Automated Heads-Up Poker Player

Hossein Karkeh Abadi Reza Takapoui

December 12, 2014

Abstract

In this paper we describe designing and implementing an automated heads-up Texas Hold'em poker player. We use reinforcement learning techniques to simultaneously explore the state space and exploit the acquired knowledge from the opponent. We consider a Bayesian model for the agent and the opponent. The implementation is done in python which consists the the user interface and the algorithm.

1 Introduction

Implementing an automated poker player is challenging since poker involves elements of uncertainty, randomness, strategic interaction, and game-theoretic reasoning. Heads-up poker is a form of Texas Hold'em poker that is played between two players. Poker has a rich story of study among game theorists, mathematicians, and economists. Recently, there has been lots of research into developing strong programs for playing poker. In [1] authors introduce the Bayesian Poker Program (BPP), which uses a Bayesian network to model poker hands and the opponent's playing behavior. In [2] an algorithm has been developed to compute the approximate jam/fold equilibrium strategies in tournaments with three players. In a jam/fold strategy, the actions player can take is restricted to either folding or going all-in. This strategy known to be the near-optimal in the two player tournaments. A heads-up no-limit Texas Hold'em poker player (we will define these terms later) called Tartanian, has been introduced in [3]. Tartanian uses a discretized betting model to reduce the size of the strategy space. It also, benefits from a card abstraction model to decrease the problem size. In [4] a poker program called Poki has been developed. Poki uses reinforcement learning techniques to explore and construct statistical models for each opponent, and exploit based on the observed patterns.

In this project, we aim to discuss implementing an automated heads-up poker player. Our approach to this problem is breaking the problem into a sequence of simplified problems. We will describe this hierarchy of problem complexities in later sections.

The rest of this report is organize as follows: in §2 we describe the rules of heads-up poker and the dynamics of the game. In §3 we will describe our main model for the project and demonstrate the hierarchy of simplified problems and their complexity. §4 describes our implementation. Finally section §5 discusses the result.

2 Rules

We are going to build our model based on *Doyle's game* which was used in the 2007 Association for the Advancement of Artificial Intelligence (AAAI) Computer Poker Competition. The game is played between two players that we will call player A and player B.

- **Blinds:** Every hand, both players start with 1000 chips. In odd hands (every other hand), player A is the *small blind* and contributes 1 chip to the pot, while player B is the *big blind* and contributes 2 chip to the pot. In even hands the role of two players is reversed.
- **Pre-flop:** Two players are dealt random cards (face down) which are called *hole cards*. Then the small blind can either *fold* (*i.e.*yield all the chips in the pot to the other player), *call* (contribute chips to the pot such that the number of contributed chips from two players are equal), or *raise* (contributing more chips to the pot than the opponent). Notice that the famous *all-in* action is a especial case of raising. The betting process goes on until one player stops raising (and folds or simply calls).
- **Flop:** Three community cards from the rest of the deck are shown. Starting from the big blind, the betting process starts over similar to pre-flop. Unlike pre-flop where the players are only using their hole cards to make actions, here the players are getting more information about the community cards.
- **Turn:** A fourth community card dealt face up. The betting process is similar to flop.
- **River:** A fifth (and last) community card is shown. A final round of bets takes place similar to flop and turn.
- **Show down:** In the event that none of the players fold until the end of river round, two players make the best combination of five cards out of seven cards (two hole cards and five community cards). The player with a better combination wins the pot. In the case of two equally ranked hands, the pot is split.

3 Model

In this project we will implement the *limited* Texas Hold'em heads-up poker. As we described in section 2, every hand proceeds in four states pre-flop, folp, turn, and river. In the limited poker, the set of all valid actions for agents is fold, check, bet two chips. In our model, for each agent we define two hidden random variables F, and H, which are not revealed to the opponent. The vector F, is a feature vector representing the agent's properties, *i.e.*, her level of aggression, etc. and incorporates the agent's way of playing in all levels. H is a random variable representing the agent's hole cards. Note that H is unique for each hand of the game and F is unique for each agent.

3.1 How the opponent agent plays poker

As mentioned before, the agent will consider two hidden random variables H and F as described above for the opponent and tries to update her beliefs about them as she plays. Figure 1 shows the probabilistic graphical model (PGM) for each state of a particular hand that the agent uses to update her beliefs. In this model, B represents the dealt cards so far on the board, S is the story of the hand so far (*i.e.*the actions in the previous states of the hand), and A is the opponent's action. Note that B, S, and A are all observed by the agent, and she should use them to update her beliefs about hidden random variable H. In our model, we defined an $F \in \mathbf{R}^{13}$ as the latent feature vector. $F_1 \in [0, 1]$ is the probability of making a bet based on bluff (when the agent would not make a bet with his strategy). The agent ma

3.2 How the agent plays poker

The value of F, will be updated whenever the opponent's hole cards revealed at the end of a hand.

Given the updated beliefs about F and H, the agent computes the expectation reward, $\mathbb{E}(R)$ that she can get out of this hand. If $\mathbb{E}(R) < 0$, she will fold any bet, otherwise she chooses to call, check, or even bet if the expected value is above some threshold. So, whenever the agent sees an action from the opponent, she updates her beliefs, and act accordingly to maximizes her reward. The agent can sometimes chooses to explore more, by being aggressive to get more information about the opponent's hand instead of just exploiting her beliefs.

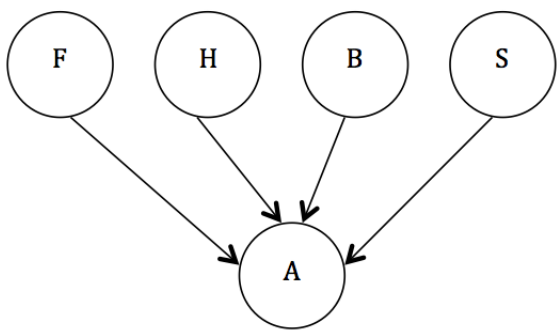

Figure 1: opponent's action PGM

We will report the increase in the expected collected reward when we move on to the more complicated models.

4 Implementation

We use Python for our implementation. So far, our implementation includes several classes that facilitate the user interface and enables us to play poker with an agent. We describe the infrastructure here:

- `pokerCards` is a class which consists of cards. Each `pokerCard` has two parameters rank and suit. Rank is a number in $\{2, 3, \cdots, 14\}$ (where 14 means Ace) and suit is a number in $\{1, 2, 3, 4\}$.

- `deck` is a class which contains a deck of cards. `deck` includes `pokerCards` as a subclass and defines methods for shuffle, pop, and peek.

- `player` consists of player objects. Each object contains information on the player like the role (small blind or big blind), hole cards, stack (the number of chips), and the set of possible actions a player can do.

- `table` is a user interface to show the table, card, players, and status of the game.

- `handEvaluator` contains functions to evaluate poker hands. In particular, the method we use is inspired by **??**. The idea here is to assign scores to sets of five, six, or seven cards such that the hand with higher rank has a higher score and equally-ranked hands have the exact score. Specifically, `hadEvaluator.handEvaluator` function receives the hole cards and the board (with three to five cards) as its arguments and assigns a real number in $[0, 1]$ to the union of hole cards and the board. If `handEvaluator(h1,b) > handEvaluator(h2,b)`, it means that the rank of best five cards in $\texttt{h1} \cup \texttt{b}$ is higher than the best five cards in $\texttt{h2} \cup \texttt{b}$. Notice that this doesn't mean the `h1` does not have any chance to win.

In order to decrease the number of states, we are working on `cards abstraction`. The basic idea is to use abstract class of cards to represent the rank of a hand. For example, if multiple cards will make a flush, we consider all of them as the same event. We will explain this more in our final report.

5 References

1 S. Ganzfried and T. Sandholm, Computing an Approximate Jam/Fold Equilibrium for 3-player No-Limit Texas Holdem Tournaments

2 K. B. Korb and A. E. Nicholson and N. Jitnah, Bayesian Poker

3 The challenge of Poker ,D. Billings, A. Davidson, J. Scharffer, D. Szafron

4 A heads-up no-limit Texas Hold?em poker player: Discretized betting models and automatically generated equilibrium-finding programs, A. Gilpin, T. Sandholm, T. B. Sorensen

5 http://www.suffecool.net/poker/evaluator.html

Figure 2: visual representation of the table