Texas Hold'em:A Study of Optimal Strategy

William Yang

February 21, 2024

Abstract

This paper introduces various strategies played in a game of Texas Hold'em and implements those strategies into bots. The performance of the bots in stack size, number of wins, and time to make a decision shows the effectiveness of the strategies. With the complexity of a game of poker, finding a way to reduce the time and space complexity will be crucial in finding an optimal solution in the uncertain nature and non-deterministic game of poker. The paper will present ways to reduce the time and space complexity of poker and create an optimal solution capable of accurately modeling the opponent and their behavior to develop a counter strategy.

1 Introduction

Texas Hold'em is a widely popular game with over 100 million players worldwide. The game is played with two to nine players, each with a specific position at the table that rotates after a winner in each round. Players are dealt two cards from the deck and take turns in a circular pattern starting from the small blind position, betting, checking, raising, and folding. After the pre-flop round, when everyone has two cards, players enter the flop round, where the dealer deals three cards for everyone to see, followed by another round of betting. The dealer then deals one more card for the next two rounds, called the turn and the river. A winner is decided when everyone has folded, or the player with the best combination of five out of the seven cards wins. The seven cards come from two cards in the player's hand and five from the board. Since the probability of winning varies throughout the game due to the uncertain nature, non-deterministic of the game, and large state space. An algorithm that can take in the probability of chance, repeated scenarios, and traverse a large state space is the best solution to determine the optimal strategy to maximize gains and minimize losses.

2 Problem Description

The problem we will be taking on is determining what is the overall best strategy to be profitable in poker. Is the best approach to be aggressive in the pre-flop and have the players fold, or is it to play more reserve with the best-starting hand? How about on the flop, is the optimal strategy a wait-and-see approach, bluff off the best hands, or try and draw towards a winning hand? This is an interesting challenge to break down

as player position, starting hand, and outs must all be considered in deciding the best option. We will not go over the details on pot odds, stack size, and player behavior as it is too difficult of problem to tackle. To break down the complex game of Texas Hold'em and the problem, we will start by looking at the positions and the starting hands.

2.1 Starting Hands and Positions

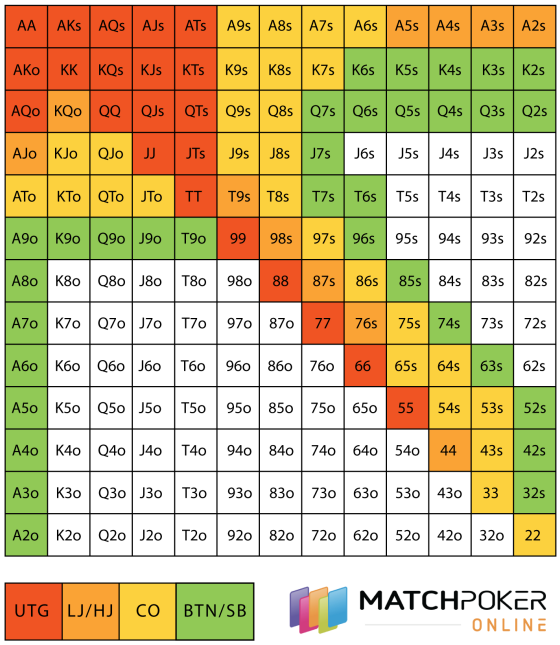

Figure 1: All combination of starting hand a dealer can deal to a player. With colors indicating what hole cards should play in a given position. The chart assumes there are eight players instead of nine [Mat].

The areas highlighted in the green show what hands players in the position in the button(dealer) and small blind (position immediately after dealer) should play. UTG or under the gun is the position after the big blind(position after the small blind). LJ low jack and HJ high jack are the positions that follow after UTG. CO or cut-off is the position between the HJ and the button. Now the letters and numbers inside each cell represent the value of the card and if they are suited or not. So "A9s" is Ace nine suited, and suited cards are essentially cards that belong to the same suit (i.e. two spades cards). The values in the main diagonal are pocket pairs (cards with the same value). The chart is just a reference for players to understand what they should fold and what they should bet.

2.2 Hand Strength and Outs

Figure 2: List of best hands a player can have with five cards from top to bottom. [wso].

The figure above showcases what hands beat other hands. So if a player has a royal flush, they would beat a player with a two-pair hand. Often a player might not have a good hand on the flop and would like to draw to a better hand. The cards potentially left in the deck and can help improve the player's hands are called outs. Outs can be calculated as follows.

$$percentOnTurn = outs * 4/100$$

$$percentOnRiver = outs * 2/100$$

If the flop comes out with two hearts and a player has two hearts in his hand and is drawing to a flush, they would have about a 36 percent chance of hitting it on the turn and 18 percent on the river(2 hearts on flop + 2 heart in player hand - 13 total heart cards = 9 outs). The calculations on the chance of hitting a flush are not accurate (off by one or two percent), however, it is a quick, easy, and reliable way of judging percentages.

3 Background

3.1 Strategies Against Player

3.1.1 Monte-Carlo Tree Search Expected Reward Distributions

There are several strategies in a game of poker, each strategies has its own strength and weaknesses. One way to verify the effectiveness of a play style is to use the Monte Carlo method. The Monte Carlo method is a mathematical technique used to estimate the possible outcomes of an uncertain event. A paper titled "Monte-Carlo Tree Search in Poker using Expected Reward Distributions" uses a variation of the Monte Carlo method called Monte Carlo Tree Search (MCTS) with additional modification to (MCTS) that models and exploits the uncertainty of expected values. The new strategy was the first to play at a reasonable level with more than two players. The new approach uses the distribution of possible rewards, where the agent can then make more informed decisions by taking into account risk to reward. When playing against a novice player resulted in a profit of four big blinds per game, the minimum bet allowed, and a draw against an experienced player [VdBDR09].

3.1.2 EpectiMax Modelling Opponent Tendencies

Testing the effectiveness of an algorithm on actual players with varying levels of skill is an effective way to measure the effectiveness of the algorithm. The paper "Opponent Modelling and Search in Poker" by Terence Schauenberg, tackles the challenge of modeling actual player tendencies and strategy in the game using a variation MinMax algorithm called expectimax. It does this by recording every state of the game, what a player does in that particular state, and the frequency of that action. Schauenberg explains how information at each point of the game is modeled and predicts an opponent's decisions: "the cards an opponent is observed revealing at each showdown can be used to predict the chance of the decision-maker winning a showdown in the expectimax search. The cards observed being dealt at each chance node can be used to predict the frequency of the cards being dealt in the expectimax search"[Sch06].

3.1.3 Counter Strategy Deviation

However, players do not tend to stick to a given strategy constantly. The player may mix things up by tossing in bluffs or changing their strategy mid-game. The article "Game theory-based opponent modeling in large imperfect-information games" has a similar approach to Schauenberg's works by recording every state of the game but differentiates by noting deviations in the opponent's strategy and adjusting to it. Their algorithm called Deviation-Based Best Response (DBBR) works by noticing anomalies in the player's strategy from their previous behavior and constructing a new model of the opponent based on these deviations [GS11]. With the new model, the algorithm can exploit the unexpected behavior of the player. While the algorithm can adapt to changes in the opponent's play style, if the opponent's play strategy never deviated and is just their play style, then the algorithm fails.

3.1.4 Game Theory Optimal

There are many strageies and play styles in poker, but there is a strategy called game theory optimal (GTO): where GTO is impenetrable to any other counter strategy. A player who plays GTO is making the decision that returns the most profit in the long run, or at worst breaks even. "Exploitability and Game Theory Optimal Play in Poker" delve further into what GTO gameplay is and the capabilities and limitation of GTO. The paper explains that a GTO strategy is a balance between bluffs and playing true to the strength of a hand and ignores the holes and exploitability in the opponent's play style. While the strategy sounds great, it assumes the opponent also plays optimally or GTO which is rarely true in poker[Li18].

3.2 Poker State Space

3.2.1 Size of State Space

A game of Texas Hold'em between two players and a limit on bets with a predetermined size will create a tree with 316,000,000,000,000,000 branches as stated by "How A.I. Conquered Poker"[Rom22]. Creating a perfect or near-optimal poker agent that can handle variation in the player strategy, behavior, number of players, and the combinations of hands and state of the game would be a state space too astronomically large to compute. Below is a table computed by the authors of "Measuring the Size of Large No-Limit Poker Games" that calculated the size of the state space in a two-player no-limit Texas Hold'em.

	Round	Sequences	Actions	Continuing	Terminal
Betting Sequences	Preflop	8.54665e31	2.564e32	8.54665e31	8.54665e31
	Flop	4.66162e44	1.39849e45	4.66162e44	4.66162e44
	Turn	1.61489e54	4.84467e54	1.61489e54	1.61489e54
	River	1.28702e62	3.86106e62	0	2.57404e62
	Total	1.28702e62	3.86106e62		2.57404e62
	Round	Infosets	Actions	Continuing	Terminal
One-Sided Canonical	Preflop	1.44438e34	4.33315e34	1.44438e34	1.44438e34
	Flop	5.99853e50	1.79956e51	5.99853e50	5.99853e50
	Turn	8.91266e61	2.6738e62	8.91266e61	8.91266e61
	River	3.12525e71	9.37575e71	0	6.2505e71
	Total	3.12525e71	9.37575e71		6.2505e71
	Round	Infosets	Actions	Continuing	Terminal
One-Sided	Preflop	1.13329e35	3.39986e35	1.13329e35	1.13329e35
	Flop	1.21154e52	3.63461e52	1.21154e52	1.21154e52
	Turn	1.97261e63	5.91782e63	1.97261e63	1.97261e63
	River	7.2317e72	2.16951e73	0	1.44634e73
	Total	7.2317e72	2.16951e73		1.44634e73
	Round	States	Actions	Continuing	Terminal
Two-Sided	Preflop	1.38828e38	4.16483e38	1.38828e38	1.38828e38
	Flop	1.30967e55	3.92901e55	1.30967e55	1.30967e55
	Turn	2.04165e66	6.12494e66	2.04165e66	2.04165e66
	River	7.15938e75	2.14781e76	0	1.43188e76
	Total	7.15938e75	2.14781e76		1.43188e76

Figure 3: Number of possible actions and states computed by an algorithm from "Measuring the Size of Large No-Limit Poker Games". Parameters of results are from one dollar small blind and two dollar big blind No-Limit Texas Hold'em with 1000 dollar stacks [Joh13].

3.3 Reducing State Space

3.3.1 Redundant State Space

The majority of the possible state space can be redundant if one player is guaranteed to win. For example, the poker agent having a royal flush the best hand strength in a poker game on the flop against a player with a pair of aces. It does not matter what the player does after the turn or river, as they are guaranteed to lose. Reducing the state space the algorithm has to traverse, and it no longer needs to keep track of what possible cards might be in the deck, but just the player's behavior and strategy. The same can be said in the pre-flop when everyone is dealt a card. If the poker agent has the best-starting hand and bets accordingly on the pre-flop with only one caller (a person who also bets the same amount), then it reduces the state space to keep track of one player instead of seven.

3.3.2 Subset of Player Actions

The "Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions" paper addresses the issue of keeping track of large states' space by considering only a subset of a player's previous actions. The algorithm presented in the article is a modified version of Monte Carlo Counterfactual Regret Minimization (MCCFR) takes into account a given player's typical strategy in most situations and develops a counter strategy[GBLS12]. By creating a counter-strategy against the average play style of a player, the poker agent only needs to record a small number of actions a player does and the hands they will play, reducing the size of the state space.

3.3.3 Predicting Actions to Reduce State Space

Instead of keeping track of all the possible player strategy, behavior, number of players, and the combinations of hands and state of the game, we can instead create an poker agent that thinks what an opponent might have in a given situation. This does require the agent to keep track of previous mentioned parameters, but only a small subset of each. As a opponent will not likely be playing 100 different strategy with and an equal amount of combinations of hands they might play. An research paper titled "ReBeL: A general game-playing AI bot that excels at poker and more" done by Meta analyze an agent thinks what a player might do or have:

> "Unlike those previous AIs, however, ReBeL makes decisions by factoring in the probability distribution of different beliefs each player might have about the current state of the game, which we call a public belief state (PBS). In other words, ReBeL can assess the chances that its poker opponent thinks it has, for example, a pair of aces"[BB20].

The AI presented by Meta is a sound reinforcement learning and search AI that is effective in a two-player zero-sum imperfect information game. The AI was capable of beating a heads-up (only two players) no-limit poker game against a professional. A poker agent that infers what an opponent might do can shrink the possible state space. Allowing the agent to be quick and efficient in deciding the best possible action.

4 Approach

An approach to finding out the best strategy in Texas Hold'em will be to create bots with different approaches and have them compete against each other one on one with the same starting stack size. Then measure the effectiveness of the strategy by the average number of wins. We would compare every combination of bots to get a good estimated performance on each approach. The bots would rotate position in the simulated environment and play until one loses its entire stack. This is to impose fairness between the bots as we want to measure how well the strategies work in the long run instead of one round session. We would also measure the average stack size positive or negative between the bots. Additionally, the number of games played between the bots will have to be large, as one bot winning five games in a row is unlikely. It does not tell us how well the strategy is, but one bot got lucky five times in a row. Applying the law of large numbers would even out the outliers and give us a more accurate measure of performance. We would also take the average time it takes for a bot to come to a decision as well. As a bot taking an hour to decide whether or not to call 100 big blind bet on a losing hand is clearly inefficient. The time will be measured to 5 decimal points. However, we would have to limit the run time of the simulation to a hour. This is will be mentioned further on as to why the simulation ends at one hour.

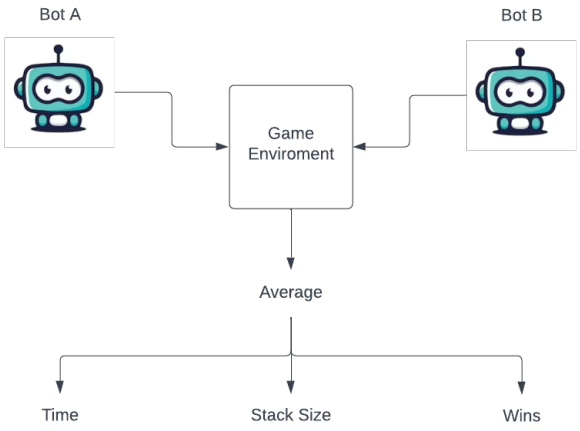

Figure 4: Example Diagram of experiment. Bot A and Bot B will play against each other in a simulated game environment. The bots will compete against each other multiple times and the average time, stack size, and wins will be taken and use to measure optimal strategy.

A base line comparison for the effectiveness of the bots different strategy would be against a bot that performs random actions. Additionally, the bots will have different ways of approximating the solution. Some of the bots will be using Monte Carlo simulation to estimate the probability of outcomes, while others will emulate the game state. The reason why some bots will use Monte Carlo simulation is to also see if grabbing a range of possible outcomes(i.e. probability) is better or faster than experience each possible out comes. An example would be knowing a coin toss is 50/50 or personally tossing a coin several times to see if the probability is 50/50.

5 Experiment Design

The experiment will be done by using the pyPokerEngine library, which is a simple framework for Texas Hold'em AI development. The library allows us to have the bots play and compete against each other in a simulated environment for multiple rounds and games. It also allows us to have the bots compete at the table of more than two players to see if a bots strategy is detrimental to another. The library also gives us the freedom to design and create our bots. Each of the bot's time to make a decision will be measured as well. Some of the bots will use pyPokerEngine predefined function called **estimate_hole_card_win_rate** that uses Monte Carlo simulation to estimate the bot's hole card probability of winning against opponents, and other bots will have hard-coded decisions. The number of simulations done will be 1000 for each bot that uses **estimate_hole_card_win_rate**. One bot will use pyPokerEngine **Emulator** class to run and simulate poker games in the given situation. The **Emulator** class is the reason why we limit the run time of the simulation to an hour. As the example mention before on knowing the probability of the coin or physically tossing the coin several times to get the same probability. We can tell right away that knowing the probability is faster than doing the physical activity of tossing the coin. This may also skew some of our results as well if the simulated environment never runs to the full number of games. Next, The number of games will be 50, and have a max 100 rounds per game. Each game bot will be given a 100 big blind stack, with small blinds being half of the big blinds. The bots will compete against each other in heads-ups (1v1) on every combination of bots. The results collected will be the amount won/lost in all games, the number of games won, and the average time for bots to make a decision. The experiment will be run three times and results will be the total average of the three. To measure the performance of the bots we will be using a code from a GitHub repository by changhongyan123, called testperf.py[cha]. The code will be modified to fit our needs.

5.1 Bots

5.1.1 Honest Player Bot

The honest player bot will be designed to only call bets that have a 100 percent chance of winning by using **estimate_hole_card_win_rate**, otherwise it will fold. The honest player bot will do this for every round of the game, but prioritize checking first to see if they can see a card at zero cost. The bot will also never bet or raise on any round. *The code for this bot is provided by pyPokerEngine[Ish].*

5.1.2 Aggressive Player Bot

The aggressive player bot will prioritize raising and betting if possible, and if no other valid action is available, it will check or call. The aggressive player bot strategy is to have other players fold constantly because of the large bet/raise size. Also forcing other players to calculate the risk to rewards, and often times the risk is too high to call the bet. This will allow the bot to collect blinds in the pre-flop, and try to bluff off other players.

5.1.3 Fish Player Bot

The fish player bot will play every single hand and call every bet size on every round of the game. A fish player is term used in poker for a bad player in general. It is a player that is usually target by other because they play too many hands. *The code for this bot is provided by pyPokerEngine[Ish].*

5.1.4 Random Player Bot

The random player bot will divide up the possible actions(raise, fold, call) equally and pick randomly among the actions. For the raise amount, it will be a random number between the minimum bet amount and the maximum bet amount. This bot will be the baseline comparison for the measure of effectiveness of strategy. *The code for this bot is provided by pyPokerEngine[Ish].*

5.1.5 Emulator Player Bot

The emulator player bot will play all hole cards and emulate the round of the game with all possible actions. By emulating the round a list of possible stack sizes corresponding to the action is calculated, and the average is taken. The best average stack size corresponding to the action will then be the bot's decision. The emulation is closely similar to **estimate_hole_card_win_rate** function where both are simulating the round. Where one simulates until the end of the game, the other simulates only the round. Additionally, **estimate_hole_card_win_rate**, calculates the percent chance of winning, while emulations calculate the bots stack size after the action. *The code for this bot is provided by pyPokerEngine[Ish].*

5.1.6 Real Player Bot

The real player bot will play a wider range(play more hole cards) but will raise the pre-flop 3-4 times the last bet on good hands. The good hands the bot will raise on are the cards highlighted in red in figure 1. For the other cards the in the pre-flop, the bot will call the previous bet and play connected cards, 1-2 gapper cards(i.e. the value of the cards, not the suit, (10,8), (5,4), (7,4)), and any suited cards. Any round after the pre-flop, the bot will calculate its win rate using **estimate_hole_card_win_rate** and bet varying pot size according to the percent chance to win.

1. If win rate $> 75\%$ then the bot will bet 80% pot
2. If win rate $> 50\%$ then the bot will bet 33.33% pot
3. If win rate $> 35\%$ then the bot will call the previous bet size or check.
4. If another bot bets and win rate < 35 % then the bot will fold

This bot will simulate a more real-life player with a wide range. By betting varying amounts of pot size bet, the bot is charging opponents to see another card. Ensuring the bot's current best hand does not lose value. Meaning if the bot has a high pair on the board, it does not lose to a flush or a straight.

Bot A vs. Bot B	Stack Size (A, B)	Wins (A, B)	Time in Seconds (A, B)
Honest vs. Aggressive	(6066.66, 3933.33)	(31.33, 19.66)	(16.52293, 0.002)
Honest vs. Fish	(-5168.66, 5168.66)	(0, 0)	(923.81881, 0.004)
Honest vs. Random	(6523, 3465.66)	(22.33, 16.33)	(87.43396, 0.007)
Honest vs. Emulator	(-425, 425)	(0, 0)	(75.46523, 4099.32708)
Honest vs. Real	(261.33, 5995.33)	(3.33, 24.33)	(597.0009, 200.96521)
Aggressive vs. Fish	(5733.33, 4266.66)	(28.66, 21.33)	(0.00013, 0.00008)
Aggressive vs. Random	(9979.66, 0)	(29.66, 0)	(0.001, 0.00166)
Aggressive vs. Emulator	(5200, 4800)	(26, 24)	(0.00066,539.08964)
Aggressive vs Real	(6866.66, 3133.33)	(34.33, 15.66)	(0, 0.00533)
Fish vs. Random	(8387, 1600)	(29, 8)	(0, 0)
Fish vs. Emulator	(-18, 18)	(0, 0)	(0, 4255.56298)
Fish vs. Real	(810, 9122)	(3.66, 44.66)	(0.001, 157.73973)
Random vs. Emulator	(1799.33, 8184.33)	(8.33, 25.33)	(0.001,3325.41667))
Random vs. Real	(6117, 3392)	(21.66, 12)	(0.00169, 8.21304)
Emulator vs. Real	(933.33, 3332.66)	(1, 11.33)	(3920.50956, 35.76597)

Table 1: Table of results comparing bots performance total average of 3 runs in stack size, wins, and time taken. Some of the values may be skewed due to hard stop at one hour run time and the number of rounds per game. The results that are skewed are from the comparison between emulator bot and other bots. The source code for the computing the results can be found at the following link. (https://github.umn.edu/yang7313/Poker-Ai-Performance)

Bot	Stack Size	Win Rate	Time in Seconds
Honest	35.4	53.68%	0.23615
Aggressive	154.69	55.32%	0
Fish	90.80	35.99%	0
Random	51.92	21.73%	0
Emulator	70.05	49.33	9.9782.
Real	121.83	56.43%	0.21301

Table 2: Average result per game of all bots results from all comparison tests. This includes the negative results as well. The win rate is calculated by the number of games played and won by the bot. This exclude incomplete games due to run time limitations and early termination of games due to limited round counts. Time in seconds is calculated by the average time for a bot to make a decision per round.

6 Analysis

6.1 Honest Bot Results

The results from the honest bot show that the strategy of only calling bets when it's guaranteed to win is only effective against the aggressive bot and the random bot. The reason for the effectiveness is when played against the aggressive approach the honest bot will fold close to 99% of the hands. When the honest bot finally has a hand that

can beat the aggressive bot it will call the bet, which leads to a massive gain in stack size. Against the random player bot, it implements a similar play style of folding the majority of hands until it has a playable hand. This strategy of always folding hands that do not have a 100% win rate has some downsides. One of which is the ante or the force bets for the non-dealer position, leading the bot to slowly lose its stack and never play a hand; we can see this result prevalent in the honest bot match-up against the fish and emulator bot. Where there is never a conclusive winner to the game as the simulation hits the 100-round limit per game. We can see that in the simulation, the honest bot stack size is a perfect negative value to the opponent stack size. Showing that the honest bot never played a hand, or if it did, it lost right away. Similar results are shown against the match-up with the real player, where honest bot stack size and the number of wins is a stark comparison to the real player. Next, for the average game results, we can see honest bot stack size was the lowest among all the other bots, this is due to the negative values included in the calculation, but if only positive values were calculated it would have done much better. The win rate percentage for the bot is decent at 53.68% compared to the baseline of 21.73% for the random bot and 35.4 stack size showing the vulnerability in the strategy of constantly folding. The time it took to make a decision is acceptable, with an average of .23615 seconds. The reason for the longer decision time is most likely due to the bot using **estimate_hole_card_win_rate** on every decision.

6.2 Aggressive Bot Results

The results for the aggressive bot show it performed well against the fish, random, emulator, and real bot in stack size. What is surprising is how well it did against the emulator and real bot. As the logic for the two bots was they both find the best action according to the probability of how much they win/lose. What most likely happen was the aggressive player forcing the opponent to call because of the percent chance of winning, and the aggressive bot got lucky. Using a larger sample size may help fix the results, causing the aggressive bot to have a smaller stack size. Overall the aggressive bot faired well against the other bots with consistently more wins and stack size against the other bots. The time for the bot to make a decision was fast, with an average of 0.0 seconds to make a decision. Likely due to the hard-coded logic and not relying on any simulation to calculate the probability of it winning. It also by far have the largest stack size average stacks per game among all other bots with 154.69, and a high win rate at 55.32%.

6.3 Fish Bot Results

The results from the fish bot were to be expected. It won against the honest bot by having it bleed out its stack, lost against the real bot because it calls any bet and will play any hand, won against the random bot, and lost to the aggressive bot by calling large bets with a low probability of winning. What is surprising about the results is how closely matched the bot was against the emulator bot. Although, the experiment did not run to completion due to the time limit. It only lost 18 from its stack in the one hour it played. What might have happened is the two bots losing and winning back and forth to each other. However, we should be expecting the emulator bot to be winning a

large stack size as it is calculating the probability of winning. The performance of the fish bot is varied, with a 90.80 in stack size and a 35.99% win rate. The low win rate is to be expected, as in a game of poker, playing every hand, in the long run, will cause the player to lose frequently, as they are often playing against better hands. The time it took for the bot to make a decision is 0.0 seconds because of the hard coded logic which matches the time taken by aggressive and random bot.

6.4 Random Bot Results

Not much can be said about the random player bot results, as it is a baseline comparison for the performance of other bots with a low average stack size per game at 51.92 and a 21.73% win rate. What is surprising from the results is how well it performed against the real player bot. The random player bot has about two times the number of win and stack size compared to the real player bot. This is an anomaly within the experiment, as the logic from both bots should indicate that the real player bot performance should be better.

6.5 Emulator Bot Results

It is hard to find definitive results for the emulator bot as most of the games are incomplete due to the round limitations or the time constraint. What we can analyze though is the bot can beat the honest bot by having it bleed its stack, close match up with the aggressive bot by calculating the risk to rewards, win against the fish bot because of playing every hand, and beat random. An interesting result is how badly it lost against the real player bot. Both use a function that calculates the probability of winning and deciding on the best action, but the emulator bot stack size is one-third of the real player bot and lost about 11 games. It seems that using **estimate_hole_card_win_rate** is more effective and efficient then emulating the game to calculate the best action. We can see the effectiveness in the average time it takes to decide at 9.9782 seconds, about 42 times longer than the next slowest bot at 0.23615 seconds. The lack of effectiveness can also be seen in the rate as well with a 49.33% win rate and 70.05 stack size, which is less than the fish bot.

6.6 Real Bot Results

Overall the results from the real bot shows it performs better than all other bots dominating them in stack size and number of wins, and only losing to aggressive and random bot. The bot has the best overall average stack size at 121.83 only losing to aggressive bot, and the highest win rate at 56.43%. The time take to make a decision per round is 0.21301 seconds which slightly faster than honest bot due to it only calculating the probability of winning after the pre-flop.

7 Conclusion

We discussed approaches to find the overall best strategy to be profitable in poker by creating bots with different strategies and having them compete against each other.

Using the results from the experiment in the stack size, number of wins, and time is taken to decide, we were able to find a bot with the best overall strategy against other strategies. For future work, we would like to increase the number of games and rounds per game to have more robust results. Additionally, removing the emulator bot from the experiment as the time taken for the bot to find the best action is too long for an increased number of games and rounds. We would also like to create a more thorough bot that implements strategies from the several papers in the background work and have them compete. Instead of the simple logic bots provided by pyPokerEngine and the bots we created. Additionally, the real player bot could be improved to have a higher win rate percentage and faster decision time by applying additional rules to limit the number of hands it will play, and relying less on **estimate_hole_card_win_rate**. All in all, implementing algorithms from the research papers and more advanced strategies into the bots will give us a more insightful analysis of what strategy is the best to be profitable in poker.

References

[BB20] Noam Brown and Anton Bakhtin. Rebel: A general game-playing ai bot that excels at poker and more, Dec 2020.

[cha] changhongyan123. Changhongyan123/mypoker: Cs3243 introduction to ai term project: Ai poker.

[GBLS12] Richard Gibson, Neil Burch, Marc Lanctot, and Duane Szafron. Efficient monte carlo counterfactual regret minimization in games with many player action. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

[GS11] Sam Ganzfried and Tuomas Sandholm. *Game theory-based opponent modeling in large imperfect-information games*, May 2011.

[Ish] Ishikota. Ishikota/pypokerengine: Poker engine for poker ai development in python.

[Joh13] Michael Johanson. Measuring the size of large no-limit poker games. Feb 2013.

[Li18] Jingyu Li. Exploitability and game theory optimal play in poker. 2018.

[Mat] *Pre-Flop Ranges (8-Max).* Match Poker Online.

[Rom22] Keith Romer. How a.i. conquered poker, Jan 2022.

[Sch06] Terence Conrad Schauenberg. *Opponent modelling and search in Poker.* PhD thesis, Library and Archives Canada = Bibliothèque et Archives Canada, 2006.

[VdBDR09] Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-carlo tree search in poker using expected reward distributions. *Lecture Notes in Computer Science*, page 367–381, Nov 2009.

[wso] *How To Play — Hand Ranking*. Caesars Interactive Entertainment.