Student Names: Eduardo Porto & Lester Hernandez Alfonso

PID: 4714449 & 4017986

Course: CAP 4630 – U01

Due Date: 12/07/2017

Team: Electric Caviar Racers

Final Project Report

1. Intro

The objective of this project was to create an agent that plays 3-player Kuhn poker, a simplified 4 card (1 card per hand) version of poker against other student's agents. The name of our agent was Electric Caviar, taken from our team name which was selected through a random team name generator. The game is played using the dealer from the computer poker competition (computerpokercompetition.org), which specifies the format for how our agent plays. Electric Caviar (our agent) is actually a heavily modified version of the "example_player" agent from computer poker competition code download package.

2. Approach

The approach we took involved using the "example_player" agent as a starting point and implementing the Nash Equilibria set of strategies described in Tables 2 and 3 from "A Parameterized Family of Equilibrium Profiles for Three-Player Kuhn Poker" (Szafron, Duane, et al.), then implementing learning around the constrained parameters that were initially chosen to be randomized (b11, b21, b23, b32, c11), but were dependent on other players' moves. The basic idea behind our agent was to estimate and remain within the equilibrium family chosen by other players in every hand, as often as possible. This would maximize our agent's utility over any statistically significant amount of hands. As described in the research paper cited above, two or more players playing different equilibrium families on the same hand is not in the best interest of any player's utility in the long run, therefore, our objective was to avoid that situation as often as possible. It is important to note that we assumed other players to be playing an equilibrium family of strategies. This assumption is certainly valid since any agent that does not attempt to play a Nash Equilibrium would always lose to one playing an equilibrium family.

The dynamic aspect of our agent was mostly ruled by the fictitious play algorithm which ensured that after every hand our agent would get closer to a real estimation of its opponents' tendencies of play. This approach is highly affected by the amount of dynamism that our opponents might exhibit, nonetheless it is a simple, and effective approach when faced against somewhat simplistic agents, which was to be expected in this project. We thought it would be interesting to implement a way to side with our weakest opponent, in the hopes of affecting the utility of our strongest opponent. As described in the research paper, we can do just that, when in any given hand our agent is second-to-move, and it holds either a J or a Q. Over statistically many hands, by often choosing the most aggressive move - that is b11 = b21 = 0.25 – our agent would transfer a maximum amount of small utility from the first-to-move to the third-to-move opponent. On the other hand, by almost never choosing the most aggressive move - that is b11 = b21 = 0 – our agent would transfer a minimum amount of small utility. The best part is that the utility of the second-to-move player is constant, as described by the research paper. Therefore, our agent's choice of utility transfer does not affect its own utility. We decided to implement this strategy because over a significant amount of hand, it would do a great job of levelling the scoreboard, which is beneficial in all circumstances.

3. Implementation

The "example_player" agent simply finds the valid moves at a given game state and plays them randomly. The only exception of the action of folding which has a low probability. We initially modified the agent so that it implemented the equilibrium a_{22}=0, b_{22}=0 and c_{22}=0 from table 3 from the paper (Szafron, Duane, et al.). It was very easy to implement as a series of if-statements that took advantage of the readily available function provided by the 'game.c' file in the code download package. Furthermore, this equilibrium was easy to verify.

The rest of equilibrium families from Table 2 and 3(Szafron, Duane, et al.) were implemented, but there were some variables that were dependent on other players' moves. These variables were the probabilities c_{11}, b_{11}, b_{21}, b_{23} and b_{32}. Since we hadn't implemented learning at that moment, we chose to randomize those parameters. Despite randomization, it was noted that they should reflect realistic values a smart player would take. For instance, c_{11} (player 3 has a jack and raises after player 1 and 2 both call) should be low for a smart player, but not zero. So, it was randomized between 0.0 and 0.5 rather than between 0.0 and 1.0. This was decided purely from the knowledge displayed in the table, which defined the values of c_{11} to be between 0 and 0.5 for the equilibrium set of families. Similar assumptions were made for the other parameter values. Since we planned to implement learning around these values it was seen as being a good starting point. We proceeded to test the modified agent against two random agents ("example_player") and the result was positive, as expected, with earnings in the ballpark of +$700 for the modified agent after 3000 hands, depending on the dealer seed chosen, and game entering position.

We began implementing learning by gathering data on the other players, specifically on how they conformed to our initially randomized variables (c_{11}, b_{11}, b_{21}, b_{23} and b_{32}). Every time these estimations were “violated” they were adjusted after each hand to estimate the opponents’ actual behavior probabilities with more precision. This meant that the longer we played against other players the closer these values would reflect the opponent’s actual strategy. This is simply a version of the fictitious play algorithm described briefly in class lectures. We simply based the probabilities of our agent’s moves on the probability of our opponents making a move that would affect how our agent would play when presented with some options in a given equilibrium situation. The parameter variables dependencies are described in Table 3. Our agent’s c_{11} probability depends on the values we predict for b_{11}, and b_{21} of our opponents. The same goes for b_{11}, which depends on our prediction of what the third-to-move player’s c_{11} is. The rest of the parameters are calculated the same way, as defined by Table 3. The fictitious play algorithm is the core of our learning strategy.

The last addition to our agent consisted of a mechanism to affect the utility of our strongest opponent by either choosing to transfer the maximum or minimum amount of utility, whenever our agent is second-to-move. We implemented a way for our agent to have its own scoreboard, keeping a record of its opponents’ earnings during a game. This scoreboard is updated after every hand. The agent identifies its opponents as right_player and left_player when the game begins, which simply indicates where each of our opponents is located from our agent’s perspective throughout the duration of the match. Our utility balancing/transferring function consisted of playing $b11 = b21 = 0$, whenever the player to our right had a better score than the one to our left, and $b11 = b21 = 0.25$, whenever the player to our left had a better score than the one to our right. This way our agent always sided with the weakest player, in an attempt to even the playing field and increase its chances of victory.

4. Further Improvements

In addition to replacing fictitious play by a more robust algorithm, which would improve our agent’s performance significantly, we had another improvement in mind that we did not get to implement. The idea that our agent could learn from other players was trivial, but in real life, humans don’t base their decision-making strategies solely on how their opponents play, but also on how opponents will react to their strategies. We wanted to implement a way for our agent to estimate not only the opponents’ strategies, but also what the opponents think our strategies are, in an effort to remain unpredictable within the thresholds of the equilibrium families. This could be achieved by simply keeping track of all opponents moves, and deviating from a set of strategies whenever an opponent seems to be “guessing” our strategies with an “abnormal” frequency, and thus earning an “abnormal” amount of utility in a predefined period of time. Of course, that is a simple implementation, and an effective function would need to be specified that would provide us with the amount of deviation from our current strategy that would be convenient based on factors like the scoreboard, or even the state of all the opponents. Our team would have appreciated more time to develop and improve our agent, as this particular subject was very appealing to us. However, we understand the constraints in play, and will certainly continue making improvements to our agent in the future.

5. Sources:

1- Szafron, Duane, et al. “A Parameterized Family of Equilibrium Profiles for Three-Player Kuhn Poker.” poker.cs.ualberta.ca/publications/AAMAS13-3pkuhn.pdf.

2- Annual Computer Poker Competition Website. http://www.computerpokercompetition.org/. (Protocol, Rules, and sample code)