2-player No Limit Texas Hold 'em AI Report

Game Theory

Tong Chen

June 21, 2020

1 Strategy Framework

My strategy is based on the winning rate of visible cards. In each round, I calculate the winning rate and choose to either fold, call, or raise. When the new public card(s) is/are revealed, I calculate the winning rate again and choose a new strategy.

1.1 Estimating the winning rate

Consider the tree representation of a extensive form game. Given the opponent strategy, the best strategy is choosing the child node with max expected utility. However, I do not have the opponent strategy, and cannot build the whole game tree. So I choose to use the expected winning rate to estimate. The expected winning probability is defined as follows,

$$W(c_i, c_p) = \mathop{\mathbb{E}}_{c_{-i}, c_{p'}}[rank(c_i + c_p + c'_p) > rank(c_{-i} + c_p + c'_p)],$$

where, c_i is the private cards of player i, c_{-i} is the private cards of the opponent player, c_p is the revealed public cards, c'_p is the unrevealed public cards, $rank()$ is the rank function of a set of cards, and the expression in the brackets represents a random variable that whether the relationship is satisfied. Similarly, we define the expected losing probability,

$$L(c_i, c_p) = \mathop{\mathbb{E}}_{c_{-i}, c_{p'}}[rank(c_i + c_p + c'_p) < rank(c_{-i} + c_p + c'_p)].$$

And let $v(c_i, c_p) = W(c_i, c_p) - L(c_i, c_p)$ be the expected rate of a state.

I choose expected rate v because it is easy to approximate by a Monte Carlo based idea. I sample c_{-i} and c'_p form the deck, and decide whether play i wins in this case. I repeat sampling many times (e.g. 10000), and then get a approximate value for $v(c_i, c_p)$.

Using expected rate v as the estimated value is reasonable for two reason. Firstly, if the two player call on all state, v is the exact expected utility of a state. Secondly, by intuition, the better card combination means higher value of a state. And the actual value can be approximated by some function of v. This approximation can be merged into our parameters of our strategy.

However, the estimation does not use the current amount in the pot and the history actions, and these factors limit the performance.

1.2 automaton strategy in each round

In the following description, I use notation v to represent the expected rate of the current state. Based on the value of winning expectation v, we decide whether to fold, call or raise. To clarify, we choose one of the four automaton strategy in this round. We use the first round as an example, and other rounds are exactly the same.

Give up If my cards is bad, I will not bet more money on this hand.

If I am asked to put more money, I fold directly. Otherwise, if two player have put the same amount of money, I call.

Wait and see If my cards is just so-so, I want to involve but do not want to bet a large amount of money.

If I am asked to put some amount of money under a threshold $p(v)$, I call. Otherwise, if the opponent has raised the price higher than the threshold $p(v)$, I fold.

Raise to some amount If my cards is good, I want to bet some money but I do not want to bet all-in.

If the minimum raise value is not higher than my expected bet value $q(v)$, I raise the price to my expected bet value. If the minimum raise value is higher than my expected bet value $q(v)$, I call. Otherwise, if the opponent has raised the price higher than a threshold $p(v)$, I fold.

Bet to the end If my cards is perfect, I want to bet as much as money on this hand.

I will raise the price to some value $q(v)$ in this step. If the opponent also raise, I will continue to raise the price to twice the current betting price.

1.3 Parameters in the strategy

In round $r(r = 0, 1, 2, 3)$, we choose three critical point $x^{(r)}, y^{(r)}, z^{(r)}$. We use **Give up** if $v < x^{(r)}$, use **Wait and see** if $x^{(r)} \leq v < y^{(r)}$, use **Raise to some amount** if $y^{(r)} \leq v < z^{(r)}$, and use **Bet to the end** if $z^{(r)} \leq$ ‘v.

Then how do we design $p^{(r)}(v)$ and $q^{(r)}(v)$? It is natural to use a constant or linear piecewise function. But I choose to use a simple but more powerful function, polynomials. Since polynomials can fit any continuous function in theoretical, I choose to use a 3-order polynomial function for $p^{(r)}(v)$ and $q^{(r)}(v)$.

$$p^{(r)}(v) = a_0^{(r)} + a_1^{(r)} \cdot v^1 + a_2^{(r)} \cdot v^2 + a_3^{(r)} \cdot v^3$$
$$q^{(r)}(v) = b_0^{(r)} + b_1^{(r)} \cdot v^1 + b_2^{(r)} \cdot v^2 + b_3^{(r)} \cdot v^3$$

Therefore, there are $4 \times (3 + 4 + 4) = 44$ parameters. And a parameter profile can determine a strategy.

2 Fine-tuning the parameters

Design a good parameter profile is not hard for human. However, it is difficult or even impossible for human to find a best profile. So I want to try or find some method for computer to find the best profile without aid of human. Unfortunately, I spent many time on this problem but resulted in limited improvement. So I mainly discuss my efforts on solving difficulties and what is the limitations.

My basic idea is as follows. I randomly modify the current parameter profile. I compare the new parameter profile with the original one by playing many hands (e.g. 1000). If the new one is better, I discard the old parameter profile and accept the new one. Otherwise, I discard the new one. I repeat this process for many times.

However, there are some fatal problem of this idea.

- The first problem is that the randomness of the comparison is very large. If a player win or loss a all-in, there will be a huge basis (± 20 / hand on average). And some small improvement will be cover by this basis.

  So, for each hand, I exchange the order the players and run twice. Moreover, I run 1000 times for multiple times and remove the extreme value before taking the average.

- The second problem is that simultaneously changing all parameters can hardly improve the performance because some parameters go to the right direction and some go the the wrong direction.

  So I separate the training into some sub-rounds. In each sub-round I only modify one parameters and find a best value given the other value. Since this is a one-variable function optimization problem, I use simulated annealing in each sub-round.

- There is another problem, the best response strategy of a certain strategy may not good enough in the global view. So I iterate may times, and use the best response strategy to replace the original strategy.

3 Further Ideas

I also have some further idea, and I have not implemented them due to time limit.

Design defense/attack strategy for specific cases Suppose the opponent always act “all-in” in the final round, my AI may give up even if I have put many money many times, and thus, I may lose many money. So It is a good idea to design some method to defend this sort of strategy. On the other hand, I also can use this method to exploit other AIs.

Embedding different strategy My AI uses same strategy in different hands along the whole match. I have an idea that I can implement some different strategy and try all of them in first a few hands (e.g. 100 hands), and use the most beneficial one in remaining hands. This method can be achieved bu using some online learning techniques.

Using Advanced learning algorithm As far as I know, many students use Reinforcement Learning and Counterfactual Regret Minimization algorithm to build a good AI. I did not use these advance techniques due to the time limit, but I think these techniques can improve the performance of my AI.