Finding Optimal Betting Strategies in Multiplayer Poker

Justin Dano

Stevens Institute of Technology

JDano@Stevens.edu

Abstract

This paper presents the design and implementation of a betting strategy for three-player Kuhn poker. The strategy profile establishes an approximated Nash equilibrium using Counterfactual Regret Minimization (CFR), a popular technique for zero-sum, imperfect-information games such as poker. While some variations of two-player poker have been solved using CFR [1], extensions to three or more players are still considered open problems in the research community. When considering multiple opponents, traditional measures of an optimal strategy are without the theoretical guarantees two-player games have. This is due to a variety of factors such as possible collusion between the opponents. Studies have shown however that while an approximated Nash equilibrium does not guarantee an optimal strategy, it can still perform very well in practice [2]. In this paper we will first explore the concepts of Kuhn Poker, followed by the CFR algorithm and its relation to Nash equilibrium. Next we will cover the nuances for training a CFR strategy for three-player Kuhn Poker. Once our strategy is trained, we evaluate it against a best response strategy developed to exploit the opponents. By comparing the utility between the CFR strategy and the best response strategy, we gain insight into how close the CFR strategy approximates equilibrium.

1. Introduction to Kuhn Poker

Variations of poker, such as Texas Hold'em have been a popular research topic given their applications to a variety of real life decision-making scenarios. Some variations of the game have even been solved, with two of the most well-known programs being Cepheus and DeepStack [3]. One of the main constraints of these programs however is that they are playing against a single opponent. Adding additional players to the game not only rescinds the theoretical guarantees relied on for the two-player variant, but also exponentially increases the number of game states that must be analysed. For example, studies show that two-player limit Hold'em has 10^{18} game states, while the three-player version has 10^{24} states to consider [2]. While our ultimate goal is to design a betting strategy for Texas Hold'em, we can gain intuition on our training algorithms by playing a much simpler version of the game called Kuhn Poker. Introduced by Joseph Kuhn in the 1950 paper *Contributions to the Theory of Games*, Kuhn poker was invented to provide a simplified version of traditional poker while maintaining the important characteristics required for the study of game-theory [1].

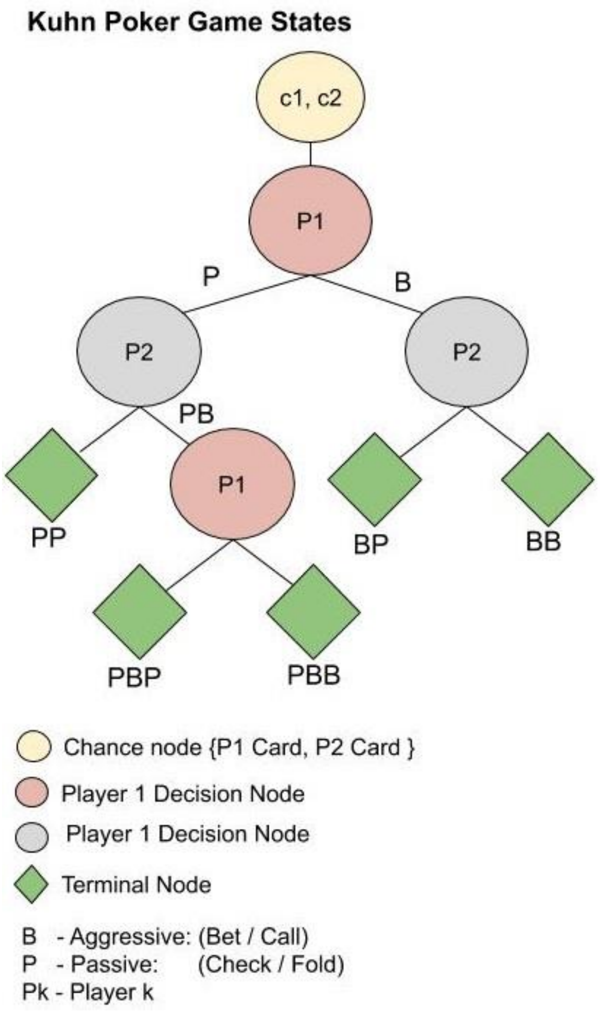


Figure 1: Two-player Kuhn poker game tree

Kuhn Poker involves just a single round, with three cards for two-player games and four cards for three-player games. For the three-player game, the cards are valued at `1`, `2`, `3`, and `4`, so a winner is guaranteed and ties cannot occur. At the beginning of the game each player antes `1` chip into the pot, followed by one card being randomly assigned to each player. `Player1` begins who may either check or bet. By checking, they make no addition to the pot, and by betting they add `1` additional chip to the pot. If a player makes a bet, the following player may either call the bet, which involves matching the ante of `1` chip, or may fold and be excluded from the round. Once a player makes a bet, all previous players who checked also receive a turn to either call the bet or fold. The game states for two-player Kuhn poker are shown in Figure 1, and three-player Kuhn poker in Figure 2.

One of the underlying characteristics of Kuhn poker—and poker games in general—is there is hidden information from the players' perspective. As a result, a different framework should be used to analyse strategies compared to traditional perfect-information games like Chess and Go. A popular approach is to construct the game state in such a way as to account for the missing information via *chance nodes* and *information sets* [4]. Every state or decision node in the game is represented as an information set which encapsulates an active player and all public information available to the player at that moment. Information sets can be thought of as a perspective the player has before they make a decision. These are the structures used to maintain the strategy profile and regret accumulation throughout counterfactual regret minimization.

Information sets can be encoded as a card value concatenated with the *history*, of the previous round. For two-player Kuhn poker there is a total of `12` information sets: `{1, 1P, 1B, 1PB, 2, 2P, 2B, 2PB, 3, 3P, 3B, 3PB}`. As an example, `3B` is an information set for `Player2` who has the `3` card, and the previous player has just betted. Since the order in which rounds are played do not change, each player has a pre-defined collection of information sets. `Player1`, who starts, has the sets `{1, 2, 3, 1PB, 2PB, 3PB}` while `Player2` has `{1P, 1B, 2P, 2B, 3P, 3B}`. As with information sets there are also a unique combination of terminal states with various payoffs. Table 1 summarizes the five different payoff scenarios possible for two-player Kuhn poker [4].

Player1	Player2	Player1	Payoff
Pass	Pass		+ 1 to player with highest card
Pass	Bet	Pass	+ 1 to Player2
Pass	Bet	Bet	+ 2 to player with highest card
Bet	Pass		+ 1 to Player1
Bet	Bet		+ 2 to player with highest card

Table 1: Two-player Kuhn poker payoffs

2. Three-Player Kuhn Poker

Three-player Kuhn poker is very similar to the two-player version, with the same rules but extended for an additional player. For example, there are 24 possible hand combinations, compared to 8 for the two-player game, 13 terminal states compared to 5, and 48 information sets that will need to be analysed, instead of 12. As previously mentioned, the main difficulty around the three-player game is figuring out how to design an optimal strategy. While CFR is guaranteed to derive a Nash equilibrium strategy profile for the two-player game, such is not the case for multi-player games [1]. Even if an equilibrium strategy could be computed, there are no guarantees that the strategy will perform well in practice. Despite the lack of theoretical backing, CFR-trained strategies for three-player Kuhn poker have been shown to do well when facing against a variety of opponents [2].

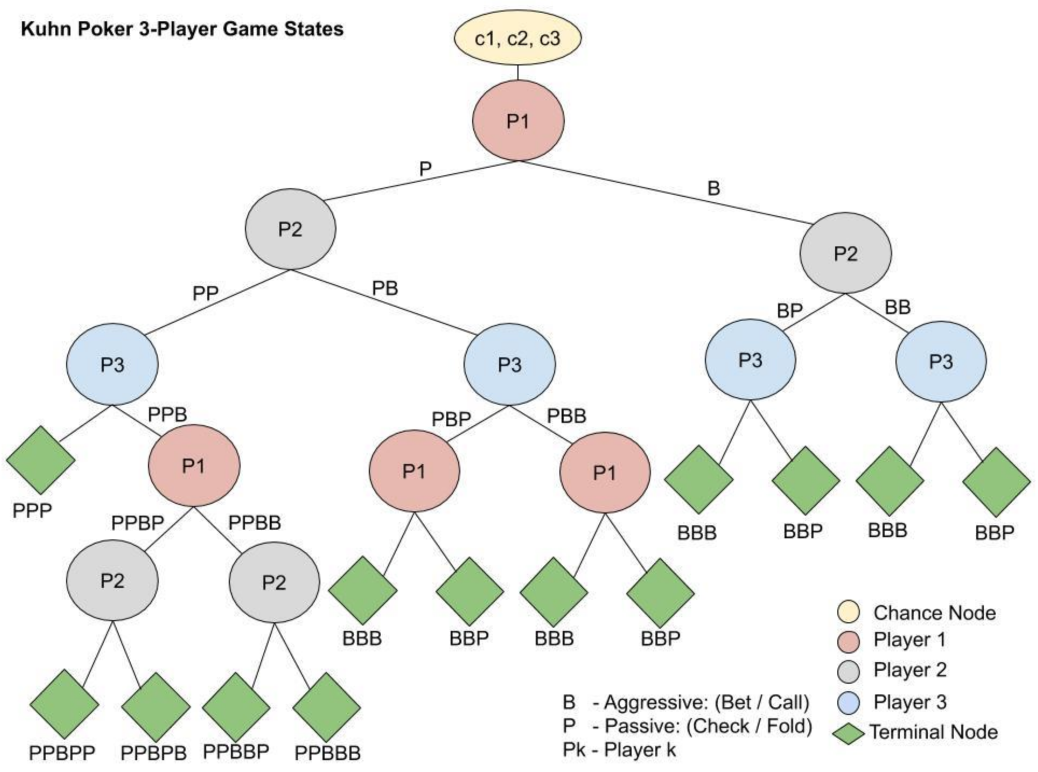


Figure 2: Three-player Kuhn poker game states

3. Counterfactual Regret Minimization

Counterfactual regret minimization is an online learning algorithm where it learns to play a game by repeatedly playing against itself. Starting with a uniform random strategy, the agent will play every action at every decision point with equal probability. After each round is over, the decisions made by the agent will be compared against alternative decisions that could have been made, and a record is kept of how much better or worse the outcome could have been if they had chosen that alternative decision. As additional rounds are played, the agent will recognize past scenarios and react by adjusting their strategy accordingly.

To define CFR formally, let us consider the following notation: Let `A` denote the set of all game actions. Possible actions include `A = {Check, Bet}` or `A = {Fold, Call}` depending on the game state. This is simplified for *non-aggressive* actions (checking and folding) encoded as `P`, and *aggressive* actions (betting and calling) encoded as `B`, resulting in `A = {P, B}`. Let `I` represent an information set, and we have a strategy profile σ^t where `t` signifies the current training iteration. Our strategy profile for three-player Kuhn poker consists of each player's strategy, $\sigma^t = (\sigma_1, \sigma_2, \sigma_3)$. We also have σ_{-i} which is a strategy profile that excludes the strategy for player `i`. We keep track of the game history `H` for all previous actions in a given round. The information set is encoded as a string with the card value followed by the history. Reach probabilities for strategy σ are encoded as π^σ. Terminal states are labelled as `z`, and $u_i(z)$ is the expected terminal utility for player `i` at terminal state `z`. For every information set, we can estimate the expected utility of each action by counterfactually playing all potential subgames and computing their utility, proportional to the probability of the subgame being played [4]. This counterfactual utility is expressed in Equation 1.

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

Eq 1: Counterfactual utility for player i given strategy σ and history h

Regret is used to learn how much utility differs between a particular action and the unequivocal best action. One could think of regret as: how much better would I have done over all the games previously played if I had just always played this one particular action at this decision, instead of playing some mixture of actions that my strategy said I should have chosen [5].

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

Eq 2: Counterfactual regret for history h

Equation 2 describe how regret is calculated for one particular realization of a game state. Positive regret means we would have played better if we had taken that action more, while negative regret means we would have done better not playing the action. Different games states however will result in different regret values, which we accumulate over an information set `I`, as shown in Equation 3. For instance, if the opponent has a higher card in one round, the regret value of our action will be different then if our card was higher.

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

Eq 3: Counterfactual regret for information set I

CFR then accumulates all of these regrets over millions of rounds, each time updating our strategy to be proportional to the regret corresponding to the rounds previously played. Overtime this allows us to learn what action has the higher probability of being successful. Total regret is presented as R_i^T in Equation 4.

$$R_i^T(I,a) = \sum_{t=1}^{T} r_i^t(I,a)$$

Eq 4: Total Regret of taking action a at information set I

For each game, the total regret for the given information set is used to determine the current strategy. Each action for the strategy is determined by positive regret, which is then normalized across all actions. If no positive regret exists, then a uniform random strategy is used.

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)} & \text{if } \sum_{a \in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases}$$

Eq 5: Updating the strategy profile

Figure 3 shows the evolution of the strategy profile and cumulative regret throughout training. Each column represents a different information set (only a card number for the first betting scenario). CFR converges quickly for some cards, such as always passing when being handed a `1`, and always betting when given a `4`. For cards `2` and `3` however there is an oscilating pattern as total regret for one action surpasses the other.

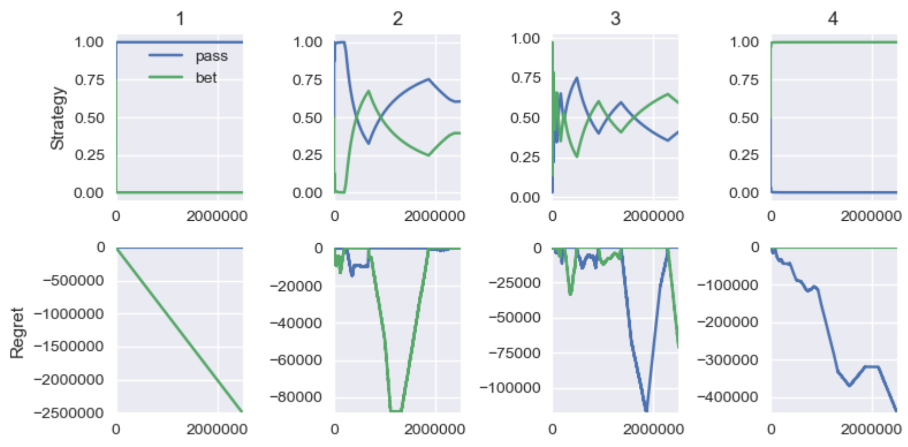


Figure 3: Sample betting strategies for Player1

Figure 4 shows betting scnearios after `Player1` passed. From the perspective of `Player2`, training regret looks similar for cards `1` and `4`. For card `2`, it follows a similar oscillating pattern, and card `3` looks to have converged to passing with a much higher probability then betting.

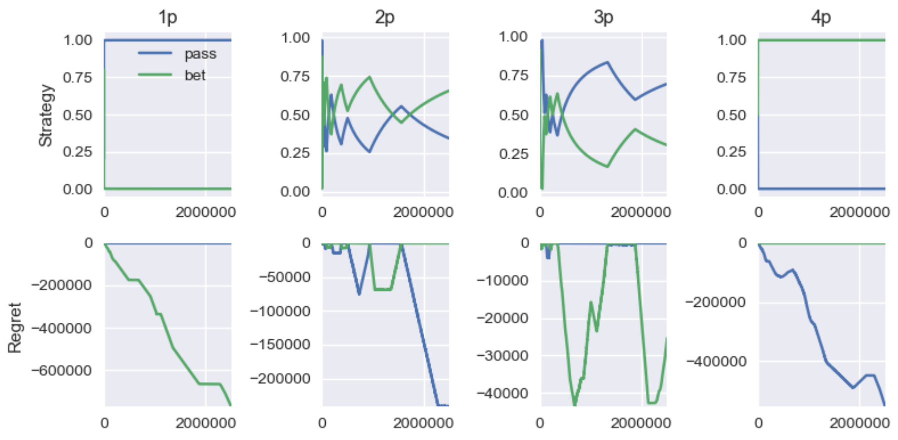


Figure 4: Sample betting strategies for Player2

Figures 3 and 4 are scaled to capture the entire range of regret, with negative regret often magnitudes larger than positive regret. Since positive regret is what actually determines the strategy profile, Figure 5 provides a scaled view of the y-axis to provide insight into how cumulative regret defines the strategy profile. The first column depicts a scenario for `Player3` who has card `2` and has observed both opponents passing. In the beginning, passing appears to have accumulated a very small amount of positive regret (< 1), whilst betting is clearly negative. This regret still greatly influences the strategy for several rounds however, causing the agent to pass at a greater frequency. A critical moment occurs around the 200th iteration when the regret for betting becomes positive. The action probabilities changes directions and even invert towards the 600th round. After a while, the regret of passing surpasses betting again at the 1000th round, resulting in another reflection point for the strategy. Soon the action probabilities cross again and appear to continue with passing as the dominant action.

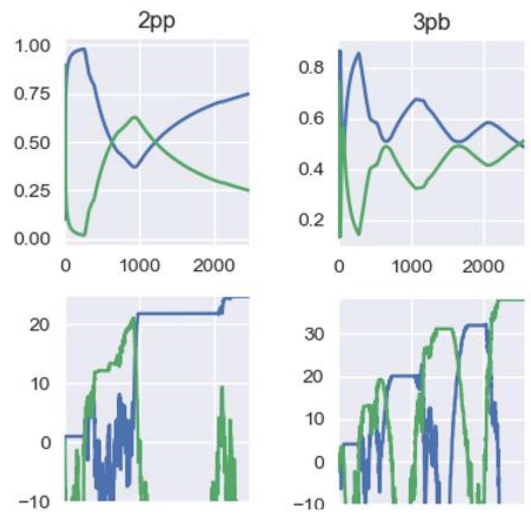


Figure 5: Betting strategies for player 3

Thinking about the actual scenario, when player three passes, they automatically lose one chip, as there is no way to win with the `2` card. By betting, the agent may bluff their way to victory, however the potential loss is double that of passing. So after considering the probability of winning the bluff, we infer how often bluffing should occur. For this small training scenario, the agent would bluff about 25% of the time when presented with this game state. The second column shows a different scenario for `Player3`, who has the `3` card and witnessed `Player1` passing, and `Player2` betting. If `Player3` decides to pass, `Player2` automatically wins. If `Player3` bets however, the probability of winning is 66%. The graph weighs both options and overtime develops a mixed strategy that may either bet or pass with around equal probability. Additional examples of how CFR calculates these values have been included in the appendix.

4. Approximate Nash Equilibrium

Our overall objective is to come up with a strategy that wins as much as possible. One common concept for this in game theory is the Nash Equilibrium [1]. Our strategy profile σ is a Nash Equilibrium profile if no player can unilaterally deviate from σ and increase their expected utility. A formal definition is presented in Equation 6. Having a strategy that is in equilibrium is useful because it can do no worse than tie, on expectation, against any other opponent strategy. If the opponent also plays a Nash equilibrium strategy, the result will be a tie, and even if the opponent learns your Nash equilibrium strategy and computes the perfect counter-strategy it will still lead to a tie. If the opponent makes mistakes however, then the Nash equilibrium strategy will win more often, in expectation. [5].

$$\max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i}) \leq u_i(\sigma) \text{ for all } i = 1, 2, \ldots, |N|.$$

Eq 6: Nash Equilibrium

For two-player Kuhn poker, the Nash equilibrium can be computed in polynomial time using CFR. Unfortunately for us, three-or-more player games cannot be computed as it belongs to the PPAD-complete class of problems [2]. As an alternative we will find an approximation to the Nash equilibria, meaning a strategy profile where no player can increase its utility by more than ε by unilaterally changing its strategy. This is known as the ε-Nash equilibrium. The approximated Nash equilibrium can be verified by developing a best response (BR) strategy σ_i^{BR} to the competing strategies σ_{-i}. This best response is essentially learning how to exploit the opponent strategy. The following procedure can be used to evaluate CFR strategies for three-player Kuhn poker [2].

1. Generate a strategy profile $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ using CFR.
2. Compute a best response strategy, σ_i^{BR} to σ_{-i} using CFR for each player `i` = 1, 2, 3.
3. Combine the three best responses into a best response strategy profile $\sigma^{BR} = (\sigma_1^{BR}, \sigma_2^{BR}, \sigma_3^{BR})$
4. Compute the utilities for each position of the strategy profile σ by playing each CFR strategy σ_1, σ_2, and σ_3 against each other.
5. Compute the utilities of the best response in each position by playing one σ^{BR} against two σ's.
6. Compare the σ^{BR}'s utilities in each position to σ's utilities to determine how much extra BR wins for each position.
7. If the best response σ^{BR}, does not improve by more than ε averaged over all positions then σ is an ε-Nash equilibrium.

The remaining sections of this paper presents the organization of the source code used to train the CFR algorithm, and the results from evaluating the strategy using the procedure above. It is important to mention however that while we may approximate an equilibrium strategy, it does not provide any indication in how well it would perform against two arbitrary opponents. To evaluate the strength of our multiplayer agent, we must simulate millions of rounds against benchmark agents [2].

5. Project Overview

The main modules for the project are `multiPlayerKuhnTrainer`, `multiPlayerKuhnPoker`, `kuhnHelper`, and `main`. The first module, `multiPlayerKuhnTrainer` will train the CFR strategy, and also train the best response strategies. The module `multiPlayerKuhnPoker` simulates Kuhn poker using pre-defined strategies which are passed in as parameters. While some of the functionality is similar to the `multiPlayerKuhnTrainer`, its purpose is different in that it will accumulate the utility of winning for each position of a strategy, whereas `multiPlayerKuhnTrainer` accumulates regret and the strategy profile. Utility is then used to compare CFR with BR strategies and calculate the approximate to Nash equilibrium. `KuhnHelper` contains all the helper functions that are used for saving output, creating graphs, and other miscellaneous operations not directly related to CFR. The `main` module is what coordinates all operations required to train the strategies, run the simulations, and persist the results. With the `main` module, the entire program can be ran by invoking a single method. Here are the main variations that can be ran with descriptions of the parameters. Option 6 is recommended to start, as it save a report in excel that is easy to view.

```
1. results = main(iterations=10000)
```

This is the vanilla-run approach. All results are returned via the `results` variable. You will not persist any of the content of the models and simulation.

```
2. results = main(iterations=10000, run_training=True, save_models=True)
```

Using the `save_models` parameter will save the trained strategies for Kuhn Poker by serializing the data using Pickle. A custom timestamp-labelled directory is generated (e.g., 2019_05_09_11_03) in your project directory which contains the CFR strategy profile and BR strategy profiles.

```
3. results = main(iterations=10000, run_training=False,
   training_mod_dir=<dir>)
```

If `run_training` is set to `False`, you will be required to also provide a directory to the `training_mod_dir` parameter. This runs the simulations of Kuhn Poker using the pre-trained strategy profiles.

```
4. results = main(iterations=10000, save_results=True)
```

Similar to `save_models` parameter, `save_results` contains the simulations of playing the Kuhn Poker game using the CFR strategy and best response strategies. It also saves the approximated Nash Equilibrium value. A helper function `KuhnHelper.load_results()` is available for easily accessing any serialized strategies or results.

```
5. results = main(iterations=10000, gen_graphs=True)
```

When setting the `gen_graphs` parameter to `True`, the training module keeps a tally of the regret sum and strategy profile for every iteration. It will then generate multiple graphs, (the same ones seen in section 3). In the current state, this is not an optimized feature. Space complexity for this is double the number of iterations multiplied by information sets, or `O(iterations*48*2)`. It is recommended to use for smaller iterations.

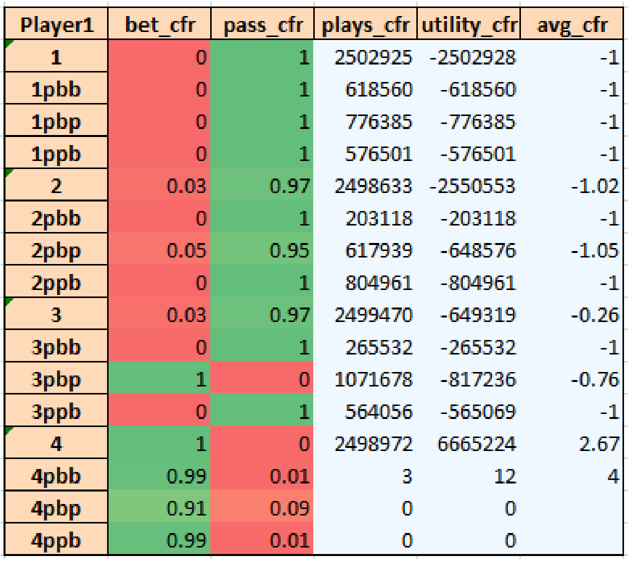
```
6. results = main(iterations=10000, gen_report=True)
```

Setting `gen_report=True` enables an excel report which captures all strategy profiles and results from simulation. Useful for quick overview of the results and sanity checks.

6. Results

Figure 6 presents the results of our CFR strategy for `Player1` after training 10^7 iterations. Each row is a possible betting scenario. The columns `bet_cfr`, and `pass_cfr` reference the action probabilities that make up the strategy profile. The next three columns are the results of the simulated games, which are also ran 10^7 times. `plays_cfr` is the number of times the betting situation had occurred, and `utility_cfr` can be thought of as the net money won or lost in that scenario using the strategy. Finally we have `avg_cfr` which shows us our average utility for each betting scenario.

When presented with cards `1` or `2`, the strategy will almost always pass, and receiving a `4` almost always results in a bet. For card `3`, it depends on the situation. If both opponents bet, player one will always pass. If only player two bets, player one will also bet (resulting in a showdown). If player three bets however, player one will pass. This could be due to the aditional uncertainty since player two will also get another chance to bet. The number of plays for the first betting scenario `1`, `2`, `3`, and `4` are each approximately 2.5 million. In fact if you add them up they will equate to exactly the number of training iterations. Betting scenarios with longer histories however occur much less frequently due to probablistic nature of them occuring. For the last two rows, they never even get played because the likelihood of the previous actions. Since player one always bets when presented the `4` card, the scenario of pass-bet-pass statistically never occurs. Figures 7 and 8 show the corresponding results for `Player2` and `Player3` respectively.



Player1	bet_cfr	pass_cfr	plays_cfr	utility_cfr	avg_cfr
1	0	1	2502925	-2502928	-1
1pbb	0	1	618560	-618560	-1
1pbp	0	1	776385	-776385	-1
1ppb	0	1	576501	-576501	-1
2	0.03	0.97	2498633	-2550553	-1.02
2pbb	0	1	203118	-203118	-1
2pbp	0.05	0.95	617939	-648576	-1.05
2ppb	0	1	804961	-804961	-1
3	0.03	0.97	2499470	-649319	-0.26
3pbb	0	1	265532	-265532	-1
3pbp	1	0	1071678	-817236	-0.76
3ppb	0	1	564056	-565069	-1
4	1	0	2498972	6665224	2.67
4pbb	0.99	0.01	3	12	4
4pbp	0.91	0.09	0	0	
4ppb	0.99	0.01	0	0	

Figure 6: Player one strategy and simulation results

Each player also has a corresponding best response strategy profile, which can be found in the appendix. To calculate epsilon for the averaged equilibrium profile we take the average across all possible scenarios for each player and compare that average with the best response strategies. These

Player2	bet_cfr	pass_cfr	plays_cfr	utility_cfr	avg_cfr
1b	0	1	881349	-881349	-1
1p	0	1	1618002	-1617998	-1
1ppbb	0	1	940	-940	-1
1ppbp	0	1	830776	-830776	-1
2b	0	1	859756	-859757	-1
2p	0.66	0.34	1641882	-2423067	-1.48
2ppbb	0	1	323	-323	-1
2ppbp	0.94	0.06	280324	-542791	-1.94
3b	0	1	855988	-857042	-1
3p	0.01	0.99	1642825	22286	0.01
3ppbb	0	1	0	0	
3ppbp	1	0	833155	-1555074	-1.87
4b	1	0	48685	146051	3
4p	1	0	2451513	6141743	2.51
4ppbb	0.5	0.5	0	0	
4ppbp	0.5	0.5	0	0	

Figure 7: Player two strategy and simulation results

Player3	bet_cfr	pass_cfr	plays_cfr	utility_cfr	avg_cfr
1bb	0	1	24782	-24782	-1
1bp	0	1	856412	-856412	-1
1pb	0	1	1078147	-1078147	-1
1pp	0	1	537738	-537747	-1
2bb	0	1	13903	-13903	-1
2bp	1	0	846939	-1693856	-2
2pb	0	1	826722	-826723	-1
2pp	0.05	0.95	813268	-770890	-0.95
3bb	0	1	11093	-11093	-1
3bp	1	0	843875	-1631710	-1.93
3pb	0.49	0.51	1097592	-961220	-0.88
3pp	0	1	548344	1096677	2
4bb	1	0	11	44	4
4bp	1	0	48763	146289	3
4pb	1	0	550754	1652262	3
4pp	1	0	1901657	4877920	2.57

Figure 8: Player three strategy and simulation results

results are then averaged across the three players to derive our ε value. Figure 9 provides a summary of average utility across the CFR strategy profile and the best response strategy profile for each player. By taking the last column and dividing by the number of players we get ε = 0.043071. This implies that the expected payoff of each player is within a relative error of 0.043071 from the best response to the opponent strategies [6]. When re-training the CFR strategy, an interesting find is some of strategies for the `2` and `3` card develop completley opposing probabilities. Examples include `2pbp`, `3`, `3p`, `2pp`, and `3ppb`. This can be expected though since optimal strategies need not be unique. They each approximate equilibrium with about 2% difference in error.

Results	p1	p2	p3
CFR	-0.27831	-0.27283	-0.06333
BR	-0.27436	-0.23585	0.024949
diff	0.003958	0.036978	0.088278

Figure 9: CFR vs BR utility comparison

7. Conclusion

This paper has provided a general overview of how to develop betting strategies for the mutliplayer poker game Kuhn poker. By using counterfactual regret minimization, a strategy is developed that approximates a Nash Equilibrium with a relative error of around 4%. Verification of the strategy is done by computing best response strategies and comparing the overall utility between the two across all positions. Further validation of the strategy profile against a variety of benchmark agents should be considered to gain a better understanding of how well the agent performs against arbitrary opponents. Extensions of the work presented here could be to apply similar techniques to more complex versions of multiplayer Poker, such as Leduc Hold'em. The complete source code for this project is available online at https://github.com/dano09.

8. References

[1] Duane Szafron, Richard Gibson, and Nathan Sturtevant. A Parameterized Family of Equilibrium Profiles for Three-Player Kuhn Poker. https://webdocs.cs.ualberta.ca/~games/poker/publications/AAMAS13-3pkuhn.pdf

[2] Nick Risk and Duane Szafron. Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents. https://poker.cs.ualberta.ca/publications/AAMAS10.pdf

[3] Kamil Czarnogorski.Coutnerfactual Regret Minimization – the core of Poker AI beating professional players. https://int8.io/counterfactual-regret-minimization-for-poker ai/#Basics_of_game_theory

[4] Todd Neller and Marc Lanctot. An Introduction to Counterfactual Regret Minimization. http://modelai.gettysburg.edu/2013/cfr/cfr.pdf

[5] Michael Johanson. What is an intuitive explanation of counterfactual regret minimization? https://www.quora.com/What-is-an-intuitive-explanation-of-counterfactual-regret-minimization

[6] Constantinos Daskalakis. On the Complexity of Approximating a Nash Equilibrium. https://people.csail.mit.edu/costis/appxhardness.pdf

A1. CFR for Two-player Kuhn Poker

Figure 10 below provides a representation of our CFR algorithm for two-player Kuhn poker after the first iteration of training has been completed. `Player1` had the number `2` card, while `Player2` had the number `3` card. The bold numbers from 1 to 8 indicate the order in which these values get computed. For any round of Kuhn poker, only 4 of the 12 information sets will get calculated.

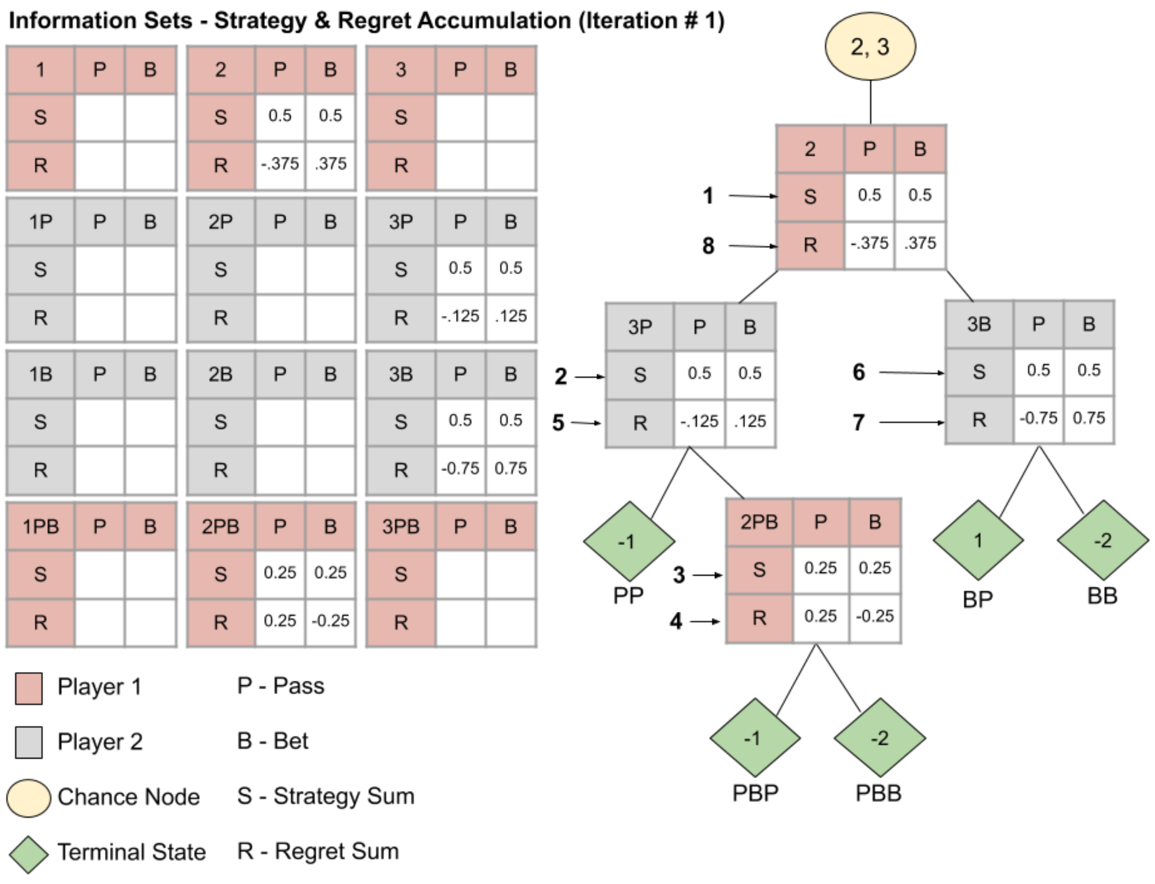


Figure 10: Two-player Kuhn poker game state after iteration 1

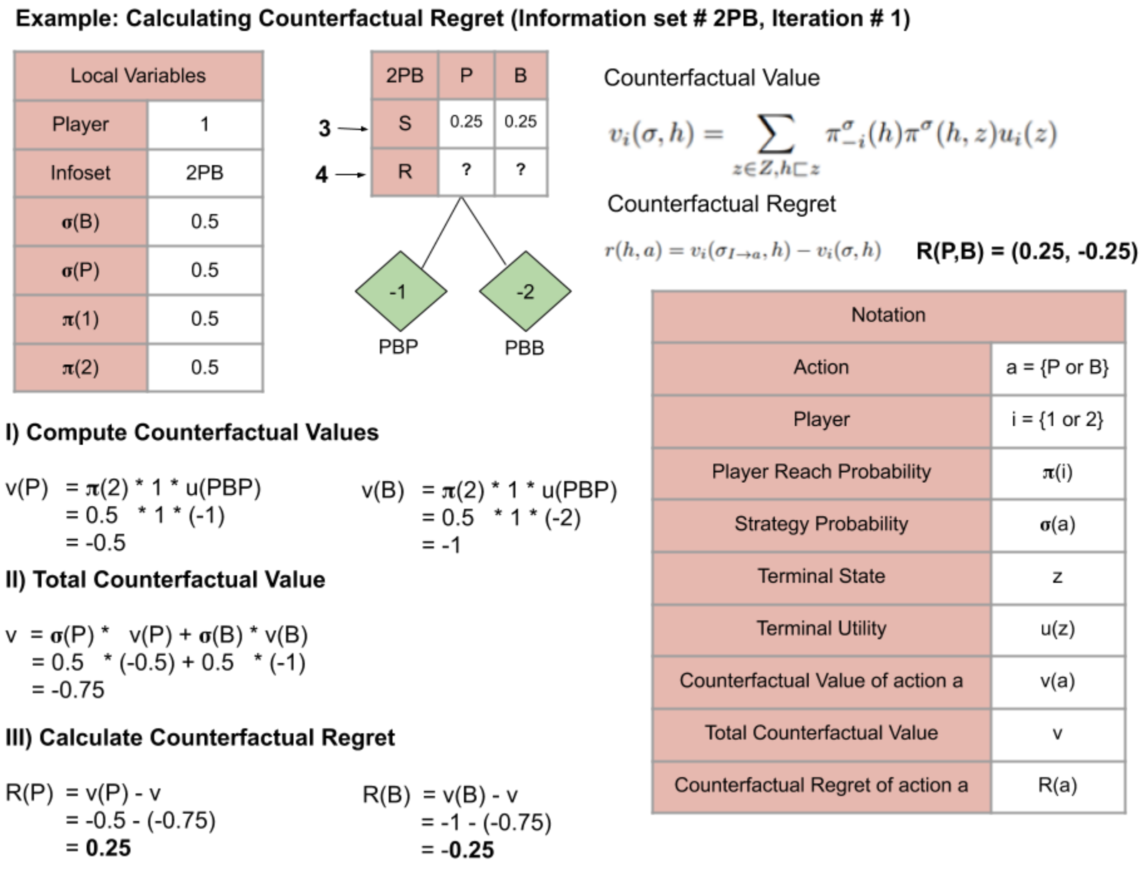
Each information set can be seen as a table that keeps track of four things: `strategy_sum (S)` and `regret_sum(R)` for each action *non-aggressive* (`P`) and *aggressive* (`B`). The `strategy sum` is used to calculate the final strategy profile after training is complete, while the `regret_sum` is used to update the strategy σ^{t+1}. To compute the regret at any node, we recursively call our CFR algorithm to identify the expected regret of future moves based on the current decision. So calculating the regret for information set `2`, which is identified by step 8, we counterfactually play `P`, which re-runs CFR from the `3P` perspective and also `B`, re-running CFR from the `3B` perspective. The resulting utility is passed back up to the parent information set.

Code Snippet 1 shows the core algorithm of CFR for two-player Kuhn Poker, which was inspired by the design in [4]. Reach probabilities `rp0` and `rp1` are probabilities of `Player1` and `Player2` reaching the current information set respectively. On line 2 we get the `strategy` from current information set and update the reach probabilities for each subgame to take into account each players strategy, which corresponds to the probability of taking that action (lines 8 and 10). The expected utility is calculated for both actions (Eq. 1) and added to the `node_util` variable. The regret calculation (Eq. 2) is done on line 16. Finally we increment the total regret for the information set on line 17 (Eq. 3) and return the node utility. The variable `self.NUM_ACTIONS` is simply the integer value 2, for betting and passing. Regret is finally calculated on line 17 but special consideration needs to be taken to account for the reach probability of the current information set. This is done by scaling the regret by the reach probability of the previous player.

```python
reach_probability = rp0 if player == 0 else rp1
strategy = node.get_strategy(reach_probability)
util = [0.0] * self.NUM_ACTIONS
node_util = 0.0
for a in range(0, self.NUM_ACTIONS):
   next_history = history + ('p' if a == 0 else 'b')
   if player == 0:
       util[a] = - self.cfr(cards, next_history, rp0*strategy[a], rp1)
   else:
       util[a] = - self.cfr(cards, next_history, rp0, rp1*strategy[a])

   node_util += strategy[a] * util[a]

# For each action, compute and accumulate counterfactual regret
for a in range(0, self.NUM_ACTIONS):
    regret = util[a] - node_util
    node.regret_sum[a] += rp1 * regret if player == 0 else rp0 * regret
return node_util
```

Code Snippet 1: CFR - Two-player Kuhn poker

Each subgame is visited until a terminal state has been reach, where the utility is calculated based on the game state and what cards each player has. The terminal utility, depending on how far down the game tree it is, will be scaled by the reach probabilities of the players.



Example: Calculating Counterfactual Regret (Information set # 2PB, Iteration # 1)

Local Variables	
Player	1
Infoset	2PB
σ(B)	0.5
σ(P)	0.5
π(1)	0.5
π(2)	0.5

	2PB	P	B
3 →	S	0.25	0.25
4 →	R	?	?

Counterfactual Value

$$v_i(\sigma, h) = \sum_{z \in Z, h \sqsubset z} \pi^{\sigma}_{-i}(h)\pi^{\sigma}(h, z)u_i(z)$$

Counterfactual Regret

$r(h, a) = v_i(\sigma_{I \to a}, h) - v_i(\sigma, h)$ **R(P,B) = (0.25, -0.25)**

Notation	
Action	a = {P or B}
Player	i = {1 or 2}
Player Reach Probability	π(i)
Strategy Probability	σ(a)
Terminal State	z
Terminal Utility	u(z)
Counterfactual Value of action a	v(a)
Total Counterfactual Value	v
Counterfactual Regret of action a	R(a)

I) Compute Counterfactual Values

v(P) = π(2) * 1 * u(PBP)
= 0.5 * 1 * (-1)
= -0.5

v(B) = π(2) * 1 * u(PBP)
= 0.5 * 1 * (-2)
= -1

II) Total Counterfactual Value

v = σ(P) * v(P) + σ(B) * v(B)
= 0.5 * (-0.5) + 0.5 * (-1)
= -0.75

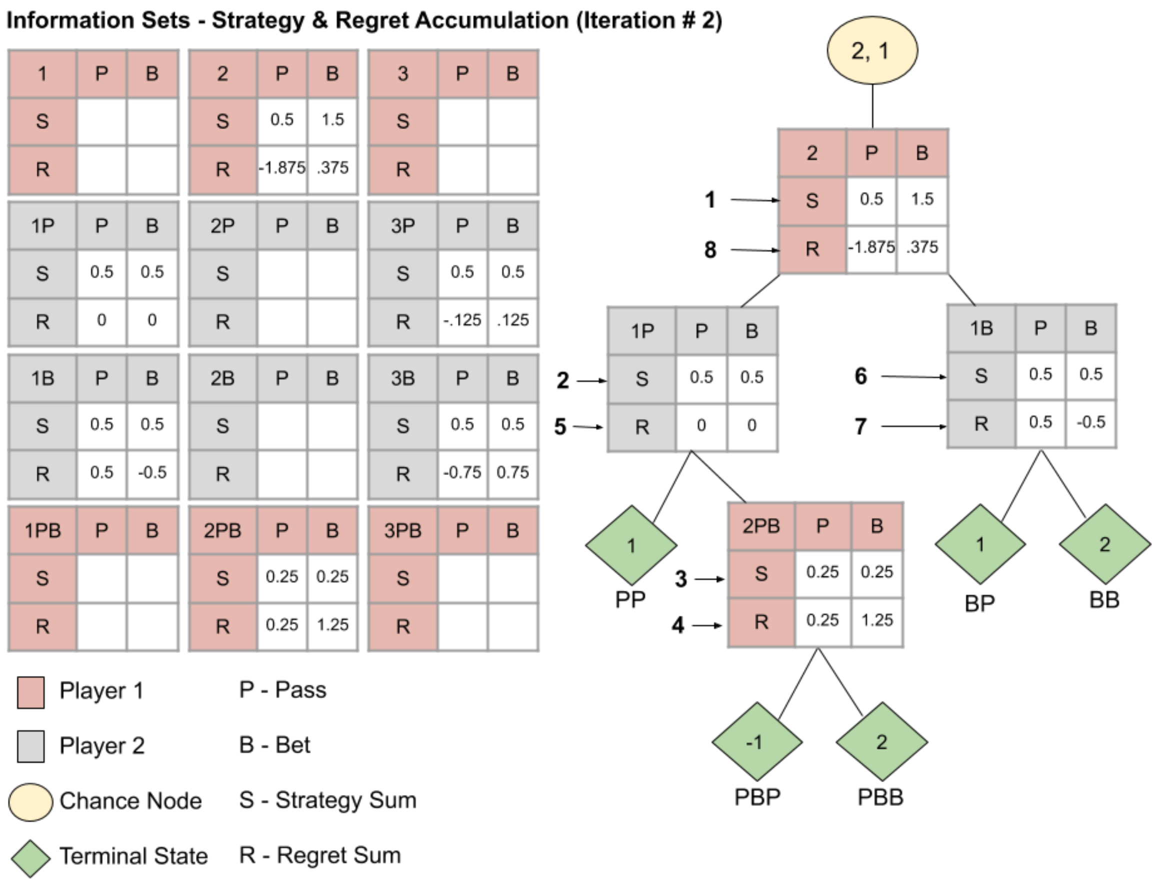
III) Calculate Counterfactual Regret

R(P) = v(P) - v
= -0.5 - (-0.75)
= **0.25**

R(B) = v(B) - v
= -1 - (-0.75)
= **-0.25**

Figure 11: Calculating CFR

After the first iteration, we repeat this process. The next figure shows the state of the information sets after the 2nd iteration. The second round shown below depicts `Player1` with the winning hand with a card value of `2`, and `Player2` having a losing hand with a card value of `1`.



Information Sets - Strategy & Regret Accumulation (Iteration # 2)
1 P B S R
2 P B S 0.5 1.5 R -1.875 .375
3 P B S R
1P P B S 0.5 0.5 R 0 0
2P P B S R
3P P B S 0.5 0.5 R -.125 .125
1B P B S 0.5 0.5 R 0.5 -0.5
2B P B S R
3B P B S 0.5 0.5 R -0.75 0.75
1PB P B S R
2PB P B S 0.25 0.25 R 0.25 1.25
3PB P B S R
Player 1
Player 2
Chance Node
Terminal State
P - Pass
B - Bet
S - Strategy Sum
R - Regret Sum
2, 1
1
8
2
5
6
7
3
4
1 PP
-1 PBP
2 PBB
1 BP
2 BB

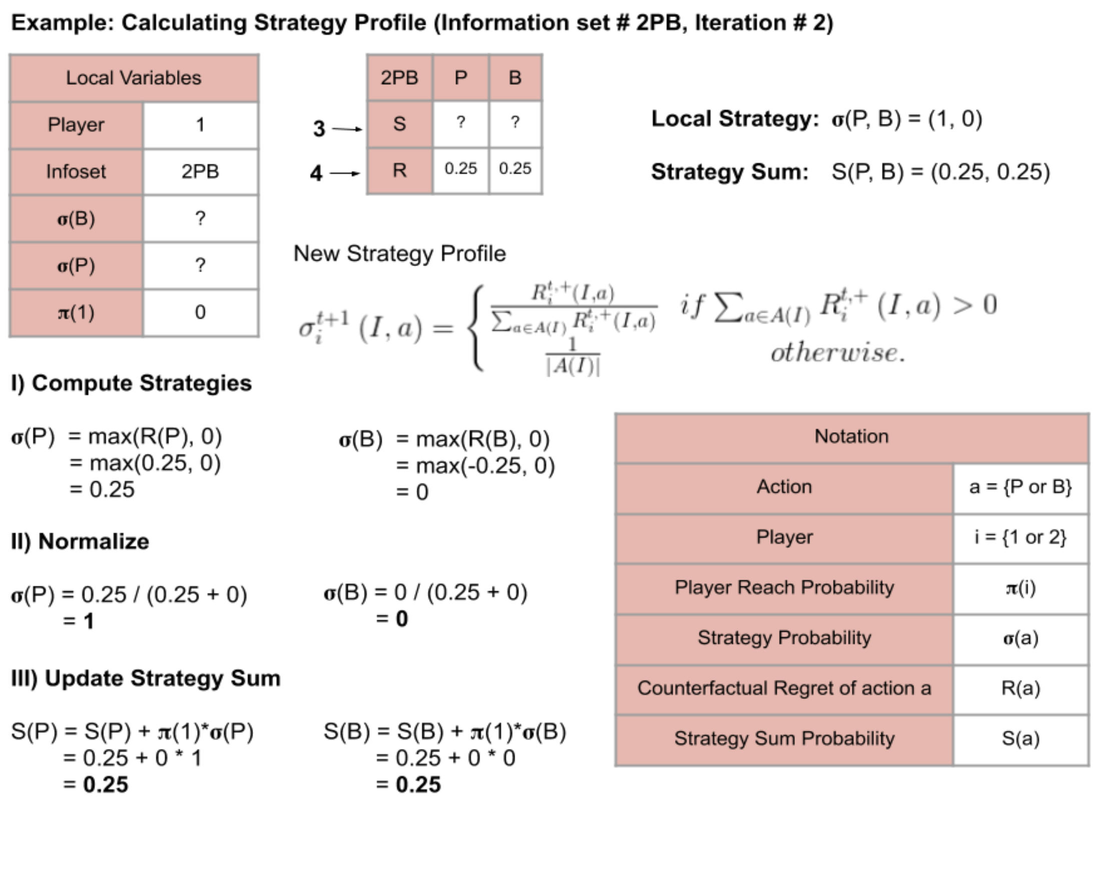
Figure 12: Two-player Kuhn poker game state after iteration 2

Now that the `1` card has been played, we start to accumulate regret and strategy profiles for those information sets as well. Also it is clear that information sets `2` and `2PB` have updated `regret_sum` and `strategy_sum` values. One subtle, but important detail to mention is that there are two strategy variables used throughout training. First there is a local variable `strategy`, or *σ,* which is used to compute probabilities of actions for reaching the sub-games. The `strategy_sum` variable is the accumulation of *σ* for a particular information set multiplied by the probability that the player reaches that information set to begin with. Code Snippet 2 shows both calculations. On line 5 our local *σ* is determined by our previous regret (Eq. 5). If no positive regret has been accumulated, a uniform distribution is used. The loop starting on line 8 is used to normalize the strategy profile. The `strategy_sum` is shown being updated on line 13. When it is updated, it takes into account the players probability of reaching the node, represented as the `realization_weight`.

```
def get_strategy(realization_weight):
    normalizing_sum = 0.0
    strategy = [0.0] * self.NUM_ACTIONS
    for a in range(0, self.NUM_ACTIONS):
    strategy[a] = max(self.regret_sum[a], 0)
    normalizing_sum += strategy[a]

    for i in range(0, self.NUM_ACTIONS):
        if normalizing_sum > 0:
        strategy[i] /= normalizing_sum
    else:
        strategy[i] = 1.0 / self.NUM_ACTIONS
    self.strategy_sum[i] += realization_weight * strategy[i]

    return strategy
```

Code Snippet 2: Choosing a strategy for two-player Kuhn poker

Figure 13 provides an example of how `strategy` and `strategy_sum` get calculated for the information set `2PB`. The `strategy_sum` is just the accumulated local `strategy` multiplied by the reach probability. The complete CFR algorithm for two-player Kuhn poker is shown in Figure 14 which was provided by [4].



Example: Calculating Strategy Profile (Information set # 2PB, Iteration # 2)

Local Variables	
Player	1
Infoset	2PB
σ(B)	?
σ(P)	?
π(1)	0

	2PB	P	B
3 →	S	?	?
4 →	R	0.25	0.25

Local Strategy: σ(P, B) = (1, 0)

Strategy Sum: S(P, B) = (0.25, 0.25)

New Strategy Profile

$$\sigma_i^{t+1}(I,a) = \begin{cases} \frac{R_i^{t,+}(I,a)}{\sum_{a\in A(I)} R_i^{t,+}(I,a)} & if \sum_{a\in A(I)} R_i^{t,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & otherwise. \end{cases}$$

I) Compute Strategies

σ(P) = max(R(P), 0)
= max(0.25, 0)
= 0.25

σ(B) = max(R(B), 0)
= max(-0.25, 0)
= 0

II) Normalize

σ(P) = 0.25 / (0.25 + 0)
= **1**

σ(B) = 0 / (0.25 + 0)
= **0**

III) Update Strategy Sum

S(P) = S(P) + π(1)*σ(P)
= 0.25 + 0 * 1
= **0.25**

S(B) = S(B) + π(1)*σ(B)
= 0.25 + 0 * 0
= **0.25**

Notation	
Action	a = {P or B}
Player	i = {1 or 2}
Player Reach Probability	π(i)
Strategy Probability	σ(a)
Counterfactual Regret of action a	R(a)
Strategy Sum Probability	S(a)

Figure 13: Calculating a strategy profile

A2. CFR for Three-Player Kuhn Poker

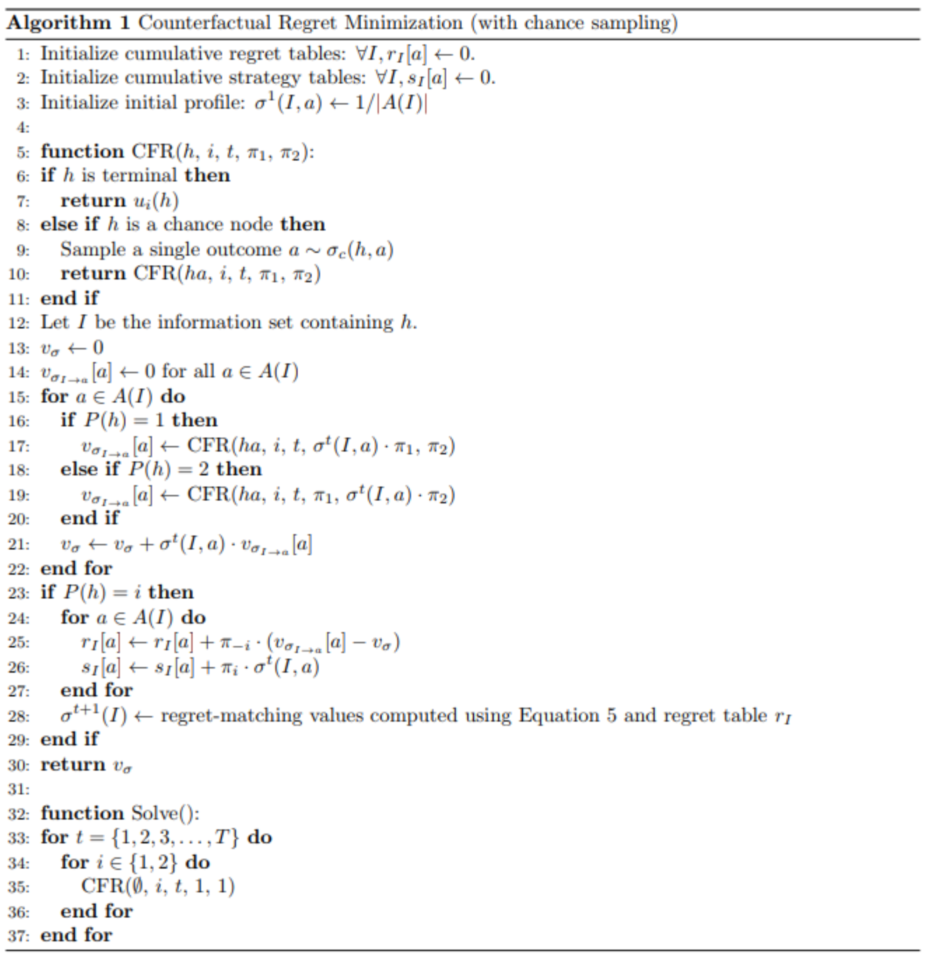
For three-player Kuhn poker, a few changes to the underlying CFR algorithm have to be made. For the two-player implementation we made use of a nice property where utility for one player is exactly the negation of the utility of the other (zero-sum game). While we still have a zero-sum game for multiplayer Kuhn-poker, we cannot simply negate the utility as we did for the two-player game (e.g., Code Snippet #1 lines 8 and 10).

Reviewing Code Snippet #3 we can discern some of the differences of CFR between the two games. Instead of returning a scalar utility value, a list of utilities for each player is returned as `terminal_utilities`. These are calculated by summing across both actions and scaling by the strategy probability as done previously, but now with an array, shown on lines 18 and 19. CFR then chooses the correct utility and stores it in `node_util` on line 21. The `current_player` variable is computed via a simple modulus function on line 3. Regret is also calculated as done before, but making sure to use the previous players reach probability.

```
1  def cfr(cards, history, reach_probabilities):
2      # -- Some code redacted -- #
3      current_player = len(history) % 3
4      for a in range(0, self.NUM_ACTIONS):
5          # For each action, recursively call cfr with additional history and probability
6      next_history = history + ('p' if a == 0 else 'b')
7      if current_player == 0:
8          child_utilities = self.cfr(cards, next_history, [rp0 * strategy[a], rp1, rp2])
9      elif current_player == 1:
10         child_utilities = self.cfr(cards, next_history, [rp0, rp1 * strategy[a], rp2])
11     else:  # Player 2
12         child_utilities = self.cfr(cards, next_history, [rp0, rp1, rp2 * strategy[a]])
13
14     # Calculating CFR for the current infoset
15     util[a] = child_utilities[current_player]
16
17     # Used to pass up to parent infoset
18     weighted_utilities = [strategy[a] * z for z in child_utilities]
19     terminal_utilities = np.add(terminal_utilities, weighted_utilities)
20
21      node_util = terminal_utilities[current_player]
22
23      # For each action, compute and accumulate counterfactual regret
24      for i in range(0, self.NUM_ACTIONS):
25          regret = util[i] - node_util
26
27      if current_player == 0:
28          info_set_node.regret_sum[i] += rp2 * regret
29      elif current_player == 1:
30          info_set_node.regret_sum[i] += rp0 * regret
31      else:
32          info_set_node.regret_sum[i] += rp1 * regret
33
34   return terminal_utilities
```

Code Snippet 3: CFR - Three-player Kuhn poker

A3. CFR Algorithm



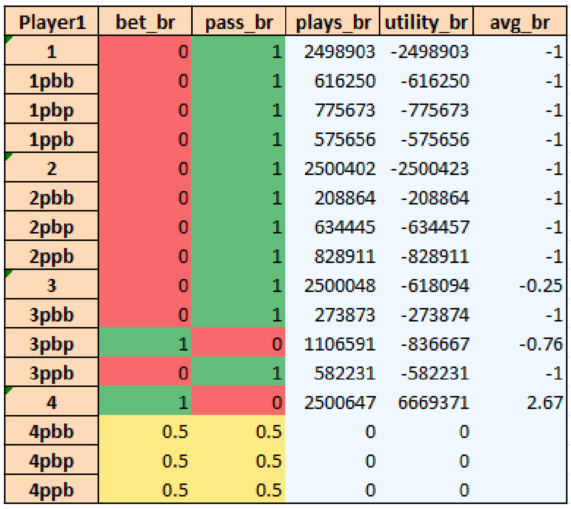
Algorithm 1 Counterfactual Regret Minimization (with chance sampling)

```
Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

function CFR($h$, $i$, $t$, $\pi_1$, $\pi_2$):
if $h$ is terminal then
    return $u_i(h)$
else if $h$ is a chance node then
    Sample a single outcome $a \sim \sigma_c(h,a)$
    return CFR($ha$, $i$, $t$, $\pi_1$, $\pi_2$)
end if
Let $I$ be the information set containing $h$.
$v_\sigma \leftarrow 0$
$v_{\sigma_{I\rightarrow a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ do
    if $P(h) = 1$ then
        $v_{\sigma_{I\rightarrow a}}[a] \leftarrow$ CFR($ha$, $i$, $t$, $\sigma^t(I,a) \cdot \pi_1$, $\pi_2$)
    else if $P(h) = 2$ then
        $v_{\sigma_{I\rightarrow a}}[a] \leftarrow$ CFR($ha$, $i$, $t$, $\pi_1$, $\sigma^t(I,a) \cdot \pi_2$)
    end if
    $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I\rightarrow a}}[a]$
end for
if $P(h) = i$ then
    for $a \in A(I)$ do
        $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I\rightarrow a}}[a] - v_\sigma)$
        $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
    end for
    $\sigma^{t+1}(I) \leftarrow$ regret-matching values computed using Equation 5 and regret table $r_I$
end if
return $v_\sigma$

function Solve():
for $t = \{1,2,3,\ldots,T\}$ do
    for $i \in \{1,2\}$ do
        CFR($\emptyset$, $i$, $t$, 1, 1)
    end for
end for
```

Figure 14: CFR Algorithm from [4]

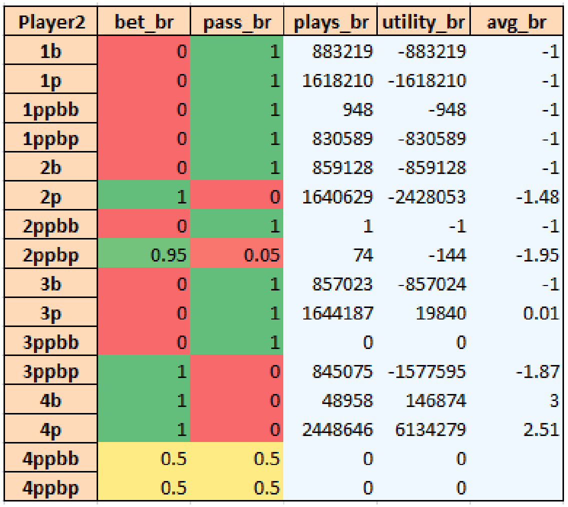
A4. Best Response Strategies

Corresponding Best Response Strategies to the CFR strategies in Section 6 Used to calculate the error.



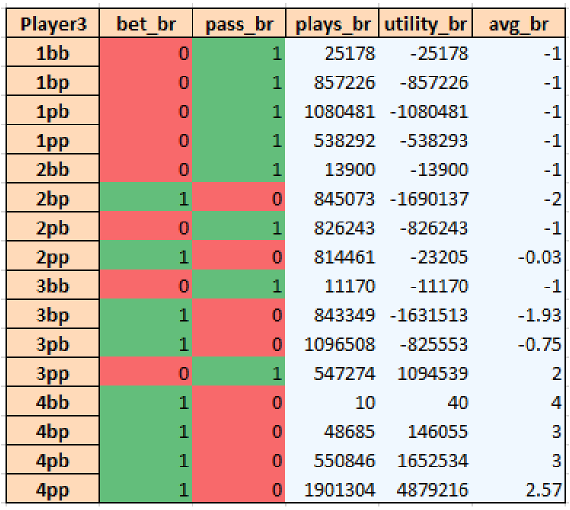
Player1	bet_br	pass_br	plays_br	utility_br	avg_br
1	0	1	2498903	-2498903	-1
1pbb	0	1	616250	-616250	-1
1pbp	0	1	775673	-775673	-1
1ppb	0	1	575656	-575656	-1
2	0	1	2500402	-2500423	-1
2pbb	0	1	208864	-208864	-1
2pbp	0	1	634445	-634457	-1
2ppb	0	1	828911	-828911	-1
3	0	1	2500048	-618094	-0.25
3pbb	0	1	273873	-273874	-1
3pbp	1	0	1106591	-836667	-0.76
3ppb	0	1	582231	-582231	-1
4	1	0	2500647	6669371	2.67
4pbb	0.5	0.5	0	0	
4pbp	0.5	0.5	0	0	
4ppb	0.5	0.5	0	0	

Figure 15: Player one strategy and simulation results



Player2	bet_br	pass_br	plays_br	utility_br	avg_br
1b	0	1	883219	-883219	-1
1p	0	1	1618210	-1618210	-1
1ppbb	0	1	948	-948	-1
1ppbp	0	1	830589	-830589	-1
2b	0	1	859128	-859128	-1
2p	1	0	1640629	-2428053	-1.48
2ppbb	0	1	1	-1	-1
2ppbp	0.95	0.05	74	-144	-1.95
3b	0	1	857023	-857024	-1
3p	0	1	1644187	19840	0.01
3ppbb	0	1	0	0	
3ppbp	1	0	845075	-1577595	-1.87
4b	1	0	48958	146874	3
4p	1	0	2448646	6134279	2.51
4ppbb	0.5	0.5	0	0	
4ppbp	0.5	0.5	0	0	

Figure 16: Player two strategy and simulation results



Player3	bet_br	pass_br	plays_br	utility_br	avg_br
1bb	0	1	25178	-25178	-1
1bp	0	1	857226	-857226	-1
1pb	0	1	1080481	-1080481	-1
1pp	0	1	538292	-538293	-1
2bb	0	1	13900	-13900	-1
2bp	1	0	845073	-1690137	-2
2pb	0	1	826243	-826243	-1
2pp	1	0	814461	-23205	-0.03
3bb	0	1	11170	-11170	-1
3bp	1	0	843349	-1631513	-1.93
3pb	1	0	1096508	-825553	-0.75
3pp	0	1	547274	1094539	2
4bb	1	0	10	40	4
4bp	1	0	48685	146055	3
4pb	1	0	550846	1652534	3
4pp	1	0	1901304	4879216	2.57

Figure 17: Player three strategy and simulation results