Beating the Odds: Theory and Algorithms of Poker

Max Jappert
Seminar: Theory and Algorithms of Puzzles and Games
University of Basel

Supervised by Patrick Ferber

November 24, 2021

Abstract

Since the dawn of Computer Science, imperfect information games like poker have fascinated many of the field's greatest minds. This paper covers both the fundamental ideas and advanced achievements in the algorithmic approach to playing poker. Starting off by covering basic concepts from game theory in an applied manner, affording an intuitive understanding of how an algorithm can deal with imperfect information, the concepts are later incrementally expanded such as to meet the requirements of increasingly complex versions of poker. The last chapter is dedicated to covering state of the art poker AIs by discussing how the previously introduced concepts are utilised within.

1 Introduction

Computers have traditionally been rather good at playing perfect-information games. The fact that an AI managed to beat the world's then leading chess player Garry Kasparov back in 1996 demonstrates this quite well (Tran 1996).

A similar claim can't be made for poker, although it played a significant role as a case study for seminal game theoretical papers midway during the last decade (e.g. von Neumann 1928), since, as an imperfect-information game, it resembles many real-life situations much more closely than its perfect-information counterparts. It is very rare that we find ourselves having complete knowledge of a scenario in which we need to make important decisions. Think debates, investing in the stock market, placing a strategic vote, or partitioning ones study-time for each exam. These are all situations in which we have to strategically work with uncertainty, which is very unlike the aforementioned games, in which the information is shared, and very much like poker, in which the aspect of uncertainty is unavoidable for strategic play.

Surprisingly, the first poker-playing AI feasibly capable of beating humans was only released in 2015. Even more surprising is that since then multiple AIs have beaten

professional players by unbelievable margins, thereby establishing themselves as some of the most significant advancements in the entire field over the past few years.

This paper is going to cover the algorithmic approach to playing poker, thereby demonstrating a technique, with which we can let computers deal with imperfect information effectively. Starting off with the formal basics and game theoretic concepts, it will incrementally introduce more complex variations of the previously introduced concepts, with the goal of affording a self-contained introduction to playing poker algorithmically.

2 Formal Concepts used in Poker Algorithms

2.1 Rules of Poker

There are many different variations of poker, such that it lies outside the scope of this paper to describe each of them in detail. What they share in common, are the following rules:

Each player gets dealt a certain amount of cards, which are only visible to each player respectively. In certain variants there are also cards lying face-up on the table, such that all can see them. How many cards these are and when they're laid on the table varies from variant to variant. Each player can bet money on them winning the round. In order to be able to win the round, a player must always meet the maximum amount which has been bet to stay in the game, which means that the highest bidder decides on how much everyone must bet. The players can at any point decide to check, meaning that they don't need to continue raising their bet, yet they also cannot win the round and loose all they have already bet for the round. Winning the round means either being the only one left who hasn't checked, or in the case where nobody wants to raise (or the variant doesn't permit further betting) having the most valuable card combination. How card combinations are valued is not going to be of importance in this paper. The player who wins the round is rewarded all the money which has been bet by all players during the given round.

The goal of an algorithm is maximizing the amount of money (we'll use the term *payoff*) which can be won. In order to achieve this, we need a theoretical framework to understand how that can be reached in the case, where each player only has a limited amount of information on the game state. That theoretical framework is game theory, whose concepts will be introduced in the following sections.

2.2 Nash Equilibrium

The concept of Nash equilibrium exceeds the scope of purely playing poker. It was introduced by mathematician John Nash in 1950 and is defined as follows:

Definition 1 (Nash equilibrium (Milovsky 2014))**.**
A set of strategies, such that each player, considering all information they have gained during the game, cannot benefit from unilaterally changing their own strategy is called a Nash equilibrium.

This paper will demonstrate the concept of a Nash equilibrium using the normal form of two games.

2.3 Normal-form games

The normal form of games can be informally defined as a representation in which the payoff of a relation of actions for 2 players and k possible actions per player can be represented as a $\mathbb{R}^{k \times k}$ matrix for each decision point in the game, with each entry representing an n-tuple of payoffs.[1] For formally defining the normal form of a game this paper will follow **Definition 2**.

Definition 2 (Normal-form game (Neller and Lanctot 2013))**.**
A normal-form game is a tuple (N, A, u), where:

- $N = \{1, ..., n\}$ *is a finite set of n players.*
- A_i *is a finite set of possible actions for player i.*
- $A = A_1 \times ... \times A_n$ *is the set of all possible simultaneous actions. Each of its elements $a \in A$ is an n-tuple called an action profile, whereby $a_i \in A_i$ denotes the action chosen by player i.*
- u *is a function mapping each action profile and each strategy profile σ (introduced later) to a vector of utilities (also referred to as payoffs) for each player. Player i's payoff function $u_i : A \longrightarrow \mathbb{R}$ calculates the payoff for player i given an action profile.*

We'll first demonstrate the concept of a Nash Equilibrium on a simple and seminal example: the prisoners dilemma. The game consists of two agents, P_1 and P_2, who are convicted of plotting a crime. They are separately interrogated, whereby they cannot communicate, and offered the following options:

1. If both don't confess, they'll each get a short prison sentence, e. g. 1 year.
2. If both confess, they'll both get the same medium prison sentence, e. g. 3 years.
3. If P_i confesses and P_j doesn't for $i, j \in \{1, 2\}$ and $i \neq j$, then P_j is free to leave and P_i gets a long prison sentence, e.g. 5 years.

These conditions can be visualized in normal form with the following $\mathbb{R}^{2 \times 2}$ payoff matrix, due to the fact that both players have $|A_i| = 2$ possible actions. Each box represents the payoff of one action profile $u(a_1, a_2)$ representing the loss for each player in the given scenario. Each can choose from their set of possible actions A_i, whose elements define the rows and the columns.

[1] This paper introduces the normal form because the payoff matrix makes visualizing the game theoretical aspects easier, but for more complex games the payoff matrix becomes impractical. E.g. for heads-up limit Texas hold'em poker there are $k = 3.589 \times 10^{13}$ possible actions (Johanson 2013), which would make a matrix representation highly impractical. More on that in later chapters.

	P_1 confesses	P_1 doesn't confess
P_2 confesses	$-3, -3$	$0, -5$
P_2 doesn't confess	$-5, 0$	$-1, -1$

2.4 Imperfect-Information Games

In a perfect information game, like e.g. chess, both players have access to the same knowledge of the game state, i.e. they have full knowledge of their opponent's future options and past actions. In the prisoner's dilemma, this is only partially the case: the strategy needs to be chosen under the condition that the other person's strategy is unknown. The same goes for poker. This notion is formalized as an imperfect-information game.

Definition 3 (Imperfect-Information Games, (Levin 2002))**.**
A game in which its players do not have common knowledge of the game is called an imperfect-information game.

3 Counterfactual Regret Minimization

A widely used algorithm for playing poker is Counterfactual Regret Minimization, henceforth abbreviated as CFR. CFR finds a strategy by approximating the Nash equilibrium for a given game. It does this by playing against itself and thereby learning how to minimize regret. It iteratively deduces a strategy (in form of a probability distribution over the set of possible actions) from the cumulative regret. Regret in this context refers to how much the algorithm "regrets" not having chosen a given action. In CFR, the counterfactual regret $r(a_i)$ is computed for each $a_i \in A_i$ by subtracting the achieved payoff $u_i(a_i)$ from the possible payoff $u_i(a_i')$ if a_i' had been chosen instead.

In more formal terms we can define this as follows for a two-player game: For a_i being player i's action and a_{-i} being the opponent's action, we can compute the immediate regret $r_i^t(a_i')$ for round $t \in \mathbb{N}$ and each possible action a_i' as follows:

$$r_i(a_i') = u_i(a_i', a_{-i}) - u_i(a_i, a_{-i}) \tag{1}$$

This corresponds to the aforementioned informal description. For calculating the strategy we divide the cumulative regret $R_i(a_i)$ for each action $a_i \in A_i$ by the total regret of all actions. The cumulative regret for action $a_i \in A_i$ is trivially computed as in equation (2), whereby $T \in \mathbb{N}$ refers to the amount of rounds played.

$$R_i^T(a_i) = \sum_{t=1}^{T} r_i^t(a_i) \tag{2}$$

In simple terms, a computer playing poker requires a way to choose an action at each decision point in a game. The way CFR accounts for this is by defining a probability distribution over the set of possible actions A_i, whereby the probability $\sigma_i(a_i)$ denotes the probability that player i chooses action $a_i \in A_i$. There are two more important concepts related to strategies: A strategy profile $\sigma \in \Sigma$ consists of a strategy σ_i for all players $i \in N$ and Σ_i denotes the set of strategies for player i.

Since the strategy is defined as a probability distribution over A_i, we need to compute a probability $\sigma(a_i)$ for each $a_i \in A_i$, such that $\sum_{a_i \in A_i} \sigma_i(a_i) = 1$. We do this by dividing the cumulative regret by the total cumulative regret if the latter is not equal to 0, since that would imply that no rounds have been played yet (and a division by 0). We formally define the computation of the strategy next strategy (i.e. for round $T+1$) in equation (3).

$$\sigma_i^{T+1}(a_i) = \begin{cases} \frac{R_i^T(a_i)}{\sum_{a_i' \in A_i} R_i^T(a_i')} & \sum_{a_i' \in A_i} R_i^T(a_i') \neq 0 \\ \frac{1}{|A_i|} & \text{otherwise} \end{cases} \tag{3}$$

3.1 Example: Rock-Paper-Scissors

This paper will use the game rock paper scissors to demonstrate how CFR uses regret minimization to approximate the Nash equilibrium. We can use the normal form for this game, due to the fact that there is only 1 decision point with $|A_i| = 3$ possible actions. The payoff matrix has the following form:

	P_1 rock	P_1 paper	P_1 scissors
P_2 rock	$0, 0$	$-1, 1$	$1, -1$
P_2 paper	$-1, 1$	$0, 0$	$-1, 1$
P_2 scissors	$1, -1$	$-1, 1$	$0, 0$

The algorithm iteratively performs the following four steps, with each cycle representing a single round it played against itself.

1. Decide on an action depending on the strategy (which is trivial due to the fact that the strategy is a probability distribution over the set of possible actions).
2. Calculate the achieved payoff from choosing this strategy using the payoff matrix.
3. Calculate the immediate regret $r_i^T(a_i)$ for each possible action $a_i \in A_i$. This is done as defined in equation (1).
4. Calculate the next strategy $\sigma_i^{T+1}(a_i)$ for the next round $T+1$ by means of equation (3).

For example, the computation would look as such for $T = 0$ and player i:

1. We start off by initializing the strategy as evenly distributed, which means that $\sigma_i^T(a_i) = \frac{1}{|A_i|}$ for all $a_i \in A_i$, as in equation (3). The probability for choosing each action is equal and we choose $a_i = "Paper"$.

2. Our opponent has chosen $a_{-i} = "Scissors"$, which means that our payoff for this round is $u_i(\sigma^T) = u_i("Paper", "Scissors") = -1$.

3. Our immediate regret for each action is computed as in equation (1), so it follows that $r_i^T(a_i') = u_i(a_i', "Scissors") - u_i("Paper", "Scissors")$. That results in the following immediate regrets:

$$\begin{array}{lcll} r_i^T("Rock") & = & 1-(-1) & = 2 \\ r_i^T("Paper") & = & -1-(-1) & = 0 \\ r_i^T("Scissors") & = & 0-(-1) & = 1 \end{array}$$

4. We update our strategy as defined in equation (3), which results in the following updated strategy:

$$\begin{array}{ll} \sigma_i^{T+1}("Rock") & = \frac{2}{3} \\ \sigma_i^{T+1}("Paper") & = 0 \\ \sigma_i^{T+1}("Scissors") & = \frac{1}{3} \end{array}$$

This process is repeated until it converges to the optimal strategy $\sigma_i^T(a_i) = \frac{1}{3}$ for all $a_i \in A_i$ at roughly $T = 5 \times 10^5$, which is somewhat obvious (although research shows that in practical play humans often don't abide to this strategy [Batzilis et al. 2019]). It is also the Nash equilibrium, because playing with this strategy entails that no player can benefit from unilaterally changing their own strategy considering their knowledge of the game. For playing against another entity than itself, the algorithm would simply take the average strategy and play accordingly.

3.2 Example: Kuhn Poker

While rock-paper-scissors is definitely not poker, it shows the idea of CFR and thereby the way an algorithm can deal with imperfect information quite well. Since real poker is quite a bit more complex, I'd like to add an intermediate step and look at how we can adapt this algorithm to deal with a toy poker game. We'll do so with the drastically simplified version of poker called Kuhn poker. It was introduced by mathematician Harold W. Kuhn as a simple sequential two-player imperfect-information game on which a complete game theoretical analysis can be performed (Kuhn 2016).

The game involves three cards whose values differ, this paper will henceforth consider them to be Jack, Queen and King, and each player is dealt one card. Before the round starts, both players have to bet one chip. Then one player starts and can either bet a further chip or check. Thereafter, the other player does the same. If one player bets and the other passes, the player who has bet wins the round (except if the starting player passes and the other player bets, then the starting player has the chance to also bet), and if both pass or both bet the player with the more valuable card wins the round.

Kuhn poker, unlike the games we've looked at so far, is a sequential game, it consists of a sequence of actions. Although it would in theory be possible to treat it in normal-form by assuming that both players calculate their strategy at the beginning of each game for all possible actions the opponent could make, but this would result in a huge and impractical matrix, whereby we must consider that each action must be treated differently depending on the point in the game at which it is played, how much has been bet, etc. Additionally, for the sake of approaching how proper poker is computed, we'll be representing Kuhn poker in *extensive form*. Thereby the game is represented as a tree with chance nodes, terminal nodes and choice nodes. Perfect-information games can also be represented as such, yet for imperfect information games we need to consider that certain nodes are indistinguishable from each other at a give game state, since the player doesn't have full knowledge of the game state (other than in chess, for example, where all past and possible moves are known and therefore all nodes are, in theory, distinguishable). We take this into account by partitioning the nodes into sets of nodes, whereby each set contains nodes, which are indistinguishable from each other for the player. We call them *information sets*.

For example: You are player 1 with a Queen and have have to decide if you want to bet or pass. For you the node in the game tree representing player 2 having Jack and the node representing player 2 having King aren't distinguishable, because you simply lack the information needed in order to make the distinction. Therefore, those two nodes are in the same information set.

Extensive-form games are best represented as a game tree. Other than with non-toy poker, Kuhn poker is simple enough, i.e. it has few enough nodes, such that this tree can be feasibly printed. Figure 1 shows the game tree for Kuhn poker.

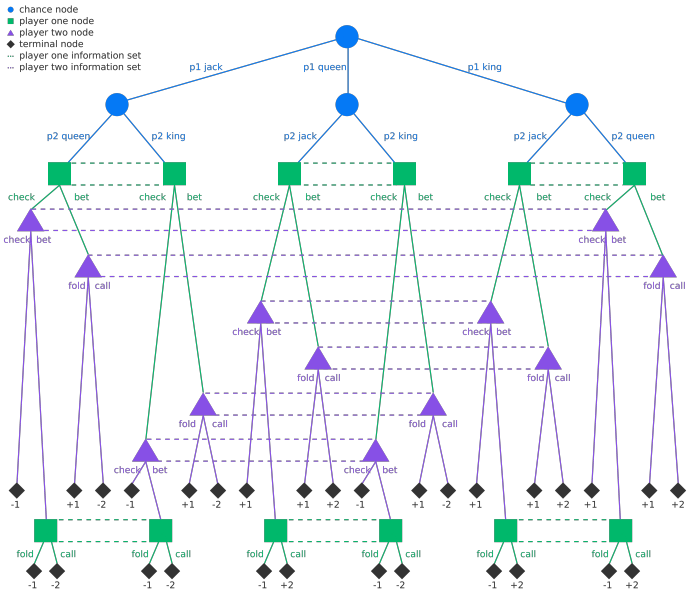
The round nodes are the chance nodes: they denote the random allocation of cards. The square and triangular nodes are the decision nodes: At each such node a decision is met, in our case to either check or bet. The rhombic nodes are the terminal nodes: when they are reached, the game is over and the payoff is determined.

Each information set is shown as one horizontally aligned bar of dotted lines. This intuitively makes sense. E.g. being at the leftmost chance node, the decision node where player 2 has Queen and the decision node where player 2 has King are not distinguishable and they are therefore in the same information set.

3.2.1 Adapting CFR for extensive-form Games

Having intuitively introduced the extensive-form representation of Kuhn poker, we will now formally define the extensive-form representation of a game with imperfect information. The definition may look intimidating at first glance (it does, after all, take up more than half a page), yet it builds upon and includes many of the concepts which have already been introduced for normal-form games. The most noteworthy addition are the histories, which describe a game state as the path which has been taken along the game tree.

Definition 4 (Finite extensive-form representation of a game with imperfect information (Zinkevich et al. 2007))**.**
A finite extensive game with imperfect information has the following components:



Source: https://justinsermeno.com/img/kuhn_game_tree.svg

Figure 1: The game tree for Kuhn poker.

- *A finite set N of players $i \in N$.*
- *A finite set H of histories, which are represented as sequences of actions. The empty sequence is in H and every prefix $h' \sqsubset h$ of a sequence $h \in H$ is also in H. $Z \subseteq H$ is the set of terminal histories, which includes all histories which have reached a terminal node and therefore aren't a prefix of any other history. For the non-terminal histories $h \in H \setminus Z$, the function $A(h) = \{a \mid (h, a) \in H\}$ returns a set of possible actions which can be performed after the sequence denoted by h.*
- *A function $P : H \setminus Z \longrightarrow N \cup \{c\}$, whereby $\{c\}$ denotes the chance events, which maps each non-terminal history to a player or a chance event, such that $P(h)$ is the player who takes an action at the end of h. This implies that if it's player i's turn in a game round described by h, then $P(h) = i$. For the case that $P(h) = c$, a chance event follows h.*
- *A function f_c which assigns a probability for each chance event c to each history $h \in H \setminus Z$ where $P(h) = c$ holds. $f_c(c \mid h)$ thereby denotes the probability that chance event c occurs given h.*
- *A partition $\mathcal{I}_i$ of the set $\{h \in H \mid P(h) = i\}$ for every player $i \in N$ with the property that $A(h) = A(h')$ for all h, h' in the same partition $I_i \in \mathcal{I}_i$. A's domain is extended such that $A(I_i)$ denotes the set $A(h)$ and $P(I_i)$ denotes the player $P(h)$. I_i is called an information set of player i, whereby the information partition $\mathcal{I}_i$ includes all information sets of player i.*

- *A utility function $u_i : Z \longrightarrow \mathbb{R}$ for every player i, which maps each terminal history (i. e. each terminal node represented as a terminal history) to a payoff in $\mathbb{R}$. If $N = \{1, 2\}$ and $u_1(z) + u_2(z) = 0$ for all $z \in Z$, the game is zero-sum.*

The histories are represented by a string of characters. Examples of such strings and how they are intepreted can be found in chapter 3.2.2. The strategies are used in a similar way as in normal-form games. The strategy of player i is also denoted by σ_i and it assigns a probability distribution over a set of actions $A(I_i)$ for each information set $I_i \in \mathcal{I}_i$, instead of mapping each action to a probability individually.

We further define the reach probability π^σ, which is the probability of history h occuring if strategy profile σ is used by the players. The reach probability consists of the contributions of all players and chance nodes π_i^σ with $i \in N \cup \{c\}$ multiplied together.

For later usage we'll define the reach probability of a terminal node given a history in equation (4).

$$\pi^\sigma(h, z) = \begin{cases} \frac{\pi^\sigma(z)}{\pi^\sigma(h)} & h \sqsubseteq z \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

For extensive-form games, we need to store the counterfactual regret for each action in for each information set separately, instead of only once, as with a normal-form game. This is due to the fact that the regret must be computed for each decision point in a game. Since the previously introduced games only consist of a single decision point, only one set of regrets had to be computed, while Kuhn poker consist of a decision point per information set. Additionally, as Zinkevich et al. (2007) have shown, the sum of counterfactual regrets is never less than the overall regret. This makes it possible to, instead of minimizing the overall regret, minimize each counterfactual regret seperately and thereby minimizing overall regret. Before presenting the equation, we need to define $\sigma_{(I \to a)}$ as the strategy profile identical to σ except that player $P(I)$ always chooses action $a \in A(I)$ in information set $I \in \mathcal{I}_i$. The formula for calculating the cumulative counterfactual regret $R_i^T(I, a)$ for player i, round T and action $a \in A(I)$ in information set $I \in \mathcal{I}$ is the following:

$$R_i^T(I, a) = \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t_{(I \to a)}, I) - u_i(\sigma^t, I)) \tag{5}$$

Although the equation looks intimidating, it's basically the same as for the normal form game. The main difference is that the utility function u_i has the domain Σ_i instead of A and the counterfactual regret for each round t is scaled by the probability $\pi_{-i}^{\sigma^t}$ of reaching the given information set I with the opponent having made the last move.

We also define $R_i^{T,+}(I, a) = \max(R_i^T(I, a), 0)$, meaning the cumulative counterfactual regret for action a in information set I such that it becomes 0 if it has a negative value, player i's probability for playing action a in information set I is denoted as σ_i^{T+1} and computed with equation (6). The overall strategy, i.e. the probability distributions over actions in information sets, is composed of those values.

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a' \in A(I)} R_i^{T,+}(I,a')} & \text{if } \sum_{a' \in A(I)} R_i^{T,+}(I,a') > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases} \tag{6}$$

The value computed if the first condition in (6) is met corresponds quite closely to how the probability for playing an action in the next round is computed for normal-form games. The cummulative regret for the given action a played in information set I is divided (and thereby normalized) by the total regret for that information set. If the total regret for the information set is 0, that means no round has been played yet and therefore the probability for being played is initialized as being equal for each action. The counterfactual regret $r(a, h)$ of not taking action a at history h or information set I is computed by means of the counterfactual value as in equations (8) and (9) respectively. The counterfactual value $v_i(\sigma, h)$ is computed as in equation (7).

$$v_i(\sigma, h) = \sum_{z \in Z, h \sqsubset z} \pi_{-i}^{\sigma}(h) \pi^{\sigma}(h, z) u_i(z) \tag{7}$$

$$r(h, a) = v_i(\sigma_{(I \to a)}, h) - v_i(\sigma, h) \tag{8}$$

$$r(I, a) = \sum_{h \in I} r(h, a) \tag{9}$$

3.2.2 Using CFR to approximate the Nash Equilibrium for Kuhn Poker

With the components introduced, we can now show how they are utilized in CFR to approximate the Nash equilibrium for Kuhn poker. On an abstract level, this is done quite similarly as with rock-paper-scissors, as in that the algorithm iteratively plays against itself and tries to minimize the overall regret by using the regret to update its strategy. On a concrete level, it does this by minimizing the counterfactual regret for each information set seperately. In order to do this, it has to perform a recursive depth-first traversal of the entire game tree for each iteration, thereby calculating the counterfactual regret for each action in each information set. With that done, a new strategy $\sigma_i^{T+1}(I, a)$ is computed by normalizing the cumulative counterfactual regret $R_i^T(I, a)$ for each action a_i in information set I.

A significant difference to CFR performing on a normal-form game is that we need to keep track of the history, as seen in the formulas above. This is implemented as a string, whereby each character represents a node we have visited. This paper assigns the following characters:

- `'r'`: Chance node
- `'c'`: Check
- `'b'`: Bet

E.g. `"rrcbc"` describes the history whereby after two chance nodes (i.e. both players being dealt cards) player 1 checks, then player 2 bets, then player 1 checks again.

As the game tree is recursively traversed, the relevant function checks if the given history represents a terminal node (as does the example history in the above paragraph), in which case the utility functions are computed and the regrets and strategies are updated. If it is not a terminal node, then the tree gets traversed further by recursively calling the same function for both versions of the extended history, i.e. with both `"c"` and `"b"` appended to the current history. Thereby the tree is traversed entirely per iteration and the counterfactual regrets are computed for each action in each information set.

After $T = 10^5$ iterations, the computed strategy profile and approximated Nash equilibrium σ (consisting of all elements $a_1 \in A_1$ and $a_2 \in A_2$) for Kuhn poker will look as such (Sermeno 2014):

	Check	Bet
J "rr"	0.79	0.21
J "rrcb"	1.00	0.00
K "rr"	0.39	0.61
K "rrcb"	0.00	1.00
Q "rr"	1.00	0.00
Q "rrcb"	0.45	0.55

Table 1: Approximated optimal strategy for player 1.

	Check	Bet
J "rrb"	1.00	0.00
J "rrc"	0.67	0.33
K "rrb"	0.00	1.00
K "rrc"	0.00	1.00
Q "rrb"	0.66	0.34
Q "rrc"	1.00	0.00

Table 2: Approximated optimal strategy for player 2.

J, Q and K refer to the card which has been dealt to the player and together with the string denoting a history refers to an information set. The floating point numbers in the range $[0, 1]$ refer to the probabilities $\sigma_1^t(a, I)$ of action a in information set I being played at $T = 10^5$. The rows denote the information sets I and the columns denote the actions a within the given information set.

3.2.3 Monte-Carlo CFR

The CFR algorithm suffices to approximate the Nash equilibrium for Kuhn poker, yet it scales badly, since the entire game tree gets traversed for every round of self-play. Considering that the size of a game tree in extentensive-form grows exponentially with additional cards, bet sizes and players, and therefore the computations required also grow exponentially, it follows that certain adjustments need to be made. One seminal adjustment to reduce the amount of computation is Monte-Carlo CFR (Lanctot et al. 2009), henceforth abreviated as MCCFR. The goal of MCCFR is to avoid traversing the entire game tree. How this is done in detail can be found in the cited paper. The authors empirically show that their method reduces time needed to converge significantly.

3.3 Modern poker AIs

MCCFR was famously used by Bowling et al. in 2015 in their seminal paper on being the first to weakly solve[2] heads-up limit hold'em poker, a two-player variant of poker with 3.16×10^{17} possible states and 3.19×10^{14} decision points.[3]

The authors of the paper additionally used a technique called sub-game solving, which this paper is not going to address in detail. The technique was introduced by Burch et al. in 2014. It allows for treating each node as responsible for a set of subgames, which can be separately solved. Bowling et al. made use of this due to the fact that, despite not using floating-point numbers and compressing the stored regrets and strategies by ratios of 13-to-1 and 28-to-1 respectively, the regrets still take up 11 TB of storage, the strategies 6 TB, which obviously exceeds any feasible memory availability. By treating the game as a set of subgames, each subgame can be solved seperately and only the relevant subset of regrets and strategies is loaded into memory at any given time.

Additionally, to even further improves the rate of convergence, they make use of CFR+, introduced by Tammelin (2014). CFR+ firstly converts the floating-point arithmetic of regret- and strategy-storagy to fixed-point integer arithmetic in order to reduce storage. Secondly, it uses regret-matching$^+$, which treats negative counterfactual regrets as being equal to 0. The authors argue that this reduces the time needed for convergence significantly.

3.3.1 DeepStack

DeepStack was the first AI to beat professional poker players by a statistically significant margin. It was introduced by Moravčík et al. only recently in 2017. What makes this more impressive is that it plays Heads-up no-limit hold'em, which has roughly 10^{160} game states. This is significantly more than the roughly 10^{14} states of heads-up limit hold'em solved by Bowling et al. This is due to the fact that there's no restriction on individual betting (hence the name "no-limit"), which trivially leads to a significant increase in possible game states.

Instead of explicit abstraction in order to reduce the amount of game states in order to calculate a compute a complete set of strategies prior to playing, it recursively iterates through a part of the game tree *while playing*. It learns how to decide which part of the game tree it should traverse in real time by means of deep learning prior to play. For deep learning it uses "examples from random poker situations" (Moravčík et al. 2017).

3.3.2 Pluribus

The last and most recently developed poker AI covered in this paper is Pluribus, which, as the name suggests, was the first AI to beat professional poker players in six-player

[2] Weakly solving a game refers to developing an algorithm which can guarantee a win or a draw against any possible opponent, if played from the beginning.

[3] These numbers differ because, as we've established throughout this paper, poker is an imperfect information game and therefore certain decision points are indistinguishable to the player and therefore grouped into information sets I.

no-limit hold'em. It was introduced by Brown and Sandholm in 2019. The multiplayer context changes a lot compared to the two-player context, which was exclusively considered in this paper up until now. In the special case of two-player zero-sum games, if all players independently compute a Nash equilibrium and let it dictate their play, the list of strategies used still ends up being a Nash equilibrium. This is not necessarily the case for games with $n > 2$ therefore it is also not necessarily beneficial to compute a Nash equilibrium for a multiplayer game.

The authors state that Pluribus' self-play does not necessarily converge to an Nash equilibrium. Yet as with Libratus, it uses MCCFR with both information- and action-abstraction to compute a blueprint strategy, which is used in the first round of betting. It later uses real-time sub-game solving. How this is done will not be further covered, since it would exceed the scope of this paper.

4 Conclusion

This paper has tried to deliver a self-contained introduction into how poker, as an imperfect-information game, can be tackled by an algorithm. We therefore introduced CFR, demonstrating its functionality on rock-paper-scissors in normal form and later extending its scope to be able to compute an optimal strategy for Kuhn poker in sequential form. Due to the fact, that many powerful poker AIs use variations of CFR, the last part was dedicated to showing how some of those adapt CFR to become more powerful and efficient.

This paper has hoped to establish the fact, that huge achievements can be accomplished using comparably simple concepts, like the Nash equilibrium and the methods to approximate it (i.e. CFR). Of course the formulas to compute the individual components become more complex as the scope of the games increases, while new measures need to be met in order to reduce memory requirements and computational complexity to a point of manageability, yet the fundamental concepts remain the same as the ones that can be used to beat rock-paper-scissors. Alone the fact that CFR (or rather variants thereof) are used in all the seminal poker AIs introduced in chapter 3.3 is somewhat impressive.

And despite this, it took some of the greatest minds in the field decades to even develop an algorithm such as CFR, let alone weakly beat the game of poker. But once that milestone (i.e. weakly solving poker) had been reached, new milestones were achieved on nearly a yearly basis. Today, AIs such as Pluribus are considered unbeatable, even by the best players the game has to offer.

References

Batzilis, D., Jaffe, S., Levitt, S., List, J. A., and Picel, J. (2019). Behavior in strategic settings: Evidence from a million rock-paper-scissors games. *Games*, 10(2):18.

Bowling, M., Burch, N., Johanson, M., and Tammelin, O. (2015). Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149.

Brown, N. and Sandholm, T. (2019). Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890.

Burch, N., Johanson, M., and Bowling, M. (2014). Solving imperfect information games using decomposition. In *Twenty-eighth AAAI conference on artificial intelligence*.

Johanson, M. (2013). Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008*.

Kuhn, H. W. (2016). 9. a simplified two-person poker. In *Contributions to the Theory of Games (AM-24), Volume I*, pages 97–104. Princeton University Press.

Lanctot, M., Waugh, K., Zinkevich, M., and Bowling, M. (2009). Monte carlo sampling for regret minimization in extensive games. *Advances in neural information processing systems*, 22:1078–1086.

Levin, J. (2002). Games of incomplete information. *Standford Education. Retrieved*.

Milovsky, N. (2014). The basics of game theory and associated games.

Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49.

Neller, T. W. and Lanctot, M. (2013). An introduction to counterfactual regret minimization. In *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, volume 11.

Sermeno, J. (2014). Vanilla counterfactual regret minimization for engineers. https://justinsermeno.com/posts/cfr/. [Accessed on 29.10.2021].

Tammelin, O. (2014). Solving large imperfect information games using cfr+. *arXiv preprint arXiv:1407.5042*.

Tran, M. (1996). Deep blue computer beats world chess champion. *the Guardian*.

von Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. (2007). Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20:1729–1736.