B351 AI Project: Texas Hold 'em

Samuel Maginot[1] and James Gregory[1]

[1] *Indiana University, Bloomington, IN, USA*

May 5, 2017

In this project we attempted to create an algorithm that accurately models the decision making process of a poker player playing the card game Texas Hold'em. To do so we utilized basic probability principles that a poker player may realistically consider, such as the number of hands that could beat theirs at any point in time. We did not utilize more complex AI principles, such as A*, in this algorithm, but rather collapsed the decision making process into a single computation.

1 Introduction

Ever since the beginnings of human civilization, games have been used to test the intelligence and aptitude of people in a nonviolent context. Thus it is not surprising that games continue to play a large role in society to this day, as one can see from the everpresent emphasis on sports one sees in seemingly all social classes in most every developed country. There is also a great degree of interest today in many of the games that have been played for centuries or even millennia, such as chess and go, however this interest derives from a different origin than the society of today's fascination with sports. This origin is the field of Artificial Intelligence. The word itself inspires wonder and fear alike among futurists. Wonder due to the glowing promises of self driving cars and house appliances working in harmony. Fear due to the dark possibility of computer take over as prophesized in films such as *The Matrix* and *The Terminator*. There are still others who have used the topic of Machines that Think to discuss what it really means to be human, as seen in books such as *Do Androids Dream of Electric Sheep?* While many of these topics and concerns are beyond the current state of computing, the original topic, the one that caused these concerns, is not.

Seeing as games have been used to test the intelligence of humans since an early stage of civilization, it is understandable that these same notions have been brought up regarding computers. That is, the aptitude at which computers can play games can indicate the current understanding of Artificial Intelligence. This is made evident by the idea that chess would open the threshold of understanding Artificial Intelligence, much like the study of drosophila did for the field of genetics("Is Chess the drosophila of Artificial Intelligence? A Social Hostory of an Algorithm"). I do hope however that this doesn't mean that the field of Artificial Intelligence will become as arduous as breeding fruit flies and painstakingly observing them for mutations. Evidently this threshold has been met, as seen by algorithms such as Giraffe(Silver, 2016), which plays chess, and Google's go playing algorithm("Giraffe: Using Deep Reinforcement Learning to Play Chess"), which obviously plays go. There are also algorithms that play more modern games, such as *Starcraft*[1], though the algoritms that play this game seem to be less sophisticated as they typically take a more linear approach to playing the game than the previously cited examples. These algorithms show that the contemporary understanding of Artificial Intelligence is at a respectable level. However it also shows the disparity between the intelligence it takes to play games and the intelligence it takes to perform some menial tasks, such as driving cars. Although much of this may be due to the associated disparity between the danger these tasks pose. One cannot kill a pedestrian by playing chess with them, but they can by running them over with a car.

At first glance, poker seems to be fundamentally different from chess and go. Asside from the societal stigmas associated with gambling, chess and go opperate quite differently from poker. The former is played with a game board and pieces that are layed or moved around on this game board. The latter is played with cards and money. The former use more deterministic

[1].

approaches to playing games, as each player knows all information regarding the state of the game at any time. The latter however is based more upon chance and probability, as much of the information that is universally known to players of chess and go is kept secret in poker. However the approaches made to develop an Artificial Intelligence algorithm are similar across each of these games. They each typically create a state space and evaluate states in this space to determine which is the best state to proceed to. This is rather similar even in poker, where some information regarding the state of the game is left unknown, because the most important resource, the one that has the ability to change the state of the game, that is money, is uneversally known. Thus the only aspect of these algorithms that differ are the methods of evaluating possible future states.

Therefore it is not so difficult to imagine that a study of poker can advance the understanding of Artificial Intelligence much like chess is thouht to do. It can even be argued that poker can represent the real world more accurately than chess or go as much like poker, there is much that is left unknown in the real world. One does not certainly know the intentions of other drivers or pedestrians on the road. One does not certainly know that roads will be maintained properly, or even exist at all. One does not certainly know that an earthquake will not occur that may change the current state of the world drastically.

2 Playing Texas Hold 'em as an AI Problem

Our approach to Texas Hold'em did not make use of a graph of possible states, so obviously it did not evaluate different states within this graph and choose the best one. Instead it simply determines the number of possible hands that can beat the its hand and from this takes the probability that it has ha hand that could win. We decided to take this approach to Texas Hold'em as it was the most obvious approach to take. This approach is not so different to how many people play poker in real life. They determine how good their hand is and bet based on that hand. Our algorithm essentially does this. It does not attempt to track the betting antics of other players, but it does make the logical assumption that if one player is betting a lot, and there is a high probability that one has a better hand than they do, then this player has a better hand than them. However, there are certain attributes in which our algorithm is better than its human counterparts. One of these attributes is the use of random number generators to create noise to mask our betting strategy.

The method our algorithm uses to determine how many possible hands there are that are better than the robot (i.e. our AI) is relatively simple. The robot comapres its hand to all other possible hands in that state of the game. If another hand beats its hand, a count is incremented. After all possible hands are compared to the robot's hand, the ratio of the count of hands the robot lost to and the total number of possible hands gives the probability that the robot's hand will lose. This value is then subtracted from 1 to get the probability that the robot's hand will win. The formal equivalent of this explination is as follows where h_r is the robot's hand, h_o is another possible hand, H is the set of all possible hands in that state of the game[2], c_l is the number of hands that the robot loses, c_w is the number of hands that the robot wins, and $P(W)$ is the probability that the robot will win.

$$c_l = \sum_{n}^{|H|} betterThan(h_o, h_r),\ h_1, h_2 \in H$$

[3]

$$c_w = 1 - c_l$$

$$P(W) = \frac{c_w}{|H|}$$

Thus is the manner in which the algorithm calculates the probability that the robot's hand will win. The algorithm then uses this probability to determine the maximum amount of money it is willing to bet (b_M). This value is attained by multiplying $P(W)$ by itself the number of players ($|J|$, i.e. joueur, which is French for player) times and multiplies this value by the total amount of money the robot has (D, i.e. dollars). If the current bet is greater than b_M, then the robot folds. To determine the amount of money the robot will bet or raise (b_a, i.e. actual bet), the algorithm uses a non-uniform pseudo-random number generator ($RNG()$) and sets b_M minus the current call value (b_p, i.e. previous bet) as the upper threshold in this RNG (the lower threshold being zero). If the RNG returns 0, the robot calls, but if the RNG returns anything greater than 0, the robot raises by that amount. Formally, this is the proceeding:

$$b_M = P(W)^{|J|} * D$$

$$b_a = RNG(b_M - b_p)$$

$$b_M < b_p \implies fold()$$

$$b_a = 0 \implies call()$$

$$b_a > 0 \implies raise(b_a)$$

$$else\ check()$$

[4] As stated previously, we use a non-uniform RNG to produce noise to make it more difficult for other players to detect our betting strategy. This RNG uses a bell-curve distribution, so the values it returns are more

[2] note that $|H|$ will be constant for specific stages of the game, so $|H|$ that corresponds to the flop stage in one particular round will always be equal $|H|$ that corresponds to the flop stage in any other round regardless of the specific cards used.

[3] $betterThan()$ technically returns a boolean, but for the purposes of simplicity we will assume it returns 1 if it is true and 0 if false

[4] if no other option was chosen

likely to be somewhere in the middle between 0 and $b_M - b_p$. Thus the algorithm is more likely to bet relatively safely, and not bet everything if the robot receives a undeniably winning hand. This hides the fact that the robot holds a good hand from other players, so they are more likely to stay in the hand. One drawback to this approach is that other players will eventually pick up on the algorithms betting strategy despite the noise that it maskss itself with as the number of total rounds increases. However, the number of rounds is finite, and as such other players are unlikely to determine the algorithm's betting strategy during the time of the game. Unfortunately, we have not tested our algorithm with other players, so we cannot say this with certainty.

3 Code

To implement this strategy, we used relatively simple data structures. Lists were typically used to represent hands and dictionaries were used to represent some simple probability tables. For example, a dictionary was used to represent a 2-card table, which is a table that spectrumizes all of the possible combinations of cards given at the beginning of a round (i.e. the hole cards)[5]. Thus, simply sets were used to represent most everything.

One complication the use of sets could cause is an increase in time complexity. In one particular case, when evaluating the number of possible hands after the flop stage of any particular round, our algorithm takes $O(n^4)$ time to complete. However, n will always be relatively small, as the total number of cards in any game of one-deck Texas Hold'em is 52. Thus, our algorithm took always took less than 20 seconds to complete a full round on a relatively standard laptop.

4 Results

Unfortunately we do not know for certain how well our algorithm would hold up against others, as we did not test it against other human agents or other AI algorithms. We have only tested against randomly generated opponents to test if the alogrithm itself worked. However, we do believe that it would perform acceptably as the logic behind the algorithm is sound and there is some degree of noise to deter players or AIs who seek to determine our betting strategy.

Bibliography

In: URL: http://sscaitournament.com/.

In: URL: https://www.pokernews.com/strategy/on-starting-hand-charts-ranking-the-169-hands-in-hold-em-21325.htm.

Ensmenger, Nathan. "Is Chess the drosophila of Artificial Intelligence? A Social Hostory of an Algorithm". In: *Social Studies of Science*. URL: http://journals.sagepub.com/doi/full/10.1177/0306312711424596.

Lai, Matthew. "Giraffe: Using Deep Reinforcement Learning to Play Chess". In: *Review of Scientific Instruments*.

Silver, David; et al. (2016). "Mastering the Game of Go wih Deep Neural Networks and Tree Search". In: *Nature*. URL: https://gogameguru.com/i/2016/03/deepmind-mastering-go.pdf.

[5].