Modeling Texas Hold'em Poker Strategies with Bayesian Networks

Konstantin Tretyakov, Liina Kamm
University of Tartu

Bayesian Networks, MTAT.05.113, September 2009

1 Introduction

Poker is an exciting game of chance, psychology and skill. Development of an algorithm for playing poker (a *pokerbot*) has long been a beloved research topic for procrastinating computer scientists. In this work we join their ranks by presenting an example of a Bayesian network–driven approach to modeling strategy for a pokerbot. This approach is, naturally, not new – a well-known Bayesian poker player dates from 1999 [3] with improvements and variations being developed ever since [4, 5]. Nonetheless, we believe there are still numerous paths to be explored in this area. For example, it is only recently that the possibility of exhaustively enumerating all the $52!/43! = 1\,335\,062\,881\,152\,000$ nine-card combinations defining the states of two-player Texas holdem has become commonly accessible.

Although the approaches we are going to discuss might be applicable, with small modifications, to a wide variety of poker games, we fix our goal here to the game of *two player fixed limit Texas hold'em*. Besides being one of the most popular poker variations in existence, it is also simple enough to fit nicely into the limited scope of this project.

2 Poker Rules

Two player fixed limit Texas hold'em [6] is played using a single shuffled full deck of cards. Players alternatively play the role of the *dealer*. Each game consists of four rounds, typically named *preflop*, *flop*, *turn*, and *river*. The first round, *preflop*, begins with each of the two players being dealt two cards face down, which is followed by a round of betting. On the *flop* three cards are dealt face up on the table, and the players proceed with a second betting round. On the *turn* one more face-up card is added to the table and one more betting round is made. Finally, on the *river* the fifth card is dealt face up onto the table and a final betting round is played. If none of the players conceded (*folded*) during the betting rounds, they disclose their private (*hole*) cards to the opponent. The

player whose *hole* cards, when combined with the 5 cards on the table, result in the highest *combination* (see Table 1), wins *the pot* – all of the money that has been placed in bets during the betting rounds.

In each of the betting rounds the players make turns, declaring a *bet*, *check*, *raise*, *fold* or a *call* on each turn. The round typically starts with a *bet*, when the player places a fixed-amount of chips on the table[1], or a *check*, which is essentially equivalent to a bet of zero. The opponent may then respond with either a *call*, by placing an equal-sized bet, or with a *raise*, which means placing an equal-sized bet *and* adding another one on top. A player may *fold* at any moment, which makes him forfeit the game and everything that was placed in bets to that moment. A round is finished when either of the players *calls*, *folds*, or if both players *check*. In most cases the rules also limit the number of consecutive *raises* within one round.

A single poker match consists of multiple games, with the final score being the number of chips won by the player in the end.

Name	**Meaning**
High card	No combination. In case of a tie the player with the highest card wins.
One pair	A pair of cards of the same value. The higher value pair wins.
Two pairs	Two pairs of cards.
Three of a kind	Three cards of the same value.
Straight	Five cards in a sequence.
Flush	Five cards of the same suit.
Full house	A triple and a pair.
Four of a kind	Four cards of the same value.
Straight flush	Five cards of the same suit in a sequence.

Table 1: Poker combinations, in ascending order of value.

3 Poker Strategy

Poker has always been one of the holy grails of the computer science community. Up to now, despite the steady progress, no pokerbot has been demonstrated to play on par with the best human players. Interestingly, the reason for such mediocre development does not lie in the inherent complexity of the game state space, as it is the case with, for instance, Computer Go. Instead, the problem here lies in the fact that poker, similarly to the game of *Rock-Paper-Scissors*, is a *symmetric zero-sum nontransitive* game.

[1] This amount is decided before the game and may vary between the rounds. For instance, a *10/20 fixed limit hold'em* usually means that the bet amount is 10 units on preflop and flop, and 20 units during turn and river.

Firstly, the game is (asymptotically) symmetric – even if one of the players (e.g. the dealer) has an advantage in a single game, in the long run that doesn't matter because players alternate as dealers. Together with the zero-sum condition it immediately implies that there may not exist an optimal strategy, which could guarantee positive winnings. Yet for any fixed strategy there should exist another one that could beat it in the long run – a feature called *nontransitivity*.

Theorem 1 (Nontransitivity of poker)**.** *By a* strategy *we understand a (possibly stochastic) function, that, given all available information about the current match (i.e. the hole cards, the cards on the table, previous actions of the opponent), decides the player's next action. For any given known strategy s there exists another strategy that would play no worse than s.*

Informal proof. Consider two kinds of strategies – those that depend on the hole cards, and those that do not. Suppose s does not use the hole cards (i.e. it is a "blind" strategy). In this case the winning strategy for the opponent would be to use the knowledge of his hole cards and the player's strategy to compute precisely the odds of winning and act in accordance with the maximum expected utility principle. Clearly, this would guarantee an edge over s, which may not "know" the utilities of his actions as precisely and will thus act suboptimally. Now suppose that the actions of s do depend on the hole cards. In this case knowledge of s provides the opponent with a way of inferring probabilistic information about the player's cards by observing his actions. This again allows to compute the odds more precisely than whatever s does, which results in nonnegative winnings in the long run. □

As an example consider three kinds of strategies - *reserved* (only play sure hands), *active* (avoid weak hands, but otherwise play actively) and *bluff* (always play strongly). It is well known that a *reserved* strategy would most often beat an *active* one, that an *active* strategy would bust a *bluff*, and that a *bluff* would beat a *reserved* strategy.

What follows from the above observations is that we shouldn't be so much interested in a strategy, which would be optimal against all opponents, because it would necessarily have zero payoff against every particular opponent. Instead, it makes more sense to limit the potential space of opponents and then attempt to *learn* the opponent's strategy in order to gain better knowledge of his hole cards. Observations seem to indicate that the best human poker players are highly adaptive in their strategies, which means that a pokerbot aiming to compete with them must be at least as good. However, the development of a learning pokerbot is a highly nontrivial task, necessarily combining basic game theory with the statistical learning. This leads us to the answer to the question why computer poker is so hard. It also follows that any pokerbot algorithm that considers only a very limited amount of information about the opponent's play is not going to exceed level of a human amateur, at best.

4 Bayesian Network Models for Poker

Although Bayesian networks provide convenient means for probabilistic inference and graphical modeling, they are not meant for construction of adaptive algorithms. It follows that a purely Bayesian network-based pokerbot is not going to beat a proficient human player – a result that has already been observed in the work of Korb et al [3]. Nonetheless, the formalism of Bayesian networks provides a good starting point for understanding the problem and might be useful as a part of a more complex system.

4.1 Board state

Let us consider a fixed timepoint during a game. From the point of view of the player, it can be represented by a Bayesian network shown in Figure 1.

The random variable *Deal* corresponds to a sample of 9 cards picked uniformly from the deck. Two of the cards make up the player's hand, two correspond to the opponent's hand, and five remaining are the board cards. The player can observe his own hand and the cards on the board (depending on the round there will be 0, 3, 4 or 5 cards there). The variable *Win* denotes the resolution of the game, if it proceeds to showdown. It can take values 1, 0 and -1 depending on whether the complete 9-card deal is a win for the player, a draw, or a loss. The variables *Current Pot (CP)* and *Final Pot (FP)* denote the state of the pot at the current moment and by the end of the game. Both *CP* and *FP* are represented as pairs of numbers $(\mathrm{CP}_{\mathrm{pl}}, \mathrm{CP}_{\mathrm{op}})$ and $(\mathrm{FP}_{\mathrm{pl}}, \mathrm{FP}_{\mathrm{op}})$ – the amounts of money bet by each player respectively. The nodes *Player Strategy* and *Opponent Strategy* define the betting actions of the players during the game. For visual clarity we omit the dependence of the opponent's strategy on *CP*, *CB* and *OH*. The *Fold?* decision node is a shorthand representing the dependence of the outcome on the folding of either of the players. Finally, the

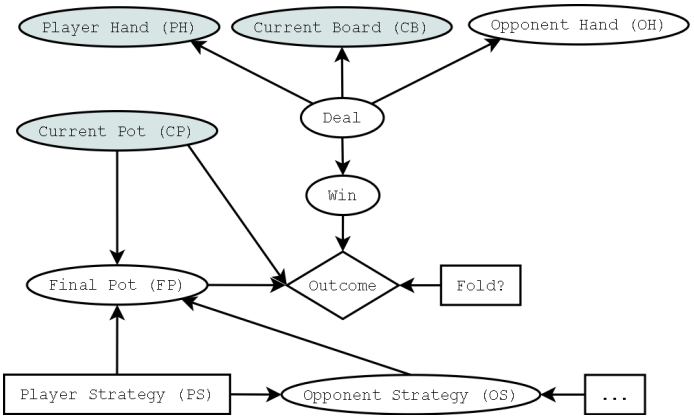

Figure 1: Poker game state.

Outcome is a utility node defined as follows:

$$\text{Outcome} = \begin{cases} \text{FP}_{\text{op}} + \text{CP}_{\text{pl}}, & \text{if } \textit{Win} = \textit{1} \text{ or opponent folds,} \\ (\text{FP}_{\text{op}} - \text{FP}_{\text{pl}})/2 + \text{CP}_{\text{pl}}, & \text{if } \textit{Win} = \textit{0}, \\ \text{CP}_{\text{pl}} - \text{FP}_{\text{pl}}, & \text{otherwise.} \end{cases}$$

In theory, we could now use the described network to compute the optimal *Player Strategy* in accordance with the maximum expected utility principle. Unfortunately, however, this would only be practical for rather simple models of *Opponent Strategy*. Besides, the network only represents the static state of the game at a given timepoint, and is therefore incapable of inferring information about *Opponent Strategy* from previous observations. Finally, the specification of conditional distibution of the *Opponent Strategy* node $\Pr[\text{OS}|\text{CB}, \text{OH}, \text{CP}, \text{PS}, \dots]$ is too unwieldy. In the following we try to address these issues.

4.2 Representing Strategy

The strategy (i.e., the set of player decisions) for fixed limit holdem games can be represented in numerous ways. In some cases a very high-level representation, such as {*reserved*, *active*, *bluff*} might be appropriate. In other cases a detailed account of the complete sequence of *bets*, *raises* and *fold* is desired. In this work we propose a middle ground, which seems both easy to represent and detailed enough for practical implementation. For each point in the game, we define the player's *current strategy* as one of the following choices, representing his decision of how the current round should be played:

Name	**Meaning**
0/0	Check. Fold on any opponent's bet.
0/1	Check. Call one opponent's bet.
1/1	Bet once or call one opponent's bet. Fold if the first bet is raised.
1/2	Bet once or call one opponent's bet. If opponent raises, call the second bet.
2/2	Try to initiate at least a double bet. If the opponent reraises to the third time, fold.
2/3	Try to initiate at least a double bet. Call a third raise.
3/3	Accept at least triple bet, but no more.
3/4	Accept a triple or a quadruple bet.

Table 2: Player's current strategy choices.

Depending on the desired granularity of representation, the overall player's strategy may now be represented as a sequence of *current strategy* values – one

per turn, one per round, one per game, or even one per match. In this work we consider a manageable case of a single *current strategy* per betting round. Clearly, the player may only observe the opponent's betting pattern but not directly the strategy. For instance, suppose the opponent bets and the player calls. The player may now infer that the opponent's strategy is **1/1** or anything higher than that. This uncertainty may be naturally modeled as an additional node in the network:

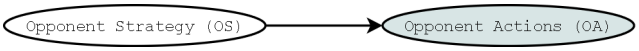

Figure 2: Strategy is observed in the form of actions.

4.3 Representing Player Hands

So far, we modeled the observable state of the board using the random variables *Player Hand (PH)* and *Current Board (CB)*. This leads to a enormous number of states for each of these nodes and is thus highly impractical. Let us observe that the player would mostly use the knowledge of these cards for only two purposes. Firstly, he can get an idea of his winning chances by estimating Player Odds = Pr[Win|PH, CB]. Secondly, he might examine the board cards in relation with the opponent's actions to get an idea of the opponent's chances and thus further improve the estimate for the *Win* variable. This immediately suggests that replacing the node *PH* with *Player Odds (PO)* will barely loose any generality. We perform a similar step for *OH*, replacing it with Opponent Odds = Pr[Win|OH, CB]. Finally, we drop the node *CB*, presuming that most of the important information from it has remained in the two newly added nodes. The resulting network (together with the modification from the previous section) is presented in Figure 3.

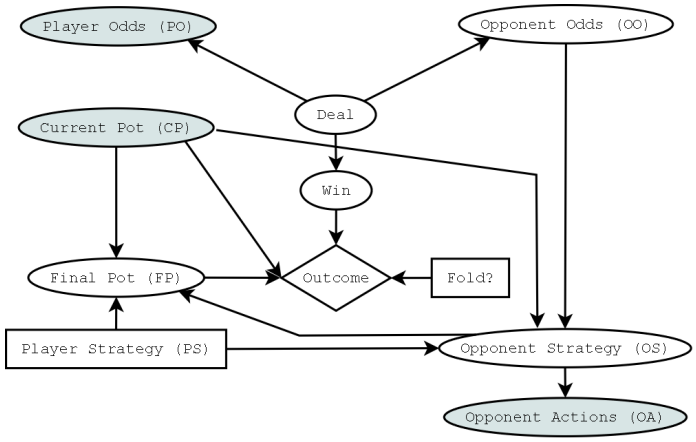

Figure 3: Network for a single round.

4.4 Modeling the Opponent

Note that the network in Figure 3 is only one step away from being a properly specified ready to use Bayesian network. What lacks is the specification of the opponent strategy, i.e. the conditional distribution $\Pr[\mathrm{OS}|\mathrm{OO}, \mathrm{CP}, \mathrm{PS}]$. As we already discussed before, the choice here is somewhat arbitrary and always suboptimal (indeed, if we knew the strategy of an "optimal" opponent, we could forget the whole Bayesian network business and simply play this strategy ourselves). For all practical purposes one would probably do well by picking something recommended by the "Poker for Dummies" textbook as a candidate for the opponent's strategy. Another good option is to learn it from the data. As both of the options met some problems on their way to the project's deadline, we must limit ourselves to a manually specified example of how a very simple opponent's strategy could look like.

A an amateur opponent's strategy might, in fact, only use the *OpponentOdds (OO)* of the cards to decide:

$$\mathrm{OS}(\mathrm{OO}) = \begin{cases} \mathbf{0/0}, & \text{if } \mathrm{OO} < 0.2, \\ \mathbf{0/1}, & \text{if } 0.2 \leq \mathrm{OO} < 0.4, \\ \mathbf{1/2}, & \text{if } 0.4 \leq \mathrm{OO} < 0.6, \\ \mathbf{2/3}, & \text{if } 0.6 \leq \mathrm{OO} < 0.8, \\ \mathbf{3/4}, & \text{if } 0.8 \leq \mathrm{OO}. \end{cases}$$

As noted in the previous sections, by using a fixed model of the opponent we ignore an important aspect of poker – the potential for adaptation. To enable the network to learn the opponent's strategy we must parameterize the variable *Opponent Strategy*, i.e., introduce an additional node *Opponent Strategy Type (OST)*. For example, this might have values {*reserved*, *active*, *bluff*}, and influence the strategy by shifting the abovementioned limits on the odds up, down, and significantly down correspondingly.

Finally, in order to be able to carry the knowledge gained about the opponent through multiple rounds and games, the separate networks for all rounds must become connected sequentially through the *OST* nodes. This would allow to learn at least those opponents, whose strategy types do not change quickly. The overall final design of the proposed network is given in Figure 4.

5 Discussion and Future Work

In the course of this project we have developed a novel architecture for a Bayesian network-based poker player. The proposed architecture is based on sound ideas, knowledge of previous work and extensive experimentation with a number of different models. Unfortunately, we have not yet found a way to execute the designed network and evaluate its performance in practice due to a hard deadline limiting the scope of the project. We have implemented a fast enumerator of 9-card poker states and still plan to run it on the grid in order

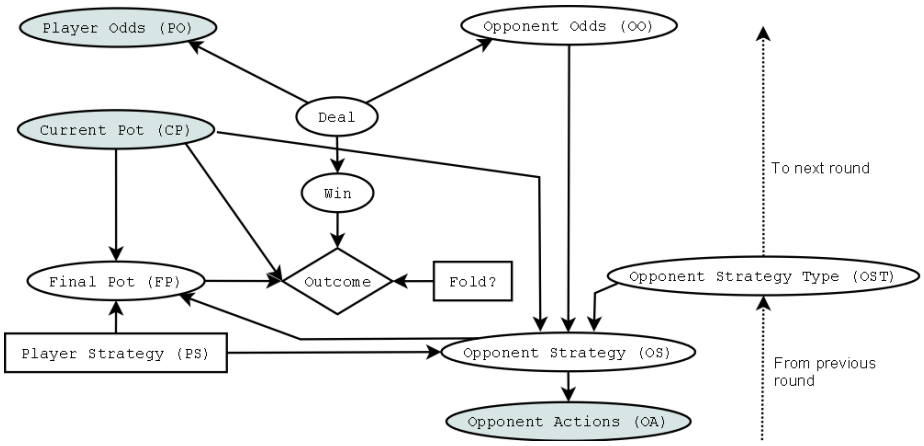

Figure 4: Final network design.

to precompute the distribution $\Pr[PO, OO, Win]$, which is required for the execution of the network. The second thing that needs to be done for that is the custom implementation of the network logic. The existing software turned out to be unsuitable for dealing with the ordinal nodes *CP* and *FP*. Finally, it would be nice to evaluate the network's performance against a different pokerbot.

References

[1] Bowling, M., Johanson, M., Burch, N., Szafron, D. *Strategy Evaluation in Extensive Games with Importance Sampling.* Proceedings of the 25th Annual International Conference on Machine Learning (ICML), pp 72-70, 2008

[2] Johanson, M., Zinkevich, M., Bowling, M. Computing Robust Counter-Strategies Advances in Neural Information Processing Systems 20 (NIPS), pp 721-728, 2008

[3] Korb, K. B., Nicholson, A. E., and Jitnah, N. *Bayesian poker.* In UAI'99 - Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence, Sweden 1999, pp. 343-350.

[4] Nicholson, A. E., Korb, K. B., and Boulton, D. *Using Bayesian Networks to Play Texas Hold'em Poker.*

[5] Terry, M. A, Mihok, B. E. *A Bayesian Net Inference Tool for Hidden State in Texas Hold'em Poker.* `http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring-2005/26C3A790-77CA-460A-B97B-CE26E3BFCCF4/0/mihokterry.pdf`, last viewed 20.06.2009

[6] Wikipedia - Texas hold'em, `http://en.wikipedia.org/wiki/Texas_hold_'em`, last viewed 15.06.2009.

[7] Zinkevich, M., Johanson, M., Bowling, M., Piccione, C. *Regret Minimization in Games with Incomplete Information.* Advances in Neural Information Processing Systems 20 (NIPS), pp 1729-1736, 2008