-call* default-policy was used.

Once the solution representation was updated to record action vectors (instead of single decisions) a variable k value was no longer required. Instead, the algorithm was updated to simply always retrieve the nearest neighbour, i.e. $k = 1$. Given further improvements to the similarity metrics used, the use of a *default-policy* was no longer required as it was no longer possible to encounter scenarios where no similar cases could be retrieved. Instead, the most similar neighbour was always returned, no matter what the similarity value. This has resulted in a much more robust system that is actually capable of generalising decisions recorded in the training data, as opposed to the initial prototype system which offered no ability for graceful degradation, given dissimilar case retrieval.

Solution Re-use Policies

The fifth area of the architecture that we discuss (Fig. 3.1, Point 5) concerns the choice of a final playing decision via the use of separate policies, given a retrieved case and its solution. Consider the probabilistic action vector, $A = (a_1, a_2, \ldots, a_n)$, and a corresponding outcome vector, $O = (o_1, o_2, \ldots, o_n)$. There are various ways to use the information contained in the vectors to make a final playing decision. We have identified and empirically evaluated several different policies for re-using decision information, which we label **solution re-use policies**. Below we outline three solution re-use policies, which have been used for making final decisions by our case-based strategies.

1. **Probabilistic** The first solution re-use policy simply selects a betting action probabilistically, given the proportions specified within the action vector, $P(a_i) = a_i$, for $i = 1 \ldots n$. These values are directly retrieved from a case's solution stored within the case-base. Betting decisions that have greater proportions within the vector will be made more often then those with lower proportions. In a game-theoretic sense, this policy corresponds to a mixed strategy.

2. **Max-frequency** Given an action vector $A = (a_1, a_2, \ldots, a_n)$, the *max-frequency* solution re-use policy selects the action that corresponds to $\arg\max_i a_i$, i.e. it selects the action that was made most often and ignores all other actions. In a game-theoretic sense, this policy corresponds to a pure strategy.

3. **Best-Outcome** Instead of using the values contained within the action vector, the *best-outcome* solution re-use policy selects an action, given the values contained within the outcome vector, $O = (o_1, o_2, \ldots, o_n)$. The final playing decision is given by the action, a_i, that cor-

Table 3.3: Results of experiments between solution re-use policies.

	Max-frequency	Probabilistic	Best-outcome	**Average**
Max-frequency		0.011 ± 0.005	0.076 ± 0.008	$\mathbf{0.044 \pm 0.006}$
Probabilistic	-0.011 ± 0.005		0.036 ± 0.009	$\mathbf{0.012 \pm 0.004}$
Best-outcome	-0.076 ± 0.008	-0.036 ± 0.009		$\mathbf{-0.056 \pm 0.005}$

responds to $\arg\max_i o_i$, i.e. the action that corresponds to the maximum entry in the outcome vector.

Given the three solution re-use policies described above, it is desirable to know which policies produce the strongest strategies. Table 3.3 presents the results of self-play experiments where the three solution re-use policies were challenged against each other. A round robin tournament structure was used, where each policy challenged every other policy. The figures presented are from the row player's perspective and are in small bets per hand. Each match consists of 3 separate duplicate matches of 3000 hands. Hence, in total 18,000 hands of poker were played between each competitor. All results are statistically significant, with 95% confidence.

Table 3.3 shows that the *max-frequency* policy outperforms its *probabilistic* and *best-outcome* counterparts. Of the three, best-outcome fairs the worst, losing all of its matches. The results indicate that simply re-using the most commonly made decision results in better performance than mixing from a probability vector and that choosing the decision that resulted in the best outcome was the worst solution re-use policy. Moreover, these results are representative of further experiments involving other third-party computerised agents.

One of the reasons for the poor performance of best-outcome is likely due to the fact that good outcomes don't necessarily represent good betting decisions and vice-versa. The reason for the success of the *max-frequency* policy is less obvious. In the author's opinion, this has to do with the type of opponent being challenged, i.e. agents that play a static, non-exploitive strategy, such as those listed in Table 3.3, as well as strategies that attempt to approximate a Nash equilibrium. As an equilibrium-based strategy does not attempt to exploit any bias in its opponent's strategy, it will only gain when the opponent ends up making a mistake by selecting an inappropriate action. The action that was made most often is unlikely to be an inappropriate action, therefore sticking to this decision avoids any exploration errors made by choosing other (possibly inappropriate) actions. Moreover, biasing playing decisions towards this action is likely to go unpunished when challenging a non-exploitive agent. On the other hand, against an exploitive opponent the bias imposed by choosing only one action is likely to be detrimental to performance in the long run and therefore it would become more important to mix up decisions.

System Training

How the system is trained is the final key area of the architecture that we discuss, in regard to system maintenance. One of the major benefits of producing case-based strategies via expert imitation, is that different types of strategies can be produced by simply modifying the data that is used to train the system. Decisions that were made by a single expert player can be extracted from hand history logs and used to train a case-based strategy. Experts can be either human or other artificial agents.

In order to train a case-based strategy, perfect information is required, i.e. the data needs to record the hidden card information of the expert player. Typically, data collected from online poker sites only contains this information when the original expert played a hand that resulted in a *showdown*. For hands that were folded before a showdown, this information is lost. It is difficult to train a strategy on data where this information is missing. More importantly, any attempt to train a system on only the data where showdowns occurred would result in biased actions, as the decision to fold would never be encountered.

It is for these reasons that our case-based strategies have been trained on data made publicly available from the Annual Computer Poker Competition [2]. This data records hand history logs for all matches played between computerised agents at a particular year's competition. The data contains perfect information for every hand played and therefore can easily be used to train an imitation-based system. Furthermore, the computerised agents that participate at the ACPC each year are expected to improve in playing strength over the years and hence re-training the system on updated data should have a follow on affect on performance for any imitation strategies produced from the data. Our case-based strategies have typically selected subsets of data to train on, based on the decisions made by the agents that have performed the best in either of the two winner determination methods used by the ACPC.

There are both advantages and disadvantages for producing strategies that rely on generalising decisions from training data. While this provides a convenient mechanism for easily upgrading a system's play, there is an inherent reliance on the quality of the underlying data in order to produce reasonable strategies. Furthermore, it is reasonable to assume that strategies produced in this way are typically only expected to do as well as the original expert(s) they are trained on. Later chapters in this thesis will address how this limitation may be overcome.

3.3.3. A Framework for Producing Case-Based Strategies in Two-Player, Limit Texas Hold'em

For the six key areas of our architecture (described above) maintenance was guided via comparative evaluation and overall impact on performance. The outcome of this intensive, systematic maintenance is the establishment of a final framework for producing case-based strategies in the domain of two-player, limit hold'em.

Here we present the details of the final framework we have established for producing case-based strategies. The following sections illustrate the details of our framework by specifying the following sufficient components:

1. A representation for encoding cases and game state information
2. The corresponding similarity metrics required for decision generalisation.

Case Representation

Table 3.4 depicts the final post-flop case representation used to capture game state information. A single case is represented by a collection of attribute-value pairs. Separate case-bases are constructed for the separate rounds of play by processing a collection of hand histories and recording values for each of the three attributes listed in Table 3.4. Each of the post-flop attribute-value pairs are now described in more detail:

1. Hand Strength: The quality of a player's hand is represented in our framework by calculating the E[HS^2] of the player's cards and then mapping these values into 1 out of 50 possible buckets. The use of 50 buckets allowed a suitable level of granularity for comparing hand strengths, while also limiting case-base growth.

2. Betting Sequence: The betting sequence is represented as a string. It records all observed actions that have taken place in the current round, as well as previous rounds. Characters in the string are selected from the set of allowable actions, $A_{limit} = \{f, c, r\}$, rounds are delimited by a hyphen.

3. Board Texture: The board texture refers to important information available, given the combination of the publicly available community cards. In total, nine board texture categories were selected by the author. These categories are displayed in Table 3.5 and are believed to represent salient information that any human player would notice. Specifically, the categories focus on whether it is possible that an opponent has made a flush (five cards of the same suit) or a straight (five cards of sequential rank), or a combination of both. The categories are broken up into **possible** and **highly-possible** distinctions. A category labelled

Table 3.4: A case is made up of three attribute-value pairs, which describe the current state of the game. A solution consists of an action and outcome triple, which records the average numerical value of applying the action ($-\infty$ refers to an unknown outcome).

Attribute	**Type**	**Example**
1. Hand Strength	Integer	1 – 50
2. Betting Sequence	String	*rc-c, crrc-crrc-cc-, r, ...*
3. Board Texture	Class	*No-Salient, Flush-Possible, Straight-Possible, Flush-Highly-Possible, ...*
Action	Triple	(0.0, 0.5, 0.5), (1.0, 0.0, 0.0), ...
Outcome	Triple	($-\infty$, 4.3, 15.6), (-2.0, $-\infty$, $-\infty$), ...

possible refers to the situation where the opponent requires two of their personal cards in order to make their flush or straight. On the other hand, a **highly-possible** category only requires the opponent to use one of their personal cards to make their hand, making it more likely they have a straight or flush.

Similarity Metrics

Each feature requires a corresponding local similarity metric in order to generalise decisions contained in a set of data. Here we present the metrics specified by our framework.

1. Hand Strength: Equation 3.2 specifies the metric used to determine similarity between two hand strength buckets (f_1, f_2).

$$sim(f_1, f_2) = max\{1 - k \cdot \frac{|f_1 - f_2|}{T}, 0\} \tag{3.2}$$

Here, T refers to the total number of buckets that have been defined, where $f_1, f_2 \in [1, T]$ and k is a scalar parameter used to adjust the rate at which similarity should decrease.

2. Betting Sequence: To determine similarity between two betting sequences we developed a custom similarity metric that involves the identification of *stepped* levels of similarity, based on the number of bets/raises made by each player. The first level of similarity (level0) refers to the situation when one betting sequence exactly matches that of another. If the sequences do not exactly match the next level of similarity (level1) is evaluated. If two distinct betting sequences exactly match for the active betting round and for all previous betting rounds the total number of bets/raises made by each player are equal then level1 similarity is satisfied and a value of 0.9 is assigned. Consider the following example where the active betting round is the *turn* and the two betting sequences are:

1. *crrc-crrrrc-cr*
2. *rrc-rrrrc-cr*

Here, level0 is clearly incorrect as the sequences do not match exactly. However, for the active betting round (*cr*) the sequences do match. Furthermore, during the preflop (1. *crrc* and 2. *rrc*) both players made 1 raise each, albeit in a different order. During the flop (1. *crrrrc* and 2. *rrrrc*) both players now make 4 raises each. Given that the number of bets/raises in the previous rounds (preflop and flop) match, these two betting sequences would be assigned a similarity value of 0.9.

If level1 similarity was not satisfied the next level (level2) would be evaluated. Level2 similarity is less strict than level1 similarity as the previous betting rounds are no longer differentiated. Consider the river betting sequences:

1. *rrc-cc-cc-rrr*
2. *cc-rc-crc-rrr*

Once again the sequences for the active round (*rrr*) matches exactly. This time, the number of bets/raises in the preflop round are not equal (the same applies for the flop and the turn). Therefore, level1 similarity is not satisfied. However, the number of raises encountered for all the previous betting rounds combined (1. *rrc-cc-cc* and 2. *cc-rc-crc*) are the same for each player, namely 1 raise by each player. Hence, level2 similarity is satisfied and a similarity value of 0.8 would be assigned. Finally, if level0, level1 and level2 are not satisfied level3 is reached where a similarity value of 0 is assigned.

3. Board Texture: To determine similarity between board texture categories a similarity matrix was derived. Matrix rows and columns in Fig. 3.3 represent the different categories defined in Table 3.5. Diagonal entries refer to two sets of community cards that map to the same category, in which case similarity is always 1. Non-diagonal entries refer to similarity values between two dissimilar categories. These values were hand picked by the author. The matrix given in Fig. 3.3 is symmetric.

$$\begin{array}{c|ccccccccc}
 & A & B & C & D & E & F & G & H & I \\
\hline
A & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
B & 0 & 1 & 0.8 & 0.7 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & 0.8 & 1 & 0.7 & 0 & 0 & 0 & 0 & 0 \\
D & 0 & 0.7 & 0.7 & 1 & 0 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 0 & 1 & 0.8 & 0.7 & 0 & 0.6 \\
F & 0 & 0 & 0 & 0 & 0.8 & 1 & 0.7 & 0 & 0.5 \\
G & 0 & 0 & 0 & 0 & 0.7 & 0.7 & 1 & 0.8 & 0.8 \\
H & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 1 & 0.8 \\
I & 0 & 0 & 0 & 0 & 0.6 & 0.5 & 0.8 & 0.8 & 1
\end{array}$$

Figure 3.3: Board texture similarity matrix.

Table 3.5: Board Texture Key

A	No salient
B	Flush possible
C	Straight possible
D	Flush possible, straight possible
E	Straight highly possible
F	Flush possible, straight highly possible
G	Flush highly possible
H	Flush highly possible, straight possible
I	Flush highly possible, straight highly possible

3.3.4. Two-Player, Limit Texas Hold'em Experimental Results

We now present a series of experimental results collected in the domain of two-player, limit Texas Hold'em. The results presented are obtained from annual computer poker competitions and data collected by challenging human opposition. For each evaluated case-based strategy, we provide an *architecture snapshot* that captures the relevant details of the parameters used for each of the six key architecture areas, that were previously discussed.

2009 IJCAI Computer Poker Competition

We begin with the results of the 2009 ACPC, held at the International Joint Conference on Artificial Intelligence. Here, we submitted our case-based agent, Sartre, for the first time, to challenge 12 other computerised agents submitted from all over the world. The following *architecture snapshot* depicts the details of the submitted agent:

1. Feature Representation
   a) Hand Strength – *categories*
   b) Betting Sequence – *string*
   c) Board Texture – *categories*
2. Similarity Assessment – *all-or-nothing*
3. Solution Representation – *single*
4. Case Retrieval – *variable k*
5. Re-Use Policy – *max-frequency*
6. System Training – *Hyperborean-08*

The *architecture snapshot* above represents a baseline strategy where no maintenance had yet to be performed. Each of the entries listed above corresponds to one of the six key architecture areas introduced in Section 3.3.2.. Notice that trivial *all-or-nothing* similarity was employed along with a *single action solution representation*, which resulted in a redundant casebase. The value for *system training* refers to the original expert whose decisions were used to train the system.

The final results are displayed in Table 3.6[3]. The competition consisted of two winner determination methods: bankroll *instant run-off* and *total bankroll*. Each agent played between 75 and 120 duplicate matches against every other agent in order to obtain the average values displayed. Each match consisted of 3000 duplicate hands. The values presented are the number of small bets per hand won or lost. Our case-based agent, Sartre, achieved a 7^{th} place finish in the instant run-off division and a 6^{th} place finish in the the total bankroll division.

[3]Full cross-tables of results can be found in Appendix A.

Table 3.6: 2009 limit heads up bankroll and runoff results, respectively

Limit Heads up Bankroll Runoff	Limit Heads up Total Bankroll	Average Won/Lost
1. GGValuta	1. MANZANA	0.186 ± 0.002
2. Hyperborean-Eqm	2. Hyperborean-BR	0.116 ± 0.002
3. MANZANA	3. GGValuta	0.110 ± 0.002
4. Rockhopper	4. Hyperborean-Eqm	0.116 ± 0.002
5. Hyperborean-BR	5. Rockhopper	0.103 ± 0.002
6. Slumbot	**6. Sartre**	0.097 ± 0.002
7. Sartre	7. Slumbot	0.096 ± 0.002
8. GS5	8. GS5	0.082 ± 0.002
9. AoBot	9. AoBot	-0.002 ± 0.003
10. GS5Dynamic	10. dcurbHU	-0.07 ± 0.002
11. LIDIA	11. LIDIA	-0.094 ± 0.002
12. dcurbHU	12. GS5Dynamic	-0.201 ± 0.002
13. Tommybot		

2010 AAAI Computer Poker Competition

Following the maintenance experiments presented in Section 3.3.2., an updated case-based strategy was submitted to the 2010 ACPC, held at the Twenty-Forth AAAI Conference on Artificial Intelligence. Our entry, once again named Sartre, used the following *architecture snapshot*:

1. Feature Representation
   a) Hand Strength – *50 buckets E[HS^2]*
   b) Betting Sequence – *string*
   c) Board Texture – *categories*
2. Similarity Assessment
   a)Hand Strength – *Euclidean*
   b) Betting Sequence – *custom*
   c) Board Texture – *matrix*
3. Solution Representation – *vector*
4. Case Retrieval – *k = 1*
5. Re-Use Policy – *probabilistic*
6. System Training – *MANZANA*

Table 3.7: 2010 limit heads up bankroll and runoff results. Values are in sb/h with 95% confidence intervals.

Limit Heads up Bankroll Runoff	Limit Heads up Total Bankroll	Average Won/Lost
1. Rockhopper	1. PULPO	0.225 ± 0.003
2. GGValuta	2. Hyperborean-TBR	0.207 ± 0.002
3. Hyperborean-IRO	**3. Sartre**	0.203 ± 0.002
4. Slumbot	4. Rockhopper	0.200 ± 0.002
5. PULPO	5. Slumbot	0.199 ± 0.002
6. Sartre	6. GGValuta	0.193 ± 0.003
7. GS6-IRO	7. Jester	0.164 ± 0.003
8. Arnold2	8. Arnold2	0.160 ± 0.003
9. Jester	9. GS6-TBR	0.139 ± 0.004
10. LittleRock	10. LittleRock	0.118 ± 0.003
11. PLICAS	11. PLICAS	-0.046 ± 0.005
12. ASVP	12. ASVP	-0.320 ± 0.006
13. longhorn	13. longhorn	-1.441 ± 0.005

Here a vector valued *solution representation* was used together with improved similarity assessment. Given the updated solution representation, a single nearest neighbour, $k = 1$, was retrieved via the k-NN algorithm. A *probabilistic* solution re-use policy was employed and the system was trained on the decisions of the winner of the 2009 total bankroll division. The winner of the total bankroll division was selected as the original expert (as opposed to the bankroll runoff division) as this agent achieved the greatest overall profit. The final results are presented in Table 3.7. Once again two winner determination divisions are presented and the values are depicted in small bets per hand. Given the improvements, Sartre was able to achieve a 6^{th} place finish in the runoff division and a 3^{rd} place finish in the total bankroll division.

2011 AAAI Computer Poker Competition

The 2011 ACPC was held at the Twenty-Fifth AAAI Conference on Artificial Intelligence. Our entry to the competition is represented by the following *architecture snapshot*:

1. Feature Representation
    a) Hand Strength – *50 buckets E[HS^2]*
    b) Betting Sequence – *string*

   c) Board Texture – *categories*
2. Similarity Assessment
   a) Hand Strength – *Euclidean*
   b) Betting Sequence – *custom*
   c) Board Texture – *matrix*
3. Solution Representation – *vector*
4. Case Retrieval – *k = 1*
5. Re-Use Policy – *max-frequency*
6. System Training – *combination*

While reasonably similar to the strategy employed for the 2010 competition, the *architecture snapshot* above exhibits some important differences. In particular, the 2011 agent consisted of a combination of multiple strategies that were each trained by observing and generalising the decisions of separate original experts, see Chapter 4 for further details. Also notice the switch from a *probabilistic* solution re-use policy to a *max-frequency* policy.

Table 3.8 presents results from the 2011 competition. For the third year in a row, Sartre was able to improve its performance in both winner determination divisions. Sartre improved from 6^{th} to 4^{th} place in the *instant runoff* division. Whereas, in the *total bankroll* division Sartre improved from 3^{rd} place to 2^{nd} place. Notice however, while Calamari was declared the winner of this competition (with Sartre placing 2nd), there is overlap in the values used to make this decision, given the standard deviations.

2012 AAAI Computer Poker Competition

Between the years 2011 to 2012 no modifications were made to the version of Sartre that was submitted to the annual computer poker competition, except to replace one of the case-bases used to make decisions, with an updated case-base, which was trained on data from the 2011 competition. Table 3.9 presents results from the 2012 ACPC, which was held at the Twenty-Sixth AAAI Conference on Artificial Intelligence. For the first year since entering the competition, Sartre experienced a degradation in performance, achieving an 8^{th} place finish in both the instant runoff and total bankroll divisions. The results from the 2012 competition suggest that competitors are rapidly improving over the years. There is however, a tremendous amount of overlap in the bankrolls of agents that did not finish in one of the top three spots. In particular, when the 95% confidence intervals are considered, there is overlap between all agents that placed 4^{th} to 8^{th}.

Table 3.8: 2011 limit heads up bankroll and runoff results. Values are in sb/h with 95% confidence intervals.

Limit Heads up Bankroll Runoff	Limit Heads up Total Bankroll	Average Won/Lost
1. Hyperborean-2011-2p-limit-iro	1. Calamari	0.286 ± 0.004
2. Slumbot	2. **Sartre**	0.281 ± 0.006
3. Calamari	3. Hyperborean-2011-2p-limit-tbr	0.225 ± 0.005
4. **Sartre**	4. Feste	0.220 ± 0.005
5. LittleRock	5. Slumbot	0.216 ± 0.006
6. ZBot	6. ZBot	0.209 ± 0.006
7. GGValuta	7. Patience	0.209 ± 0.006
8. Feste	8. 2Bot	0.192 ± 0.006
9. Patience	9. LittleRock	0.180 ± 0.005
10. 2Bot	10. GGValuta	0.145 ± 0.006
11. RobotBot	11. AAIMontybot	0.001 ± 0.012
12. AAIMontybot	12. RobotBot	-0.035 ± 0.010
13. Entropy	13. GBR	-0.051 ± 0.012
14. GBR	14. Entropy	-0.097 ± 0.013
15. player.zeta	15. player.zeta	-0.176 ± 0.017
16. Calvin	16. Calvin	-0.230 ± 0.012
17. Tiltnet.Adaptive	17. Tiltnet	-0.268 ± 0.010
18. POMPEIA	18. POMPEIA	-0.549 ± 0.006
19. TellBot	19. TellBot	-0.759 ± 0.016

Human Opponents – Limit

Finally, we provide results for our *case-based strategies* when challenging human opposition. These results were collected from an online web application[4] where any human opponent could log on via their browser and challenge the latest version of the Sartre system. While it is interesting to gauge how well Sartre performs against human opponents (not just computerised agents), care must also be taken in interpreting these results as no effort has been made to restrict the human opposition to only players of a certain quality.

Fig. 3.4. depicts the number of small bets won in total against every human opponent to challenge the system. In total just under 30,000 poker hands have been recorded and Sartre

[4]http://www.cs.auckland.ac.nz/poker/

Table 3.9: 2012 limit heads up bankroll and runoff results. Values are in sb/h with 95% confidence intervals.

Limit Heads up Bankroll Runoff	Limit Heads up Total Bankroll	Average Won/Lost
1. Slumbot	1. Slumbot	0.029 ± 0.003
2. Hyperborean.iro	2. Little Rock	0.014 ± 0.003
3. Zbot	2. Zbot	0.013 ± 0.003
4. Little Rock	4. Hyperborean.tbr	0.007 ± 0.002
5. Neo Poker Lab	5. Patience	0.007 ± 0.003
6. Patience	6. Neo Poker Lab	0.005 ± 0.003
7. Entropy	7. Entropy	0.004 ± 0.003
8. **Sartre**	8. **Sartre**	0.002 ± 0.003
9. huhuers	9. huhuers	-0.008 ± 0.002
10. Feste.iro	10. Feste.tbr	-0.034 ± 0.004
11. Little Ace	11. Little Ace	-0.038 ± 0.003

currently records a profit of 9221 small bets. This results in a final small bets per hand (sb/h) value of 0.30734. Fig. 3.5. depicts the sb/h values recorded over all hands.

Discussion

The results show a marked improvement between outcomes obtained over the years 2009 to 2011 at the annual computer poker competition, with some degradation in performance in 2012. In particular, Sartre achieved a 6^{th} place finish in the total bankroll division at the 2009 competition. The following year, using the updated strategy, Sartre now achieved a 3^{rd} place finish (out of a total of 13 agents) in the same event. In 2011, Sartre was declared *runner-up* at the 2011 *total bankroll* division and very nearly won this competition. The following year, when no maintenance was performed, Sartre's performance degraded against improved competition. These results suggest that the maintenance performed and the updated architecture did indeed have a significant impact on the quality of the case-based strategies produced.

For the data collected against human opposition, while we can not comment on the quality of the human opponents challenged, the curves in Figs. 3.4 and 3.5 show a general upward trend and a steady average profit, respectively.

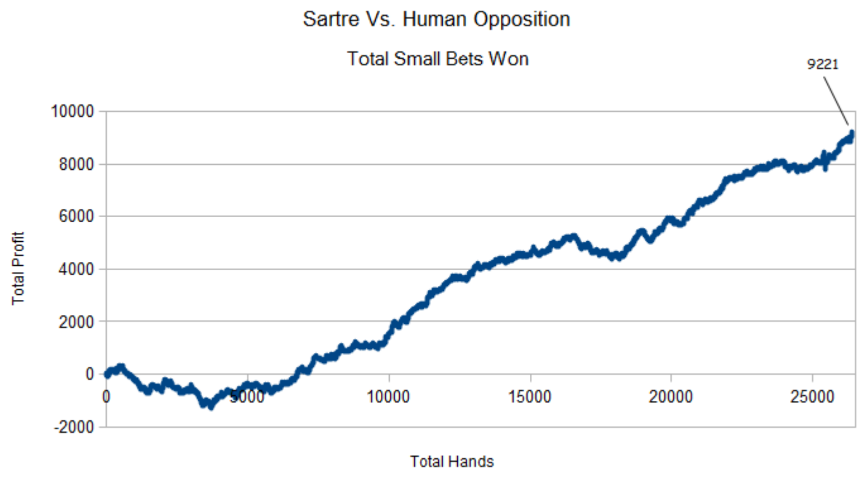

Figure 3.4: The number of small bets won in total against all human opponents to challenge Sartre.

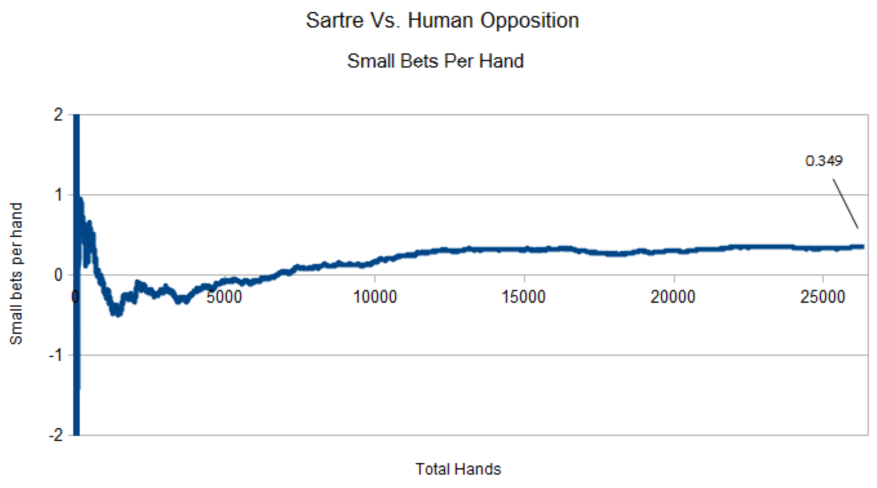

Figure 3.5: The small bets per hand (sb/h) won by Sartre over every hand played.

3.4. Two-Player, No-Limit Texas Hold'em

We now examine the application of case-based strategies in the more complicated domain of two-player, no-limit Texas Hold'em. Here we take into consideration the lessons learned during the maintenance that was performed in the two-player, limit hold'em domain. We use this information and the insights obtained in the limit domain to establish a finalised framework in the no-limit domain. However, before no-limit case-based strategies can be produced, the difficulties of handling a no-limit betting structure need to be addressed.

In the no-limit variation players' bet sizes are no longer restricted to capped amounts. Instead, a player can wager any amount they wish, up to the total amount of chips they possess. This simple rule change has a profound effect on the nature of the game, as well as on the development of computerised agents that wish to handle a no-limit betting structure. In particular, the transition to a no-limit domain results in unique challenges that are not encountered in a limit poker environment. First, there is the issue of establishing a set of abstract betting actions that all real actions will be mapped into during game play. This is referred to as **action abstraction** and it allows the vast, continuous domain of no-limit hold'em to be approximated by a much smaller *abstract* state space. Second, given an established set of abstract actions, a **translation** process is required that determines how *best* to map real actions into their appropriate abstract counterparts, as well as a reverse translation that maps abstract actions back into appropriate real-world betting decisions.

3.4.1. Abstraction

Recall that *abstraction* is a concept used by game theoretic poker agents that derive ϵ-Nash equilibrium strategies for the game of Texas Hold'em. As the actual hold'em game tree is much too large to represent and solve explicitly, it becomes necessary to impose certain abstractions that help restrict the size of the original game. For Texas Hold'em, there are two main types of abstraction:

1. **Chance abstraction** — which reduces the number of chance events that are required to be dealt with. This is typically achieved by grouping strategically similar hands into a restricted set of buckets.

2. **Action abstraction** — which restricts the number of actions a player is allowed to perform.

Action abstractions can typically be avoided by poker agents that specialise in limit poker, where there are only 3 actions to choose from: fold (f), check/call (c) or bet/raise (r). However in no-limit, where a raise can take on any value, some sort of action abstraction is required. This is achieved by restricting the available bet/raise options to a discrete set of categories based on

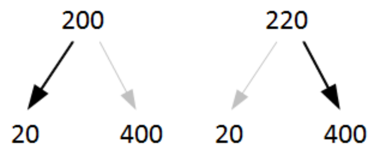

Figure 3.6: Using the *fcpa* abstraction, the amounts above will either be mapped as a *pot* sized bet – 20 or an *all-in* bet – 400.

fractions of the current pot size. For example, a typical abstraction such as: *fcpa*, restricts the allowed actions to:

- *f* – fold,
- *c* – call,
- *p* – bet the size of the pot
- *a* – all-in (i.e. the player bets all their remaining chips)

Given this abstraction, all actions are interpreted by assigning the actual actions into one of their abstract counterparts. While our case-based strategies do not attempt to derive an ϵ-Nash equilibrium solution for no-limit hold'em, they are still required to define an action abstraction in order to restrict the number of actions allowed in the game and hence reduce the state space.

3.4.2. Translation

Given that all bets need to be mapped into one of the abstract actions, a translation process is required to define the appropriate mapping. The choice of translation needs to be considered carefully as some mappings can be easily exploited. The following example illustrates how the choice of translation can lead to exploitability.

Consider a translation that maps bets into abstract actions based on absolute distance, i.e. the abstract action that is closest to the bet amount is the one that the bet gets mapped in to. Given a pot size of 20 chips and the *fcpa* abstraction (from above) any bet between 20 and a maximum of 400 chips will either be mapped into a *pot* (p) sized bet or an *all-in* (a) bet. Using this translation method, a bet amount of 200 chips will be considered a pot-sized bet, whereas a bet amount of only 20 chips more, 220, will be considered an all-in bet. See Fig. 3.6.

There can be various benefits for a player in making their opponent think that they have made a pot-sized bet or an all-in bet. First, consider the situation where an *exploitive* player bets 220 chips. This bet amount will be mapped into the *all-in* abstract action by an opponent that uses the *fcpa* abstraction. In other words, a player has made their opponent *believe* that they have bet all 400 of their chips, when in reality they have only risked 220. In this situation, it is likely that an opponent will fold most hands to an *all-in* bet, however even when the opponent calls, the *exploitive* player has still only wagered 220 chips as opposed to 400.

On the other hand, by betting just 20 chips less (i.e. 200 instead of 220), this alternative situation can have an even more dramatic effect on the outcome of a hand. When an *exploitive* player makes a bet of 200 chips this bet amount will be mapped as a pot-sized bet and if their opponent decides to call they will *believe* that they have only contributed 20 chips to the pot when in reality they would have invested 200 chips. If this is followed up with a large *all-in* bet, an opponent that believes they have only invested 20 chips in the pot is much more likely to fold a mediocre hand than a player that has contributed ten times that amount. As such, an *exploitive* player has the ability to make their opponent believe they are only losing a small proportion of their stack size by folding, when in reality they are losing a lot more. This can lead to large profits for the *exploitive* player.

The above example shows that a translation method that uses deterministic mapping based on absolute distance has the ability to be exploited simply by selecting particular bet amounts. Schnizlein *et al.* [103] formalise this type of translation as *hard translation*. **Hard translation** is a many to one mapping that maps an unabstracted betting value into an abstract action based on a chosen distance metric. Given a unique unabstracted betting value, hard translation will always map this value into the same abstract action. A disadvantage of hard translation is that an opponent can exploit this mapping simply by selecting particular betting values. To overcome this problem, a more robust translation procedure is required, one that cannot be exploited in such a way. Schnizlein *et al.* [103] also formalise an alternative *soft translation* that addresses some of the shortcomings of hard translation. **Soft translation** is a probabilistic state translation that uses normalised weights as similarity measures to map an unabstracted betting value into an abstract action. The use of a probabilistic mapping ensures that soft translation cannot be exploited like hard translation can.

Having considered the issues to do with *abstraction* and *translation*, we are now able to present our framework for producing case-based strategies in the domain of no-limit hold'em.

3.4.3. A Framework for Producing Case-Based Strategies in Two-Player, No-Limit Texas Hold'em

We now present the final framework we have established for producing case-based strategies in the domain of two-player, no-limit Texas Hold'em. In order to define the framework it is necessary to specify the following four conditions:

Table 3.10: The action abstraction used by our case-based strategies.

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

1. The action abstraction used

2. The type of state translation used and where this occurs within the architecture

3. A representation for encoding cases and game state information

4. The corresponding similarity metrics required for decision generalisation.

The conditions specified above are an extrapolation of those that were used to define the framework for producing case-based strategies in the limit hold'em domain.

Action Abstraction

Within our framework, we use the following action abstraction: *fcqhipdvta*. Table 3.10 provides an explanation of the symbols used. Here fold (f) and call (c) are the same actions that occur in the limit variation of the game. However, there are now many betting decisions, which are based on the proportion of the current pot size.

State Translation

Define $A = \{q, h, i, p, d, v, t, a\}$ to be the set of abstract betting actions. Actions f and c are omitted from A as these require no mapping. The exact translation parameters that are used differ depending on where translation takes place within the system architecture, as follows:

1. During case-base construction hand history logs from previously played hands are required to be encoded into cases. Here *hard translation* is specified by the following function $T_h : \Re \rightarrow A$:

$$T_h(b) = \begin{cases} a & \text{if } \frac{a}{b} > \frac{b}{c} \\ c & \text{otherwise} \end{cases} \tag{3.3}$$

   where $b \in \Re$ is the proportion of the total pot that has been bet in the actual game and $a, c \in A$ are abstract actions that map to actual pot proportions in the real game and $a <= b < c$. The fact that hard translation has the capability to be exploited is not a concern during case-base construction. Hard translation is used during this stage to ensure that re-training the system with the same hand history data will result in the same case-base.

2. During actual game play real betting actions observed during a hand are required to be mapped into appropriate abstract actions. This is equivalent to the translation process required of ϵ-Nash equilibrium agents that solve an abstract extensive form game, such as [6, 46, 103]. Observant opponents have the capability to exploit deterministic mappings during game play, hence a soft translation function is used for this stage, $T_s : \Re \rightarrow A$, given by the following probabilistic equations:

$$P(a) = \frac{\frac{a}{b} - \frac{a}{c}}{1 - \frac{a}{c}} \tag{3.4}$$

$$P(c) = \frac{\frac{b}{c} - \frac{a}{c}}{1 - \frac{a}{c}} \tag{3.5}$$

   where once again, $b \in \Re$ is the proportion of the total pot that has been bet in the actual game and $a, c \in A$ are abstract actions that map to actual pot proportions in the real game and $a <= b < c$. Note that when $b = a, P(a) = 1$ and $P(c) = 0$ and when $b = c, P(a) = 0$ and $P(c) = 1$. Hence, a betting action that maps directly to an abstract action in A does not need to be probabilistically selected. On the other hand, when $b \neq a$ and $b \neq c$, abstract actions are chosen probabilistically. Note that in Equations (3.4) and (3.5), $P(a) + P(c) \neq 1$ and hence a final abstract action is probabilistically chosen by first normalising these values.

3. A final reverse translation phase is required to map a chosen abstract action into a real value to be used during game play. A *reverse mapping* is required to map the abstract action into an appropriate real betting value, given the current game conditions. The following function is used to perform reverse translation, $T_r : A \rightarrow \Re$:

Table 3.11: The case representation used for producing case-based strategies in the domain of no-limit Texas Hold'em.

Feature	Type	Example
1. Hand Strength	Integer	1 – 50
2. Betting Sequence	String	*pdc-cqc-c, cc-, dc-qc-ci, ...*
3. Stack Commitment	Integer	1,2,3,4
4. Board Texture	Class	*No-Salient, Flush-Possible, Straight-Possible, Flush-Highly-Possible,* ...
Action	n-tuple	(0.0, 1.0, 0.0, 0.0, ...), ...
Outcome	n-tuple	(-∞, 36.0, -∞, -∞, ...), ...

$$T_r(a) = a' \pm \Delta a' \tag{3.6}$$

where $a \in A$ and $a' \in \Re$ is the real value corresponding to abstract action a and $\Delta a'$ is some random proportion of the bet amount that is used to ensure the strategy does not always map abstract actions to their exact real world counterparts. For example, when $a' = 100$ and $\Delta = 0.3$, any amount between 70 and 130 chips may be bet.

Case Representation

Table 3.11 depicts the case representation used to capture important game state information in the domain of two-player, no-limit Texas Hold'em. As in the limit domain, a case is represented by a collection of attribute-value pairs and separate case-bases are constructed for the separate betting rounds by processing a collection of hand histories and recording values for each of the attributes listed.

Three of the four attributes (*hand strength*, *betting sequence*, *board texture*) are the same as those used within the limit framework. The *stack commitment* attribute was introduced especially for the no-limit variation of the game. All attributes were selected by the author, given their importance in determining a final betting decision.

Each case also records a solution. Once again a solution is made up of an action vector and an outcome vector. The entries within each vector correspond to a particular betting decision. Given the extended set of betting actions that are available in the no-limit domain, the solution vectors are represented as n-tuples (as opposed to triples, which were used in the limit domain). Once again, the entries within the action vector must sum to one.

Each of the attribute-value pairs are described in more detail below.

1. **Hand Strength:** As in the limit domain, a player's hand is represented by calculating the E[HS^2] of the player's cards and mapping these values into 1 out of 50 possible buckets. A standard bucketing approach is used where the 50 possible buckets are evenly divided.

2. **Betting Sequence:** Once again a string is used to represent the *betting sequence*, which records all observed actions that have taken place in the current round, as well as previous rounds. Notice however, that the characters used to represent the betting sequence can be any of the abstract actions defined in Table 3.10. As such, there are a lot more possible no-limit betting sequences than there are limit sequences. Once again rounds are delimited by hyphens.

3. **Stack Commitment:** In the no-limit variation of Texas Hold'em players can wager any amount up to their total stack size. The proportion of chips committed by a player, compared to the player's stack size, is therefore of much greater importance, compared to limit hold'em. The betting sequence maps bet amounts into discrete categories based on their proportion of the pot size. This results in information that is lost about the total amount of chips a player has contributed to the pot, relative to the size of their starting stack. Once a player has contributed a large proportion of their stack to a pot, it becomes more important for that player to remain in the hand, rather than fold, i.e. they have become **pot committed**.

   The **stack commitment** feature maps this value into one of N categories, where N is a specified granularity:

$$[0-\frac{1}{N}],[\frac{1}{N}-\frac{2}{N}],\ldots,[\frac{N-2}{N}-\frac{N-1}{N}][\frac{N-1}{N}-1]$$

   Hence, for a granularity of $N = 4$, a stack commitment of 1 means the player has committed less than 25% of their initial stack, a stack commitment of 2 means that player has contributed somewhere between 25% and 50% of their total stack, and so forth.

4. **Board Texture:** The details of the board texture attribute are exactly as those described in the limit domain (see Section 3.3.3.).

Table 3.12: Bet Discretisation String

q	*h*	*i*	*p*	*d*	*v*	*t*

Similarity Metrics

Each feature requires a corresponding local similarity metric in order to generalise decisions contained in a set of data. Here we present the metrics specified by our framework.

1. Hand Strength: Similarity between hand strength values is determined by the same metric used in the limit hold'em domain, as specified by Equation 3.2.

2. Betting Sequence: Recall that the following bet discretisation is used: *fcqhipdvta*. Within this representation there are some non-identical bet sizes that are reasonably similar to each other. For example, a bet of half the pot (h) is quite close to a bet of three quarters of the pot (i). The betting sequence similarity metric we derived compares bet sizes against each other that occur at the same location within two betting sequences.

Let S_1 and S_2 be two betting sequences made up of actions $a \in A \cup \{f, c\}$, where the notation $S_{1,i}$, $S_{2,i}$ refers to the i^{th} character in the betting sequences S_1 and S_2, respectively.

For two betting sequences to be considered similar they first need to satisfy the following conditions:

1. $|S_1| = |S_2|$
2. Both $S_{1,i} = c$ and $S_{1,j} = a$ whenever $S_{2,i} = c$ and $S_{2,j} = a$

i.e. each sequence contains the same number of elements and any calls (c) or all-in bets (a) that occur within sequence S_1 must also occur at the same location in sequence S_2[5].

Any two betting sequences that do not satisfy the initial two conditions above are assigned a similarity value of 0. On the other hand, if the two betting sequences do satisfy the above conditions their bet sizes can then be compared against each other and a similarity value assigned.

Exactly how dissimilar two individual bets are to each other can be quantified by how far away from each other they occur within the bet discretisation string, displayed in Table 3.12.

[5]A betting sequence consists of one or more betting rounds, the above conditions must be satisfied for all betting rounds within the betting sequence.

As h and i are neighbours in the discretisation string they can be considered to occur at a distance of 1 away from each other, $\delta(h,i) = 1$, as opposed to say $\delta(q,t) = 6$, which are at opposite ends on the discretisation string.

For two betting sequences S_1, S_2 overall similarity is determined by (3.7):

$$sim(S_1, S_2) = \begin{cases} 1 - \sum_{i=0}^{|S_1|} \delta(S_{1,i}, S_{2,i})\alpha & \text{if } |S_1| = |S_2|, \\ & S_{1,i} = c \Rightarrow S_{2,i} = c, \\ & S_{1,j} = a \Rightarrow S_{2,j} = a \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

where α is some constant rate of decay parameter, which ensures dissimilar sequences result in lower similarity values.

The following is a concrete example of how similarity is computed for two non-identical betting sequences. Consider two betting sequences, $S_1 = ihpc$ and $S_2 = dqpc$. Here, $|S_1| = 4$ and $|S_2| = 4$ and wherever there exists a check/call (c) in S_1, there exists a corresponding c in S_2. As both conditions are satisfied we can evaluate the top half of Equation (3.7):

$$\begin{aligned} sim(S_1, S_2) &= 1 - [\delta(i,d)\alpha + \delta(h,q)\alpha + \delta(p,p)\alpha + \delta(c,c)\alpha] \\ &= 1 - [2\cdot\alpha + 1\cdot\alpha + 0\cdot\alpha + 0\cdot\alpha] \\ &= 1 - 3\alpha \end{aligned}$$

Using a rate of decay (selected by the author) of $\alpha = 0.05$, gives a final similarity of: $1 - 0.15 = 0.85$.

3. **Stack Commitment:** The stack commitment metric uses an exponentially decreasing function.

$$sim(f_1, f_2) = e^{(-|f_1 - f_2|)} \tag{3.8}$$

where, $f_1, f_2 \in [1, N]$ and N refers to the granularity used for the stack commitment attribute. This function was chosen as small differences between two stack commitment attributes (f_1, f_2) should result in large drops in similarity.

4. **Board Texture:** Similarity between board texture values is determined by the same similarity matrix used in the limit hold'em domain, as specified in Fig. 3.3.

3.4.4. No-Limit, Heads-Up Experimental Results

Once again we provide experimental results that have been obtained from annual computer poker competitions, where our case-based strategies have challenged other computerised agents. We also provide results against human opposition.

2010 AAAI Computer Poker Competition

We submitted an early version of our two-player, no-limit case-based strategy to the 2010 computer poker competition. The submitted strategy was trained on the hand history information of the previous year's winning agent (i.e Hyperborean) and used a probabilistic decision re-use policy.

In the no-limit competition, the *Doyle's Game* rule variation is used where both players begin each hand with 200 big blinds. All matches played were duplicate matches, where each competitor played 200 duplicate matches (each consisting of $N = 3000$ hands) against every other competitor. The final results are displayed in Table 3.13. Our entry is SartreNL.

Out of a total of five competitors, SartreNL placed 4^{th} in the *total bankroll* division and 2^{nd} in the *instant runoff* division. One important thing to note is that the strategy played by SartreNL was an early version and did not fully adhere to the framework described in Section 3.4.3.. In particular, the following abstraction was used: *fcqhipdvt*. While this abstraction is very similar to that described in Section 3.4.3., it is missing the *all-in* action, as this was only added after the agent had been submitted. A further difference is that the strategy submitted only used hard translation.

Table 3.13: 2010 no-limit heads up bankroll and runoff results, respectively

No-Limit Bankroll Runoff	No-Limit Total Bankroll	Average Won/Lost
1. Hyperborean.iro	1. Tartanian4.tbr	2.156 ± 0.048
2. **SartreNL**	2. PokerBotSLO	1.458 ± 0.184
3. Tartanian4.iro	3. Hyperborean.tbr	1.212 ± 0.026
4. PokerBotSLO	4. **SartreNL**	0.537 ± 0.034
5. c4tw.iro	5. c4tw.tbr	-5.362 ± 0.201

2011 AAAI Computer Poker Competition

The agent submitted to the 2011 competition fully adhered to the framework described in Section 3.4.3.. In the 2011 competition, SartreNL placed 2^{nd} in both winner determination divisions, out of a total of seven competitors. Table 3.14 presents the final results. Full cross-table results can be found in Appendix A.

Table 3.14: 2011 no-limit heads up bankroll and runoff results, respectively

No-Limit Bankroll Runoff	No-Limit Total Bankroll	Average Won/Lost
1. Hyperborean-iro	1. Lucky7	1.567 ± 0.053
2. **SartreNL**	2. **SartreNL**	1.302 ± 0.042
3. hugh	3. Hyperborean-tbr	1.133 ± 0.026
4. Rembrant	4. player.kappa.nl	1.026 ± 0.105
5. Lucky7	5. hugh	0.968 ± 0.054
6. player.kappa.nl	6. Rembrant	0.464 ± 0.024
7. POMPEIA	7. POMPEIA	-6.460 ± 0.051

2012 AAAI Computer Poker Competition

The SartreNL agent submitted to the 2012 competition was unchanged from the 2011 entry, apart from the case-base which was used to make betting decisions. For the 2012 competition, SartreNL's case-base was created by training on the decisions of the best 2011 competitor from the instant runoff division (i.e Hyperborean-iro). Once again, we observe in Table 3.15 that, in general, competitors have improved over the year. In 2011, SartreNL placed 2^{nd} in both divisions of the heads-up no-limit competition, whereas a year later the same architecture now achieves a 7^{th} place finish in the total bankroll division and a 5^{th} place finish in the instant runoff division, out of eleven competitors.

Human Opponents – No-Limit

Once again, we have gathered results against human opponents in the no-limit domain via our browser based application. As with the real world limit hold'em results (see Section 3.3.4.), these results are presented as a guide only and care must be taken in drawing any final conclusions. Fig. 3.7. records the number of big blinds won in total against all human opponents. SartreNL achieves a final profit of 7539.5 big blinds in just under 9000 hands. Fig. 3.8. shows the big blinds per hand (bb/h) won over all hands played, the final bb/h value recorded is 0.9023.

Table 3.15: 2012 no-limit heads up bankroll and runoff results, respectively

No-Limit Bankroll Runoff	No-Limit Total Bankroll	Average Won/Lost
1. Hyperborean	1. Little Rock	1.117 ± 0.029
2. Tartanian5	2. Hyperborean	0.596 ± 0.033
3. Neo Poker Lab	3. Tartanian5	0.580 ± 0.029
4. Little Rock	4. Neo Poker Lab	0.515 ± 0.024
5. **SartreNL**	5. Hugh	0.452 ± 0.20
6. Hugh	6. Spewy Louie	0.319 ± 0.031
7. Spewy Louie	7. **SartreNL**	0.088 ± 0.028
8. Lucky7_12	8. Azure Sky	-0.756 ± 0.073
9. Azure Sky	9. Lucky7_12	-1.077 ± 0.063
10. dcubot	10. UNI-mb_poker	-1.833 ± 0.052
11. UNI-mb_poker		

Discussion

The results of the 2010 ACPC were somewhat mixed as SartreNL performed very well in the *instant runoff* division, placing 2^{nd}, but not so well in the *total bankroll* division where the average profit won $+0.537 \pm 0.034$ was lower than all but one of the competitors. This relatively poor overall profit is most likely due to the 2010 version of SartreNL not encoding an *all-in* action in its abstraction, which would have limited the amount of chips that could be won.

After updating the strategy for the 2011 competition to accurately reflect the framework described, SartreNL's performance and total profit improves to $+1.302 \pm 0.042$. As such SartreNL is able to perform well in both winner determination procedures, placing 2^{nd} in both divisions of the 2011 competition out of seven competitors. The same agent (with an updated case-base) placed 5^{th} out of an increased number of competitors (eleven in total) at the 2012 competition.

Results were also presented for SartreNL against human opposition. While it is interesting to get an initial idea of how SartreNL does against human opponents, it would be unwise to draw any conclusions from this data, especially because not nearly enough hands have been played (given the variance involved in the no-limit variation of the game) to make any sort of accurate assessment.

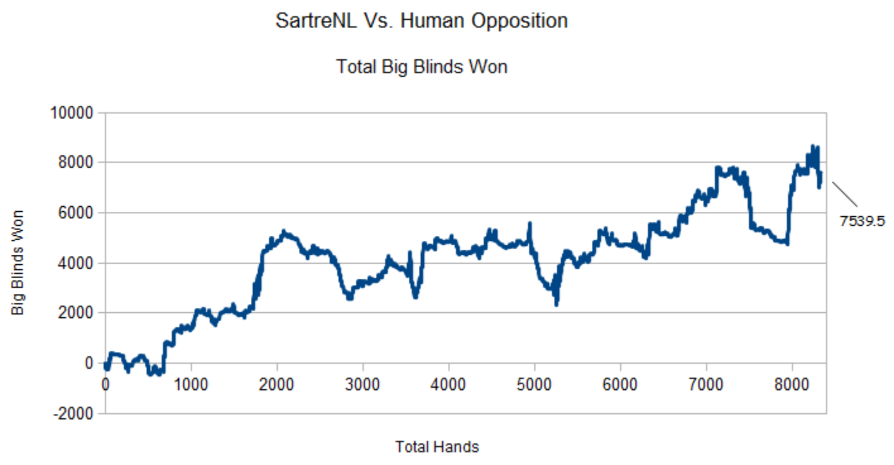

Figure 3.7: The number of big blinds won in total against every human opponent to challenge SartreNL.

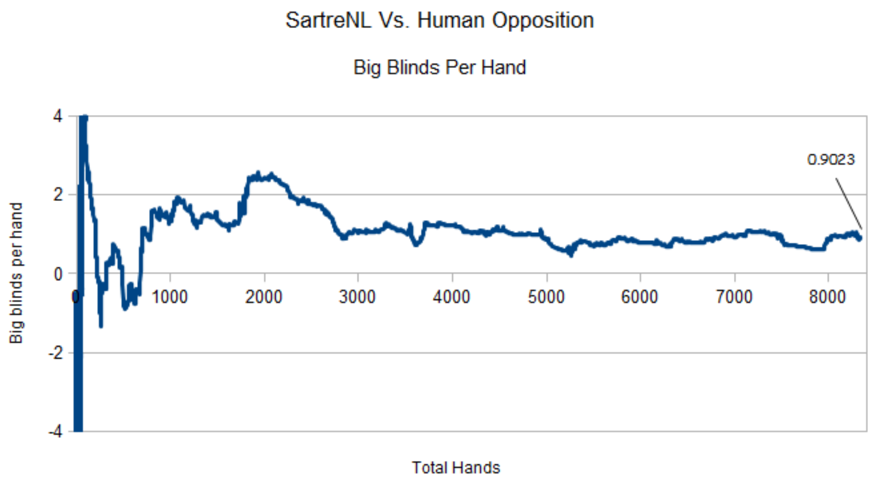

Figure 3.8: The big blinds per hand (bb/h) won by SartreNL over every hand played.

3.5. Multi-Player, Limit Texas Hold'em

The final sub-domain that we have applied and evaluated case-based strategies within is multi-player Texas Hold'em. Specifically, three-player, limit Texas Hold'em, where an agent is required to challenge two opponents instead of just one.

Once again, we use the lessons learned by applying maintenance in the two-player, limit hold'em domain in order to finalise a framework that can handle multi-player betting.

3.5.1. A Framework for Producing Case-Based Strategies in Multi-Player, Limit Texas Hold'em

We present the framework established for producing multi-player, limit case-based strategies. Our framework consists of:

1. A representation for encoding cases and game state information
2. The corresponding similarity metrics required for decision generalisation.

Case Representation

Table 3.16 depicts the final post-flop case representation used to capture game state information in the multi-player, limit hold'em domain. The main difference between Table 3.16 and the post-flop case representation for limit hold'em 3.4 has to do with the values used to represent the *betting sequence* attribute. As multi-player sequences involve more players, the sequences that record betting actions are larger and more complicated than those in the two-player domain.

Each of the attribute-value pairs are described in more detail below.

1. Hand Strength: As in the previous domains, a player's hand is represented by calculating the E[HS^2] of the player's cards and mapping these values into 1 out of 50 possible buckets.

2. Betting Sequence: The details of the betting sequence attribute are similar to those described in the two-player, limit domain where actions are chosen from $A_{limit} = \{f, c, r\}$, and rounds are delimited by hyphens. However, as more players are involved in a hand, the betting sequences are different than those produced in the two-player domain.

3. Board Texture: The nine board texture categories, defined in Table 3.5, are re-used in the multi-player domain to capture important public card information.

Table 3.16: The case representation used for producing case-based strategies in the domain of multi-player, limit Texas Hold'em.

Attribute	Type	Example
1. Hand Strength	Integer	1 – 50
2. Betting Sequence	String	*rcrrcc-crrcrrcc-ccr, rcc-ccc-ccrf, ...*
3. Board Texture	Class	*No-Salient, Flush-Possible, Straight-Possible, Flush-Highly-Possible, ...*
Action	Triple	(0.0, 0.95, 0.05), (0.6, 0.1, 0.3), ...
Outcome	Triple	($-\infty$, 9.0, -3.0), (-2.0, -4.5, -7.0), ...

Similarity Metrics

Once again, each feature requires a corresponding local similarity metric in order to generalise decisions contained in a set of data.

1. Hand Strength: Similarity between hand strength values is determined by the same metric used in the two-player limit hold'em domain, as specified by Equation 3.2.

2. Betting Sequence: To determine similarity between three-player betting sequences, a stepped level similarity metric is used that defines three levels of similarity (level0, level1, level2), similar to the two-player limit domain.

Given two betting sequences:

level0: is satisfied if the two betting sequences exactly match one another.

level1: is satisfied if the number of bets/raises made by each active player is the same for each individual, non-current betting round.

level2: is satisfied if the total number of bets/raises made by each active player is the same for all non-current betting rounds combined.

For level1 and level2 similarity above, the current betting round must match exactly for two betting sequences to be considered similar. As in the two-player domain, similarity values of 1.0, 0.9 and 0.8 are assigned to the satisfied similarity level, respectively.

3. Board Texture: Similarity between board texture values is determined by the same similarity matrix used in previous domains, as specified in Fig. 3.3.

3.5.2. Multi-Player Limit Experimental Results

We now present results obtained given the case-based strategies produced in the domain of three-player, limit Texas Hold'em. We present results from the 2011 and 2012 Annual Computer Poker Competitions, where our multi-player case-based strategies were challenged against some of the best multi-player limit agents in the world.

2011 AAAI Computer Poker Competition

We submitted a multi-player case-based strategy for the first time at the 2011 ACPC held at the Twenty-Fifth AAAI Conference on Artificial Intelligence. Our entry, Sartre3P, competed in the three-player limit Texas Hold'em competition. The results of the 2011 competition are given in Table 3.17. As with previous ACPC results, two winner determination procedures are displayed - *instant runoff* and *total bankroll*. Sartre3P placed 2^{nd} in the *instant runoff* division, out of 9 competitors. In this division Sartre3P was beaten only by the competitor Hyperborean.iro. For the *total bankroll* division, Sartre3P placed 1st and was the overall winner of this division. Sartre3P's average profit was 0.266 ± 0.024 sb/h, which was significantly greater than all other competitors in the competition. Full cross-table results are presented in Appendix A.

Table 3.17: 2011 multi-player limit bankroll and runoff results. Values are in sb/h with 95% confidence intervals.

Multi-player Bankroll Runoff	Multi-player Total Bankroll	Average Won/Lost
1. Hyperborean-3p-iro	1. **Sartre3P**	0.266 ± 0.024
2. **Sartre3P**	2. Hyperborean-3p-tbr	0.171 ± 0.023
3. LittleRock	3. AAIMontybot	0.130 ± 0.045
4. dcubot3plr	3. LittleRock	0.122 ± 0.022
5. Bnold3	5. OwnBot	0.016 ± 0.035
6. AAIMontybot	6. Bnold3	-0.084 ± 0.028
7. OwnBot	7. Entropy	-0.099 ± 0.043
8. Entropy	8. player.zeta.3p	-0.521 ± 0.040
9. player.zeta.3p		

2012 AAAI Computer Poker Competition

The same version of Sartre3P that participated in the 2011 competition was re-used for the 2012 competition. No updates were made to the case-bases that Sartre3P used to make its decisions. The results of the multi-player limit competition are shown in Table 3.18. In both divisions a third place tie was declared between Sartre3P and Neo Poker Lab. Notice, however, that statistical significance is not guaranteed with these results. When the standard deviations are taken into account, it is evident that there is a large amount of overlap in bankroll values for all but the 1st place finisher.

Table 3.18: 2012 multi-player limit bankroll and runoff results. Values are in sb/h with 95% confidence intervals.

Multi-player Bankroll Runoff	Multi-player Total Bankroll	Average Won/Lost
1. Hyperborean	1. Hyperborean	0.028 ± 0.005
2. Little Rock	2. Little Rock	-0.004 ± 0.007
3. Neo Poker Lab	3. Neo Poker Lab	-0.011 ± 0.005
3. **Sartre3P**	3. **Sartre3P**	-0.012 ± 0.007
5. dcubot		

Discussion

The version of Sartre3P submitted to the 2011 competition employed a novel *strategy switching* concept that has not yet been discussed. *Strategy switching* involves re-using strategies trained from one poker domain within a separate poker domain. In Chapter 6 of this thesis, we provide full details of *strategy switching* and its evaluation.

Overall, the results presented in Table 3.17 provide strong support for the efficacy of *case-based strategies* that have been trained on hand history data from previous year's competitions. Moreover, the results demonstrate that strategies produced in this *top-down* fashion actually have the capability to defeat their *bottom-up* counterparts, in certain situations.

3.6. Concluding Remarks

In this chapter, we have provided a comprehensive overview of our *case-based* strategies that employ a *top-down* approach by generalising decisions from a collection of data. We began with a description of the systematic maintenance performed on our strategies in the domain of two-player, limit Texas Hold'em. The final result was a framework we have employed to produce *case-based* strategies that have achieved top place finishes at international computer poker competitions, where the best computer poker agents in the world are challenged against each other. Using the lessons learned and insights obtained from the two-player, limit hold'em domain, we extrapolated our framework to handle the more complicated domains of two-player, no-limit Texas Hold'em and multi-player, limit Texas Hold'em. In the no-limit domain, our case-based strategies produced the 2^{nd} best computer poker agent as judged by the results of the 2011 ACPC. In the multi-player, limit domain our agent was the winner of the total bankroll competition at the 2011 ACPC and achieved the greatest profit against all other competitors. These results show that the *case-based* frameworks we have presented are able to produce strong, sophisticated strategies that are competitive at an international level. In the remaining chapters we will investigate extending the basic framework introduced in this chapter and we will show that by augmenting *imitation*-based, *top-down* strategies with additional capabilities, it is possible to outperform the experts whose decisions were used to train the strategies in the first place.

Chapter 4

Implicit Opponent Modelling via Dynamic Case-Base Selection

[1]In this and the next few chapters, we discuss extensions to the basic frameworks that were introduced in Chapter 3. Each extension augments the basic framework for producing case-based strategies in different ways. Recall that the case-based strategies from Chapter 3 were static, non-adaptive strategies that performed no opponent modelling. The first augmentation procedure we present introduces *implicit opponent modelling* into the architecture. Standard opponent modelling approaches construct a model of their opponent's play and use this to inform how to modify their own strategy. Rather than deciphering the details of an opponent's strategy, implicit opponent modelling uses only performance information to determine how to modify its own strategy. Given a choice of playing actions, decisions that led to positive outcomes against the opponent are more likely to be selected and actions that led to losses against the opponent are less likely to be repeated. We detail an implicit opponent modelling approach in this chapter that constructs multiple case-bases by training on different sets of data and selects at runtime an appropriate case-base to search based on how well each case-base does against the current opponent. Each case-base attempts to capture a particular style of play of an original expert. By correctly determining which style to choose against varying opponents, gains in performance should be possible.

[1]The contents of this chapter have been modified from an original publication which appeared in the proceedings of IJCAI 2011. Jonathan Rubin & Ian Watson. On Combining Decisions from Multiple Expert Imitators for Performance. In *IJCAI-11, Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence.*

4.1. Multiple Case-Bases

Performance in the game of poker is known to have an intransitive relationship, i.e. while player A beats player B and player B beats player C, it does not necessarily follow that A beats C. Given the intransitive nature of poker strategies, the successful combination of different styles of play has the potential to largely improve overall performance. By training on experts with different styles of play, we produce several case-bases and create multiple expert imitators in the domains of limit and no limit Texas Hold'em. We determine whether the performance of individual expert imitators can be improved by combining their decisions. We compare two approaches for combining decisions.

The first approach combines decisions using ensemble voting [30], where a final decision is made by having each expert imitator vote on their preferred action. The second approach combines decisions by dynamically selecting a single expert imitator for each hand of play. This approach attempts to choose the expert imitator that achieves the greatest profit against a particular style of play in order to maximise overall profit against a range of opponents. This approach was originally applied to the game of limit Texas Hold'em in [54, 55] where the UCB1 [7] allocation strategy was applied to dynamically select experts during play. Here we follow the same procedure described in [55] for dynamically selecting case-bases. Our own research extends this idea to the more complicated domain of no limit Texas Hold'em. Furthermore, our work focusses specifically on combining the decisions from multiple *expert imitators*. A benefit of this approach is that producing a new strategy simply requires replacing the training data that is used to construct the case-base. Figure 4.1 depicts an extension to the high-level architecture, previously presented in Chapter 3, to reflect dynamic case-base selection.

While any learning algorithm that uses training data from a single expert to train a system can be considered an *expert imitator*, some learning algorithms are more suited to this particular purpose than others. For example, the *lazy learning* [3] of case-based reasoning is particularly suited to expert imitation where expert observations can be recorded and stored for later use at decision time. Depending on the particular domain, some CBR systems have demonstrated successful imitation after making only a handful of observations [68]. In the games domain, imitation-based learning was successfully demonstrated in simulated robot soccer to control a multi-agent soccer team [33] and in real-time strategy games, where expert humans were observed and their decisions used to generate unique strategies [82].

Many classification systems have made use of *ensemble* decision making to improve accuracy. Typically a collection of different learned hypotheses will vote for a particular category. The category with the most (possibly weighted) votes is the one that is selected. Davidson [28] used an ensemble approach to predict future actions of an opponent in the domain of limit

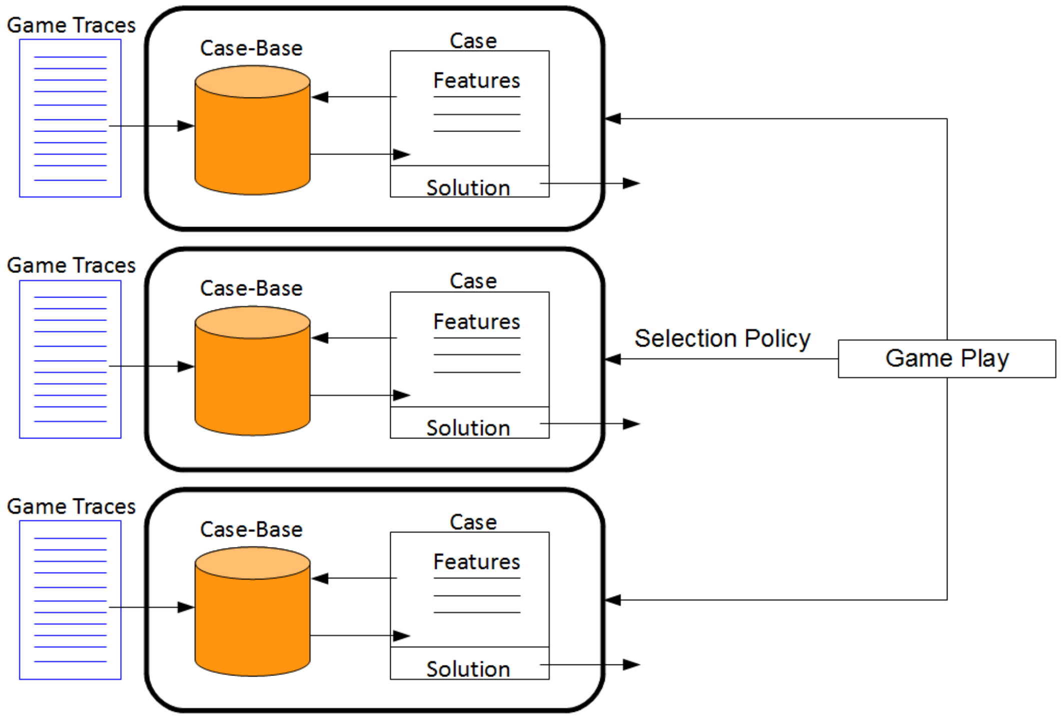

Figure 4.1: An extension to the basic architecture which allows dynamic case-base selection.

Texas Hold'em. The results showed that having multiple learning algorithms vote improved overall accuracy compared to just a single learning algorithm.

Rather than having different hypotheses vote on a solution, a more difficult problem involves dynamically selecting an appropriate hypothesis at runtime in order to maximise overall payoff. Johanson et al. [54] created a collection of exploitive strategies in the game of limit Texas Hold'em that were designed to exploit particular opponents. They employed the UCB1 update policy together with showdown DIVAT [12] analysis to select the best exploitive player against a particular opponent during game play. In this work we apply a similar idea to selecting *expert imitators* from separate case-bases in the domain of limit and no-limit Texas Hold'em.

4.2. Expert Imitators

The expert imitators we describe in this work are lazy learners created using the case-based reasoning methodology. Each expert imitator, I_j, is made up of a collection of cases

$$C_j = \{c_{j,1}, c_{j,2}, \ldots, c_{j,n}\}$$

where j refers to the particular expert being imitated and

$$\forall c \in C_j, c = (x, a)$$

where x is a feature vector that captures game state information and a is an action vector that specifies the probability of taking a certain action, given game state x.

Given a game state described by c_t, a decision is made by processing the collection of stored cases and maximising a global similarity metric to retrieve the most similar case to c_t.

$$c_{max} = \arg\max_{c_k} sim(c_k, c_t), \forall c_k \in C_j \tag{4.1}$$

where, $sim(c_1, c_2)$ is some global similarity metric that determines the similarity between the feature vectors of the two cases c_1, c_2.

The retrieved case, c_{max}, suggests an action (a_{max}) that I_j uses to approximate the decision that the original expert would have taken, given the current game state.

4.2.1. Action Vectors

The action vectors used to make a betting decision differ depending on whether the limit or no limit variation is being played. In the domain of limit hold'em the action vector used by the expert imitator has the following format:

Table 4.1: A no limit action vector. Bet amounts are mapped into one of the discrete categories listed.

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

$$a = (f, c, r)$$

In the no limit variation, players are allowed to bet or raise any amount including all of the money or chips they possess (called going *all-in*). For no-limit we use the mapping introduced in section 3.4.3. to assign quantitative bet amounts into discrete categories. The no-limit action vector used by the expert imitators is given by the following:

$$a = (f, c, q, h, i, p, d, v, t, a)$$

The significance of each action is reproduced in Table 4.1 for convenience. Once again each entry corresponds to the probability of taking that particular action and all entries in the vector sum to 1.0.

4.3. Combining Decisions

Recall that given an action vector, the final action an expert imitator selects can be determined by one of the following policies:

1. **Probabilistic** – Select an action probabilistically based on the values specified in the action vector. Or,

2. **Max Frequency** – Select the action within the triple that has the greatest probability value.

Depending on how we wish to combine decisions, the above two policies are applied in different ways. The first procedure we evaluate for combining decisions from multiple case-bases uses ensemble voting and requires the use of the *max frequency* policy for decision making.

4.3.1. Ensemble-based Decision Combination

Given an action vector $a = (a_1, a_2, \ldots, a_n)$ and a collection of expert imitators I_1, I_2, …, I_m, where each imitator applies the *max frequency* decision policy, we can derive a vector $v = (v_1, v_2, \ldots, v_n)$, whose elements correspond to the total number of votes each action received.

The final action taken by the ensemble of imitators is given by selecting the action, a_j, that corresponds to the *strictly* maximum number of votes, v_j, if one exists. If a strictly maximum number of votes does not exist, then the action specified by imitator I_1 is used, where I_1 is an arbitrary case-base selected by the author.

4.3.2. Dynamic Case-Base Selection

Rather than using ensemble voting to merge action vectors retrieved from separate case-bases, a more sophisticated decision combination procedure involves selecting the best action vector to use from the most appropriate case-base, given the current situation. Combining decisions in this way is reminiscent of the multi-arm bandit problem.

Multi-arm Bandit Problem

The multi-arm bandit problem is a well known problem in statistics and artificial intelligence. The problem consists of a gambler and a collection of slot machines (one-armed bandits). Each slot machine has a particular reward distribution associated with it, which is unknown to the gambler. The gambler seeks to maximise their overall profit by playing the right machines. The problem for the gambler is to determine which machine to play next based on the sequence of rewards they have witnessed so far.

The situation described above depicts a classic example of the **exploration/exploitation** trade off. The gambler needs to choose between selecting the machine that has so far offered the greatest reward (*exploitation*), versus selecting a different machine which may offer yet a greater reward than that witnessed so far (*exploration*). The UCB1 [7] algorithm offers a simple, computationally efficient solution to the multi-arm bandit problem and the **exploration/exploitation** trade off. Successfully handling the trade off between exploration and exploitation is a problem that is consistently faced in statistics and artificial intelligence.

UCB1

To address the problem of dynamic case-base selection we adapt the UCB1 algorithm. The UCB1 algorithm defines a policy for strategy selection based on the concept of **regret**. *Regret* refers to the difference between the reward that would have been received, had the optimal strategy always been selected, compared to the actual reward received. The UCB1 algorithm offers a logarithmic bound on regret.

Given a collection of case-bases, $\{C_j : j = 1 \dots N\}$, we perform implicit opponent modelling by dynamically selecting an appropriate case-base to use against a particular opponent. An individual case-base is dynamically selected at the beginning of every hand. In order to dynamically select an appropriate case-base at runtime, the following application of UCB1 is used:

1. Select each case-base once
2. Select case-base, C_j, that maximises the following equation:

$$\arg\max_j \bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}} \tag{4.2}$$

where,

$\bar{x}_j$ is the average amount won by case-base C_j, when challenging the current opponent,
n is the total number of hands played so far, and
n_j is the total number of times case-base C_j has been selected.

The UCB1 algorithm above determines the order in which to select different case-bases, given their performance against a particular opponent. In the computer poker domain, $\bar{x}_j$ in equation (4.2) refers to the average utility (profit or loss) achieved by case-base, C_j, while playing against the current opponent. Similarly, n_j refers to the total number of times C_j has been chosen to play against the current opponent. However, due to the inherent variance present in the game of Texas Hold'em, strategy selection based on profit alone can severely bias the results of the UCB1 allocation policy. Hence, efforts to reduce this inherent variance are required in order to stabilise the results.

Variance Reduction

As in [55], we employ the use of DIVAT analysis [12] in order to improve the average utility values used by the UCB1 allocation strategy. DIVAT (ignorant value assessment tool) is a perfect information variance reduction tool developed by the University of Alberta Computer Poker Research Group. Recall that the basic idea behind DIVAT is to evaluate a hand based on the expected value (EV) of a player's decisions, not the actual outcome of those decisions. DIVAT achieves this by comparing the EV of the player's decisions against the EV of some baseline strategy, see equation 4.3.

$$EV(ActualActions) - EV(BaselineActions) \tag{4.3}$$

Equation 4.3 attempts to factor out the effects of luck as both the actual strategy and the baseline strategy experience the same lucky or unlucky sequence of events. When the player is able to achieve a greater EV than the baseline, the player is rewarded. Alternatively, if the player's actions result in a lower EV then they are punished with a negative outcome. If the EV of both strategies is the same the outcome is 0.

Finally, DIVAT requires perfect information about both players' hole cards. When either player folds this information is not available. Therefore, to calculate the utility values used by the UCB1 allocation strategy, DIVAT analysis is employed only when a showdown occurs. If a fold occurs the actual monetary outcome of the hand is used instead.

4.4. Methodology

We provide experimental results in domains of both limit and no limit Texas Hold'em. In each domain three basic expert imitators (case-bases) were constructed. All expert imitators were trained on hand history data from the annual computer poker competition [2]. Each expert imitator was trained on separate data determined as follows:

1. The first expert imitator was trained on decisions made by the agent that won the bankroll division of the 2010 AAAI computer poker competition.

2. The second expert imitator was trained on decisions made by the agent that won the instant runoff division of the 2010 AAAI computer poker competition.

3. The final expert imitator was our own entry into the 2010 AAAI computer poker competition.

Both *max frequency* and *probabilistic* decision re-use policies were investigated resulting in a total of three case-bases and six basic expert imitators.

From these six expert imitators we constructed a further two players that combined decisions in the following ways:

1. An *ensemble-based* player was constructed where each of the basic *max frequency* expert imitators voted for a particular action and the action with the most votes was selected.

The decision of one designated expert imitator, selected by the author, was used in case all expert imitators voted for a separate action.

2. A *dynamic-selection* based player was constructed that dynamically selected at runtime the best basic expert imitator to use against the current opponent. Selection was based on the UCB1 allocation strategy together with showdown DIVAT analysis. In the limit variation a bet-for-value baseline strategy was adopted during DIVAT analysis. In no-limit a more simplistic *always-call* strategy was used as the baseline.

The Dynamic and Ensemble strategies, along with all six (single case-base only) expert imitators, were then challenged against two different types of computerised players. All matches played were duplicate matches. A duplicate match involves playing the same 3000 hands twice. In the first run, 3000 unique hands are played, after which the players' memories are wiped and the 3000 hands are played again, but in the reverse direction, i.e. the cards that were initially given to player A are instead given to player B. This way both players get to play both sets of cards and this reduces the variance that is involved in simply playing a set of hands in one direction only. All case-based strategies played five duplicate matches against each of the two computerised opponents. In total, 480,000 hands of poker were played.

4.5. Experimental Results

4.5.1. Limit Results

In the limit domain each expert imitator challenged the following competitors:

1. **Fell Omen 2** – A solid Nash equilibrium-based agent commonly used as a benchmark for testing limit hold'em agents.

2. **AlistairBot** – An exploitive agent that uses Monte-Carlo simulation to determine the decision with the best EV against the current opponent.

Table 4.2 presents the results of the 8 expert imitators against the above two competitors. The results are presented in small bets per hand (sb/h), where the total number of small bets won or lost are divided by the number of hands played. Each original expert imitator begins with the name of the original expert used to train the system followed by the decision policy used, i.e. either *max frequency* or *probabilistic*. The Sartre-max agent was our entry into the 2010 computer poker competition. Figures 4.2 and 4.3 show the total number of times each expert imitator was selected via the UCB1 allocation policy (used by Dynamic) to play against each of the competitors.

Table 4.2: Expert Imitator Results against Fell Omen 2 and AlistairBot

	FellOmen2		**AlistairBot**		**Average**	
Dynamic	**0.01342**	±**0.006**	0.68848	±0.009	**0.35095**	±**0.0075**
Ensemble	0.00830	±0.010	0.67348	±0.010	0.34089	±0.0100
Rockhopper-max	-0.01445	±0.009	**0.69504**	±**0.009**	0.34030	±0.0090
PULPO-max	-0.00768	±0.006	0.66053	±0.024	0.32643	±0.0150
Sartre-max	0.00825	±0.014	0.63898	±0.019	0.32362	±0.0165
PULPO-prob	-0.00355	±0.011	0.63385	±0.018	0.31515	±0.0145
Sartre-prob	-0.01816	±0.006	0.64535	±0.015	0.31360	±0.0105
Rockhopper-prob	-0.02943	±0.011	0.63870	±0.020	0.30464	±0.0155

4.5.2. Limit Discussion

Overall, the results presented in Table 4.2 supports the idea that combining decisions from multiple case-bases can improve performance over a single case-base alone. Against Fell Omen 2, Dynamic does better than any other expert imitator. Against AlistairBot, the expert imitator trained on the Rockhopper agent, using a *max frequency* betting policy, fares the best. Combining the results against the two opponents and taking the average sees Dynamic achieve the best average outcome, followed by Ensemble. However, care must be taken in interpreting these results as there is overlap between the standard deviations.

Also depicted in the results are plots showing the total number of times each original expert imitator was selected via the UCB1 allocation strategy to play against Fell Omen 2 (Figure 4.2) and AlistairBot (Figure 4.3). Table 4.2 suggests that the expert imitator Sartre-max did the best out of all the original imitators against Fell Omen 2, achieving a profit of $+0.00825$ sb/h. In particular, Sartre-max was the only original imitator to achieve a slight profit against Fell Omen 2, whereas the other 5 achieved slight losses. It was therefore thought by the author that this agent would be selected the most by Dynamic when challenging Fell Omen 2, however this appears not to be the case. Instead, Figure 4.2 shows that Rockhopper-max was selected the most overall, followed by Sartre-prob and Sartre-max in third place. A similar scenario is depicted in Figure 4.3, where PULPO-prob is selected by far the most, whereas the obvious choice (Rockhopper-max, as it achieves the greatest profit of $+0.69504$ sb/h against AlistairBot) is only 3rd on the list. These discrepancies are likely due to a combination of variability in the data and possibly exaggerated utility values within the UCB1 allocation strategy, caused when variance reduction is not able to take place, such as when a fold occurs.

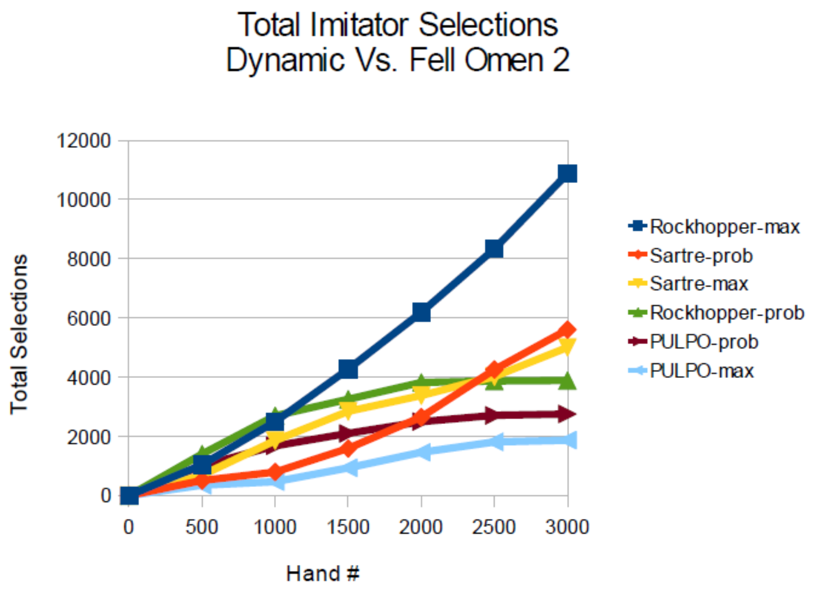

Figure 4.2: Shows the total number of times each expert imitator was selected to challenge Fell Omen 2

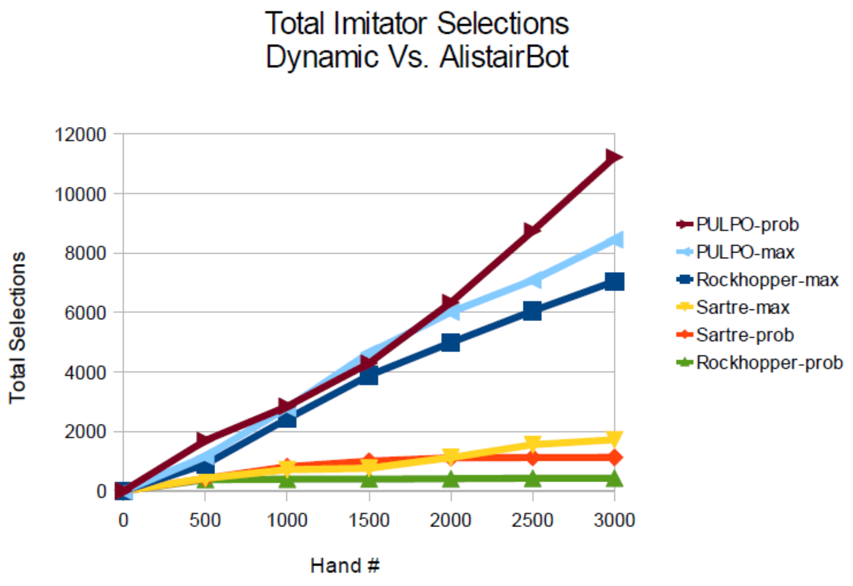

Figure 4.3: Shows the total number of times each expert imitator was selected to challenge AlistairBot

Table 4.3: Expert Imitator Results (No Limit) against MCTSBot and SimpleBot

	MCTSBot		**SimpleBot**		**Average**	
Hyperborean-max	**1.7781**	**±0.216**	**0.7935**	**±0.095**	**1.2858**	**±0.156**
Dynamic	1.3332	±0.164	0.5928	±0.065	0.9630	±0.115
Ensemble	1.3138	±0.053	0.5453	±0.065	0.9295	±0.059
Hyperborean-prob	1.1036	±0.184	0.5075	±0.104	0.8055	±0.144
Sartre-max	0.9313	±0.131	0.5248	±0.036	0.7281	±0.086
Sartre-prob	0.4524	±0.118	0.3933	±0.082	0.4228	±0.100
Tartanian-max	0.1033	±0.505	0.3450	±0.098	0.2242	±0.302
Tartantian-prob	-0.0518	±0.142	0.4221	±0.036	0.1852	±0.089

4.5.3. No-Limit Results

In the no-limit domain each expert imitator challenged the following opponents:

1. **MCTSBot** – an exploitive agent that uses Monte-Carlo Tree Search [115].
2. **SimpleBot** – a no-limit rule-based agent.

The *opentestbed*[2] project was used to gather results as the above no-limit agents were made publicly available within this framework. Table 4.3 presents the no-limit results. The results are in big blinds per hand. Once again Figures 4.4 and 4.5 show the total number of times an expert imitator was selected via the UCB1 allocation strategy to play against the above two competitors.

4.5.4. No-Limit Discussion

While both Dynamic and Ensemble achieve high placings in the no-limit results (2nd and 3rd, respectively). Table 4.3 suggests that they were not able to improve upon the performance of a single case-base, i.e. Hyperborean-max. Here Hyperborean-max performs the best against both MCTSBot and SimpleBot, suggesting that an alternative choice of opponent may be required in order to receive any benefit by combining decisions from multiple case-bases. In both Figures 4.4 and 4.5, Sartre-max was selected the most by Dynamic, followed by Hyperborean-max.

The altered betting structure from limit to no-limit had a few follow on effects when it came to dynamically selecting case-bases. Firstly, the amount of money invested by each player is typically larger than in limit poker. This can result in larger swings in the utility values used by

[2]http://code.google.com/p/opentestbed/

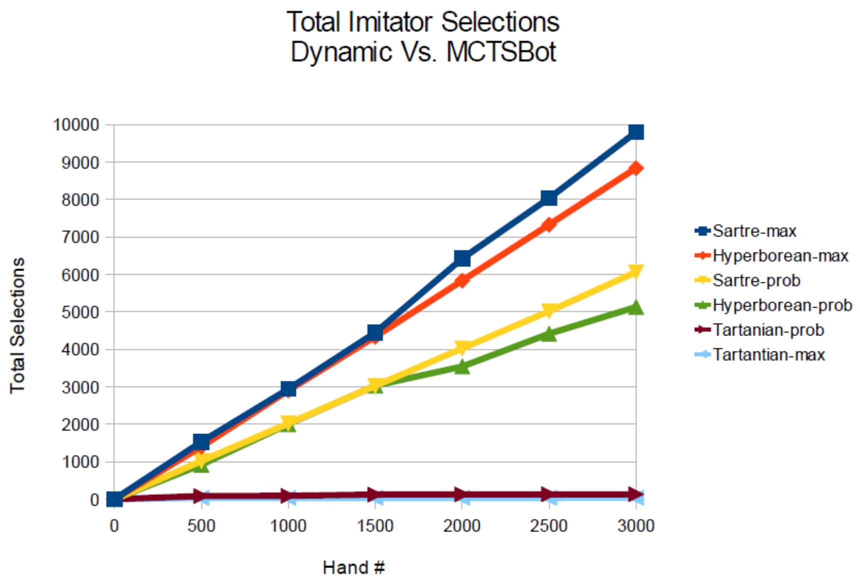

Figure 4.4: Shows the total number of times each no limit expert imitator was selected to challenge MCTSBot

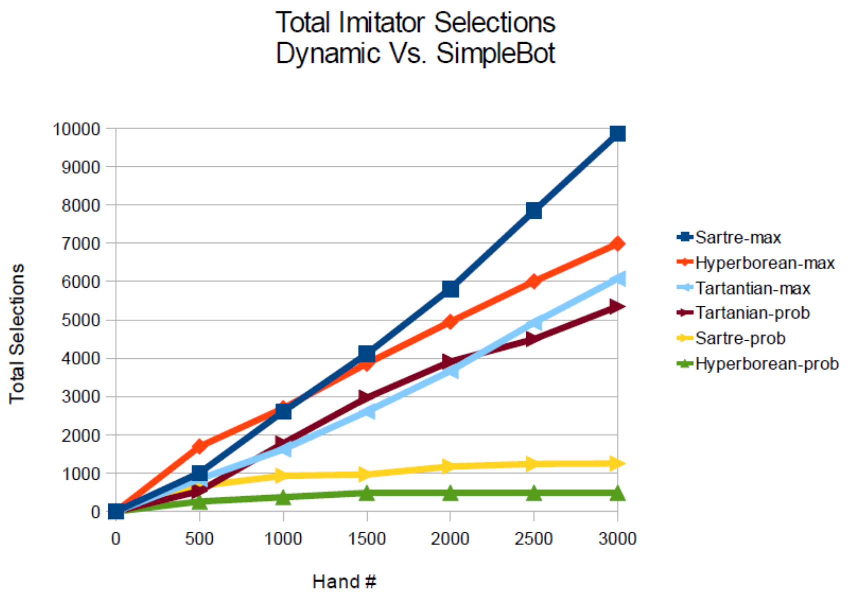

Figure 4.5: Shows the total number of times each no limit expert imitator was selected to challenge SimpleBot

the UCB1 allocation strategy. Furthermore, in no-limit a lot more hands end when one player folds to another player's bet, compared to limit where a lot more hands proceed all the way to a showdown. Showdown DIVAT (as its name suggests) can only be applied when a showdown is reached. The increased proportion of fold actions observed in no-limit poker results in less variance reduction taking place on the utility values used by UCB1 and hence could have resulted in possibly selecting an inappropriate case-base a lot more often than was necessary.

4.6. Concluding Remarks

In this chapter, we extended the basic architecture introduced in Chapter 3 by constructing multiple case-bases and combining the different solutions they suggested in order to determine playing decisions. We compared two separate approaches for combining decisions from multiple case-bases in an attempt to improve performance. The first approach used ensemble voting to merge action vectors. The second approach dynamically selected case-bases at runtime, based on their performance against a particular opponent. This is a form of implicit opponent modelling, as decisions were made based solely on performance information, rather than by constructing more sophisticated models of the opponent.

Overall, combining decisions either via ensemble voting or dynamic case-base selection at runtime appears to improve performance. In the limit variation, combining decisions was able to produce better results than relying on one expert alone. This is likely due to the intransitive nature of poker strategies. In the no-limit variation, dynamic selection did better than most single expert imitators alone, but was not the overall best strategy to use. This could be due to the fact that we simply have not tested against a diverse enough set of opponents and therefore, no benefit was received by selecting different expert imitators, or that further improvements to the variance reduction techniques used in the no-limit domain are required to stabilise the results.

In the next chapter, we present a second extension to the architecture of Chapter 3. Our second augmentation procedure performs explicit opponent modelling by capturing information about the opponent's style of play. As will be discussed in Chapter 5, this opponent modelling information is made use of to adapt case solutions against specific opponents. This leads to both adaptive and exploitive strategies.

Chapter 5

Explicit Opponent Modelling via Opponent Type Solution Adaptation

[1]In the domain of computer poker, exploitive strategies can be produced by performing either *implicit* or *explicit* opponent modelling. The previous chapter introduced *implicit opponent modelling*, which does not attempt to decipher the details of an adversary's strategy, but instead attempts to select an appropriate response from a range of strategies based on performance information against the opponent. On the other hand, *explicit opponent modelling* does concern itself with discovering details about an opponent's strategy and, in particular, any weaknesses within that strategy. This information is then used to alter the agent's own strategy in an attempt to improve overall performance. In this chapter, we describe a second augmentation procedure that uses explicit opponent modelling to produce exploitive and adaptive *case-based strategies*. We describe how *adaptation* can be applied to a precomputed, static *case-based strategy* in order to allow the strategy to rapidly respond to changes in an opponent's playing style. The work described in this chapter differs from Chapter 4, as rather than implicitly exploiting opponents by selecting from a range of static strategies, exploitation instead occurs in a more explicit manner by modifying the details of a single strategy directly via adaptation, given a certain opponent type. The exploitive strategies produced by this approach tend to *hover* around a precomputed solid strategy and adaptation is applied directly to the precomputed strategy once enough information has been gathered to classify the current *opponent type*. The use of a precomputed, *seed* strategy avoids performance degradation that can take place when little is known about an opponent. Augmenting our case-based strategies in this way has an advantage over other

[1]The contents of this chapter originally appeared in the proceedings of the Twentieth International Conference on Case-Based Reasoning (ICCBR 2012). Jonathan Rubin & Ian Watson. Opponent Type Adaptation for Case-Based Strategies in Adversarial Games. In *Case-Based Reasoning Research and Development - 20th International Conference (ICCBR 2012)* [96].

exploitive strategies whose playing decisions rely on large, individual opponent models constructed from scratch.

5.1. Opponent Type Based Adaptation

Actions selected by a case-based strategy can either be applied directly or can be modified/adapted to better account for the current situation. In previous chapters, solution vectors were simply re-used and no attempt was made to adapt these solutions at all. *Adaptation* is typically required when the current environment and similar past environments are no longer identical, but instead differ in particular ways.

In some gaming environments, making use of knowledge recorded about an adversary can dramatically improve overall performance. In this chapter, we describe an extension to the frameworks introduced in Chapter 3 that records information about particular *opponent types* and uses the information gathered to affect the selection of actions via adaptation.

Rather than employing adaptation to handle moderate differences in gaming environments, as done by other case-based strategies in adversarial environments [80, 112, 8, 38] we apply adaptation based on the playing style of a current opponent. In this way we are able to generate case-based strategies that are both exploitive and adaptive i.e. they can rapidly respond to changes in the playing style of a current opponent and they do so in a way that attempts to improve overall performance.

To begin with we construct an offline model that captures information about a set of opponent types. The information contained within this model becomes the basis for informing future adaptation based on the type of opponent currently being challenged. The exploitive strategies produced by this approach *hover* around a precomputed solid strategy and adaptation is applied directly to the precomputed strategy once enough information has been gathered about the current opponent *type*. Figure 5.1 illustrates a high-level overview of the augmentation procedure. Here the environment refers to the actual game being played. Given information from the environment, opponents are classified into a certain type. Once the opponent type is known, an opponent model is queried that dictates the type of adaptation that should be applied to case solutions against this opponent. Given the adaptation suggested by the opponent model, this is applied to the current environment.

Our approach has two major advantages over exploitive strategies that construct large opponent models in real-time:

1. Firstly, our approach relies on opponent *type* classification and not on the construction of a fully-fledged, real-time opponent model. As such, the time required to collect data

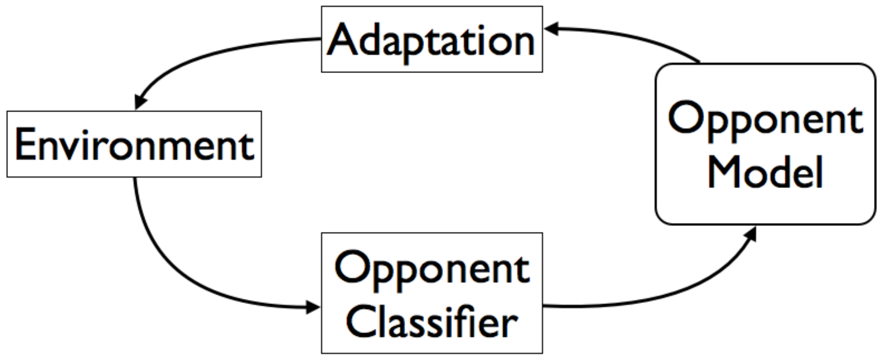

Figure 5.1: Opponent modelling guided adaptation overview

to classify an opponent is substantially less than the time required to build an accurate opponent model from scratch.

2. Second, exploitive strategies whose playing decisions rely on detailed opponent models require an expensive exploration phase during which sparsely populated opponent models can negatively impact performance. On the other hand, *seeding* a game playing agent with a robust, precomputed strategy, avoids the performance degradation that can take place when little is known about an opponent.

5.2. Alternative Approaches

Alternative approaches that perform explicit opponent modelling construct exploitive strategies by starting with an empty opponent model and populating the model in real-time during game play. Two such efforts that use this approach are described in [19] and [102]. Both approaches use imperfect information game tree search in order to construct adaptive and exploitive strategies. They build specific models about an opponent and use these to inform exploitive betting actions. A disadvantage of this approach is that the opponent models required are typically very large and constructing the opponent model can initially take some time, which will negatively impact performance when little is known about the current opponent. The work described in this chapter overcomes this problem by effectively seeding the agent with a precomputed, solid strategy and later applying adaptation to this strategy once enough information has been gathered about the current opponent's playing style. Moreover, a fully-fledged opponent model is not required to be constructed at runtime. Instead, all that is required is opponent *type* information, captured by a collection of summary statistics. Gathering this information at runtime is quick, resulting in rapid classification, which allows exploitive adaptation to take effect earlier.

Adaptation also plays an important role in the area of case-based planning [49]. Instead of constructing plans from scratch, case-based planning retrieves previously stored plans based on how similar they are to the current environment. Both case-based planning and case-based plan adaptation [74] are areas of research that have received considerable attention. Case-based plan adaptation has proved successful in adversarial, gaming environments, such as real-time strategy games [112, 80], where rather than simply re-using a previously stored plan, retrieved plans are modified to better suit the environment an agent currently finds itself in. The adaptation that we perform in this work differs from that of case-based plan adaptation as we do not rely on plans to determine an agent's actions. Moreover, case-based plan adaptation typically does not consider the specific type of opponent when determining how to adapt plans, instead adaptation is affected only by differences between the present environment and similar past environments. In this work the adaptation performed varies in response to the specific opponent type encountered.

5.3. Adaptation

Within a case-based strategy, case features are used to capture the current state of the environment and a case's solution dictates which actions are appropriate to apply, given the current state. Within our experimental domain of computer poker, case solutions are represented by probabilistic action vectors that specify the proportion of the time to select a particular action.

Adaptation is applied directly to the case's solution vector. Adapting a solution vector simply consists of shifting probabilities from one action to another. By shifting probability values, solution vectors can be made more aggressive (*aggressify*) or more defensive (*defensify*). A vector is made more aggressive by reducing the probability of playing defensive actions, such as checking or calling, and increasing the probability of more aggressive actions, such as betting or raising. Consider the following example that *aggressify's* an initial probability vector by 10%. The vector involves three possible actions (*fold*, *call*, or *raise*).

$$(0.1, 0.4, 0.5) \xrightarrow{aggressify\{0.1\}} (0.1, 0.36, 0.54)$$

In the example above, 10% of the call portion of the vector has been shifted over to the *raise* portion, resulting in a strategy that will act more aggressively.

Solution vectors can also be made more defensive, as in the following example:

$$(0.1, 0.4, 0.5) \xrightarrow{defensify\{0.1\}} (0.1, 0.45, 0.45)$$

here the probability of performing an aggressive action is lessened in favour of performing a more defensive check or call action.

5.3.1. Offline Opponent Type Model Construction

Given that we have a straightforward procedure for adapting probabilistic solution vectors, we now describe the construction of an *offline opponent type model* that employs the adaptation procedure.

5.3.2. Aggression Response Trends

The adaptation procedure described above allows an existing solid strategy to lay the foundation for an eventual exploitive strategy. While the details of *how* the adaptation is performed are straightforward, it is less obvious exactly *what* adaptation should occur, i.e. should the action vector be *aggressified* or *defensified*, and if so, by how much? In order to answer these questions it was necessary to construct an initial model of what impact adaptation has on performance against a collection of opponents with diverse playing styles. We refer to the set of opponents used to construct the offline model as the *known* set of opponents.

Figure 5.2 shows how performance can be altered against certain opponent types by systematically varying the type and amount of adaptation applied to solution vectors. We refer to the performance trends produced in Figure 5.2 as *aggression trends*. The aggression trends capture important information about the effect adaptation has on strategy performance against particular types of players. The aggression trends in Figure 5.2 were derived by blindly adapting action vectors, i.e. by selecting a single adaptation factor and holding it constant. Adaptation factors within the spectrum of -0.1 to +0.1 were selected, where a negative sign represents *defensifying* an action vector and a positive sign indicates *aggressifying* the vector. In other words, the leftmost point in each figure depicts the result of *defensifying* the action vectors by 10% and the right most point depicts the result of *aggressifying* vectors by 10% where all adaptation factors in between vary by 1%. *Null adaptation* (i.e. an adaptation factor of 0) is represented by the middle point in the figures. The aggression trends shown in Figure 5.2 highlight the fact that different adaptation strategies are required against different types of opponents. In particular, the leftmost trend in Figure 5.2 depicts a response against a particular type of player that is inclined towards increased aggression. The middle trend depicts a situation where it is not appropriate to adapt solution vectors at all against this opponent as *null adaptation* performs best. Finally, the rightmost trend depicts the opposite situation to the first trend, where it is appropriate against this opponent to further *defensify* solution vectors.

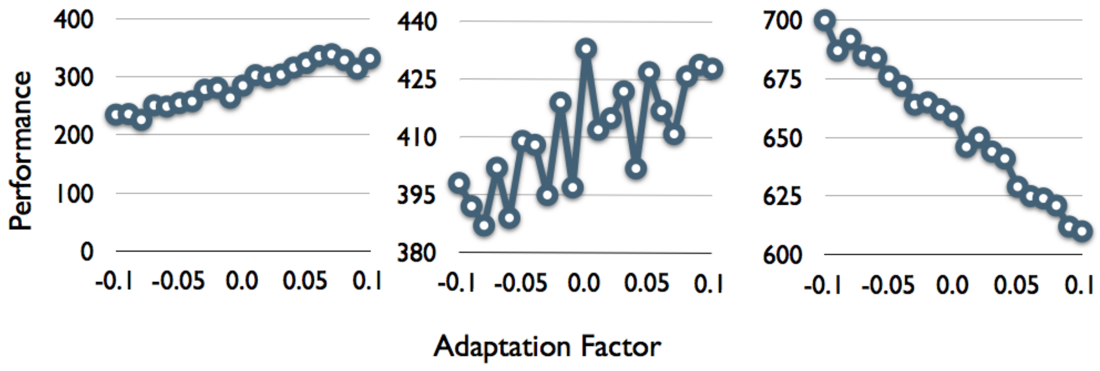

Figure 5.2: A subset of aggression trends computed during an initial training phase. Adaptation was performed within the spectrum of -0.1 to +0.1, with stepped gradations of 0.01 in between.

5.3.3. Opponent Type

Obtaining knowledge of *aggression response trends* associated with particular players is beneficial in a number of respects. First of all, we have evidence that the adaptation procedure described previously can have a beneficial impact on overall performance. Secondly, when challenging the same opponent again we can expect to improve overall performance by adapting solution vectors according to the resultant aggression trend. However, we would be severely restricted if we could only ever use the computed aggression trends against the exact same opponent. Instead, we wish to be able to use the information we have gathered to exploit further opponents that have never been encountered before. In order to do this we need to associate each aggression trend not with a particular opponent, but with a particular *type* of opponent.

There are various ways opponents could be classified to belong to a certain type. One approach that human online poker players employ to classify opponents is to keep track of statistical information about an opponent, such as the amount of money they voluntarily commit to the pot, the proportion of times they perform the *raise* action in a certain round, or their overall aggression factor. This information attempts to capture the general flavour of an opponent's style of play, such as how aggressive or defensive their strategy is. By recording action frequency information about an opponent, salient information can be captured about their particular style of play.

Within our current model, we capture information about *opponent types* by recording (for each of the four betting rounds) frequency information about *fold*, *check*, *call* and *bet* actions that were taken by the opponent. Therefore, in total each *opponent type* is described by a vector

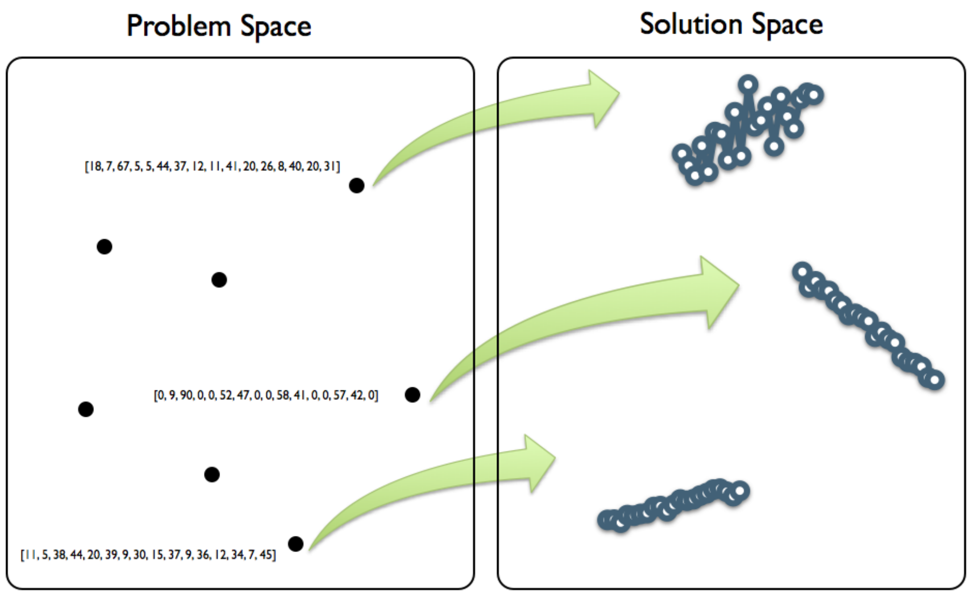

Figure 5.3: Problem-Solution space mapping for opponent types.

of 16 numerical values. By representing opponent types in this way we are able to perform similarity assessment between opponent types using Manhattan distance.

Taken together these opponent type action frequencies and their corresponding aggression trends compose an *offline opponent type model* that can be used to adapt solution vectors when challenging novel opponents by computing similarity between opponent type vectors. Figure 5.3 illustrates the idea of a mapping between opponent types (the problem space) and their corresponding aggression trends (solution space) that can be used to adapt vectors against that type of opponent.

5.4. Online Adaptation

5.4.1. Adapted Strategies

Once the *opponent type* model has been constructed we use the information contained within the model to inform how solution vectors should be adapted. Equation 5.1 uses information from the *opponent type* model to probabilistically select an appropriate adaptation factor for each hand of play:

$$\forall_{i \in 1 \ldots N},$$
$$P(x_i) = \frac{max\{outcome(x_i) - outcome(x_{pivot}), 0\}}{\sum_i^N max\{outcome(x_i) - outcome(x_{pivot}), 0\}} \tag{5.1}$$

In Equation 5.1, x_i is an adaptation factor that dictates the proportion of actions that should be *defensified/aggressified* within the original vector, $outcome(x_i)$ refers to the result of applying adaptation factor x_i against this particular type of opponent, which is retrieved from the model, $outcome(x_{pivot})$ is the result of applying *null adaptation* against this opponent type (once again retrieved from the model) and finally, N refers to the total number of adaptation factors to consider per hand, i.e. not all adaptation factors are considered for use, only a subset of the top N factors.

In other words, Equation 5.1 selects adaptation factors based on how much they were able to improve upon not applying adaptation at all, as specified by the outcome values in our model. When $N = 1$, the adaptation factor that achieved the largest improvement in outcome is selected with probability 1.0, whereas when $N = 5$, the top 5 (profitable) adaptation factors are selected probabilistically. In the case where $\forall_{i \in 1 \ldots N}, P(x_i) = 0$, null adaptation is performed.

5.4.2. Adaptation during Game Play

During game play, betting frequencies (*fold*, *check*, *call* and *bet*) for all four rounds of play are recorded about the current opponent and used to classify them as a certain *type*. However, instead of keeping track of every action the opponent has taken, only a subset of actions are recorded that affect opponent type classification. This subset includes only the latest M hands of play, where M is some specified constant[2]. This ensures that our adapted strategies can respond to recent changes in an opponent's playing style. For the first M set of hands played against an opponent no adaptation is performed and actions are selected directly from the precomputed strategy. Once M hands have been played, adaptation can occur and an appropriate *aggression trend* is selected from the model by retrieving the opponent type with the most similar bet frequency information, using Manhattan distance.

5.5. Methodology

In order to evaluate our adapted strategies we require a series of opponents, both for training the initial model and for challenging once the model has been constructed. At the 2011 AAAI

[2]Within our current implementation we use a value of $M = 300$

Computer Poker Competition a number of computer poker agents were submitted to the limit Texas Hold'em event. The agents submitted to the AAAI competition represent some of the top computer poker players in the world, so it would be beneficial to evaluate opponent type adaptation against this set of opponents. While it is not possible to challenge the original agents submitted to the competition (as none of them are publicly available), we are able to make use of the publicly available hand history information to create *expert imitators* that attempt to imitate and generalise their style of play. By making use of the expert imitation based framework described in Chapter 3 and training on the subset of decisions made by a particular *expert* agent, we constructed a set of 20 opponents that imitate each of the agents submitted to the 2011 AAAI Computer Poker Competition. These *imitation*-based agents were used within our experimental set up.

These 20 opponents were challenged against 5 adaptation strategies given by Equation 5.1, where N varied from 1 to 5, plus 1 *null-adapted* strategy that involved no adaptation. From the set of original opponents two groups were created – a *known* set of opponents and an *unknown* set. Of the 20 original opponents 75% (15 opponents) were randomly assigned to the *known* group and were used to derive the model that would inform adaptation, the remaining 25% (5 opponents) were assigned to the *unknown* group that were left out of the model construction.

Results were gathered against both the *known* and *unknown* set of opponents, where all 5 adapted strategies and the null-adapted strategy played 10 seeded duplicate matches against each opponent. Each duplicate match consisted of 6000 hands in total, where 3000 hands are initially played, after which players switch seats and the same 3000 hands are played again, so that each player receives the cards their opponent would have received before. This type of duplicate match set up was used to decrease variance. In order to further decrease overall variance, common seed values were used for duplicate matches so that each separate opponent match-up that occurred resulted in the same set of cards being dealt between players.

In the first experiment, adapted strategies were challenged against the *known* set of opponents. This represents a situation where previous knowledge about an opponent can inform present playing decisions and is akin to a scenario where a human poker player challenges a known opponent who they may have challenged before. Both human and artificial poker players are also required to challenge opponents they have never encountered before. The second experiment, involving the *unknown* set of opponents, represents this scenario, where no aggression trend or previous bet frequency information is known about the opponent and instead adaptation must occur by comparing how similar a novel opponent is to a previously *known* opponent style.

5.6. Experimental Results

Table 5.1 presents the results against the first set of known opponents (i.e. opponents whose exact aggression trends have been computed). The outcome and 95% confidence interval of each match is depicted in milli-big blinds per hand from the perspective of the row player (i.e. NoAdapt, Adapt1, Adapt2, . . ., Adapt5). Milli-big blinds record the average number of big blinds won per hand, multiplied by 1000. The opponent challenged is given as the column heading and the table is split into two sections (due to spatial limitations). The outcomes presented in the table are colour-coded as follows: grey cells represent the outcome against an opponent when no adaptation was performed. Darkly shaded cells indicate statistically significant improvements (green) or degradations (red), after applying adaptation. Lightly shaded cells indicate the observed outcome, after adapting case solutions, was statistically insignificant (once again light green for improvement and light red for degradation in performance).

Table 5.2 presents the results against the second set of unknown opponents (i.e. opponents where no aggression trend information is previously known). The same colour-coding scheme is used.

5.7. Discussion

A large proportion of Table 5.1 consists of darkly shaded green cells, indicating significant improvement in performance after solution adaptation. On average (the bottom right column in Table 5.1), all adaptation strategies ($N = 1 \ldots 5$) were able to improve on the performance of no adaptation and in all but one of these cases ($N = 2$) the improvement witnessed was statistically significant with 95% confidence. The results suggest that the adaptation information recorded within the opponent type model is appropriate and can be used to improve overall performance against a set of opponents that have been previously challenged.

In Table 5.2, less of the squares are darkly shaded green than in Table 5.1, however there is still more improvement (both significant and insignificant) witnessed than there is degradation in performance. The results in Table 5.2 show that even though no information was previously known about this set of opponents, our approach still results in an improvement in performance for all adaptation strategies ($N = 1 \ldots 5$) and in two of those cases ($N = 4$ and $N = 5$) the outcome is a statistically significant improvement in performance with 95% confidence. These results suggest that our approach has made an appropriate determination in regard to the adaptation to apply when challenging a novel opponent, given the similarity of the opponent to a known opponent.

	AAIMontybot	**Calvin**	**Feste**	**GBR**	**GGValuta**	**Hyperborean-tbr**	**POMPEIA**	**Patience**
NoAdapt	312 ± 4	548 ± 4	42 ± 2	433 ± 4	80 ± 2	25 ± 2	659 ± 3	56 ± 3
Adapt1	312 ± 3	553 ± 5	32 ± 2	429 ± 4	82 ± 2	37 ± 1	698 ± 4	57 ± 3
Adapt2	309 ± 4	549 ± 4	40 ± 3	417 ± 4	72 ± 2	28 ± 2	696 ± 3	41 ± 2
Adapt3	314 ± 4	552 ± 4	42 ± 3	421 ± 2	78 ± 3	31 ± 3	690 ± 3	57 ± 4
Adapt4	316 ± 3	566 ± 5	36 ± 2	412 ± 4	75 ± 3	26 ± 3	688 ± 2	44 ± 2
Adapt5	319 ± 4	575 ± 6	35 ± 2	409 ± 2	76 ± 3	21 ± 1	690 ± 3	49 ± 3
	RobotBot	**Slumbot**	**TellBot**	**Tiltnet**	**Tiltnet-Adaptive**	**Zbot**	**player-zeta**	**Average**
NoAdapt	285 ± 3	17 ± 3	886 ± 4	696 ± 4	673 ± 5	28 ± 2	593 ± 3	355 ± 3
Adapt1	324 ± 4	29 ± 2	900 ± 4	716 ± 4	682 ± 3	47 ± 3	596 ± 3	366 ± 3
Adapt2	330 ± 3	23 ± 4	888 ± 4	716 ± 4	674 ± 3	36 ± 2	591 ± 3	361 ± 3
Adapt3	327 ± 4	28 ± 3	892 ± 4	718 ± 2	684 ± 4	50 ± 2	590 ± 4	365 ± 3
Adapt4	326 ± 4	21 ± 2	897 ± 4	710 ± 2	705 ± 3	44 ± 2	590 ± 3	364 ± 3
Adapt5	334 ± 4	18 ± 2	888 ± 4	721 ± 4	680 ± 3	57 ± 1	600 ± 4	365 ± 3

Table 5.1: Opponent type adaptation results against first set of known opponents.

Table 5.2: Opponent type adaptation results against second unknown set of opponents.

	Entropy	Hyperborean-iro	2Bot	LittleRock	Calamari	Average
NoAdapt	436 ± 4	25 ± 2	61 ± 3	6 ± 2	-4 ± 2	105 ± 2
Adapt1	456 ± 5	25 ± 3	65 ± 3	-4 ± 3	-1 ± 1	108 ± 3
Adapt2	454 ± 4	21 ± 2	61 ± 2	-5 ± 2	1 ± 2	106 ± 2
Adapt3	448 ± 4	20 ± 2	56 ± 3	9 ± 3	8 ± 2	108 ± 3
Adapt4	463 ± 4	23 ± 2	57 ± 3	18 ± 2	1 ± 2	112 ± 2
Adapt5	462 ± 4	23 ± 1	70 ± 3	2 ± 2	-4 ± 2	111 ± 3

5.8. Concluding Remarks

In this chapter, we presented an augmentation procedure that extends the basic frameworks introduced in Chapter 3. We detailed the augmented procedure for adapting case-based strategies within adversarial environments. Rather than adapting case solutions based on differences between environmental states our adaptation procedure takes into account information about opponent types and adapts solutions in an attempt to exploit an opponent currently being challenged. This was achieved by first constructing an offline opponent type model that recorded appropriate aggression trends in response to a set of opponent types. The information captured within this model was later extrapolated during game play in order to inform probabilistic adaptation strategies. Our experimental results show that altering a precomputed strategy via adaptation was able to successfully improve overall performance, compared to not adapting the precomputed strategy. Moreover, the adaptation strategies produced were shown to be successful against both known and unknown opponents. In the next chapter we introduce the third and final extension to the frameworks of Chapter 3. Our final augmentation procedure employs transfer learning to leverage information recorded in case-bases from separate domains in order to improve overall performance.

Chapter 6

Strategy Switching via Case-Based Transfer Learning

[1]Chapter 4 introduced an augmentation procedure that made playing decisions by selecting cases from multiple case-bases and described procedures for combining those decisions. In Chapter 4, all cases-bases that could influence the final playing decision were from the same poker domain. When multiple case-bases exist from separate poker domains, such as limit/no-limit domains or single/multiple opponent domains, the question arises whether we can make use of the information captured from one domain and apply it within a separate domain which it was not originally intended for. In order to do so, we require a suitable mapping between domains. This chapter describes possible mappings that allow this type of transfer learning [84] to take place. We introduce the concept of *strategy switching*, which identifies moments during game-play when it may be useful to leverage information within case-bases from alternative domains. Figure 6.1 extends the architecture initially introduced in Figure 3.1 from Chapter 3. Distinct case-bases are constructed by recording decisions observed within different game domains. An inter-domain mapping is required to bridge the gap between these distinct case-bases. Given a reasonable mapping, information collected from one domain may be leveraged by a case-based strategy that makes playing decisions in a totally separate environment.

[1]The contents of this chapter have been modified from a short paper that was presented at the First Computer Poker Symposium at AAAI 2012. Jonathan Rubin & Ian Watson. Sartre3P: A Case-Based Multiplayer Poker Agent. In the *Twenty-Sixth AAAI Conference on Artificial Intelligence, Computer Poker Symposium, 2012* [98].

6.1. Strategy Switching

In this thesis we have investigated three poker sub-domains as experimental test-beds: two-player limit; two-player no-limit and three-player limit. Each domain produced one or more case-bases. None of the augmentation procedures from previous chapters attempted to search case-bases from multiple, separate poker domains. In this chapter, we investigate the use of transfer learning between the two-player limit and multi-player limit domains, i.e. we determine how we can we leverage information recorded in a two-player case-base to make better decisions in a multi-player domain. In the three-player domain, when one opponent *folds* it would be useful if the case-based strategies, previously developed for heads-up play, could be used to make a betting decision. Consider the following pseudocode, where s refers to the choice of a particular strategy:

```
$s \leftarrow \emptyset$
**if** fold occurred **then**
  **if** $\exists$ inter-domain mapping **then**
    $s \leftarrow$ *heads-up* strategy
  **else**
    $s \leftarrow$ *multi-player* strategy
  **end if**
**else**
  $s \leftarrow$ *multi-player* strategy
**end if**
```

The pseudocode above requires a mapping between two separate domains in order to allow a heads-up strategy to be applied within a multi-player environment. One way that this can be achieved is to develop a similarity metric that is able to gauge how similar a game state encountered in a multi-player domain is, compared to a corresponding state in heads-up play. Given our case representation for heads-up strategies, presented in Chapter 3 Table 3.4, we require a suitable inter-domain mapping for the *betting sequence* attribute.

6.1.1. Inter-Domain Mapping

A mapping can occur between domains as long as one player has folded in the three-player domain. There are various ways an inter-domain mapping can take place between a three-player betting sequence and a two-player betting sequence. To determine how similar a three-player betting sequence is to a two-player betting sequence, we need to first map the actions of the two remaining players in the three-player domain to actions of an appropriate player in

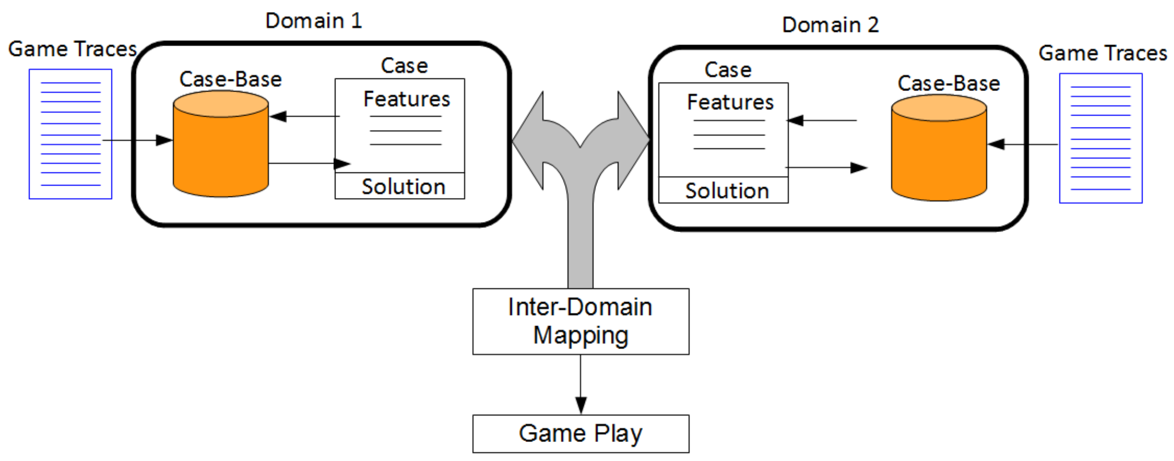

Figure 6.1: Case-based transfer learning architecture extension.

the two-player domain. An appropriate player is selected by considering the order in which the players act. We refer to this as an inter-domain *player mapping*.

We can determine how similar two betting sequences from separate domains are by counting the total number of *bet/raise* decisions taken by an active player in the three-player domain and comparing this with the total number of *bet/raise* decisions made by the corresponding player in the two player domain, as specified by the *player mapping*. This ensures that the mapping between domains retains the strength that each player exhibited by betting or raising.

Fig. 6.2 illustrates a possible *player mapping* that takes place when a *fold* has occurred during the preflop round of play. The abbreviations in Fig. 6.2 stand for *Dealer* (D), *Non Dealer* (ND), *Small Blind* (SB) and *Big Blind* (BB). The connections between the two domains (i.e. the red arrows) specifies the *player mapping* and are further explained below.

1. $D \rightarrow D$: As the *Dealer* is the last player to act postflop (in both domains) the total number of preflop raises made by D in the three-player domain, must match the total number of preflop raises made by D in the two-player domain for the inter-domain sequences to be considered similar.

2. $SB \rightarrow ND$: As the SB is the first player to act postflop (in the three-player domain) and the ND is the first player to act postflop (in the two-player domain), the total number of preflop raises made by the SB must match the total number of preflop raises made by the ND for the inter-domain sequences to be considered similar.

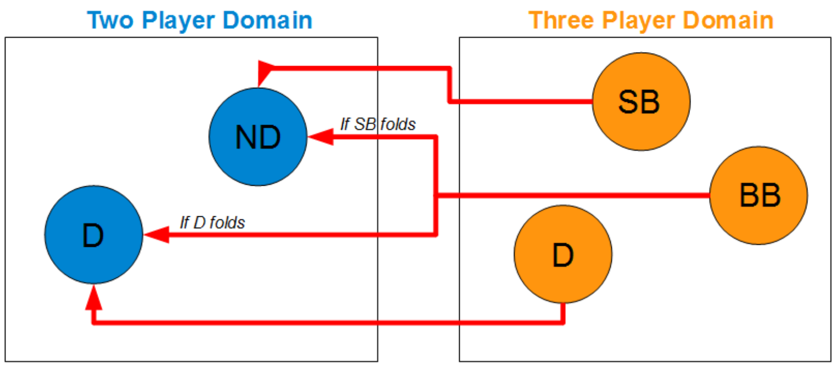

Figure 6.2: Demonstrates how inter-domain mapping is performed for betting sequences when a fold takes place on the first betting round in the three player domain. Each node represents a particular player position within their domain.

3. $BB \rightarrow D/ND$: For the final mapping, the order in which the BB acts depends on which player folded in the three-player domain (SB or D). If the SB folded, the BB will act first postflop and hence, the total number of preflop raises made by the BB must match the total number of preflop raises made by the ND. On the other hand, if the D was the player that folded, the BB will act last postflop and hence, the total number of preflop raises made by the BB must match the total number of preflop raises made by the D instead.

The above mapping works by ensuring a two-player betting sequence exists where the total number of preflop raise actions made by each player are equal to the total preflop raise actions made by the two remaining players in the three-player domain.

6.2. Inter-Domain Mapping Example

Here we provide a concrete example that illustrates the result of this inter-domain mapping.

Consider the three-player betting sequence that takes place on the flop:

$$frc - r$$

The preflop action proceeds as follows: D folds, the SB raises and the BB calls. As a fold has occurred we can apply inter-domain mapping. The *player mapping* introduced in Fig. 6.2 tells us that:

$$SB \rightarrow ND$$
$$BB \rightarrow D$$

Hence, we require a two-player betting sequence that involves a single raise by the ND in the first round. The only legal two-player betting sequence that is appropriate, is as follows:

$$crc - r$$

Where, D checks, the ND raises and the D calls. As a sufficiently similar sequence has been found, future betting decisions would be based on the above two-player sequence.

Notice that the two-player betting sequence:

$$rc - r$$

which looks similar to the original three-player sequence ($frc - r$), would not be considered similar via the inter-domain mapping. While this sequence looks similar to the three-player sequence above, it nevertheless confuses the order in which bets/raises were made and would not accurately capture the important strength information that is captured by the mapping we have described.

6.3. Methodology

We evaluate the effect of *strategy switching* by comparatively evaluating *non-switching* strategies with their *switching* counterparts in the domain of three-player, limit Texas Hold'em. The metric used to determine similarity between *betting sequences* differs depending on whether *strategy switching* is enabled or not.

Without Switching When *strategy switching* is not enabled a three-player case-base will be searched that contains three-player betting sequences only.

With Switching When *strategy switching* is enabled an inter-domain similarity metric is required that is able to determine similarity between the current three-player betting sequence and the two-player sequences contained within the case-base. We use the metric described in section 6.1.1..

We produced two sets of case-based strategies (one with *switching*, and one without) in order to determine the affect that strategy switching had on performance. Both strategies were challenged against the two multi-player, limit agents: *dpp* and *akuma*, which have been made publicly available by the Knowledge Engineering Group at Technische Universität Darmstadt [27]. The agent *dpp* is described as a "mathematically fair bot" that does no opponent modelling when making decisions, whereas *akuma* uses Monte-Carlo simulation with opponent modelling.

Both *non-switching* and *switching* case-based strategies were challenged against *dpp* and *akuma* for a total of 10 matches, where each match played consisted of 1000 duplicate hands. Recall from Section 1.2.2., that a three-player duplicate match involves six seat enumerations in order for each agent to experience the same game scenarios as their opponents, therefore each match involved playing 6000 hands. Finally, each of the 10 duplicate matches were seeded so that the scenarios encountered by the *non-switching* strategy were the same as those encountered by the *switching* strategy. This reduces the influence that the stochastic nature of the game has on the results.

6.4. Experimental Results

The final results are presented in small bets won per hand with a 95% confidence interval in Table 6.1. Results are from the perspective of the row player. In the first column our player, CaseBased3P, didn't use switching. The second column presents results when switching was enabled. Against akuma and dpp, our agent achieved a win rate of 0.2101 sb/h using a non-switching strategy. When strategy switching was employed, our agent improved this win rate to 0.2218 sb/h – an increase of 0.0117 sb/h. For both sets of matches, akuma also achieved a positive win rate. When challenging dpp and our non-switching strategy, akuma's win rate was 0.0898 sb/h and when challenging dpp and our switching agent this dropped slightly to 0.0875. For both sets of matches, dpp was not able to achieve a profit.

We submitted our multi-player agent, which employed strategy switching, to the 2011 AAAI Computer Poker Competition. Our entry to the 2011 multi-player limit competition was Sartre3P. Table 6.2 reproduces the results of this competition. Out of a total of 9 competitors, Sartre3P placed second in the instant runoff division and won the total bankroll division of the competition, achieving a win rate of 0.266 sb/h. Table 6.2 shows that Sartre3P wins this competition by a substantial margin. In particular, Sartre3P achieves 0.095 sb/h more than the second place finisher Hyperborean.

Table 6.1: Multi-player case-based strategy results with and without strategy switching.

	Non-Switching	*Switching*
CaseBased3P	0.2101 ± 0.004	0.2218 ± 0.004
akuma	0.0898 ± 0.003	0.0875 ± 0.004
dpp	-0.2995 ± 0.004	-0.3093 ± 0.004

Table 6.2: 2011 multi-player limit bankroll and runoff results.

Multi-player Bankroll Runoff	Multi-player Total Bankroll	Average Won/Lost
1. Hyperborean-3p-iro	1. **Sartre3P**	0.266 ± 0.024
2. **Sartre3P**	2. Hyperborean-3p-tbr	0.171 ± 0.023
3. LittleRock	3. AAIMontybot	0.130 ± 0.045
4. dcubot3plr	3. LittleRock	0.122 ± 0.022
5. Bnold3	5. OwnBot	0.016 ± 0.035
6. AAIMontybot	6. Bnold3	-0.084 ± 0.028
7. OwnBot	7. Entropy	-0.099 ± 0.043
8. Entropy	8. player.zeta.3p	-0.521 ± 0.040
9. player.zeta.3p		

6.5. Discussion

The results presented in Table 6.1 and 6.2 provide further support regarding the efficacy of top-down, case-based strategies when compared with agents that employ alternative approaches. Results from Table 6.1 show that extending the basic case-based frameworks introduced in Chapter 3 leads to performance improvements. Firstly, notice that the case-based strategies produced (whether *switching* or *non-switching*) are able to beat both opponents at a significant level. Second, the results show that multi-player case-based strategies *with switching* are able to achieve a greater overall profit than strategies that don't switch. The win rate difference observed is significant with 95% confidence. As long as the mapping does not destroy the information about the current game state, we are able to make use of the more detailed heads-up strategies, which positively affects performance.

Results from the 2011 ACPC in Table 6.2 show that the combination of case-based strategies with strategy switching can produce very strong play. In this competition our top-down, case-based strategy outperforms other agents that have been constructed via approaches such as counterfactual regret minimisation and Monte-Carlo simulation. Here, Sartre3P convincingly wins the bankroll competition, achieving a greater bankroll than all other competitors by a large margin. While not directly measured, we believe strategy switching has a large part to play in this win rate.

6.6. Concluding Remarks

Overall, these results suggest that the inter-domain mapping we have defined is appropriate and can improve performance by making use of strategies that have previously been learned in a separate poker domain. Moreover, the result presented in this chapter demonstrate that augmented case-based strategies actually have the capability to defeat their *bottom-up* counterparts. In particular, Sartre3P was trained on hand history data containing the decisions of the Hyperborean.tbr agent from the 2010 three-player, limit competition, yet the results from the 2011 competition show that, for this particular pool of opponents, the strategy used by Sartre3P, together with *strategy switching*, was able to outperform the 2011 version of Hyperborean.tbr. While the differences between the 2010 and the 2011 versions of Hyperborean.tbr are not known, it is reasonable to expect that the quality of the 2011 version of Hyperborean.tbr is more likely to have improved (rather than degraded) over the year.

Chapter 7

Conclusions and Future Work

7.1. Conclusions

The focus of this thesis was on the construction, empirical evaluation and analysis of strategies that employ the case-based reasoning methodology in the domain of computer poker. This domain was chosen as a testbed for case-based strategies as it is a large, complex environment that exhibits many real-world characteristics such as the requirement to handle imperfect information and chance events, the ability to deal with one or more competing adversaries, as well as a necessity to handle deception. These types of domain characteristics pose challenging issues, which are not typically addressed by many case-based reasoning systems.

Within this challenging domain, we constructed top-down, case-based strategies that were trained on hand history logs recorded from previous poker matches. The resulting agents were able to play at a world-class level and were shown, in some cases, to outperform alternative state-of-the-art approaches in computer poker,such as counterfactual regret minimisation, Monte-Carlo simulation, imperfect information game tree search and rule-based systems.

In this thesis, we have focused on three poker domains:

1. Heads-Up, Limit Texas Hold'em
2. Heads-Up, No-Limit Texas Hold'em, and
3. Multi-player, Limit Texas Hold'em

We presented frameworks for the construction of case-based strategies in these three subdomains. The frameworks were the result of a period of iterative maintenance and improvement. During this period of maintenance, issues were addressed to do with case representation, similarity assessment and overall system architecture. Where possible, empirical evaluations were conducted to confirm the efficacy of the decisions made. Changes introduced into the architecture were empirically evaluated and modifications that resulted in a positive effect on performance were incorporated into the final framework. A performance trajectory of the strategies produced using this approach was presented by evaluating our case-based agent, Sartre, against some of the best computer poker agents in the world between 2009 to 2012 at the Annual Computer Poker Competition. Overall, the results presented indicate that the use of the case-based reasoning methodology within computer poker's imperfect information environment is reasonable. Our results provide support that CBR principles and techniques can be employed to handle complex domain characteristics. The same issues and challenges that we encountered and solved would need to be dealt with in order to apply CBR to related domains, such as trade markets, combat environments, politics or any environment that involves complex negotiations between multiple competing parties.

The frameworks introduced in this thesis used expert imitation to produce *top-down, case-based* strategies. Employing this basic framework produced strong, sophisticated agents that were able to perform well against both computerised opponents and human players. A reasonable criticism of top-down, case-based strategies constructed via expert imitation is that the agents produced will only be as good as the original expert used to train the system. Furthermore, the strategies produced and evaluated, by the basic frameworks, were all static strategies that never varied their play. These strategies performed no opponent modelling, whereby information about an opponent is collected and used to affect decision making. To overcome the above limitations, we introduced the concept of an *augmentation procedure*, where the basic framework was extended in various ways to allow further functionality and to improve overall performance.

The first augmentation procedure used *implicit opponent modelling*. This involved choosing an appropriate strategy to use against an opponent, by selecting from a set of static strategies at runtime. Multiple static strategies were constructed using our basic framework. This resulted in multiple case-bases, each of which were trained on separate sets of data and imitated a different expert's style of play. Static strategies were selected based upon how well they performed against the opponent. An adapted UCB1 allocation policy was employed to handle the exploration/exploitation tradeoff. Using this augmentation procedure, no direct modelling of the opponent took place, i.e. it was not necessary to decipher the details of an opponent's strategy. Instead, opponent modelling took place *implicitly* by recording the profit or loss associated with each case-base. By employing this augmentation procedure, dynamic strategies that change over time were produced, as opposed to the non-waiving, static strategies that result from the basic framework. We presented results in the domains of heads-up limit and heads-up no-limit Texas Hold'em. Overall, improvements were witnessed in the limit domain, but results were mixed in the more complicated no-limit domain.

The strategies produced by our first augmentation procedure were not *exploitive* strategies, as they did not recognise opponent weaknesses and capitilise on those weaknesses. The second augmentation procedure produced both *dynamic* and *exploitive* strategies. Our second augmentation procedure extended the basic framework to allow *explicit opponent modelling* to take place. Details of an opponent's strategy were captured and used to dynamically alter the decisions made by our own case-based strategies. Instead of constructing fully-fledged models of individual opponents, which can take a considerable amount of time, our augmentation procedure classified opponent types using a feature vector representation based on bet frequency information. Using this approach, we were able to quickly assess the similarity of the current opponent to opponents that had previously been encountered. Adaptation strategies which had been successful against previous similar opponents, were reused against the current opponent to directly alter the probability distribution of playing actions. We evaluated a range of adaptation strategies against a set of both previously encountered and completely novel opponents. In both cases, the results produced showed improvement in performance compared to the basic framework that used no opponent modelling.

Overall, extending the basic framework with either *implicit* or *explicit* opponent modelling tended to result in improvements in performance, compared to simply employing top-down, case-based strategies produced by the basic framework. Our third and final augmentation procedure performed no opponent modelling, but instead used *transfer learning* in order to leverage information contained in case-bases from separate poker domains. We introduced a player mapping that allowed similarity to be determined between cases from separate domains. We showed that by specifying a reasonable inter-domain mapping, it was possible to perform *strategy switching*, whereby cases recorded in one domain were able to be applied in a separate domain. We evaluated this augmentation procedure by leveraging cases from the two-player, limit domain within the multi-player, limit domain. As the two-player domain case-base contained a larger set of more fine-grained cases, this allowed better decisions to be made when relevant heads-up scenarios were encountered in the multi-player environment. This resulted in large improvements in performance as witnessed via a comparative evaluation between *switching* and *non-switching* strategies.

In conclusion, the limitations of *top-down*, *case-based* strategies have been overcome by extending the basic frameworks with *augmentation procedures*. The augmentation procedures described in this thesis gather supplementary information, which they make use of to produce both *dynamic* and *exploitive* case-based strategies. By introducing augmentation procedures, we have addressed the criticism of *expert imitation* that restricts the performance of the strategies produced to the level of the original expert used to train the system. We have presented evidence that it is possible to overcome this limitation. In particular, consider the results of the

2011 AAAI computer poker competition, where our agent (Sartre3P) won the bankroll division of the multi-player, limit Texas Hold'em competition. Here, Sartre3P constructed its case-base by imitating the decisions of the best agent from the previous year's competition, i.e. Hyperborean. By augmenting the static strategy produced, with *strategy switching*, Sartre3P was able to outperform all competitors in the 2011 competition, by a large margin, including the latest version of Hyperborean.

Throughout this thesis, the emphasis has been on the performance of the case-based strategies produced, with the goal of constructing strong strategies that are able to play at an advanced level of skill. The results presented provide strong support for the efficacy of case-based strategies that have been trained on hand history data and results from the annual computer poker competition confirm that our case-based strategies do reach a level of sophisticated play. This focus on performance has inevitably resulted in some specific tailoring of the frameworks to the computer poker domain. However, the architecture of the augmentation procedures we have introduced are all reusable and are able to be applied in other gaming environments, as well as real-world domains that involve imperfect information and competing adversaries.

7.2. Future Work

We have identified several possible areas of future work. The first area of future work involves the construction and evaluation of case-based strategies in more complex poker sub-domains. The contents of this thesis focused entirely on the domains of heads-up, limit; heads-up, no-limit; and three-player limit Texas Hold'em. These domains were selected as they have been offered as competitions at the annual computer poker competition since 2009. The next obvious poker sub-domain to explore would be three-player, no-limit Texas Hold'em. The combination of both multiple opponents and a no-limit betting structure makes this a challenging domain for computerised poker agents.

In addition to multi-player, no-limit Texas Hold'em, an emphasis on tournament play should also be considered. In a tournament, players are continually eliminated and play proceeds until only one player holds all the chips. Players are required to deal with fluctuating chip stacks and increasing blind levels. This is the type of play used to determine a world champion at the annual World Series of Poker. All domains considered in this thesis were cash game environments, where infinite bankrolls and fixed blind levels (i.e. where blind values do not increase over time) were assumed. Fixing these domain characteristics limits the stochasticity of the environment and allows a more accurate assessment of a strategy's quality. While varying blind and chip stack levels introduce further complexity and noise into the environment, if the goal of computer poker researchers is to construct world champion-level poker agents, then tournament play needs to be taken into account.

As increasingly more complex domains are considered, a related area of future work has to do with the applicability of transfer learning between those domains. In our current work, we evaluated transfer learning between heads-up limit and the three-player limit domain. Further investigation would be required to determine if leveraging cases between domains still holds, as the gap between the number of players associated with those domains widens. Furthermore, transfer learning between no-limit domains would be worthy of investigation, where more complex betting sequences pose an added difficulty in assessing similarity between subdomains.

Finally, the work presented in this thesis largely focused on bot vs. bot play for evaluation. While some results against human competitors were presented, as recorded from our browser-based web application, a more rigorous evaluation against human opposition would further validate the efficacy of our approach. A related area of possible future work would involve more of a focus on constructing case-based agents that have been trained with real-world decisions made by expert human poker players. Most of the strategies evaluated in this thesis were constructed with artificial data from the annual computer poker competition. ACPC data was used to train our strategies due to its quality and availability. Cased-based strategies trained on real-world data-sets from human play, would require further efforts (e.g. applying DIVAT analysis) to ensure that the quality of the decisions used for training were reasonable.

Appendices

Appendix A

Annual Computer Poker Competition Cross-Tables

Below are selected crosstables of matches from the Annual Computer Poker Competition for the years 2009 – 2012. Results are from the perspective of the row player. Values are in milli big blinds per hand with a 95% confidence interval. Some agent names may have been shortened due to space considerations.

	dcurbhu	AoBot	GGValuta	GS5	GS5Dynamic	Hyperborean-BR	Hyperborean-Eqm	LIDIA	Rockhopper	Slumbot	tommybot	MANZANA	Sartre
dcurbhu	–	−66 ± 9	−208 ± 3	−241 ± 4	−117 ± 4	−230 ± 4	−208 ± 4	−145 ± 3	−217 ± 4	−218 ± 3	1006 ± 5	−269 ± 3	−217 ± 3
AoBot	66 ± 9	–	−132 ± 4	−110 ± 4	93 ± 4	−138 ± 3	−143 ± 4	73 ± 5	−134 ± 4	−100 ± 4	856 ± 7	−225 ± 10	−131 ± 6
GGValuta	208 ± 3	132 ± 4	–	51 ± 3	162 ± 4	21 ± 3	12 ± 2	182 ± 4	14 ± 2	42 ± 3	440 ± 6	7 ± 2	46 ± 3
GS5	241 ± 4	110 ± 4	−51 ± 3	–	145 ± 4	−44 ± 4	−46 ± 3	177 ± 4	−43 ± 3	−16 ± 4	552 ± 8	−47 ± 3	8 ± 3
GS5Dynamic	117 ± 4	−93 ± 4	−162 ± 4	−145 ± 4	–	−177 ± 4	−151 ± 4	−42 ± 5	−149 ± 4	−130 ± 5	−1190 ± 9	−173 ± 4	−119 ± 4
Hyperborean-BR	230 ± 4	138 ± 3	−21 ± 3	44 ± 4	177 ± 4	–	−25 ± 2	239 ± 5	−8 ± 2	24 ± 4	598 ± 6	−19 ± 2	24 ± 3
Hyperborean-Eqm	208 ± 4	143 ± 4	−12 ± 2	46 ± 3	151 ± 4	25 ± 2	–	194 ± 4	8 ± 2	27 ± 3	455 ± 7	7 ± 2	51 ± 3
LIDIA	145 ± 3	−73 ± 5	−182 ± 4	−177 ± 4	42 ± 5	−239 ± 5	−194 ± 4	–	−169 ± 4	−171 ± 4	565 ± 9	−243 ± 4	−145 ± 4
Rockhopper	217 ± 4	134 ± 4	−14 ± 2	43 ± 3	149 ± 4	8 ± 2	−8 ± 2	169 ± 4	–	27 ± 3	487 ± 6	−3 ± 2	33 ± 3
Slumbot	218 ± 3	100 ± 4	−42 ± 3	16 ± 4	130 ± 5	−24 ± 4	−27 ± 3	171 ± 4	−27 ± 3	–	656 ± 7	−29 ± 3	12 ± 3
tommybot	−1006 ± 5	−856 ± 7	−440 ± 6	−552 ± 8	1190 ± 9	−598 ± 6	−455 ± 7	−565 ± 9	−487 ± 6	−656 ± 7	–	−1205 ± 5	−765 ± 7
MANZANA	269 ± 3	225 ± 10	−7 ± 2	47 ± 3	173 ± 4	19 ± 2	−7 ± 2	243 ± 4	3 ± 2	29 ± 3	1205 ± 5	–	38 ± 2
Sartre	217 ± 3	131 ± 6	−46 ± 3	−8 ± 3	119 ± 4	−24 ± 3	−51 ± 3	145 ± 4	−33 ± 3	−12 ± 3	765 ± 7	−38 ± 2	–

2009 Heads-up Limit Texas Hold'em

	Arnold2	ASVP	GGValuta	GS6.iro	GS6.tbr	Hyperborean.iro	Hyperborean.tbr	
Arnold2	–	475 ± 8	−32 ± 5	−12 ± 6	−12 ± 6	−32 ± 5	−36 ± 5	
ASVP	−475 ± 8	–	−461 ± 9	−689 ± 8	−41 ± 18	−574 ± 9	−593 ± 8	
GGValuta	32 ± 5	461 ± 9	–	31 ± 6	31 ± 6	3 ± 6	−3 ± 5	
GS6.iro	12 ± 6	689 ± 8	−31 ± 6	–	–	−31 ± 5	−40 ± 6	
GS6.tbr	12 ± 6	41 ± 18	−31 ± 6	–	–	–	−41 ± 6	
Hyperborean.iro	32 ± 5	574 ± 9	−3 ± 6	31 ± 5	–	–	−3 ± 4	
Hyperborean.tbr	36 ± 5	593 ± 8	3 ± 5	40 ± 6	41 ± 6	3 ± 4	–	
Jester	−25 ± 6	722 ± 9	−47 ± 5	−35 ± 4	−33 ± 6	−49 ± 6	−54 ± 5	
LittleRock	−38 ± 5	420 ± 9	−77 ± 5	−47 ± 4	−52 ± 8	−70 ± 5	−69 ± 5	
longhorn	−1396 ± 12	−1836 ± 10	−1437 ± 13	−1438 ± 13	−1514 ± 11	−1427 ± 13	−1421 ± 13	
PLICAS	−191 ± 6	419 ± 17	−219 ± 7	−197 ± 6	−196 ± 7	−231 ± 6	−234 ± 7	
PULPO	39 ± 5	724 ± 8	−1 ± 5	32 ± 6	33 ± 6	−2 ± 4	−9 ± 4	
Rockhopper	36 ± 5	492 ± 8	6 ± 5	37 ± 6	38 ± 6	3 ± 4	1 ± 4	
Sartre	15 ± 5	762 ± 8	−18 ± 5	3 ± 5	5 ± 5	−15 ± 5	−23 ± 5	
Slumbot	34 ± 6	562 ± 9	−4 ± 5	34 ± 6	34 ± 6	−1 ± 4	−5 ± 4	
	Jester	**LittleRock**	**longhorn**	**PLICAS**	**PULPO**	**Rockhopper**	**Sartre**	**Slumbot**
Arnold2	25 ± 6	38 ± 5	1396 ± 12	191 ± 6	−39 ± 5	−36 ± 5	−15 ± 5	−34 ± 6
ASVP	−722 ± 9	−420 ± 9	1836 ± 10	−419 ± 17	−724 ± 8	−492 ± 8	−762 ± 8	−562 ± 9
GGValuta	47 ± 5	77 ± 5	1437 ± 13	219 ± 7	1 ± 5	−6 ± 5	18 ± 5	4 ± 5
GS6.iro	35 ± 4	47 ± 4	1438 ± 13	197 ± 6	−32 ± 6	−37 ± 6	−3 ± 5	−34 ± 6
GS6.tbr	33 ± 6	52 ± 8	1514 ± 11	196 ± 7	−33 ± 6	−38 ± 6	−5 ± 5	−34 ± 6
Hyperborean.iro	49 ± 6	70 ± 5	1427 ± 13	231 ± 6	2 ± 4	−3 ± 4	15 ± 5	1 ± 4
Hyperborean.tbr	54 ± 5	69 ± 5	1421 ± 13	234 ± 7	9 ± 4	−1 ± 4	23 ± 5	5 ± 4
Jester	–	−5 ± 5	1399 ± 13	184 ± 7	−52 ± 5	−52 ± 5	−27 ± 5	−45 ± 5
LittleRock	5 ± 5	–	1475 ± 11	91 ± 6	−125 ± 5	−77 ± 5	−72 ± 5	−69 ± 6
longhorn	−1399 ± 13	−1475 ± 11	–	−1038 ± 16	−1478 ± 16	−1437 ± 12	−1445 ± 13	−1416 ± 12
PLICAS	−184 ± 7	−91 ± 6	1038 ± 16	–	−242 ± 6	−221 ± 6	−218 ± 7	−219 ± 7
PULPO	52 ± 5	125 ± 5	1478 ± 16	242 ± 6	–	−7 ± 5	27 ± 4	−3 ± 5
Rockhopper	52 ± 5	77 ± 5	1437 ± 12	221 ± 6	7 ± 5	–	26 ± 5	5 ± 5
Sartre	27 ± 5	72 ± 5	1445 ± 13	218 ± 7	−27 ± 4	−26 ± 5	–	−21 ± 5
Slumbot	45 ± 5	69 ± 6	1416 ± 12	219 ± 7	3 ± 5	−5 ± 5	21 ± 5	–

2010 Heads-up Limit Texas Hold'em

	c4tw.iro	c4tw.tbr	Hyperborean.iro	Hyperborean.tbr	PokerBotSLO	SartreNL	Tartanian4.iro	Tartanian4.tbr
c4tw.iro	-	-	−4181 ± 161	-	−7557 ± 739	−2289 ± 110	−5534 ± 109	-
c4tw.tbr	-	-	-	−3562 ± 97	−6977 ± 729	−2241 ± 124	-	−8669 ± 168
Hyperborean.iro	4181 ± 161	-	-	−83 ± 43	775 ± 47	200 ± 39	248 ± 49	122 ± 38
Hyperborean.tbr	-	3562 ± 97	83 ± 43	-	795 ± 45	272 ± 35	364 ± 42	220 ± 39
PokerBotSLO	7557 ± 739	6977 ± 729	−775 ± 47	−795 ± 45	-	−193 ± 39	−108 ± 46	−159 ± 40
SartreNL	2289 ± 110	2241 ± 124	−200 ± 39	−272 ± 35	193 ± 39	-	42 ± 38	−13 ± 33
Tartanian4.iro	5534 ± 109	-	−248 ± 49	−364 ± 42	108 ± 46	−42 ± 38	-	−80 ± 23
Tartanian4.tbr	-	8669 ± 168	−122 ± 38	−220 ± 39	159 ± 40	13 ± 33	80 ± 23	-

2010 Heads-Up No-Limit Texas Hold'em

	2Bot	AAIMontybot	Calamari	Calvin	Entropy	Feste	GBR	GGValuta	Hyper-iro	Hyper-tbr	
2Bot	–	235 ± 18	−75 ± 8	513 ± 11	280 ± 14	−0 ± 8	269 ± 16	−13 ± 9	−73 ± 8	−65 ± 9	
AAIMontybot	−235 ± 18	–	−325 ± 19	70 ± 15	538 ± 18	−270 ± 17	27 ± 13	−219 ± 18	−288 ± 33	−266 ± 18	
Calamari	75 ± 8	325 ± 19	–	531 ± 11	454 ± 11	53 ± 7	429 ± 22	41 ± 5	−11 ± 7	−8 ± 7	
Calvin	−513 ± 11	−70 ± 15	−531 ± 11	–	53 ± 18	−495 ± 12	24 ± 12	−455 ± 9	−502 ± 12	−522 ± 12	
Entropy	−280 ± 14	−538 ± 18	−454 ± 11	−53 ± 18	–	−344 ± 14	−6 ± 18	−105 ± 13	−259 ± 13	−237 ± 12	
Feste	0 ± 8	270 ± 17	−53 ± 7	495 ± 12	344 ± 14	–	347 ± 21	−3 ± 8	−51 ± 8	−60 ± 9	
GBR	−269 ± 16	−27 ± 13	−429 ± 22	−24 ± 12	6 ± 18	−347 ± 21	–	−202 ± 14	−327 ± 36	−309 ± 18	
GGValuta	13 ± 9	219 ± 18	−41 ± 5	455 ± 9	105 ± 13	3 ± 8	202 ± 14	–	−42 ± 8	−33 ± 7	
Hyper-iro	73 ± 8	288 ± 33	11 ± 7	502 ± 12	259 ± 13	51 ± 8	327 ± 36	42 ± 8	–	–	
Hyper-tbr	65 ± 9	266 ± 18	8 ± 7	522 ± 12	237 ± 12	60 ± 9	309 ± 18	33 ± 7	–	–	
LittleRock	38 ± 8	236 ± 16	−16 ± 7	494 ± 10	131 ± 13	30 ± 8	278 ± 17	7 ± 8	−27 ± 8	−35 ± 9	
Patience	−8 ± 9	262 ± 22	−65 ± 8	500 ± 11	268 ± 16	0 ± 9	359 ± 19	−5 ± 9	−62 ± 9	−72 ± 8	
player.zeta	−427 ± 14	−201 ± 43	−585 ± 14	164 ± 21	63 ± 49	−515 ± 16	−353 ± 73	−350 ± 15	−476 ± 14	−450 ± 16	
POMPEIA	−525 ± 6	−514 ± 10	−613 ± 9	−565 ± 9	−686 ± 9	−565 ± 7	−604 ± 8	−467 ± 7	−526 ± 8	−528 ± 7	
RobotBot	−271 ± 12	177 ± 15	−276 ± 11	328 ± 11	−35 ± 17	−248 ± 10	228 ± 14	−240 ± 10	−292 ± 10	−306 ± 10	
Sartre	57 ± 8	327 ± 17	−18 ± 5	507 ± 10	470 ± 12	37 ± 7	417 ± 20	36 ± 7	−19 ± 6	−24 ± 8	
Slumbot	63 ± 10	259 ± 17	7 ± 6	490 ± 12	240 ± 14	47 ± 8	309 ± 15	39 ± 6	−6 ± 7	3 ± 8	
TellBot	−703 ± 17	−1028 ± 26	−899 ± 15	−734 ± 23	−1091 ± 33	−853 ± 19	−1359 ± 29	−367 ± 17	−580 ± 15	−541 ± 21	
Tiltnet	−573 ± 11	−446 ± 26	−716 ± 11	36 ± 17	56 ± 21	−633 ± 13	−266 ± 24	−344 ± 12	–	−500 ± 16	
TiltAdapt	−571 ± 13	−386 ± 11	−668 ± 12	213 ± 21	−156 ± 24	−618 ± 12	−200 ± 63	−357 ± 13	−511 ± 15	–	
ZBot	47 ± 9	249 ± 18	−31 ± 7	473 ± 11	274 ± 13	54 ± 8	309 ± 11	17 ± 9	−33 ± 8	−35 ± 9	

	LittleRock	Patience	player.zeta	POMPEIA	RobotBot	Sartre	Slumbot	TellBot	Tiltnet	TiltAdapt	ZBot
2Bot	−38 ± 8	8 ± 9	427 ± 14	525 ± 6	271 ± 12	−57 ± 8	−63 ± 10	703 ± 17	573 ± 11	571 ± 13	−47 ± 9
AAIMontybot	−236 ± 16	−262 ± 22	201 ± 43	514 ± 10	−177 ± 15	−327 ± 17	−259 ± 17	1028 ± 26	446 ± 26	386 ± 11	−249 ± 18
Calamari	16 ± 7	65 ± 8	585 ± 14	613 ± 9	276 ± 11	18 ± 5	−7 ± 6	899 ± 15	716 ± 11	668 ± 12	31 ± 7
Calvin	−494 ± 10	−500 ± 11	−164 ± 21	565 ± 9	−328 ± 11	−507 ± 10	−490 ± 12	734 ± 23	−36 ± 17	−213 ± 21	−473 ± 11
Entropy	−131 ± 13	−268 ± 16	−63 ± 49	686 ± 9	35 ± 17	−470 ± 12	−240 ± 14	1091 ± 33	−56 ± 21	156 ± 24	−274 ± 13
Feste	−30 ± 8	−0 ± 9	515 ± 16	565 ± 7	248 ± 10	−37 ± 7	−47 ± 8	853 ± 19	633 ± 13	618 ± 12	−54 ± 8
GBR	−278 ± 17	−359 ± 19	353 ± 73	604 ± 8	−228 ± 14	−417 ± 20	−309 ± 15	1359 ± 29	266 ± 24	200 ± 63	−309 ± 11
GGValuta	−7 ± 8	5 ± 9	350 ± 15	467 ± 7	240 ± 10	−36 ± 7	−39 ± 6	367 ± 17	344 ± 12	357 ± 13	−17 ± 9
Hyper-iro	27 ± 8	62 ± 9	476 ± 14	526 ± 8	292 ± 10	19 ± 6	6 ± 7	580 ± 15	–	511 ± 15	33 ± 8
Hyper-tbr	35 ± 9	72 ± 8	450 ± 16	528 ± 7	306 ± 10	24 ± 8	−3 ± 8	541 ± 21	500 ± 16	–	35 ± 9
LittleRock	–	45 ± 10	388 ± 12	520 ± 8	288 ± 9	6 ± 6	−27 ± 7	383 ± 17	466 ± 12	452 ± 14	−9 ± 8
Patience	−45 ± 10	–	478 ± 14	576 ± 7	260 ± 10	−48 ± 9	−66 ± 10	790 ± 17	605 ± 12	589 ± 14	−57 ± 8
player.zeta	−388 ± 12	−478 ± 14	–	722 ± 9	137 ± 24	−617 ± 16	−448 ± 16	764 ± 46	224 ± 17	383 ± 22	−396 ± 13
POMPEIA	−520 ± 8	−576 ± 7	−722 ± 9	–	−815 ± 5	−647 ± 9	−530 ± 8	35 ± 10	−509 ± 6	−353 ± 8	−532 ± 8
RobotBot	−288 ± 9	−260 ± 10	−137 ± 24	815 ± 5	–	−213 ± 11	−296 ± 11	349 ± 21	364 ± 18	207 ± 19	−299 ± 11
Sartre	−6 ± 6	48 ± 9	617 ± 16	647 ± 9	213 ± 11	–	−16 ± 7	979 ± 14	748 ± 12	722 ± 12	16 ± 7
Slumbot	27 ± 7	66 ± 10	448 ± 16	530 ± 8	296 ± 11	16 ± 7	–	520 ± 18	493 ± 11	493 ± 13	25 ± 8
TellBot	−383 ± 17	−790 ± 17	−764 ± 46	−35 ± 10	−349 ± 21	−979 ± 14	−520 ± 18	–	−1515 ± 21	−879 ± 28	−579 ± 17
Tiltnet	−466 ± 12	−605 ± 12	−224 ± 17	509 ± 6	−364 ± 18	−748 ± 12	−493 ± 11	1515 ± 21	–	–	−543 ± 13
TiltAdapt	−452 ± 14	−589 ± 14	−383 ± 22	353 ± 8	−207 ± 19	−722 ± 12	−493 ± 13	879 ± 28	–	–	−546 ± 13
ZBot	9 ± 8	57 ± 8	396 ± 13	532 ± 8	299 ± 11	−16 ± 7	−25 ± 8	579 ± 17	543 ± 13	546 ± 13	–

2011 Heads-up Limit Texas Hold'em

	hugh	Hyperborean-iro	Hyperborean-tbr	Lucky7	player.kappa.nl	POMPEIA	Rembrant	SartreNL
hugh	-	−597 ± 41	−603 ± 37	684 ± 39	705 ± 312	4563 ± 26	735 ± 29	−279 ± 35
Hyperborean-iro	597 ± 41	-	-	535 ± 36	2735 ± 110	2194 ± 49	599 ± 35	145 ± 34
Hyperborean-tbr	603 ± 37	-	-	576 ± 39	2715 ± 110	2112 ± 43	620 ± 34	173 ± 29
Lucky7	−684 ± 39	−535 ± 36	−576 ± 39	-	1318 ± 225	9852 ± 108	−87 ± 39	−419 ± 35
player.kappa.nl	−705 ± 312	−2735 ± 110	−2715 ± 110	−1318 ± 225	-	17824 ± 231	−1825 ± 113	−5105 ± 254
POMPEIA	−4563 ± 26	−2194 ± 49	−2112 ± 43	−9852 ± 108	−17824 ± 231	-	−2572 ± 28	−1835 ± 27
Rembrant	−735 ± 29	−599 ± 35	−620 ± 34	87 ± 39	1825 ± 113	2572 ± 28	-	−348 ± 39
SartreNL	279 ± 35	−145 ± 34	−173 ± 29	419 ± 35	5105 ± 254	1835 ± 27	348 ± 39	-

2011 Heads-up No-Limit Texas Hold’em

	AAIMontybot								
	Bnold3	dcubot3plr	Entropy	Hyperborean-iro	Hyperborean-tbr	LittleRock	OwnBot	player.zeta.3p	Sartre3p
AAIMontybot	–	–	–	–	–	–	–	–	–
Bnold3	–	−30 ± 23	−231 ± 125	−50 ± 56	−37 ± 58	−25 ± 26	−20 ± 58	55 ± 41	−26 ± 36
dcubot3plr	147 ± 43	–	129 ± 51	26 ± 34	–	77 ± 39	118 ± 32	286 ± 47	51 ± 39
Entropy	−181 ± 74	−529 ± 78	–	−523 ± 41	−491 ± 67	−498 ± 105	−223 ± 66	−142 ± 85	−718 ± 78
Hyperborean-iro	145 ± 37	133 ± 62	211 ± 51	–	–	99 ± 36	183 ± 50	441 ± 33	133 ± 48
Hyperborean-tbr	123 ± 45	–	129 ± 47	–	–	130 ± 42	181 ± 69	370 ± 53	102 ± 21
LittleRock	98 ± 34	67 ± 34	128 ± 58	58 ± 41	34 ± 27	–	99 ± 53	376 ± 57	49 ± 26
OwnBot	25 ± 64	−62 ± 68	−9 ± 55	−118 ± 69	−111 ± 79	−74 ± 77	–	323 ± 106	−124 ± 74
player.zeta.3p	−408 ± 40	−631 ± 60	−350 ± 78	−684 ± 43	−672 ± 78	−655 ± 52	−659 ± 60	–	−849 ± 45
Sartre3p	153 ± 67	162 ± 41	397 ± 69	76 ± 39	108 ± 42	137 ± 30	191 ± 63	567 ± 51	–

	Bnold3							
	dcubot3plr	Entropy	Hyperborean-iro	Hyperborean-tbr	LittleRock	OwnBot	player.zeta.3p	Sartre3p
AAIMontybot	−118 ± 39	412 ± 69	−94 ± 76	−86 ± 70	−73 ± 38	−5 ± 65	353 ± 62	−127 ± 92
Bnold3	–	–	–	–	–	–	–	–
dcubot3plr	–	103 ± 16	10 ± 9	–	44 ± 10	79 ± 12	283 ± 17	18 ± 8
Entropy	191 ± 27	–	89 ± 23	107 ± 25	152 ± 24	275 ± 28	235 ± 34	−54 ± 23
Hyperborean-iro	100 ± 9	228 ± 15	–	–	103 ± 9	181 ± 13	451 ± 17	91 ± 11
Hyperborean-tbr	–	173 ± 17	–	–	83 ± 9	124 ± 13	356 ± 15	80 ± 9
LittleRock	39 ± 9	120 ± 13	10 ± 9	22 ± 8	–	89 ± 12	313 ± 17	25 ± 8
OwnBot	−35 ± 16	4 ± 22	−113 ± 19	−92 ± 18	−61 ± 18	–	273 ± 25	−122 ± 18
player.zeta.3p	−320 ± 25	−117 ± 37	−444 ± 23	−397 ± 25	−364 ± 23	−312 ± 27	–	−526 ± 19
Sartre3p	76 ± 10	344 ± 15	33 ± 11	34 ± 9	67 ± 8	163 ± 12	489 ± 16	–

	dcubot3plr						
	Entropy	Hyperborean-iro	Hyperborean-tbr	LittleRock	OwnBot	player.zeta.3p	Sartre3p
AAIMontybot	399 ± 100	−159 ± 56	–	−144 ± 54	−55 ± 63	344 ± 43	−213 ± 42
Bnold3	−294 ± 18	−110 ± 9	–	−83 ± 8	−44 ± 11	37 ± 19	−94 ± 10
dcubot3plr	–	–	–	–	–	–	–
Entropy	–	−225 ± 22	–	−189 ± 22	117 ± 25	252 ± 32	−378 ± 24
Hyperborean-iro	194 ± 14	–	–	65 ± 11	151 ± 12	448 ± 17	56 ± 10
Hyperborean-tbr	–	–	–	–	–	–	–
LittleRock	109 ± 16	−14 ± 10	–	–	80 ± 13	375 ± 14	3 ± 9
OwnBot	−5 ± 24	−180 ± 17	–	−129 ± 18	–	359 ± 24	−187 ± 16
player.zeta.3p	−180 ± 32	−681 ± 22	–	−661 ± 21	−453 ± 26	–	−792 ± 21
Sartre3p	325 ± 16	23 ± 10	–	43 ± 8	156 ± 12	537 ± 15	–

2011 3-player Limit Texas Hold'em (1 of 2)

	Entropy					
	Hyperborean-iro	Hyperborean-tbr	LittleRock	OwnBot	player.zeta.3p	Sartre3p
AAIMontybot	312 ± 63	361 ± 60	371 ± 90	232 ± 84	492 ± 42	320 ± 63
Bnold3	−317 ± 17	−280 ± 18	−272 ± 18	−278 ± 19	−118 ± 30	−290 ± 15
dcubot3plr	31 ± 14	–	80 ± 14	−112 ± 19	−72 ± 17	53 ± 15
Entropy	–	–	–	–	–	–
Hyperborean-iro	–	–	200 ± 14	94 ± 16	225 ± 24	191 ± 15
Hyperborean-tbr	–	–	158 ± 16	3 ± 19	152 ± 19	151 ± 14
LittleRock	61 ± 15	83 ± 14	–	−31 ± 18	63 ± 18	87 ± 15
OwnBot	−96 ± 22	−61 ± 19	−42 ± 25	–	453 ± 35	−127 ± 24
player.zeta.3p	−285 ± 31	−253 ± 34	−224 ± 34	−680 ± 36	–	−430 ± 33
Sartre3p	262 ± 14	290 ± 14	322 ± 15	197 ± 19	431 ± 22	–

	Hyperborean-iro					Hyperborean-tbr			
	Hyperborean-tbr	LittleRock	OwnBot	player.zeta.3p	Sartre3p	LittleRock	OwnBot	player.zeta.3p	Sartre3p
AAIMontybot	–	−157 ± 57	−66 ± 63	243 ± 61	−209 ± 71	−163 ± 58	−70 ± 74	303 ± 67	−210 ± 50
Bnold3	–	−113 ± 8	−68 ± 14	−6 ± 16	−124 ± 10	−105 ± 9	−32 ± 9	41 ± 17	−114 ± 11
dcubot3plr	–	−51 ± 10	30 ± 13	233 ± 16	−79 ± 10	–	–	–	–
Entropy	–	−262 ± 24	2 ± 25	60 ± 38	−453 ± 22	−241 ± 25	58 ± 23	102 ± 35	−441 ± 22
Hyperborean-iro	–	–	–	–	–	–	–	–	–
Hyperborean-tbr	–	–	–	–	–	–	–	–	–
LittleRock	–	–	58 ± 15	287 ± 14	−29 ± 9	–	64 ± 12	327 ± 14	−28 ± 8
OwnBot	–	−200 ± 18	–	130 ± 26	−253 ± 16	−173 ± 19	–	238 ± 22	−229 ± 17
player.zeta.3p	–	−742 ± 22	−524 ± 29	–	−884 ± 21	−708 ± 22	−518 ± 28	–	−852 ± 18
Sartre3p	–	−5 ± 9	121 ± 12	443 ± 15	–	−3 ± 8	123 ± 10	487 ± 12	–

	LittleRock			OwnBot		player.zeta.3p
	OwnBot	player.zeta.3p	Sartre3p	player.zeta.3p	Sartre3p	Sartre3p
AAIMontybot	−24 ± 54	279 ± 60	−186 ± 55	336 ± 76	−67 ± 64	282 ± 92
Bnold3	−27 ± 12	52 ± 16	−93 ± 9	39 ± 25	−41 ± 14	37 ± 13
dcubot3plr	50 ± 11	286 ± 15	−46 ± 8	94 ± 22	31 ± 13	255 ± 15
Entropy	73 ± 22	161 ± 34	−409 ± 22	226 ± 38	−70 ± 27	−1 ± 35
Hyperborean-iro	142 ± 12	455 ± 17	34 ± 10	394 ± 23	132 ± 12	441 ± 15
Hyperborean-tbr	110 ± 13	381 ± 14	31 ± 8	280 ± 22	106 ± 12	365 ± 11
LittleRock	–	–	–	247 ± 22	68 ± 13	326 ± 13
OwnBot	–	250 ± 23	−201 ± 18	–	–	200 ± 21
player.zeta.3p	−497 ± 25	–	−821 ± 22	–	−655 ± 23	–
Sartre3p	134 ± 13	495 ± 15	–	455 ± 20	–	–

2011 3-player Limit Texas Hold'em (2 of 2)

	Entropy	Feste.iro	Feste.tbr	huhuers	Hyperborean.iro	Hyperborean.tbr	Little Ace	Little Rock	Neo Poker Lab	Patience	Sartre	Slumbot	Zbot
Entropy	–	27 ± 6	35 ± 5	8 ± 5	−17 ± 5	−1 ± 5	41 ± 6	−10 ± 5	−6 ± 6	−7 ± 5	6 ± 3	−22 ± 5	−3 ± 6
Feste.iro	−27 ± 6	–	6 ± 5	−14 ± 5	−43 ± 6	−55 ± 6	8 ± 7	−44 ± 5	−37 ± 6	−36 ± 6	−27 ± 4	−46 ± 5	−33 ± 6
Feste.tbr	−35 ± 5	−6 ± 5	–	−20 ± 6	−44 ± 6	−56 ± 6	1 ± 6	−45 ± 6	−32 ± 6	−38 ± 6	−34 ± 5	−47 ± 5	−36 ± 6
huhuers	−8 ± 5	14 ± 5	20 ± 6	–	−34 ± 5	−17 ± 6	22 ± 6	−22 ± 5	−10 ± 5	−16 ± 5	−7 ± 5	−26 ± 5	−18 ± 5
Hyperborean.iro	17 ± 5	43 ± 6	44 ± 6	34 ± 5	–	30 ± 5	52 ± 5	16 ± 3	24 ± 5	18 ± 6	15 ± 5	−4 ± 3	14 ± 3
Hyperborean.tbr	1 ± 5	55 ± 6	56 ± 6	17 ± 6	−30 ± 5	–	52 ± 5	−8 ± 6	−2 ± 5	−3 ± 6	6 ± 5	−32 ± 6	−18 ± 6
Little Ace	−41 ± 6	−8 ± 7	−1 ± 6	−22 ± 6	−52 ± 5	−52 ± 5	–	−40 ± 6	−30 ± 6	−41 ± 6	−50 ± 5	−55 ± 5	−44 ± 6
Little Rock	10 ± 5	44 ± 5	45 ± 6	22 ± 5	−16 ± 3	8 ± 6	40 ± 6	–	7 ± 6	7 ± 6	17 ± 5	−18 ± 3	−3 ± 3
Neo Poker Lab	6 ± 6	37 ± 6	32 ± 6	10 ± 5	−24 ± 5	2 ± 5	30 ± 6	−7 ± 6	–	1 ± 6	7 ± 5	−22 ± 5	−11 ± 5
Patience	7 ± 5	36 ± 6	38 ± 6	16 ± 5	−18 ± 6	3 ± 6	41 ± 6	−7 ± 6	−1 ± 6	–	7 ± 5	−26 ± 5	−11 ± 5
Sartre	−6 ± 3	27 ± 4	34 ± 5	7 ± 5	−15 ± 5	−6 ± 5	50 ± 5	−17 ± 5	−7 ± 5	−7 ± 5	–	−19 ± 5	−7 ± 5
Slumbot	22 ± 5	46 ± 5	47 ± 5	26 ± 5	4 ± 3	32 ± 6	55 ± 5	18 ± 3	22 ± 5	26 ± 5	19 ± 5	–	21 ± 4
Zbot	3 ± 6	33 ± 6	36 ± 6	18 ± 5	−14 ± 3	18 ± 6	44 ± 6	3 ± 3	11 ± 5	11 ± 5	7 ± 5	−21 ± 4	–

2012 Heads-up Limit Texas Hold'em

	AzureSky	dcubot	Hugh	Hyperborean	Little Rock	Lucky7_12	Neo Poker Lab	SartreNL	Spewy Louie	Tartanian5	Uni-MB_poker
AzureSky	–	4848 ± 66	-2877 ± 116	-2333 ± 132	-5170 ± 139	2295 ± 162	-2009 ± 102	559 ± 146	-3237 ± 156	-2954 ± 96	8920 ± 230
dcubot	-4848 ± 66	–	-756 ± 29	-498 ± 27	-1106 ± 37	-1605 ± 41	-706 ± 35	-333 ± 35	-683 ± 35	-754 ± 36	316 ± 31
Hugh	2877 ± 116	756 ± 29	–	-165 ± 36	-284 ± 39	794 ± 71	-314 ± 52	78 ± 33	290 ± 57	102 ± 30	693 ± 38
Hyperborean	2333 ± 132	498 ± 27	165 ± 36	–	199 ± 64	571 ± 65	186 ± 34	125 ± 37	503 ± 63	161 ± 36	1121 ± 96
Little Rock	5170 ± 139	1106 ± 37	284 ± 39	-199 ± 64	–	1317 ± 104	44 ± 51	223 ± 61	582 ± 64	-165 ± 63	2802 ± 111
Lucky7_12	-2295 ± 162	1605 ± 41	-794 ± 71	-571 ± 65	-1317 ± 104	–	-871 ± 67	-736 ± 88	-956 ± 81	-536 ± 94	-1617 ± 237
Neo Poker Lab	2009 ± 102	706 ± 35	314 ± 52	-186 ± 34	-44 ± 51	871 ± 67	–	2 ± 53	434 ± 54	-8 ± 31	1237 ± 66
SartreNL	-559 ± 146	333 ± 35	-78 ± 33	-125 ± 37	-223 ± 61	736 ± 88	-2 ± 53	–	141 ± 76	-56 ± 38	952 ± 61
Spewy Louie	3237 ± 156	683 ± 35	-290 ± 57	-503 ± 63	-582 ± 64	956 ± 81	-434 ± 54	-141 ± 76	–	-614 ± 60	1239 ± 119
Tartanian5	2954 ± 96	754 ± 36	-102 ± 30	-161 ± 36	165 ± 63	536 ± 94	8 ± 31	56 ± 38	614 ± 60	–	1148 ± 61
Uni-MB_poker	-8920 ± 230	-316 ± 31	-693 ± 38	-1121 ± 96	-2802 ± 111	1617 ± 237	-1237 ± 66	-952 ± 61	-1239 ± 119	-1148 ± 61	–

2012 Heads-up No-Limit Texas Hold'em

	dcubot				Hyperborean			Little Rock		Neo Poker Lab
	Hyperborean	Little Rock	Neo Poker Lab	Sartre3p	Little Rock	Neo Poker Lab	Sartre3p	Neo Poker Lab	Sartre3p	Sartre3p
dcubot	–	–	–	–	-65 ± 11	-65 ± 10	-64 ± 11	-58 ± 11	-62 ± 10	-60 ± 9
Hyperborean	–	43 ± 8	48 ± 9	50 ± 10	–	–	–	23 ± 8	26 ± 7	34 ± 6
Little Rock	22 ± 8	–	31 ± 9	36 ± 9	–	-9 ± 9	-9 ± 8	–	–	6 ± 8
Neo Poker Lab	16 ± 7	27 ± 8	–	33 ± 7	-14 ± 6	–	-15 ± 7	–	-5 ± 8	–
Sartre3p	14 ± 9	26 ± 10	27 ± 9	–	-17 ± 9	-19 ± 8	–	-1 ± 8	–	–

2012 3-player Limit Texas Hold’em

Bibliography

[1] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59, 1994.

[2] ACPC. The Annual Computer Poker Competition, 2013. http://www.computerpokercompetition.org/.

[3] David W. Aha. Editorial. *Artificial Intelligence Review*, 11(1-5):7–10, 1997.

[4] David W. Aha, Matthew Molineaux, and Marc J. V. Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In *6th International Conference on Case-Based Reasoning, ICCBR 2005*, pages 5–20. Springer, 2005.

[5] Marv Andersen. Web posting at pokerai.org forums, general forums, August 2009. http://pokerai.org/pf3/viewtopic.php?t=2259&start=18.

[6] Rickard Andersson. Pseudo-optimal strategies in no-limit poker. Master's thesis, Umea University, 2006.

[7] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[8] Bryan Auslander, Stephen Lee-Urban, Chad Hogg, and Héctor Muñoz-Avila. Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. In *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008*, pages 59–73, 2008.

[9] Bruce W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artif. Intell.*, 21(3):327–350, 1983.

[10] Brien Beattie, Garrett Nicolai, David Gerhard, and Robert J. Hilderman. Pattern classification in no-limit poker: A head-start evolutionary approach. In *Canadian Conference on AI*, pages 204–215. 2007.

[11] Darse Billings. *Algorithms & Assessment In Computer Poker*. PhD thesis, University of Alberta, 2006.

[12] Darse Billings and Morgan Kan. A tool for the direct assessment of poker decisions. *The International Association of Computer Games Journal*, 2006.

[13] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI/IAAI*, pages 493–499. 1998.

[14] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Poker as testbed for AI research. In *Advances in Artificial Intelligence, 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 228–238. 1998.

[15] Darse Billings, Lourdes Peña Castillo, Jonathan Schaeffer, and Duane Szafron. Using probabilistic knowledge and simulation to play poker. In *AAAI/IAAI*, pages 697–703, 1999.

[16] Darse Billings, Denis Papp, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron. Using selective-sampling simulations in poker. In *AAAISS-99, Proceedings of the AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*. 1999.

[17] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artif. Intell.*, 134(1-2):201–240, 2002.

[18] Darse Billings, Neil Burch, Aaron Davidson, Robert C. Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[19] Darse Billings, Aaron Davidson, Terence Schauenberg, Neil Burch, Michael H. Bowling, Robert C. Holte, Jonathan Schaeffer, and Duane Szafron. Game-tree search with adaptation in stochastic imperfect-information games. In *Computers and Games, 4th International Conference, CG 2004*, pages 21–34, 2004.

[20] Michael Bowling, Nicholas Abou Risk, Nolan Bard, Darse Billings, Neil Burch, Joshua Davidson, John Hawkin, Robert Holte, Michael Johanson, Morgan Kan, Bryce Paradis, Jonathan Schaeffer, David Schnizlein, Duane Szafron, Kevin Waugh, and Martin Zinkevich. A demonstration of the polaris poker system. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1391–1392. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009. ISBN 978-0-9817381-7-8.

[21] George W. Brown. Iterative solutions of games by fictitious play. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. Wiley, New York, 1951.

[22] Michael Buro and David Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106–108, 2012.

[23] Lourdes Peña Castillo. Probabilities and simulations in poker. Master's thesis, University of Alberta, 1999.

[24] William Cheetham and Ian D. Watson. Fielded applications of case-based reasoning. *Knowledge Eng. Review*, 20(3):321–323, 2005.

[25] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games, 5th International Conference, CG 2006*, pages 72–83. 2006.

[26] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967. ISSN 0018-9448.

[27] Technische Universität Darmstadt. Computer poker bots, 2013. http://www.ke.tu-darmstadt.de/resources/poker.

[28] Aaron Davidson. Opponent modeling in poker: Learning and acting in a hostile and uncertain environment. Master's thesis, University of Alberta, 2002.

[29] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence*, pages 1467–1473. 2000.

[30] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems, First International Workshop, MCS 2000*, pages 1–15, 2000.

[31] William Dudziak. Using fictitious play to find pseudo-optimal solutions for full-scale poker. In *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, pages 374–380. 2006.

[32] Ian Fellows. Artificial Intelligence Poker, 2013. http://www.deducer.org/pmwiki/index.php?n=Main.ArtificialIntelligencePoker.

[33] Michael W. Floyd and Babak Esfandiari. An active approach to automatic case generation. In *Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009*, pages 150–164, 2009.

[34] Michael W. Floyd, Babak Esfandiari, and Kevin Lam. A case-based reasoning approach to imitating robocup players. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference*, pages 251–256, 2008.

[35] Robert I. Follek. Soarbot: A rule-based system for playing poker. Master's thesis, Pace University, 2003.

[36] Sam Ganzfried and Tuomas Sandholm. Computing an approximate jam/fold equilibrium for 3-player no-limit Texas Hold'em tournaments. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 919–925. 2008.

[37] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[38] Kellen Gillespie, Justin Karneeb, Stephen Lee-Urban, and Héctor Muñoz-Avila. Imitating inscrutable enemies: Learning from stochastic policy observation, retrieval and reuse. In *Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010*, pages 126–140, 2010.

[39] Andrew Gilpin and Tuomas Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*. 2006.

[40] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 192–200, 2007.

[41] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *J. ACM*, 54(5), 2007.

[42] Andrew Gilpin and Tuomas Sandholm. Expectation-based versus potential-aware automated abstraction in imperfect information games: An experimental comparison using poker. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 1454–1457. 2008.

[43] Andrew Gilpin and Tuomas Sandholm. Speeding up gradient-based algorithms for sequential games. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1463–1464. 2010.

[44] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding Nash equilibria in extensive form games. In *Internet and Network Economics, Third International Workshop, WINE 2007*, pages 57–69, 2007.

[45] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 50–57. 2007.

[46] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 911–918, 2008.

[47] Matthew L. Ginsberg. Gib: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 584–593. 1999.

[48] Mehmet H. Göker, Robert J. Howlett, and Joseph E. Price. Case-Based Reasoning for diagnosis applications. *Knowledge Eng. Review*, 20(3):277–281, 2005.

[49] Kristian J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14(3):385–443, 1990.

[50] Alec Holt, Isabelle Bichindaritz, Rainer Schmidt, and Petra Perner. Medical applications in case-based reasoning. *Knowledge Eng. Review*, 20(3):289–292, 2005.

[51] ICGA. International computer games association website, 2013. http://ticc.uvt.nl/icga/.

[52] Michael Johanson. Web posting at pokerai.org forums, general forums, August 2009. http://pokerai.org/pf3/viewtopic.php?p=20456#p20456.

[53] Michael Johanson and Michael Bowling. Data biased robust counter strategies. In *Twelfth International Conference on Artificial Intelligence and Statistics*, pages 264–271. 2009.

[54] Michael Johanson, Martin Zinkevich, and Michael H. Bowling. Computing robust counter-strategies. In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.

[55] Michael Bradley Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[56] Morgan Hugh Kan. Postgame analysis of poker decisions. Master's thesis, University of Alberta, 2007.

[57] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artif. Intell.*, 6 (4):293–326, 1975.

[58] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning*, pages 282–293. 2006.

[59] D. Koller and A. Pfeffer. Generating and solving imperfect information games. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1185–1192. Montreal, Canada, August 1995.

[60] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, pages 750–759. 1994.

[61] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[62] Kevin B. Korb, Ann E. Nicholson, and Nathalie Jitnah. Bayesian poker. In *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 343–350. 1999.

[63] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: an architecture for general intelligence. *Artif. Intell.*, 33(1):1–64, 1987. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(87)90050-6.

[64] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 1078–1086. 2009.

[65] Michael Littman and Martin Zinkevich. The AAAI computer poker competition. *Journal of the International Computer Games Association*, 29, 2006.

[66] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Procedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[67] Raphaël Maîtrepierre, Jérémie Mary, and Rémi Munos. Adaptive play in Texas Hold'em poker. In *ECAI 2008 - 18th European Conference on Artificial Intelligence*, pages 458–462. 2008.

[68] Pedro-Pablo Gómez Martín, David LLansó, Marco Antonio Gómez Martín, Santiago Ontañón, and Ashwin Ram. MMPM: a generic platform for case-based planning research. In *Case-Based Reasoning Research and Development Workshop on CBR for Computer Games*, pages 45–54, 2010.

[69] Patrick McCurley. An artificial intelligence agent for Texas Hold'em poker, 2009. Undergraduate Dissertation, University of Newcastle Upon Tyne.

[70] Nicholas Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. ISSN 01621459.

[71] Peter Bro Miltersen and Troels Bjerre Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, page 191. 2007.

[72] Kinshuk Mishra, Santiago Ontañón, and Ashwin Ram. Situation assessment for plan retrieval in real-time strategy games. In *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008*, pages 355–369, 2008.

[73] Peter Morris. *Introduction to game theory*. New York : Springer-Verlag, 1994.

[74] Héctor Muñoz-Avila and Michael T. Cox. Case-based plan adaptation: An analysis and review. *IEEE Intelligent Systems*, 23(4):75–81, 2008.

[75] Yu. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM J. on Optimization*, 16(1):235–249, 2005. ISSN 1052-6234. doi: http://dx.doi.org/10.1137/S1052623403422285.

[76] Ann E. Nicholson, Kevin B. Korb, and Darren Boulton. Using Bayesian decision networks to play Texas Hold'em poker. Technical report, Faculty of Information Technology, Monash University, 2006.

[77] Garrett Nicolai and Robert Hilderman. No-limit Texas Hold'em poker agents created with evolutionary neural networks. In *CIG-2009, IEEE Symposium on Computational Intelligence and Games*, pages 125–131. 2009.

[78] Jason Noble. Finding robust Texas Hold'em poker strategies using pareto coevolution and deterministic crowding. In *Proceedings of the 2002 International Conference on Machine Learning and Applications - ICMLA 2002*, pages 233–239. 2002.

[79] Jason Noble and Richard A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In *Proceedings of the*

Genetic and Evolutionary Computation Conference, GECCO-2001, pages 493–500. Morgan Kaufmann, 2001.

[80] Santi Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. On-line case-based planning. *Computational Intelligence*, 26(1):84–119, 2010.

[81] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In *7th International Conference on Case-Based Reasoning, ICCBR 2007*, pages 164–178, 2007.

[82] Santiago Ontañón, Kane Bonnette, Prafulla Mahindrakar, Marco A. Gómez-Martín, Katie Long, Jainarayan Radhakrishnan, Rushabh Shah, and Ashwin Ram. Learning from human demonstrations for real-time case-based planning. In *IJCAI-09 Workshop on Learning Structural Knowledge From Observations*, 2009.

[83] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. Cambridge, Mass. : MIT Press, 1994.

[84] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359, 2010.

[85] Denis Richard Papp. Dealing with imperfect information in poker. Master's thesis, University of Alberta, 1998.

[86] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Mateo, Calif. : Morgan Kaufmann Publishers, 1988.

[87] Marc Ponsen, Marc Lanctot, and Steven de Jong. MCRNR: Fast computing of restricted Nash responses by means of sampling. In *Proceedings of Interactive Decision Theory and Game Theory Workshop (AAAI 2010)*. 2010.

[88] Jay H. Powell, Brandon M. Hauff, and John D. Hastings. Utilizing case-based reasoning and automatic case elicitation to develop a self-taught knowledgeable agent. In *Proceedings of the Workshop on Challenges in Game AI, Nineteenth National Conference on Artificial Intelligence (AAAI-2004), San Jose, California, USA*. AAAI Press, 2004.

[89] Hanyang Quek, Chunghoong Woo, Kay Chen Tan, and Arthur Tay. Evolving Nash-optimal poker strategies using evolutionary computation. *Frontiers of Computer Science in China*, 3(1):73–91, 2009.

[90] PokerAI.org Research. Pokerftp hand history database, March 2011. http://pokerftp.com/.

[91] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1989. ISBN 0898597676.

[92] Nicholas Abou Risk. Using counterfactual regret minimization to create a competitive multiplayer poker agent. Master's thesis, University of Alberta, 2009.

[93] Nicholas Abou Risk and Duane Szafron. Using counterfactual regret minimization to create competitive multiplayer poker agents. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 159–166, 2010.

[94] RoboCup. Robocup world championship, 2013. http://www.robocup.org/.

[95] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175 (5-6):958–987, 2011.

[96] Jonathan Rubin and Ian Watson. Opponent type adaptation for case-based strategies in adversarial games. In *Case-Based Reasoning Research and Development - 20th International Conference, ICCBR 2012*, pages 357–368, 2012.

[97] Jonathan Rubin and Ian Watson. Case-based strategies in computer poker. *AI Communications*, 25(1):19–48, 2012.

[98] Jonathan Rubin and Ian Watson. Sartre3P: A case-based multiplayer poker agent. In *Twenty-Sixth AAAI Conference on Artificial Intelligence, Computer Poker Symposium*, 2012.

[99] Teppo Salonen. The bluffbot website, 2013. http://www.bluffbot.com/.

[100] Jonathan Schaeffer. The games computers (and people) play. *Advances in Computers*, 52: 190–268, 2000.

[101] Jonathan Schaeffer. A gamut of games. *AI Magazine*, 22(3):29–46, 2001.

[102] Terence Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[103] David Schnizlein, Michael H. Bowling, and Duane Szafron. Probabilistic state translation in extensive games with large action sets. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 278–284, 2009.

[104] David Paul Schnizlein. State translation in no-limit poker. Master's thesis, University of Alberta, 2009.

[105] Immanuel Schweizer, Kamill Panitzek, Sang-Hyeun Park, and Johannes Frnkranz. An exploitative Monte-Carlo poker agent. Technical report, Technische Universität Darmstadt, 2009.

[106] Alex Selby. Optimal heads-up preflop poker, 1999. www.archduke.demon.co.uk/simplex.

[107] Manu Sharma, Michael P. Holmes, Juan Carlos Santamaría, Arya Irani, Charles Lee Isbell Jr., and Ashwin Ram. Transfer learning in real-time strategy games using hybrid CBR/RL. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1041–1046, 2007.

[108] Brian Sheppard. World Championship Caliber Scrabble. *Artif. Intell.*, 134(1-2):241–275, 2002.

[109] Jiefu Shi and Michael L. Littman. Abstraction methods for game theoretic poker. In *Computers and Games, Second International Conference, CG*, pages 333–345. 2000.

[110] David Sklansky. *The Theory of Poker.* Two Plus Two Publishing, Las Vegas, Nevada, 2005.

[111] David Sklansky and Mason Malmuth. *Hold'em Poker For Advanced Players.* Two Plus Two Publishing, Las Vegas, Nevada, 1994.

[112] Neha Sugandh, Santiago Ontañón, and Ashwin Ram. On-line case-based plan adaptation for real-time strategy games. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 702–707, 2008.

[113] K. Sycara, D. Navin Chandra, R. Guttal, J. Koning, and S. Narasimhan. Cadet: a case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 4(2): 157–188, 1991. ISSN 0894-9077.

[114] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3): 58–68, 1995.

[115] Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo tree search in poker using expected reward distributions. In *Advances in Machine Learning, First Asian Conference on Machine Learning, ACML 2009*, pages 367–381, 2009.

[116] Robert J. Vanderbei. *Linear programming: foundations and extensions.* Boston : Kluwer Academic, 2001.

[117] Ian Watson. *Applying Case-Based Reasoning : techniques for enterprise systems.* San Francisco, Calif. : Morgan Kaufmann, 1997.

[118] Kevin Waugh, Nolan Bard, and Michael Bowling. Strategy grafting in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 2026–2034. 2009.

[119] Kevin Waugh, David Schnizlein, Michael H. Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 781–788. 2009.

[120] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael H. Bowling. A practical use of imperfect recall. In *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009*. 2009.

[121] Bob Wilson. Wilson software: Official site of Turbo Poker, 2013. http://www.wilsonsoftware.com/.

[122] Martin Zinkevich, Michael H. Bowling, and Neil Burch. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 788–793. 2007.

[123] Martin Zinkevich, Michael Johanson, Michael H. Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.