An Introduction to the Current Approaches in Computer Poker

Richard Gibson
Department of Computing Science,
University of Alberta, Edmonton, Canada
rggibson@cs.ualberta.ca

Abstract

Deterministic, full-information games have been a well-studied testbed for artificial intelligence research. Poker, however, is a game involving chance where players have private information that no other player can see and has become an exciting and challenging new domain of research. In this survey, we take a look at some of the different approaches to creating computer programs for playing poker. In one particular approach, we summarize how to create Nash equilibrium strategies using the counter-factual regret minimization algorithm. Bots using the strategies found by this algorithm are currently the state-of-the-art of poker programs.

1 Introduction

Autonomous computer agents capable of meaningful interactions with the world is the holy grail of artificial intelligence research. Every day, humans must make decisions based on an incomplete view of a world, experienced only through limited observations. In addition, the consequences of these decisions often cannot be fully known at the time of the decision-making. These decisions can be short term, such as quick decisions made while driving, or long term, such as whether or not to move to a new city. To further complicate matters, the world is full of millions of people and the actions of others can often affect our own. For instance, if we see a lot of vehicles on the highway nearby, we may believe that it will be faster to take a detour to work rather than potentially being slowed by the traffic. We would expect any fully autonomous robot to be capable of making such decisions, despite lacking all pertinent information (like a full traffic report). Unfortunately, current science and technology is far from achieving such complete autonomous behaviour.

For at least the past few decades, games have been an exceptional platform for artificial intelligence research. Many successful computer programs have been developed which play at expert levels in games such as Othello [5], checkers [12], and chess [6]. Each of these games is deterministic and both players share complete information about the game states. These games are handled well by classic artificial intelligence techniques, such as heuristic search [5] [6] [12]. In real-life problems, non-determinism (chance) abounds and we often do not have complete information. Recently, poker has become popular as a domain for AI research (examples include [2], [3], [4], [7], [8], [13], and [14]). Poker involves stochastic elements with some information hidden from the players (the player's private cards) and classical approaches like heuristic search are not appropriate when states are not completely known. Thus, computer poker research provides an excellent domain for investigating problems of decision making under conditions of uncertainty and serves as a better platform for modeling general human decision making.

In this survey, we discuss some of the exciting poker research that has developed over the past decade. In Section 2, we introduce the poker game Texas Hold'em, its variants, and the fundamental terminology used in the computer poker domain. A particular variant, Heads-Up Limit Texas Hold'em, is the focus of much of our discussion throughout the survey. We then present some early approaches to creating poker bots in Section 3 and acquaint the reader with the concept of a Nash equilibrium. In Section 4, the counter-factual regret minimization (CFR) algorithm for finding equilibria is presented. CFR-based poker programs are the strongest to date, and we present evidence of this in Section 4. Finally, we summarize the survey and discuss future avenues of poker research in Section 5.

2 Poker Terminology

The poker game we are interested in is Texas Hold'em. In this section, we explain the rules and variants of Texas Hold'em, followed by a light introduction of the essential concepts needed for studying games such as Texas Hold'em.

2.1 The rules of Texas Hold'em

In a hand of Texas Hold'em, each player receives two private cards, or *hole cards*, that are kept secret throughout the betting rounds. One player is designated as the dealer, the player to the left of the dealer places a mandatory half-bet (called the *small blind*), and the next player to the left places a mandatory bet (called the *big blind*). Then, a betting round, often called the *preflop*, begins with the player to the left of the big blind player. In a betting round, players in turn may either *fold* (forfeit the round), *call* (match the current bet, called a *check* if no bets have been made), or *raise* (increase the current bet). Once all players have either folded or matched the bets, the next round begins. In betting rounds other than the preflop, the player closest to the left of the dealer that has not folded acts first. After the preflop, 3 community cards are dealt face up for all players to see and another betting round begins. This is called the *flop*. After the flop round is the *turn*, where another community card is dealt face up and another round of betting begins. Following the turn is the *river*, where a final community card is dealt face up and a final round of betting occurs. Finally, all players remaining in the hand enter the *showdown* where they reveal their hole cards. The player able to make the best 5-card poker hand, using the 2 hole cards and the 5 community cards, wins all of the bets (known as the *pot*). The player to the left of the dealer then becomes the new dealer for the next hand.

Texas Hold'em itself has a number of variants. In Limit Texas Hold'em, each bet must be of some fixed amount. In the turn and river rounds, the fixed bet size is twice that of the bet size in the preflop and flop rounds. For example, bets may be \$5 during the preflop and flop rounds, and \$10 during the turn and river rounds. In addition, there are a maximum of 4 raises in Limit Texas Hold'em per betting round. Another variant of Texas Hold'em is No Limit. Here, each bet can be of any amount ranging from the size of the previous bet increase in the round to all the player's money on the table. Finally, each of these variants are either played as a *Ring* game (more than two players), or as a *Heads-Up* game (exactly two players). In Heads-Up play, the dealer posts the small blind and the other player posts the big blind, with action beginning with the dealer in the preflop round.

2.2 Extensive form games and hand strength

Here, we describe the components of an *extensive form game* [11], which is used to model a hand of Texas Hold'em. We will refer to these concepts throughout the rest of this survey.

Briefly, we model the game as a directed tree with a single root node, representing the initial state of the game.

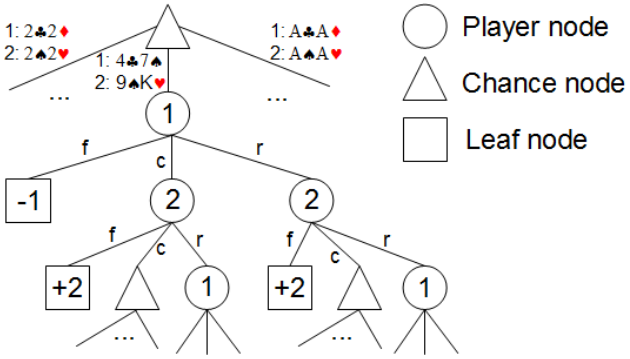

Figure 1. A portion of the extensive form game tree for Heads-Up Limit Texas Hold'em.

The players' actions, and the stochastic card deals (often denoted by a *chance player*), are represented by the edges of the tree and lead to the state resulting from that action. The leaves of the tree represent the end of a hand, and with each leaf we associate a gain or payoff amount z_i to player i, equaling the number of bets gained (or lost) by player i. In the case of Texas Hold'em,

$$\sum_i z_i = 0 \tag{1}$$

at all leaf nodes, as one player's win always equals the total number of bets lost by the other players. A game satisfying (1) at all leaf nodes is called a *zero-sum* game. Figure 1 shows the top portion of the tree for Limit Texas Hold'em with two players, labeled 1 (the dealer) and 2, where the bets have a value of 2 during the preflop and payoffs are represented from player 1's perspective.

As a player's hole cards cannot be viewed by the other players, Texas Hold'em is an imperfect information game. In the extensive form game tree, this fact is represented by partitions, one for each player, of the game states into groups called *information sets*. They are built so that two states are indistinguishable to player i if and only if they are contained in the same information set of player i. See Figure 2 for an example of two states falling into the same information set I.

A *strategy* σ_i for player i is an assignment mapping each information set to a probability distribution of taking each action at the information set. For instance, assigning the probability of a raise to 1 at information set I of Figure 2,

$$\sigma_1(I, \text{raise}) = 1,$$

and setting fold, call, and raise probabilities to be equal at every other information set I',

$$\sigma_1(I', a) = \frac{1}{3} \text{ for } a \in \{\text{fold, call, raise}\},$$

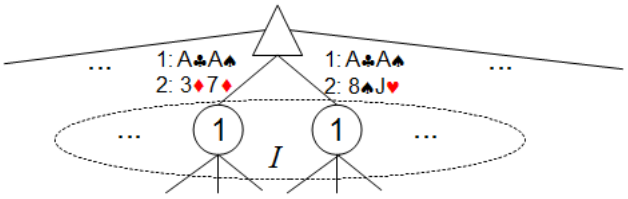

Figure 2. An example of two states which are in the same information set for player 1.

is a strategy for player 1 (though not a very good one) for playing Limit Texas Hold'em. A vector of strategies $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ for an N-player game is called a *strategy profile*. For ease of notation, (σ'_i, σ_{-i}) represents the strategy profile σ, except with σ_i replaced by σ'_i.

With each player i, we also associate a *utility function* u_i which that player attempts to maximize. The utility function u_i of player i maps each strategy profile σ to a real number by computing the weighted sum of the payoffs z_i over all terminal nodes, according to the probability of that terminal node being reached if all players play according to σ.

To finish this section, we introduce the concept of *hand strength*, as described in [8]. As we will see in Section 3, it is often useful to be able to rank different hands according to a particular metric. During the river and showdown phases of Texas Hold'em, the hand strength HS of a hand is simply the probability that the hand is better than a uniformly random opponent hand. During the preflop, flop, and turn rounds, we can compute the expected hand strength, $E[HS]$, of a hand by rolling out all possible future community cards in addition to rolling out all possibilities of an opponent's hole cards. Similarly, we can also compute the expected hand strength squared, or $E[HS^2]$. This metric incorporates the variance in hand strength of a particular hand and is a useful metric for detecting hands that can drastically improve in later rounds. We expect $E[HS]$ and $E[HS^2]$ to be good estimates of how likely we are to win in a showdown.

3 Early Efforts

In this section, we present some of the computer poker bots that have been created through techniques different from those used by the current state-of-the-art programs. We also introduce the basic concepts applied by the best computer poker programs today in Section 3.3. A more detailed and broadened discussion can be found in [8], and pieces of that discussion are revisited below.

3.1 Simulation-based approach

Poki [2] is an early poker bot which played Limit Texas Hold'em, and was primarily designed to play competently in Ring games. In brief, Poki works by using expert knowledge for preflop strategy, followed by Monte Carlo roll-outs to determine post-flop play. These roll outs are essentially "random" simulations of how the hand could play out. For each possible action Poki can take, the winnings of the simulations corresponding to that action are averaged together. The action with the highest-valued simulation average is then taken.

The random nature of the action evaluation was poor, as it is an inaccurate model of how competent opponents play. The program was thus never a match for professional human players, particularly in heads up scenarios. As computer poker research has shifted to Heads-Up play, further exploration of simulation-based programs has been put aside.

3.2 Adaptive programs

We now focus on programs for Heads-Up Limit Texas Hold'em. This 2-player version is much smaller in terms of the number of game states, but is also more strategically difficult to play. A player must play more hands than in a Ring game, as otherwise too many blinds will be forfeited to the opponent.

An adaptive program often creates a model of the opponent's strategy and attempts to exploit that strategy as best as possible. The programs Vexbot [3] and BRPlayer [13] are two such opponent modeling programs. Throughout a series of hands against a single opponent, Vexbot and BRPlayer record the frequency at which the opponent makes each action at each stage of a hand. These frequencies are then used as the action probabilities for our opponent within a miximix search [7], an extension of traditional minimax search to games with stochasticity and where probabilistic action selection is desired. In addition, on each showdown, these programs update a histogram which stores the frequencies of hand strengths HS revealed by the opponent for the current pot size. The histogram is used to help estimate the value of the terminal nodes within miximix search. However, there are many different hands which the opponent could be holding, and so to generalize learning, the frequencies of similar hand strengths are combined. The only difference between Vexbot and BRPlayer is how this opponent showdown information is modeled; BRPlayer distinguishes between more types of hands than Vexbot [13, p. 73].

The advantages of adaptive programs like Vexbot and BRPlayer are typically outweighed by the disadvantages. After several hands, the opponent model can become very accurate at predicting play, which can lead to good ex-

ploitation of the opponent. However, Vexbot and BRPlayer typically require several thousands of hands before such a model is achieved. Before a decent model is acquired, they play very poorly, as often the initial deals can greatly skew their beliefs. For example, if the opponent catches some unlucky breaks early on with the card deals, Vexbot and BRPlayer can unwisely believe that they can win every hand with high probability. Finally, these bots are no match for the equilibrium bots, which we now introduce.

3.3 Equilibrium bots

The current best known approach to building a computer poker bot for Heads-Up Limit Texas Hold'em is by approximating a Nash equilibrium [10] (see Section 4.3 for evidence). For a general extensive-form game with N players, a strategy profile $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ is a *Nash equilibrium* if no individual player i can gain utility by deviating their strategy from σ_i; more precisely, for all $i = 1, 2, ..., N$,

$$\max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i}) \leq u_i(\sigma).$$

The concept of a Nash equilibrium is most powerful in 2-player, zero-sum settings. If we (player i) play a strategy which is a component of a Nash equilibrium strategy profile σ, then we *guarantee* our expected utility to be no worse than $u_i(\sigma)$; if our opponent deviates from this equilibrium, then we can only do better. Note that an arbitrary extensive form game can have an arbitrary number of Nash equilibria associated with it; however, for two-player, zero-sum games, such as Heads-Up Limit Texas Hold'em, all Nash equilibria have the same value to both players. This means that if σ and σ' are Nash equilibria of a two-player, zero-sum extensive form game, then

$$u_1(\sigma) = u_1(\sigma') = -u_2(\sigma') = -u_2(\sigma).$$

So, how can we compute a Nash equilibrium of Heads-Up Limit Texas Hold'em? The game has over 10^{17} nonterminal game states [8, p. 22] and is far too large for current techniques to find an exact solution. Instead, we must approximate a Nash equilibrium strategy. [8] describes two steps for doing so, which we now present.

First, we relax the problem from finding an exact solution to finding a solution within ϵ of a Nash equilibrium, for some small value of ϵ. An *ϵ-Nash equilibrium* for an extensive form game with N players is defined as a strategy profile $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ where each player i can gain no more than ϵ by deviating from σ_i; i.e., for all $i = 1, 2, ..., N$,

$$\max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i}) \leq u_i(\sigma) + \epsilon.$$

Thus, in practice, we find ϵ-Nash equilibria for as small a value of ϵ as possible. One approach to finding ϵ-Nash equilibria is constructing a linear program from the extensive form game tree and solving it with an LP solver [9].

Finding an ϵ-Nash equilibrium, however, is still intractable for games as large as Heads-Up Limit Texas Hold'em. To combat this, we must employ a state-space abstraction to reduce the number of game states down to a more feasible size. One method for abstracting Texas Hold'em is to partition all possible combinations of a player's hole cards and community cards into a fixed number of buckets, based on some similarity metric. We then create an *abstract game* by regarding all information states in the full game that fall into the same bucket as a single state in the abstract game. Thus, in an abstract game, all hands within the same bucket are indistinguishable, and hence are played in the same manner by a strategy for the abstract game. If we bucket the cards in a way which places hands warranting similar strategic plays together, then a good strategy in the abstract game is likely to perform well in the full game too.

So, equilibrium bots for Heads-Up Limit Texas Hold'em compute an ϵ-Nash equilibrium strategy of an abstract game, and use it directly as a strategy for playing the full game. For bucketing abstractions, hands with similar $E[HS]$ or $E[HS^2]$ values (or both) are commonly grouped together. Several ϵ-Nash equilibrium bots have been created using the techniques outlined above, including the PsOpti family of bots [4]. Note that equilibrium bots do not adjust their strategies in any way during a match and do not attempt to find weaknesses in their opponent's play like adaptive programs do. Thus, rather than playing to win a lot of money, they are playing not to lose a lot of money.

None of the early equilibrium bots were world class players. This was mainly because the linear program could only fit into computer memory when the abstraction radically reduced the size of the state space. It was not until a breakthrough algorithm for finding ϵ-Nash equilibria in larger abstract games was developed before computer poker programs could achieve professional-level play. We discuss this algorithm in Section 4.

4 Counter-factual Regret Minimization

While linear programming approaches can generate ϵ-Nash equilibrium strategies in small abstract games, we want to be able to find such strategies in much larger abstractions of Heads-Up Limit Texas Hold'em. The Counter-Factual Regret Minimization algorithm (CFR) [14] allows us to do just this. Section 4.1 presents the theories behind CFR, Section 4.2 summarizes the steps of the algorithm, and Section 4.3 discusses the successes of CFR.

4.1 Theoretical underpinnings of CFR

CFR relies mainly on the concept of *regret*, which measures how much we "regret" playing a given sequence of T

strategies, each played for one hand, as opposed to a single fixed strategy for all T hands. More formally, consider a sequence of strategy profiles $S = (\sigma^1, \sigma^2, ..., \sigma^T)$. The *average overall regret* [14] of player i with respect to S is

$$R_i^T = \frac{1}{T} \max_{\sigma_i^*} \sum_{t=1}^{T} \left(u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t) \right).$$

The key ingredient linking regret and Nash equilibria is the following theorem:

Theorem 4.1 ([14], Theorem 2) *In a two-player, zero-sum game, if both players' average overall regret with respect to $S = (\sigma^1, ..., \sigma^T)$ is less than ϵ, then $\bar{\sigma}^T$ is a 2ϵ-Nash equilibrium, where $\bar{\sigma}^T$ is the average strategy profile of S.*

The average strategy profile of S can be quickly acquired by calculating normalized weighted sums; see [14] for the details.

So, we would like to find a sequence of strategy profiles S that exhibit $R_i^T < \epsilon$ for $i = 1, 2$ and some small value of ϵ. This would allows us, by Theorem 4.1, to build an approximate Nash equilibrium by simply taking the average of the strategy profiles in S. But how can we find S? The answer comes in two steps. First, for every information set I of player i ($i = 1, 2$), we can measure an amount of positive regret $R_i^{T,+}(I, a)$ of not taking action a at information set I (for formal details, see [14]). We can use these positive regret values to bound R_i^T:

Theorem 4.2 ([14], Theorem 3)

$$R_i^T \leq \sum_I \max_a R_i^{T,+}(I, a).$$

Then, starting with an arbitrary initial strategy profile σ^0, we can iteratively construct S by applying a regret minimizing technique known as *regret matching* [14, Appendix 2]; for $t = 1, ..., T$:

$$\sigma_i^t(I, a) = \begin{cases} \frac{R^{t-1,+}(I,a)}{\sum_{a' \in A(I)} R^{t-1,+}(I,a')} & \text{if } \sum_{a' \in A(I)} R^{t-1,+}(I, a') > 0, \\ \frac{1}{|A(I)|} & \text{otherwise,} \end{cases} \tag{2}$$

where $A(I)$ is the set of actions available at information set I. In a nutshell, (2) says that the more you regret not taking action a in the past, the more often you should take action a now. This procedure bounds

$$\max_a R_i^{T,+}(I, a) \leq \frac{C}{\sqrt{T}}$$

for a constant C [14, Appendix 2], and hence sends $\max_a R_i^{T,+}(I, a)$ to zero as T approaches infinity. Thus by Theorem 4.2, the average overall regret R_i^T converges to 0, as desired.

Algorithm 1 CFR(T)

1: **for** $i = 1, 2$, I an info set of player i, $a \in A(I)$ **do**
2: $\quad R_i^+(I, a) \leftarrow 0$
3: **end for**
4: **for** $t = 1$ to T **do**
5: $\quad$ **for** $i = 1, 2$, I an info set of player i, $a \in A(I)$ **do**
6: $\quad\quad$ **if** $\sum_{a' \in A(I)} R_i^+(I, a') > 0$ **then**
7: $\quad\quad\quad \sigma_i^t(I, a) \leftarrow \frac{R_i^+(I,a)}{\sum_{a' \in A(I)} R_i^+(I,a')}$
8: $\quad\quad$ **else**
9: $\quad\quad\quad \sigma_i^t(I, a) \leftarrow \frac{1}{|A(I)|}$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **for** $i = 1, 2$, I an info set of player i, $a \in A(I)$ **do**
13: $\quad\quad R_i^+(I, a) \leftarrow$ calculatePosRegret$(I, a, \sigma^1, \sigma^2, ..., \sigma^t)$
14: $\quad$ **end for**
15: **end for**
16: $\epsilon \leftarrow \frac{2}{T} \max_{i \in \{1,2\}} \sum_I \max_{a \in A(I)} R_i^+(I, a)$
17: **return** Average$(\sigma^1, \sigma^2, ..., \sigma^T)$, ϵ

4.2 Summary of the CFR algorithm

While [14, Algorithm 1] gives a very detailed outline of an implementation, we give high-level pseudocode for CFR in Algorithm 1. The steps involved are as follows. CFR takes in an integer T for the number of iterations to run, and is assumed to have knowledge of the extensive form abstract game we are using. First, the regret values are initialized to zero (line 2). Then, using (2), we repeatedly create new strategy profiles based on the current regret values at the information sets (line 7). The first strategy profile created is the uniformly random strategy profile (line 9). After each new strategy profile σ^t is complete, we update the regret values to account for σ^t (line 13). Once we have T strategy profiles, we compute an upper bound for how far off our average strategy profile is from a Nash equilibrium of the abstract game (line 16), and finally return this bound along with the average strategy profile (line 17). The outputted strategy profile is thus an ϵ-Nash equilibrium of our abstract game. Note that in practice, we do not need to store all T strategy profiles in memory; instead, we can keep just one profile in memory, along with a running average of the strategy profiles seen thus far.

4.3 CFR bots

Poker programs playing ϵ-Nash equilibrium strategies found using CFR outperform previous approaches and also perform quite admirably against human competition. Other equilibrium-finding techniques have not been able to compete; for example, the linear programming approach fails in comparison as the linear program is simply too large to store in memory for any decently-sized abstract game. On

the other hand, CFR can find ϵ-Nash equilibria for much larger abstract games, which tends to lead to much better play [14, Table 1]. In addition, CFR-based bots under the name "Polaris" competed in both the 2007 and 2008 Man vs Machine Poker Competitions against top human poker professionals. While narrowly losing to professionals Phil Laak and Ali Eslami in 2007, Polaris finished on top in 2008 by defeating professionals Matt Hawrilenko and IJay Palansky in the final match [1].

5 Conclusion

This survey has discussed the foundations of computer poker research and some of the approaches that have been taken on the road to world-class play in Heads-Up Limit Texas Hold'em. Poki, Vexbot, and BRPlayer were competent programs, but they have been surpassed by bots playing strategies from an approximate Nash equilibrium of an abstract game. CFR has provided the best strategies currently known and has resulted in programs capable of defeating professional human players.

There are many more directions to explore in computer poker. Since the creation of Poki, Ring poker has received little attention, likely because of the additional challenges it presents. Firstly, the theoretical guarantee that CFR finds a Nash equilibrium (Theorem 4.1) only applies in the two-player case. Secondly, even if a Nash equilibrium can be found, it is still unclear whether playing a strategy from an equilibrium is a good idea in Ring games. Another difficult variation is Heads Up No Limit Texas Hold'em. Here, the number of actions available at each information state goes from 3 in Limit games, to a number proportional to the number of chips a player currently has. This dramatically increases the branching factor of the extensive form game tree. Thus in order to work with No Limit games, we must also abstract the actions of the players. Both Ring poker and No Limit poker are areas of current research.

Poker is a challenging game due to incomplete information and stochastic elements. In everyday life, people must cope with similar circumstances and thus the decision processes occurring at the poker table act as a simple analogue for studying how we make our commonplace choices. For this reason, we can see computer poker research having a large impact on the artificial intelligence of autonomous agents in the coming years.

References

[1] The second man-machine poker competition. `http://poker.cs.ualberta.ca/man-machine`, 2008.

[2] D. Billings. *Algorithms and Assessment in Computer Poker*. PhD thesis, University of Alberta, 2006.

[3] D. Billings, M. Bowling, N. Burch, A. Davidson, R. Holte, J. Schaeffer, and T. Schauenberg. Game tree search with adaptation in stochastic imperfect information games. In *International Conference on Computers and Games (CG)*, pages 21–34, 2004.

[4] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 2003 International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[5] M. Buro. The Othello match of the year: Takeshi murakami vs. logistello. *International Computer Chess Association Journal*, **20**:189–193, 1997.

[6] M. Campbell, A.J. Hoane Jr., and F. h. Hsu. Deep blue. *Artificial Intelligence*, **134**:57–83, 2002.

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

[8] M. Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[9] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Annual ACM Symposium on Theory of Computing, STOC'94*, pages 750–759, 1994.

[10] J. Nash. Non-cooperative games. *The Annals of Mathematics*, **54**:286–295, 1951.

[11] M. Osborne and A. Rubenstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[12] J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The world Man-Machine Checkers Champion. *AI Magazine*, **17**:21–29, 1996.

[13] T. Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[14] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, 2008.