Artificial Intelligence 175 (2011) 958–987

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Computer poker: A review

Jonathan Rubin, Ian Watson *

Department of Computer Science, University of Auckland, New Zealand

ARTICLE INFO

Article history:
Received 19 February 2010
Accepted 21 December 2010
Available online 21 January 2011

Keywords:
Computer poker
Imperfect information games
Nash equilibrium
Computational game theory
Opponent modelling

ABSTRACT

The game of poker has been identified as a beneficial domain for current AI research because of the properties it possesses such as the need to deal with hidden information and stochasticity. The identification of poker as a useful research domain has inevitably resulted in increased attention from academic researchers who have pursued many separate avenues of research in the area of computer poker. The poker domain has often featured in previous review papers that focus on games in general, however a comprehensive review paper with a specific focus on computer poker has so far been lacking in the literature. In this paper, we present a review of recent algorithms and approaches in the area of computer poker, along with a survey of the autonomous poker agents that have resulted from this research. We begin with the first serious attempts to create strong computerised poker players by constructing knowledge-based and simulation-based systems. This is followed by the use of computational game theory to construct robust poker agents and the advances that have been made in this area. Approaches to constructing exploitive agents are reviewed and the challenging problems of creating accurate and dynamic opponent models are addressed. Finally, we conclude with a selection of alternative approaches that have received attention in previously published material and the interesting problems that they pose.

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. 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* [81]:

> 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, these days all sorts of competitions exist for many different games. The International Computer

* Corresponding author.
E-mail addresses: jrubin01@gmail.com (J. Rubin), ian@cs.auckland.ac.nz (I. Watson).
URL: http://www.cs.auckland.ac.nz/research/gameai (I. Watson).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.12.005

Games Association [39] presently organises regular Computer Olympiads, where competitions are held for board and card games such as chess, checkers, bridge and Go. RoboCup [73] is an annual competition for both simulated and robotic soccer matches. There are also annual competitions for poker [1], real-time strategy games [19] and general game playing [29]. For the above domains, competitions are successfully used to drive research and bring about improvements to 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 [16] 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. However, as yet a comprehensive review of the algorithms and approaches used to construct these numerous poker agents has been lacking.

Many past reviews have focused on games in general. In *The Games Computers (and People) Play*, Schaeffer [80] provides a history of program development and case-studies for classic board and card games including backgammon, bridge, checkers, chess, Othello, poker and Scrabble. Schaeffer identifies poker as a game where human supremacy may soon be challenged and highlights the use of Monte-Carlo simulations within the poker domain due to an inability to directly apply alpha-beta search.

Fürnkranz [27] adopts a *problem-oriented* approach in his survey of machine learning approaches in game playing. The problem of opponent modelling is identified as a somewhat neglected area of research within game playing domains and its importance in a game such as poker is highlighted.

More recently (and more specific to the poker domain), Darse Billings' PhD thesis [8] provides a comprehensive description and analysis regarding the evolution of computer poker algorithms from 1997 to 2005 that he and his colleagues at the University of Alberta CPRG have investigated. While many of the approaches discussed in [8] are also addressed in this work, Billings' obvious focus is on the personal contribution made by himself and his colleagues, whereas this document incorporates previously published work from other researchers that has further extended and evaluated the original ideas presented in [8], along with further advances that have occurred over recent years.

As the vast majority of published work focuses on one particular variant of poker – Texas Hold'em, this will be the sole focus of this review paper. Furthermore, previous computer poker research can roughly be split into two broad categories [16]. 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 paper we will mostly restrict our attention to the second category, i.e. published work that attempts to tackle the full game of Texas Hold'em by producing autonomous poker playing agents that have been evaluated by challenging human or computerised opposition.

Each approach presented within this document will roughly follow the same pattern. First, the approach being reviewed 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 begins with a review of early work on knowledge-based poker agents, followed by Section 3 which introduces Monte-Carlo simulation. Section 4 focuses on ϵ-Nash equilibrium based agents produced via both linear programming and state-of-the-art iterative algorithms. Section 5 reviews exploitive agents that attempt to adapt to their opponents' playing style by constructing accurate opponent models and Section 6 briefly reviews some alternative approaches that have received attention in the literature, such as the use of case-based approaches, Bayesian poker and evolutionary algorithms. Finally, Section 7 concludes this review.

Before reviewing the avenues of research that have been investigated in the area of computer poker, the remainder of this section presents 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.

mention this in the introduction after chapter description

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 [90], or for further information on poker in general see [89].

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

The game of Texas Hold'em is played in 4 stages – **preflop**, **flop**, **turn** and **river**. During the preflop 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. The player to the left of the big blind, known as **under the gun**, then begins the betting by either folding, calling or raising. The possible betting actions common to all variations of poker are described as follows:

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 continue 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 players can wager up to the total amount of chips they possess in front of them. Once the betting is complete, as long as at least two players still remain in the hand, play continues on to the next stage. Each further 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.

During each stage players combine their hole cards with the public community cards to form their best 5 card poker hand. Each stage also involves its own round of betting and play continues as long as there are players left who have not folded their hands. 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 two or more players have the same best hand then the pot is split amongst the winners.

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.

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*** 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 [59]. Currently, it is intractable to compute exact Nash equilibria for full-scale Texas Hold'em, but by applying simplifying abstractions to the game (see Section 4) 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. 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[2] is a commercial software product 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.

Annual Computer Poker Competition The Annual Computer Poker Competition[3] (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. Entrance into the competition is open to anyone and the agents submitted typically represent the current state of the art in computer poker. The ACPC uses a duplicate match structure. For a heads-up match a duplicate match proceeds as follows: N hands are played between two agents after which the agent's memories are wiped and the N hands played again, but in the reverse direction, i.e. the cards that were initially given to player A are instead given to player B and vice-versa. This way both players get to play both sets of N cards and this reduces the variance that is involved in simply playing a set of N hands in one direction only. Many duplicate matches are played in order to achieve a significant result.

The ACPC typically involves two winner determination methods. The first is known as the **instant run-off competition** and the second is the **bankroll competition**. The instant run-off competition uses a recursive winner determination algorithm that repeatedly removes the agents that performed the worst against a current pool of players. The bankroll competition simply rewards agents that are able to maximally exploit other agents, resulting in increased bankrolls.

Man-Machine Poker Championship Another competition that has been cited in the results of some of the literature reviewed in this work 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.

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

2.1. 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 Fig. 1. In this example, a rule is created for the preflop round where the agent holds a pair of Aces (the best starting hand

[2] http://www.poker-academy.com/.
[3] http://www.computerpokercompetition.org/.

```
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...
}
```

Fig. 1. A hypothetical rule within a knowledge-based system for Texas Hold'em.

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 Fig. 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.

2.2. 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 [22,11]. Various methods are available for computing hand strength [22,13], some of which are mentioned below.

2.2.1. Hand strength computations

Enumeration techniques can be used to calculate the *immediate hand rank* (IHR) of a postflop hand [22]. 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:

$$IHR = \big(wins + (ties/2)\big)/(wins + ties + losses) \tag{1}$$

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* [22], 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) [11,22] combines the results of immediate hand strength and positive potential:

$$EHS = IHS + (1 - IHS) \times PositivePotential \tag{2}$$

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) [44].

2.2.2. 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:

$$PotOdds = \frac{c}{p + c} \tag{3}$$

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.[4]

2.3. 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 [15,11,56]. 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 [90], where they identify and rank various equivalence classes of starting hands. [11,22] 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 [90].

Turbo Texas Hold'em [98] 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, [8] still reports that the system is easily beaten by mediocre competition after initial exposure.

For his masters thesis project Follek [26] developed SoarBot, a rule-based expert system for multi-player, limit Texas Hold'em, built upon the SOAR rule-based architecture [51]. Overall, Follek reports that SoarBot's play was mediocre, stating [26]:

> 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 [66,15] and the formula-based version of Poki [22,11], 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 [11]. Furthermore, a member of the Poki family of agents achieved first place at the 2008 ACPC 6-Player limit hold'em competition [1].

More recently [56] 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 5). McCurley [56] 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 [10] 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 [14]. In general, there exist too many scenarios in the complete game of Texas Hold'em for a knowledge-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.

[4] 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.

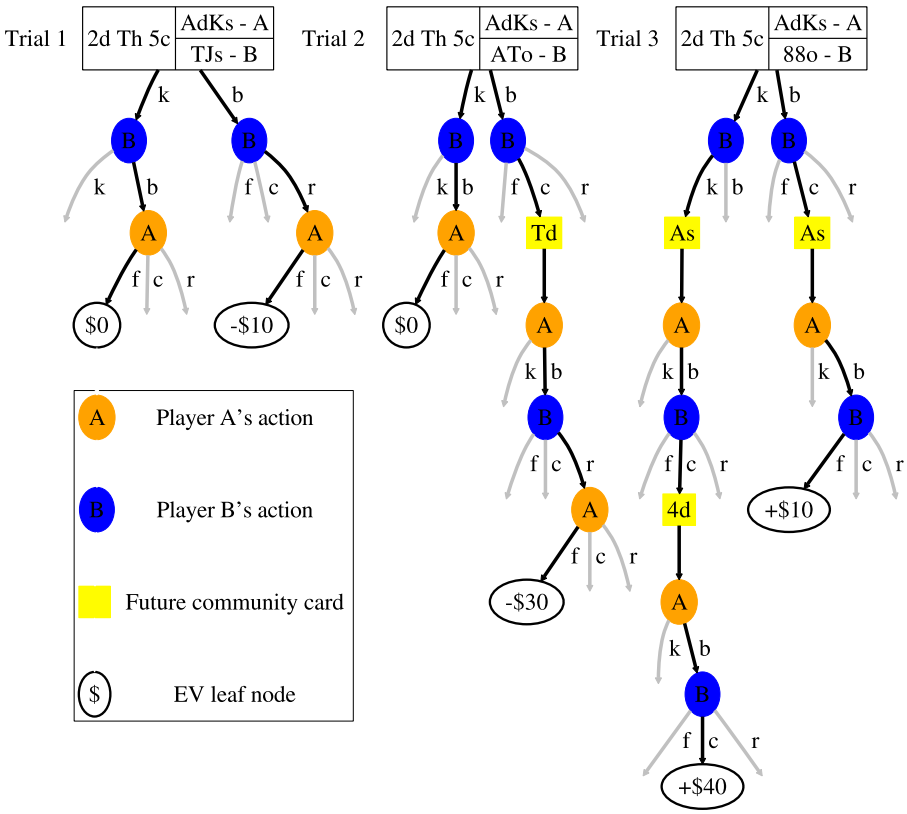

Fig. 2. Three hypothetical trials within a simulation. Each trial contributes to the final estimation of the EV for checking or betting. The betting decisions are as follows: k-check, b-bet, f-fold, c-call, r-raise.

3. Monte-Carlo and enhanced simulation

3.1. Introduction

Dynamic search of a state space usually offers an advantage over the static knowledge-based approaches mentioned in Section 2. A typical game tree search, such as minimax search with *alpha-beta* pruning [45], involves exhaustively searching the breath 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 [57], 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 [91], as well as games that involve imperfect information such as Scrabble [87] and bridge [38].

3.2. Simulation example

The following example describes the simulation process as applied to the game of Texas Hold'em poker. Fig. 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♢ T♡ 5♣.
4. Player A's personal cards are: A♢ K♠.
5. Player A is first to act and wishes to determine the expected value of either checking (k) or betting (b).

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 Fig. 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 Fig. 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.[5]
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 Fig. 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 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, Fig. 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.

3.3. 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 [66,15]. 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 [10,14,20].

Billings et al. [10] 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 [10]. 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 [10]. 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 [10,14,11].

Loki was later re-written and renamed Poki [22,11]. 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 [22,23]. Analysis reported by Davidson [22] reiterated earlier findings [10] about the emergence of

[5] 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.

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 [22] 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 [22]:

> 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 [22] 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[6] is required that is not as sensitive to bias as selective-sampling simulations are. Billings [8] 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 [22] and [8], there have been other efforts, outside of the University of Alberta CPRG, to develop poker agents based on Monte-Carlo Simulation. AKI-RealBot [85] 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 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 [85] 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 [1].

A recent algorithm that makes use of Monte-Carlo simulations is Monte-Carlo Tree Search (MCTS) [21]. 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 [46]) 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 game of Go [21]. Standard *alpha-beta* search procedures failed in the domain of Go due to the games 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. [92] have recently applied MCTS to the game of no limit hold'em. Van den Broeck et al. [92] 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. [92] 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. [92] present results that indicate the MCTS agent was able to successfully exploit naive, non-adaptive rule-based players.

4. 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

[6] The new method proposed is *miximax* and *miximix* search discussed in Section 5.1.

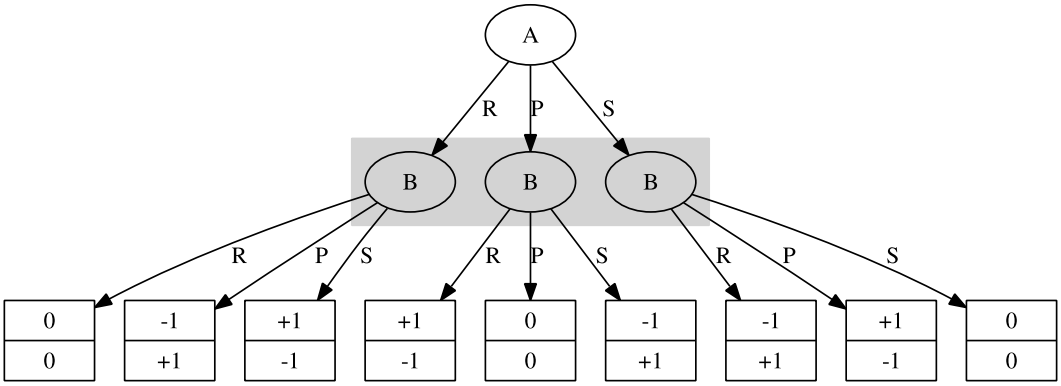

Fig. 3. The game of Rock-Paper-Scissors represented as a tree.

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.

4.1. Game theory preliminaries

The field of game theory provides analytical tools to study situations where multiple agents compete and interact within a particular environment [65,59]. Agents have individual goals and objectives that they try to achieve. Typically, a game will involve the following components (replicated from [59]):

- 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.

4.2. Rock-paper-scissors example

A game, as described above, can be represented either by rules, as a tree or as a matrix. Fig. 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*.

Fig. 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 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 Fig. 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. Fig. 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 Fig. 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

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

Fig. 4. The game of Rock-Paper-Scissors represented as a matrix.

every possible game state a player may encounter. Referring again to the game tree in Fig. 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 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 [59]. 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 [59] or interior point methods [93]) 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.

4.3. 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 corresponding 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* [47], 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 [48], it is still computationally infeasible to represent and solve for games that involve massive game trees, such as full-scale Texas Hold'em. Fig. 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 [88]. 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 [33] 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 4.4) 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.

4.4. 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 [9,31,4]. Each of these programs requires the use of a combination of abstractions to reduce the size of the game tree.

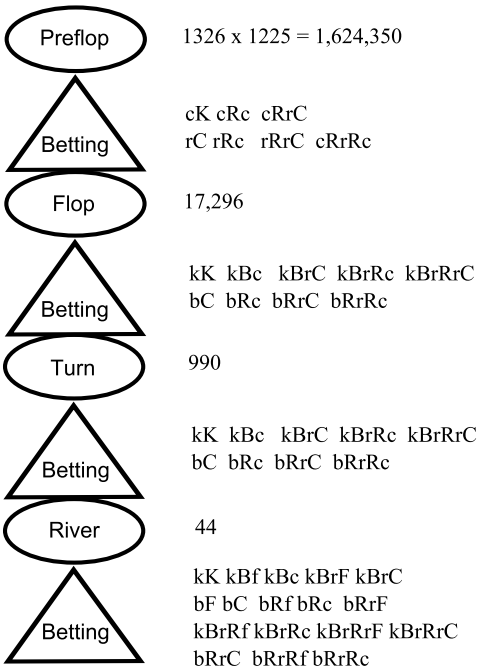

Fig. 5. A high level view of the game tree for 2-player Texas Hold'em, which consists of approximately 10^{18} game states. Based on original image presented in [9].

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 Fig. 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 AKs (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 TJo (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 [88,9,31,4]. 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.

4.4.1. 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 [43] 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 [90].

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.

4.4.2. Potential-aware automated abstraction

A separate automated abstraction method is *potential-aware* automated abstraction, developed by Gilpin and Sandholm at Carnegie Mellon University [36]. *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. [36] 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* [54] 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. [36] 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. Gilpin et al. [36] 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. [36] 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 the how many children each of the turn buckets should have.

Gilpin et al. [34] 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, [34] 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.

4.4.3. 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 [97], 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 to 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. [97] present results that suggest this is not an issue in practise. Their results show imperfect recall abstractions are superior to their perfect recall counterparts.

4.4.4. 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 [9].

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.

4.5. 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.

4.5.1. Heads-up limit

Shi and Littman [88] 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 [86] 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. [9] 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. [9] 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. [9] 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 [53].

Teppo Salonen's limit version of BluffBot [78] 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 [78].

GS1 [31], developed by Andrew Gilpin and Thomas 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* [33] 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 [32]. The next agent they developed, GS2 [32], used a better automated abstraction routine that relied on *k-means clustering* [54] 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 [32]. 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 [53].

4.5.2. 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. For example, assuming a player begins a match with 1000 chips, after paying a forced small blind of 1 chip they then have the option to either fold, call 1 more chip or raise by contributing anywhere between 3 and 999 extra chips.[7] 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,

[7] 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.

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 [37]. Furthermore, increasing the starting stack sizes of the players has the effect of exponentially increasing the size of the game tree.

Aggrobot [4] 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 [4,37].

Due to processor and memory limitations [4] 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 [78]. Starting with BluffBot2.0 which placed first in the heads-up, no limit competition at the 2007 ACPC [1]. 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, [58] 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 [58] 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 [28] used an extension of *fictitious play* (see Section 4.6.1) 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 [9] were forced to combine separate preflop and postflop models to approximate an equilibrium strategy. [31,32] 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 [8] due to the fact that simply gluing disjoint strategies together is not guaranteed to produce a complete and coherent equilibrium strategy.

Waugh et al. [95] 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.

4.6. 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.

4.6.1. Fictitious play

An alternative method for computing Nash equilibrium is via fictitious play [18]. 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 [24] 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 [25]. 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.

4.6.2. Range of Skill

The *Range of Skill* algorithm is an iterative procedure that was developed by Zinkevich et al. [99] 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. [99] 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. [99] shows that this offers a significant improvement upon previous approaches which rely on connecting together separate smaller models such as [9,31].

4.6.3. Excessive gap technique

Gilpin et al. [30] present an adapted version of Nesterov's *excessive gap technique* [60] 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 [30] 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:

$$\max_x \min_y x^T A y = \min_y \max_x x^T A y \tag{4}$$

where A is the payoff matrix for player 1, x is player 1's strategy and y is a strategy for player 2. Eq. (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. [30] 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 [36]. The result of solving a non-truncated game, together with the use of potential-aware automated abstraction (see Section 4.4.2) 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 [36].

Gilpin, Sandholm & Sørensen have also used EGT to create ϵ-Nash equilibrium strategies for no limit poker. Tartanian [37], is able to build models for starting chip stacks of up to 1000 chips (cf. [4] 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 [37] were similar to those chosen by [4] with the removal of the $2\times$pot bet. Gilpin et al. [37] 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 [37,1].

Recently, Gilpin and Sandholm have presented two speed up improvements for EGT algorithms [35]. 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 [35].

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

4.6.4. 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 [100,43]. 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 Fig. 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 [43].

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:

$$u\left(a^*\right) - u(a)$$

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 [43] and no limit variations [84], as well as multi-player games [71]. 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.

4.6.5. 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.

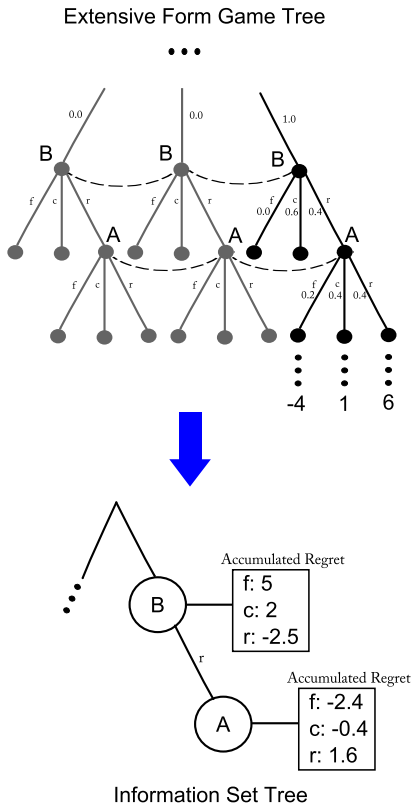

Fig. 6. Depicts a partial extensive form game tree for 2-player Texas Hold'em (top) and the corresponding portion of the information set tree (bottom). The extensive form game tree is required to be traversed, but not stored in memory. The information set tree is required to be stored in memory, it records accumulated regret values at each information set.

The 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 the algorithm proceeds. For a more detailed discussion on the theoretical background of the algorithm consult [100].

We refer again to Fig. 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.

$$f\text{: } -4 - 2 = -6$$
$$c\text{: } 1 - 2 = -1$$
$$r\text{: } 6 - 2 = 4$$

- 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.

$$f\text{: } -6 \times 0.4 = -2.4$$

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

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

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 [9,31]. 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 [100]. 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* [96] for why this is not always the case). Zinkevich et al. [100] and Johanson [43] 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 traversing every game state can offer significant improvements on convergence time. Mote 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 [52].

4.6.6. CFR-based poker agents

The University of Alberta CPRG have used CFR to produce ϵ-Nash equilibrium strategies for both Hyperborean-Eqm and Polaris [17] since 2007.[8] 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 [97].

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 [1]. GGValuta differs from Hyperborean-Eqm in the abstractions that are used to produce the model. In particular, GGValuta used a modified *k-means*

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

clustering algorithm to assign postflop hands into groups, based on hand strength and potential [1]. In the same competition Hyperborean-Eqm placed 2nd. The 2009 version of Hyperborean-Eqm was constructed by solving separate independent subgames (*grafts*) via the CFR algorithm and using *strategy grafting* [95] to combine the grafts via a base strategy.

Recently the CFR algorithm was applied to the 3-player game of limit hold'em [72]. 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 [71]. Nevertheless, the resulting player, HyperboreanRing-Eqm, was able to beat all 6 opponents at the 2009 ACPC 3-player limit competition [1].

CFR has also been used to create no limit agents [84]. 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 [83].

5. 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.

5.1. 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 [82]. 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 [6]. 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.

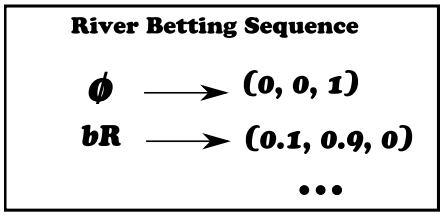

Fig. 7. Part of player A's opponent model of player B.

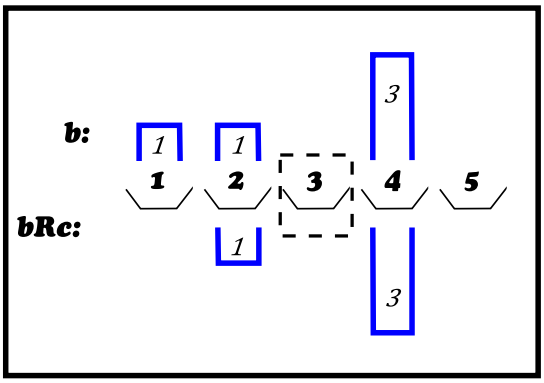

Fig. 8. Two showdown histograms for betting sequences on the river. Player A's hand rank is in bucket 3.

The University of Alberta CPRG have developed two algorithms based on the above ideas, **miximax** and **miximix** [22,12]. 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 procedure avoids always choosing the maximum action so as to balance the exploration/exploitation trade-off and avoid predictable behaviour of the agents produced.

5.1.1. 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 Figs. 7 and 8. Fig. 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%. Fig. 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.

We can now combine the information contained in the opponent model with a depth first search on the imperfect information game tree (see Fig. 9). A simple formula is used to determine the leaf node values [12]:

$$EV(x) = Pr(Win) \times TotalPot - PlayerInvestment \tag{5}$$

where $Pr(Win)$ is the probability that player A wins and x refers to a particular node in the game tree. In Fig. 9 all nodes have been labelled with numbers to clarify the calculation. A depth first search on the game tree in Fig. 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 (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

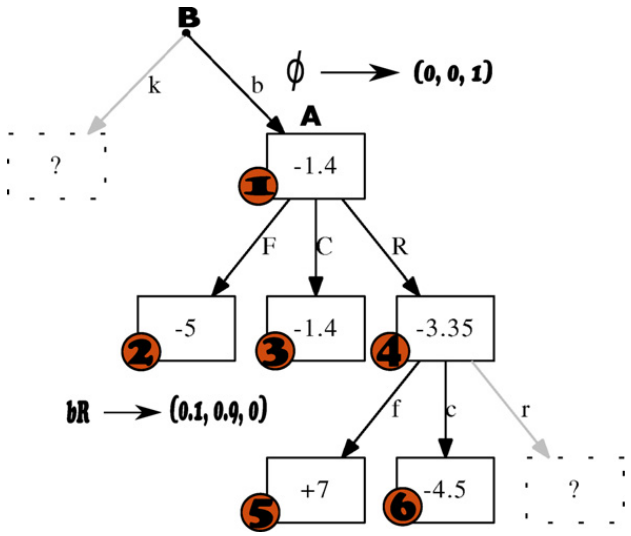

Fig. 9. Expected value calculation using the miximax algorithm.

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. Recall that, based on A's opponent model in Fig. 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 (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 Fig. 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.

5.1.2. 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 [12] 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 [82]. 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 [82]. Results

reported for Vexbot show that it is able to exploit a wide range of computerised opponents [12]. 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 [12].

McCurley [56] applied the miximax algorithm to heads-up, no limit hold'em. Rather than recording action frequencies to predict opponent actions, McCurley [56] 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 [56] indicate that the agent was still profitable against human opposition on an online poker site.

Maîtrepierre et al. [55] 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. [55] 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 [5] 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 [55,1].

5.2. 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.

5.2.1. Frequentist best response

Frequentist Best Response (FBR) [42,43] 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 4.4). 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 [43] 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 [43] 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.

5.2.2. Restricted Nash Response

The Restricted Nash Response (RNR) [42,43] 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 [43]. 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) [68] is one such algorithm that allows a faster rate of convergence than its non-sampled counterpart.

5.2.3. 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) [41]. 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. [41] 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 [41] 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 [40], which placed second in the 2009 ACPC 2-player limit bankroll competition.

5.2.4. 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 [43] describes the use of a *coach*, that determines the appropriate strategy to employ against an opponent using the UCB1 algorithm [5]. 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. [42] 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.

6. Alternative approaches

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

6.1. Case-based reasoning

Case-based reasoning (CBR) [70,49] is a type of *lazy learning* [2] algorithm, where a set of cases are stored and maintained that encode knowledge of previously encountered situations, along with their solutions. Cases are typically represented as feature-value pairs. When a new problem requires a solution, a case is created by binding appropriate values to its features. The novel case is then compared against the stored cases in the case-base and the most similar cases retrieved, e.g. via the *k*-nearest neighbour algorithm.

The application of CBR to Texas Hold'em poker involves storing cases that describe previous hands, along with their solutions and outcomes, and re-using the solutions from previous similar cases to aid future betting decisions. Sandven and Tessem [79] used CBR to create a limit hold'em agent called CASEY. CASEY began with an empty case-base. As it played more and more games, cases were created and stored in the case-base along with their associated outcome. As the case-base was initially empty, CASEY was forced to populate the case-base by making random decisions and recording their outcomes. As CASEY recorded more hands it began to re-use decisions that had a positive outcome in similar past situations. As expected, CASEY could initially form some incorrect beliefs about the value of certain decisions, however after an initial training period CASEY'S play became less erratic.

Rubin and Watson [75], Rubin [74] have further investigated the use of CBR to derive betting strategies. Rubin and Watson [75] created a case-based limit hold'em agent nicknamed CASPER (CASe based Poker playER). CASPER was designed to play Texas Hold'em at a full table consisting of up to 10 players. Rather than learning a case-base from scratch, as in [79], CASPER generated its case-base by observing hand histories which involved formula-based and simulation-based versions of Poki within Poker Academy Pro 2.5. Each case recorded information such as hand rank and potential, the number of active players, the number of players yet to act, the number of bets to call, the pot odds and so forth. A solution to a case consisted of a betting decision. By retrieving the most similar cases a probability triple was constructed and used to determine a betting action. CASPER was shown to be competitive against a range of computerised opponents offered through Poker Academy Pro, including Poki. CASPER was also tested against human opposition on an online poker site for both fake and real money. While CASPER made a consistent profit at the non-real money tables, a small loss was made when playing for miniature stakes at the real money tables [94,74].

A separate CBR system nicknamed Sartre (Similarity Assessment Reasoning for Texas hold'em via Recall of Experience) specialises in heads-up, limit hold'em [76,77]. Sartre was trained on the hand history logs of the top ϵ-Nash equilibrium agents from previous year's AAAI Computer Poker Competitions. Sartre competed in the limit hold'em competition at the 2009 ACPC, placing 6th out of 13 competitors in the bankroll division. In 2010 Sartre improved to 3rd place, in the same competition. A no limit version of Sartre also placed 2nd at the 2010 ACPC in the no limit, instant run-off competition [1].

6.2. Evolutionary algorithms and neural networks

The use of evolutionary algorithms, to automatically evolve strong poker agents, has also been the subject of investigation [62–64,7,69]. 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 [63]. 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 [64].

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 doesn't necessarily follow that A beats C.

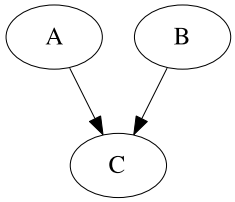

Fig. 10. A directed acyclic graph. A, B and C are random variables. Nodes A and B are the parents of node C and therefore they influence node C, represented by the edges.

Rather, Noble attempts to evolve robust strategies by selecting members of a population based on multi-dimensional optimisation. To do so Noble [63] 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 [63] 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 [62] 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 [62] 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 [62] 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 [7]. 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 [1]. While relatively few details of MANZANA's design and implementation are available, it is known that separate ANNs are used for preflop and postflop play [3]. 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 [1].

6.3. 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 [67]. The edges between nodes in the graph represent dependency relationships (see Fig. 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 [50] is illustrated in Fig. 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 Ann Nicholson and Kevin Korb from Monash University have spent many years investigating the use of Bayesian networks within the domain of poker, beginning with five-card stud [50] and later adapting this approach to Texas Hold'em [61]. 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 [61]. 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.

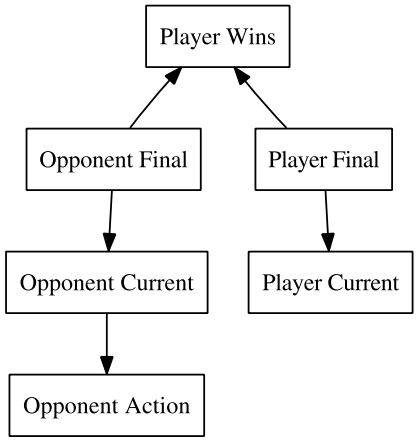

Fig. 11. A Bayesian network for poker, adapted from original image presented in [50].

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 [53]. 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 [1].

7. Conclusion

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, as well as the question of whether strong strategies can successfully be approximated via observation and generalisation. While significant advances have been made in some of the above areas there are still many challenges left to address. In particular, due to the large computational costs required to produce sophisticated strategies, much of the work presented here has focused solely on two-player, limit hold'em, with a relatively recent transition to no limit play. The difficulties of producing sophisticated strategies to challenge multiple opponents still requires further investigation. Furthermore, almost all research has focused on hand by hand play with fixed blind levels as in a poker *cash game*, with almost no consideration given to *tournament play* where increasing blind levels give rise to changes in strategy. Considering the challenges encountered so far and the difficulties imposed by the many variations of the game, it is evident that the poker domain is a vast, rich environment that seems likely to continue to offer challenging problems and opportunities for AI researchers.

Acknowledgements

The authors wish to thank all those whose work was mentioned in this review paper. Great efforts have been taken to present third party research as accurately as possible and any errors in interpretation are solely the responsibility of the authors. We further acknowledge the comments and corrections provided by the anonymous reviewers. Finally, we wish to thank the University of Alberta Computer Poker Research Group and those involved with organising the Annual Computer Poker Competitions.

References

[1] ACPC, The annual computer poker competition, http://www.computerpokercompetition.org/, 2010.
[2] D.W. Aha, Editorial, Artificial Intelligence Review 11 (1997) 7–10.
[3] M. Andersen, Web posting at pokerai.org forums, general forums, http://pokerai.org/pf3/viewtopic.php?t=2259&start=18, 2009.
[4] R. Andersson, Pseudo-optimal strategies in no-limit poker, Master's thesis, Umea University, 2006.
[5] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2002) 235–256.
[6] B.W. Ballard, The *-minimax search procedure for trees containing chance nodes, Artif. Intell. 21 (1983) 327–350.

[7] B. Beattie, G. Nicolai, D. Gerhard, R.J. Hilderman, Pattern classification in no-limit poker: A head-start evolutionary approach, in: Canadian Conference on AI, 2007, pp. 204–215.
[8] D. Billings, Algorithms & assessment in computer poker, Ph.D. thesis, University of Alberta, 2006.
[9] D. Billings, N. Burch, A. Davidson, R.C. Holte, J. Schaeffer, T. Schauenberg, D. Szafron, Approximating game-theoretic optimal strategies for full-scale poker, in: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, 2003, pp. 661–668.
[10] D. Billings, L.P. Castillo, J. Schaeffer, D. Szafron, Using probabilistic knowledge and simulation to play poker, in: AAAI/IAAI, 1999, pp. 697–703.
[11] D. Billings, A. Davidson, J. Schaeffer, D. Szafron, The challenge of poker, Artif. Intell. 134 (2002) 201–240.
[12] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M.H. Bowling, R.C. Holte, J. Schaeffer, D. Szafron, Game-tree search with adaptation in stochastic imperfect-information games, in: Computers and Games, 4th International Conference, CG 2004, 2004, pp. 21–34.
[13] D. Billings, M. Kan, A tool for the direct assessment of poker decisions, Int. Assoc. Comput. Games J. (2006).
[14] D. Billings, D. Papp, L. Peña, J. Schaeffer, D. 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.
[15] D. Billings, D. Papp, J. Schaeffer, D. Szafron, Opponent modeling in poker, in: AAAI/IAAI, 1998, pp. 493–499.
[16] D. Billings, D. Papp, J. Schaeffer, D. Szafron, Poker as testbed for AI research, in: Advances in Artificial Intelligence, 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, 1998, pp. 228–238.
[17] M. Bowling, N.A. Risk, N. Bard, D. Billings, N. Burch, J. Davidson, J. Hawkin, R. Holte, M. Johanson, M. Kan, B. Paradis, J. Schaeffer, D. Schnizlein, D. Szafron, K. Waugh, M. Zinkevich, A demonstration of the polaris poker system, in: AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009, pp. 1391–1392.
[18] G.W. Brown, Iterative solutions of games by fictitious play, in: T. Koopmans (Ed.), Activity Analysis of Production and Allocation, Wiley, New York, 1951, pp. 374–376.
[19] M. Buro, Orts rts game AI competition, 2009, http://webdocs.cs.ualberta.ca/~mburo/orts/AIIDE09/.
[20] L.P. Castillo, Probabilities and simulations in poker, Master's thesis, University of Alberta, 1999.
[21] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, in: Computers and Games, 5th International Conference, CG 2006, 2006, pp. 72–83.
[22] A. Davidson, Opponent modeling in poker: Learning and acting in a hostile and uncertain environment, Master's thesis, University of Alberta, 2002.
[23] A. Davidson, D. Billings, J. Schaeffer, D. Szafron, Improved opponent modeling in poker, in: Proceedings of the 2000 International Conference on Artificial Intelligence, 2000, pp. 1467–1473.
[24] W. 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, 2006, pp. 374–380.
[25] I. Fellows, Artificial intelligence poker, 2010, http://www.deducer.org/pmwiki/index.php/.
[26] R.I. Follek, SoarBot: A rule-based system for playing poker, Master's thesis, Pace University, 2003.
[27] J. Fürnkranz, Machine learning in games: A survey, in: Machines That Learn to Play Games, Nova Science Publishers, 2001, pp. 11–59 (Chapter 2).
[28] S. Ganzfried, T. 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), 2008, pp. 919–925.
[29] M.R. Genesereth, N. Love, B. Pell, General game playing: Overview of the AAAI competition, AI Magazine 26 (2005) 62–72.
[30] A. Gilpin, S. Hoda, J. Peña, T. Sandholm, Gradient-based algorithms for finding Nash equilibria in extensive form games, in: Internet and Network Economics, Third International Workshop, WINE 2007, 2007, pp. 57–69.
[31] A. Gilpin, T. 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.
[32] A. Gilpin, T. 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), 2007, pp. 192–200.
[33] A. Gilpin, T. Sandholm, Lossless abstraction of imperfect information games, J. ACM 54 (2007).
[34] A. Gilpin, T. 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, 2008, pp. 1454–1457.
[35] A. Gilpin, T. Sandholm, Speeding up gradient-based algorithms for sequential games, in: 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), 2010, pp. 1463–1464.
[36] A. Gilpin, T. Sandholm, T.B. 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, 2007, pp. 50–57.
[37] A. Gilpin, T. Sandholm, T.B. 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), 2008, pp. 911–918.
[38] M.L. Ginsberg, Gib: Steps toward an expert-level bridge-playing program, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999, pp. 584–593.
[39] ICGA, International computer games association website, 2010, http://ticc.uvt.nl/icga/.
[40] M. Johanson, Web posting at pokerai.org forums, general forums, 2009, http://pokerai.org/pf3/viewtopic.php?p=20456#p20456.
[41] M. Johanson, M. Bowling, Data biased robust counter strategies, in: Twelfth International Conference on Artificial Intelligence and Statistics, 2009, pp. 264–271.
[42] M. Johanson, M. Zinkevich, M.H. Bowling, Computing robust counter-strategies, in: Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, 2007.
[43] M.B. Johanson, Robust strategies and counter-strategies: Building a champion level computer poker player, Master's thesis, University of Alberta, 2007.
[44] M.H. Kan, Postgame analysis of poker decisions, Master's thesis, University of Alberta, 2007.
[45] D.E. Knuth, R.W. Moore, An analysis of alpha-beta pruning, Artif. Intell. 6 (1975) 293–326.
[46] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Machine Learning: ECML 2006, 17th European Conference on Machine Learning, 2006, pp. 282–293.
[47] D. Koller, N. Megiddo, 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), 1994, pp. 750–759.
[48] D. Koller, A. Pfeffer, Generating and solving imperfect information games, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Montreal, Canada, 1995, pp. 1185–1192.
[49] J. Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[50] K.B. Korb, A.E. Nicholson, N. Jitnah, Bayesian poker, in: UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999, pp. 343–350.
[51] J.E. Laird, A. Newell, P.S. Rosenbloom, Soar: An architecture for general intelligence, Artif. Intell. 33 (1987) 1–64.
[52] M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, Monte Carlo sampling for regret minimization in extensive games, in: Advances in Neural Information Processing Systems 22 (NIPS), 2009, pp. 1078–1086.

[53] M. Littman, M. Zinkevich, The AAAI computer poker competition, J. Int. Comput. Games Assoc. 29 (2006).
[54] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability, vol. 1, University of California Press, 1967, pp. 281–297.
[55] R. Maîtrepierre, J. Mary, R. Munos, Adaptive play in Texas hold'em poker, in: ECAI 2008 – 18th European Conference on, Artificial Intelligence, 2008, pp. 458–462.
[56] P. McCurley, An artificial intelligence agent for Texas hold'em poker, Undergraduate dissertation, University of Newcastle Upon Tyne, 2009.
[57] N. Metropolis, S. Ulam, The Monte Carlo method, J. Amer. Statist. Assoc. 44 (1949) 335–341.
[58] P.B. Miltersen, T.B. 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), 2007, p. 191.
[59] P. Morris, Introduction to Game Theory, Springer-Verlag, New York, 1994.
[60] Y. Nesterov, Excessive gap technique in nonsmooth convex minimization, SIAM J. Optim. 16 (2005) 235–249.
[61] A.E. Nicholson, K.B. Korb, D. Boulton, Using bayesian decision networks to play Texas hold'em poker, Technical Report, Faculty of Information Technology, Monash University, 2006.
[62] G. Nicolai, R. Hilderman, No-limit Texas hold'em poker agents created with evolutionary neural networks, in: CIG-2009, IEEE Symposium on Computational Intelligence and Games, 2009, pp. 125–131.
[63] J. 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, 2002, pp. 233–239.
[64] J. Noble, R.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, Morgan Kaufmann, 2001, pp. 493–500.
[65] M.J. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, Cambridge, MA, 1994.
[66] D.R. Papp, Dealing with imperfect information in poker, Master's thesis, University of Alberta, 1998.
[67] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, San Mateo, Calif., 1988.
[68] M. Ponsen, M. Lanctot, S. 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.
[69] H. Quek, C. Woo, K.C. Tan, A. Tay, Evolving Nash-optimal poker strategies using evolutionary computation, Frontiers Comput. Sci. China 3 (2009) 73–91.
[70] C.K. Riesbeck, R.C. Schank, Inside Case-Based Reasoning, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1989.
[71] N.A. Risk, Using counterfactual regret minimization to create a competitive multiplayer poker agent, Master's thesis, University of Alberta, 2009.
[72] N.A. Risk, D. Szafron, Using counterfactual regret minimization to create competitive multiplayer poker agents, in: 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), 2010, pp. 159–166.
[73] RoboCup, Robocup world championship, 2010, http://www.robocup.org/.
[74] J. Rubin, Casper: Design and development of a case-based poker player, Master's thesis, University of Auckland, 2007.
[75] J. Rubin, I. Watson, Investigating the effectiveness of applying case-based reasoning to the game of Texas hold'em, in: Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, 2007, pp. 417–422.
[76] J. Rubin, I. Watson, A memory-based approach to two-player Texas hold'em, in: AI 2009: Advances in Artificial Intelligence, 22nd Australasian Joint Conference, 2009, pp. 465–474.
[77] J. Rubin, I. Watson, Similarity-based retrieval and solution re-use policies in the game of Texas hold'em, in: Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, in: Lecture Notes in Computer Science, vol. 6176, Springer-Verlag, 2010, pp. 465–479.
[78] T. Salonen, The bluffbot website, 2009, http://www.bluffbot.com/.
[79] A. Sandven, B. Tessem, A case-based learner for poker, in: Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006), 2006.
[80] J. Schaeffer, The games computers (and people) play, Adv. Comput. 52 (2000) 190–268.
[81] J. Schaeffer, A gamut of games, AI Magazine 22 (2001) 29–46.
[82] T. Schauenberg, Opponent modelling and search in poker, Master's thesis, University of Alberta, 2006.
[83] D. Schnizlein, M.H. Bowling, D. Szafron, Probabilistic state translation in extensive games with large action sets, in: IJCAI 2009, Proceedings of the 21st International Joint Conference on, Artificial Intelligence, 2009, pp. 278–284.
[84] D.P. Schnizlein, State translation in no-limit poker, Master's thesis, University of Alberta, 2009.
[85] I. Schweizer, K. Panitzek, S.H. Park, J. Fürnkranz, An exploitative Monte-Carlo poker agent, Technical Report, Technische Universität Darmstadt, 2009.
[86] A. Selby, Optimal heads-up preflop poker, 1999, www.archduke.demon.co.uk/simplex.
[87] B. Sheppard, World-championship-caliber scrabble, Artif. Intell. 134 (2002) 241–275.
[88] J. Shi, M.L. Littman, Abstraction methods for game theoretic poker, in: Computers and Games, Second International Conference, CG, 2000, pp. 333–345.
[89] D. Sklansky, The Theory of Poker, Two Plus Two Publishing, Las Vegas, Nevada, 2005.
[90] D. Sklansky, M. Malmuth, Hold'em Poker For Advanced Players, Two Plus Two Publishing, Las Vegas, Nevada, 1994.
[91] G. Tesauro, Temporal difference learning and td-gammon, Commun. ACM 38 (1995) 58–68.
[92] G. Van den Broeck, K. Driessens, J. Ramon, Monte-Carlo tree search in poker using expected reward distributions, in: Advances in Machine Learning, First Asian Conference on Machine Learning, ACML 2009, 2009, pp. 367–381.
[93] R.J. Vanderbei, Linear Programming: Foundations and Extensions, Kluwer Academic, Boston, 2001.
[94] I. Watson, J. Rubin, Casper: A case-based poker-bot, in: AI 2008: Advances in Artificial Intelligence, 21st Australasian Joint Conference on Artificial Intelligence, 2008, pp. 594–600.
[95] K. Waugh, N. Bard, M. Bowling, Strategy grafting in extensive games, in: Advances in Neural Information Processing Systems 22 (NIPS), 2009, pp. 2026–2034.
[96] K. Waugh, D. Schnizlein, M.H. Bowling, D. Szafron, Abstraction pathologies in extensive games, in: 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), 2009, pp. 781–788.
[97] K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, M.H. Bowling, A practical use of imperfect recall, in: Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, 2009.
[98] B. Wilson, Wilson software: Official site of turbo poker, 2008, http://www.wilsonsoftware.com/.
[99] M. Zinkevich, M.H. Bowling, N. Burch, A new algorithm for generating equilibria in massive zero-sum games, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, 2007, pp. 788–793.
[100] M. Zinkevich, M. Johanson, M.H. Bowling, C. 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.