Poker Agent Literary Review

Eli Bildman

3/5/2021

In creating an intelligent poker agent of any kind, there are many challenging and unique problems an engineer must tackle. No limit Texas Hold'em (NLHE) is one of the most recent games to be effectively played at a high level by a computer. Top poker players were still defeating the best bots until 2019, when researchers at CMU changed that [6]. This is in contrast to chess masters, who were defeated in the late 90's [5], and Go masters, who famously lost to Google's AlphaGo in 2016 [4]. There seems to be something either uniquely human or uniquely nuanced about the abilities of strong poker players. This review covers the publications of the leading computer scientists solving these problems, and the strategies they used to create a no limit Texas Hold'em AI that can efficiently compete with the best poker players today.

Poker is a game of imperfect information. This is in contrast to games like Chess, Checkers, or Connect Four, games of perfect information [3]. In Poker, players don't have complete information about the state of the game when making their decisions: what cards other players at the table are holding, as well as what cards will be revealed as play progesses. This is what makes poker interesting for humans, but also what makes it challenging for computers.

The most generic game solving involves creating and traversing a game tree [3]. Game trees are trees where any state of the game is represented by a node. For any node β, β is connected to parent node α that represents the gamestate one move or random event before β. Leaf nodes represent terminal states of the game where no more decisions can be made. These states have deterministic payoff for all players, called utility. An incomplete game tree for 2-player Poker could look like this:

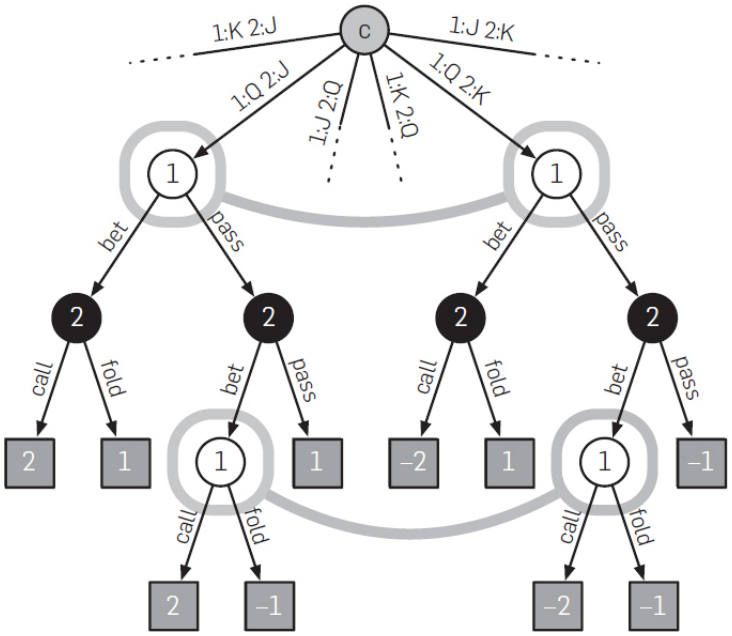

Figure 1: Incomplete Heads-up Poker Gametree [1]

This tree contains a small fraction of the total states possible, but shows how connections are made between states and their predecessors. The full 2-player poker tree has around 10^{17} states [1]. For simple games, this tree can be fully traversed in a reasonable amount of time. Meaning a computer could always make the optimal decision based on what choice leads to the highest utility leaf node. In games of more realistic size, like chess, the game tree is far too large to be traversed, so this sort of deterministic decision making won't work. Instead, the tree is traversed to the extent it can be, then the deepest states found are evaluated based on some evauluation function, and treated as leaf nodes. For example, in chess, an evaluation function could look at the number of pieces a player has at a game state and the relative value of those pieces to determine an approximate value. This can also be a good application for machine learning, as often there are patterns that indicate good game standing that are too complicated to program by hand. With an accurate enough evaluation function, this strategy can produce strong decisions.

A similar strategy could be applied to poker, but since some information is unknown to the player when making a decision, it's impossible to tell what gamestate is currently at play. Instead, the set of game states with identical public information, called an 'information set' is considered [7]. To similarly solve this problem with an evaluation function, every possible state of the

hidden information on the board has to be considered, and evaluated for. Including this uncetainty adds another layer of complexity. In poker there are $\binom{52}{2}$, or 1326 different combinations cards a player can be dealt. With 2 players, there are $\binom{52}{4}$ or 2.7×10^5 different combinations of dealings. As in the example earlier, the smallest form of poker has 10^{17} states. Even if our evaluation function could reuce this to 10^5 states and still make usful insights, this leaves us with upwards of 10^{10} nodes to explore. In order to reduce the number of computations needed, more creative algorithms are created.

In order to study these algorithms we define these standard variables and functions for imperfect information games [7]. A is the set of all game actions. I is an information set, including a number of states that are identical in public information and possible decisions. $A(I)$ is the set of possible game actions at information set I. T and t represent time, which increments one for each move by a player or random event. A strategy σ_i^t is the set of probabilities that player i at information set I_i will take any action $a \in A(I_i)$ at time t. All player strategies at time t coalesced create strategy profile σ^t. $\sigma_{I \to a}$ is a strategy profile equivilent to σ, except action a is always chosen at I. History h is a sequence of actions and random events starting from the begining of the game. $\pi^\sigma(h)$ is the probability of history h happening with strategy profile σ. Similarly, $\pi^\sigma(I)$ is the probability of reaching information set I with strategy profile σ. The counterfactual reach probability of information set I, denoted $\pi^\sigma_{-i}(I)$ is the probability of reaching I with strategy profile σ except every decision player i makes is considered to have probability of 1. Z denotes the set of terminal game histories. So, if $h \subset z$, $h \neq z$ and $z \in Z$ then h is a nonterminal game history. $\pi^\sigma(h, z)$ represents the probability of terminal history z given non terminal history h. Lastly, $u_i(z)$ is defined as the utility, or value recived for player i at terminal history z.

The most popular starting algorithm is called "counterfactual regret minimization" or CFR [7]. CFR is the process of trying to minimize "regret," which is calculated as a loss of "counterfactual value." Counterfactual value at a nonterminal history h with strategy profile σ is defined as:

$$v_i(\sigma, h) = \sum_{z \in Z, h \subset z} \pi^\sigma_{-i}(h) \pi^\sigma(h, z) u_i(z) \tag{1}$$

This is the counterfactual probability of reaching an endstate, meaning the actions of player i are considered given, multiplied by the value of that state, summed for every state reachable. This is similar to the expected value of a gamestate, but done counterfactually. Counterfactual regret builds on this and is defined as:

$$r(h, a) = v_i(\sigma_{I \to a}, h) - v_i(\sigma, h) \tag{2}$$

Regret can be considered the difference in value between changing the strategy to choose action a and not. If this regret is positive, it would be beneficial to change the strategy to make action a. Lastly, counterfactual regret at an infoset I at time T is defined as:

$$r(I, a) = \sum_{h \in I} r(I, a) \tag{3}$$

This needs to be calculated becuase, as discussed earlier, players must consider states in infosets, and can never know that true history h they're in.

From here, CFR creates a vector of running regret sums for every infoset. Whenever a given infoset is reached, new regrets are calculated and added to these sums. Strategy σ_i is then updated to represent the positive regret sums normalized to sum to one. For example, if we had a regret sum vector $[-5, 10, 8, -2]$, our resulting strategy for I would be $[0, \frac{10}{18}, \frac{8}{18}, 0]$. This process is repeated an arbitrarily large number of times in a training process, and it's been shown that [9], if implemented correctly, the strategy profile created converges to the optimal values.

CFR removes the need to consider every possible state of the unknown variables of a game because all calculations are done retroactively, when unknown variables are uncovered. However, CFR still requires calculating the value of every move that could have been made, which can take too long in large games. To tackle this issue CFR+ was developed [8]. CFR+ makes the key change of only considering positive regrets when calculating regret sums. This small difference means the training spends less time rewarding good decisions, and more time punishing bad ones which, based on studies in the same paper, makes the values converge faster in most cases.

The last addition to the algorithm is called "Regret Discounting" [2]. It's notable that for some payout distributions, CFR and CFR+ take exceptionally long to converge due to regrets calculated in early iterations. And example given in "Solving Imperfect Information Games via Discounted Regret Minimization" is the case of three choices with payoffs $[0, 1, -1000000]$. In this case, the positive regrets would be $[333332.5, 333334.5, 0]$. Since the third option has zero regret, it will never be chosen, and the strategy will consist of picking between options one and two with about 50% odds. The difference in regret between these choices is very small, so finally converging to the point where option two is chosen over one will take hundreds of thousands of iterations. The solution to this is to add a weight to the calculation of the normalized regrets. The normalization step in CFR is really taking the average of the regrets of every iteration of the training so far. Since the regrets of early iterations are relatively less important than the regrets found more

recently, these regrets can be less heavily weighted. The weight of a regret iteration is found with the equation $w = max\{T - d, 0\}$. Where d is some optional delay value to keep early iterations from being affected.

Overall, there're many approaches to solving the unique problems poker and other games of imperfect information present. These approaches tend to revolve around CFR and therefore, CFR and the additions that have been made to it will be my main design strategy in creating my no-limit Texas Hold'em agent.

References

[1] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218), 2015.

[2] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. *arXiv preprint arXiv:1809.04040*, 2009.

[3] Neil Burch. *Time and Space: Why Imperfect Information Games are Hard*. PhD thesis, University of Alberta, 2017.

[4] Sam Byford. Google's alphago AI beats Lee Se-dol again to win Go series 4-1. *The Verge*, 2016.

[5] Patrick Gebhardt. The history of chess AI. *Paessler*, 2019.

[6] Douglas Heaven. No limit: AI poker bot is first to beat professionals at multiplayer game. *Nature*, 2019.

[7] Todd W. Neller and Marc Lanctot. An introduction to counterfactual regret minimization. *Model AI Assignments*, 2013.

[8] Oskari Tammelin. Solving large imperfect information games using cfr+. *arXiv preprint arXiv:1407.5042*, 2007.

[9] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *NeurlPS Proceedings*, 2007.