CFR for Solving Imperfect-Information Games

Abstract. This essay introduces the counter-factual regret minimization algorithm (CFR), whose variants have been widely used to solve large imperfect-information games such as Poker. The necessary theoretical background is first introduced, and then an intuitive understanding of the operation of the algorithm is presented through an example with application to Kuhn Poker. The essay concludes with a review and comparison of the two notable Poker-playing agents Tartanian-7 and Cepheus.

1 Introduction

Solving large sequential games like Poker is a challenge not only because of the large game tree, but also due to their imperfect-information nature.The most commonly used approach for solving large imperfect-information games involves two stages [1]: first by generating an abstraction of the game , then using an equilibrium-finding algorithm to solve the abstract game, and finally apply post processing techniques to map the generated solution for the abstracted game to the actual game. The abstraction can be done by hand, but algorithms have been developed to do this automatically [2]. The aim of abstraction is to reduce the state-space of the game into a tractable size that can be solved. Typically, an abstraction algorithm would group actions that are strategically similar together. Consider the first betting round of Texas Hold'em where each player is dealt 2 cards. There are 1,326 distinct 2-card hands, but this distinction can be redundant since there are only 169 strategically distinct cards [10]. So an abstraction algorithm could, for example, not distinguish between the hands $AsAc$ and $AhAd$.

The most commonly used equilibrium-finding algorithm in the strongest Poker agents is CFR [2] and its variants, which will be the primary focus of this discussion. CFR uses the concept of regret of past actions in order to improve its playing strategy through iterative self-play. The playing strategy is updated to minimize overall regret as the game is repeatedly played. Previous Poker agents used Linear Programming (LP) to solve simpler poker games like Rhode Island Hold'em, but it cannot handle the more complex poker games [1, 11]. The next section will provide the necessary background for introducing CFR, then the algorithm is explored in depth in section 4.

2 Background

This section provides the necessary background before introducing CFR. This includes the concepts of regret, Nash-Equilibrium, relevant notation, and the concept of information sets.

2.1 Regret minimization and Nash-equilibrium

The regret-matching algorithm first introduced by [3] will be at the core of this discussion. Regret in the context of games quantifies how much the player's pay-off on a given round would have been better had he taken a different action on that round. Actions with greater expected pay-offs are valued higher (regretted more). The algorithm updates the players' strategies such that future actions are more likely to be chosen in proportion to these calculated positive past regrets.

In order to formalize the notion of regret and further the discussion, consider a modified 2-player Rock-Paper-Scissors game (RPS) [8]. Let σ_i denote the strategy followed by player i whenever it is his turn to play. A strategy is a probability distribution over the set of available actions, and $\sigma_i(a)$ is the probability of choosing action a. For example, σ=$(0.2, 0.3, 0.5)$ corresponding to the set of actions $A = \{R, P, S\}$, and $\sigma(R) = 0.2$. Let $u_1(a_1, a_2)$ denote the pay-off for player 1 on some given round of interest, when players 1 and 2 take actions a_1 and a_2 respectively. Whenever a reference to ourselves is made in the context of a game, it is a reference to player 1.

On a round of the game, each player bets a dollar, then both players simultaneously take an action $a \in \{R, P, S\}$ according to their respective strategies. The winning player receives the two dollars at the end of the round; in the case of a draw, a dollar is received by each player. Let's say that we begin with the uniform strategy $\sigma_1 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. We are unlucky on the first round with our choice of $a = P$ and the resulting utility of $u_1(P, S) = 0$. Then we can use the regret-matching algorithm to update our strategy. We begin by calculating our regrets of not having taken a different action, that is the regrets of not having taken actions R and S. The regret of not having played $a = R$ on this round is $u_1(R, S) - u_1(P, S) = 2 - 0 = 2$, and the regret for not having played S is $u_1(S, S) - u_1(P, S) = 1$. Let $\boldsymbol{r}$ be our vector of cumulative regrets: $\boldsymbol{r} = (2, 0, 1)$. For the second round, we update our strategy using the regret-matching algorithm as follows:

$$\sigma_1 \leftarrow \boldsymbol{r}/sum(r) = (2, 0, 1)/3 = (\frac{2}{3}, 0, \frac{1}{3})$$

Here, '$\leftarrow$' denotes assignment, '/' is the element-wise division operator, and $sum(\boldsymbol{r})$ is simply the element-wise summation function. Our new strategy is the cumulative regrets from a previous round, normalized by the total sum of the regrets. On the second round, we are a little more lucky with our choice of R with probability $\frac{2}{3}$, leading to a draw and utility $u_1(R, R) = 1$. In this round our regrets are $\boldsymbol{r} = (0, 1, 0)$, and we have cumulative regrets of $(2, 1, 1)$, which can be normalized to obtain the new strategy for the next round $\sigma_1 = (\frac{2}{5}, \frac{1}{5}, \frac{1}{5})$.

A solution for a game can be defined to be a strategy profile for all players, such that no player can expect further improvement in performance by changing strategy alone, given that the other players don't change theirs. This is a simple statement of the solution concept known as a Nash-equilibrium.

2.2 Games with Imperfect Information

Sequential games such as chess can be represented as a game-tree, with each node having an associated player and an edge denoting a transition upon taking an action. Imperfect-information games, where some components of the game states (opponent cards in the case of Poker) may not be available for all players at all times. Poker is an example of an imperfect-information game, which can be represented using a game-tree with the additional component of information sets. A more simplified version of the relevant formal notation is presented first, with subsequent sections providing examples and intuition. The notation used is based on the game-theoretic model of extensive games, for which [9, p. 200] serves as a complete reference on the details of the model and additional notation.

2.3 Notation for Extensive Games

H is a finite set of histories, where each $h \in H$ is a sequence of actions, e.g. $a_1a_2...a_6$. $Z \subseteq H$ is the set of all terminal histories. $A(h)$ denotes the set of actions available after history h has been reached, and similary $P(h)$ is the player after history h has been reached; in the case of two-player games $P(h) \in \{c, 1, 2\}$ where c is the chance "player".

In order to formalize the notion of information sets before providing the intution in the next subsection, let's denote the information partition $\mathcal{I}_i$ for player i, to be the set of all histories $h \in H : P(h) = i$. Each member $I \in \mathcal{I}_i$ in turn is a subset of histories, such that given any two histories $h, h' \in I$, $A(h) = A(h')$. i.e., the set of available actions for two different histories within the same information set are identical. If this was not the case, then the player could simply distinguish between the two histories within the same information set based on the available actions at that history, and therefore the two histories are not contained within the same information set.

σ_i is defined for player i as previously to be a function that assigns a probability distribution over $A(I_i)$ for each $I_i \in \mathcal{I}_i$. The subscript i for variable names is used to identify the player, and $-i$ refers to all players except player i. In the case of a two-player game, if σ_i is player 1's strategy, then $\sigma_{-i} = \sigma_2$ is player 2's strategy.

$\pi^\sigma(h)$ denotes the probability of reaching history h if all players play according to the strategy profile σ, i.e, the product of action probabilities along h of all players (including chance) along this history. $\pi_i^\sigma(h)$ is player i's contribution to this probability.

2.4 Information Sets

Intuitively, an information set is a set of possibly different histories, such that the player is unable to distinguish between them; as a consequence of this, the player will use the same strategy on all histories within the information set where it is his turn to play. A player can however distinguish between individual information sets from the limited information they do receive. As an example,

consider a modified game of chess where we are an optimal chess playing agent, but we are playing against an unfair opponent who hides her pieces and only reveals them after we made our move without knowing what her move was. In the beginning of the game, the opponent makes a hidden move, so our information set on our turn is the finite set of all possible initial opponent moves (1st pawn moved, 2nd pawn moved,... etc.). Since we don't know which move has been made, we will be using a strategy tailored for all possible histories in our current information set (in practice, often a uniform strategy $\frac{1}{\#actions}$ is used on the first round of a game). After we play our turn, the opponent's pieces are revealed. At this stage we will likely have regrets for having made the wrong move given that now we do know what the opponent's move was. In principle, we train ourselves to succeed in this game by repeatedly playing it, calculating our regrets for each information set, and updating our strategy such that we are more likely to make the moves we regretted the most on previous plays, and hence minimizing regrets in the future. The idea of minimizing our regrets for individual information sets is they key idea in CFR.

3 Counterfactual Regret Minimization

The concept of *counter-factual regret* for sequential games with imperfect information is introduced in [14]. The key idea is to decomposed overall regret into individual additive regret terms which can be minimized individually for each information set. In this section, the main algorithm is introduced with a focus on developing an intuitive understanding of the algorithm. For proofs, theoretical background and formal definitions, the reader is referred to [14].

The detailed algorithm (stated similarly as in [7]) is shown in **Algorithm 1**, modified for clarity. "Chance sampling" simply means that, if a node is a chance node, we sample an outcome on this node, i.e lines 10 to 11. The algorithm is called recursively for all players individually on each given iteration. Below is a summary of the key points of the algorithm, with an example run presented in the next subsection.

First, the cumulative regrets table r_I over each information set for each action is initialized. This table holds the cumulative *counter-factual* regret values defined over each individual information set and each action in the information set. These counter-factual regret values are computed on line 25. The cumulative strategies table μ_I is used to compute the *average cumulative strategies*. This average strategy (not the final strategy σ) is what converges to a Nash Equilibrium [8], and this will in turn indicate that an optimal regret-minimizing strategy for each player has been found [14, Theorem 1]. The third initialization statement simply initializes all strategies to uniform strategies. σ^t denotes the strategy profile at iteration t. $\sigma_{I \to a}$ is a strategy identical to σ, with the exception that the action a is always taken on information set I, and $v_{\sigma_{I \to a}}$ is the value of such a strategy. On a given iteration, the algorithm is called for each player, starting with player 1. The algorithm traces the game tree by taking each possible action at each node and computes expected utilities. The player

Algorithm 1 Counterfactual Regret Minimization with chance sampling

// Initializations:
$\forall I, r_I[a] \leftarrow 0$
$\forall I, \mu_I[a] \leftarrow 0$
$\forall I, a, \sigma^1(I,a) \leftarrow \frac{1}{|A(I)|}$

function CFR(h, i, t, π_1, π_2):
    if h is terminal **then**
        return $u_i(h)$
    else if h is a chance node **then**
        $s \leftarrow$ rnd_sample($\sigma_c(h)$) // sample a chance outcome
        return CFR(hs, i, t, π_1, π_2)

    //Let I be the local information set, i.e. I contains h
    $v_\sigma \leftarrow 0$
    $\forall a \in A(I), v_{\sigma_I \to a}[a] \leftarrow 0$
    for $s \in A(I)$ **do**
        if $P(h) = 1$ **then**
            $v_{\sigma_I \to a}[s] \leftarrow \text{CFR}(hs, i, t, \sigma^t(I,s) \cdot \pi_1, \pi_2)$
        else if $P(h) = 2$ **then**
            $v_{\sigma_I \to a}[s] \leftarrow \text{CFR}(hs, i, t, \pi_1, \sigma^t(I,s) \cdot \pi_2)$
        $v_\sigma \leftarrow v_\sigma + \sigma^t(I,s) \cdot v_{\sigma_I \to a}[s]$

    if $P(h) = i$ **then**
        for $s \in A(I)$ **do**
            $r_I[s] \leftarrow r_I[s] + \pi_{-i} \cdot (v_{\sigma_I \to a}[s] - v_\sigma)$
            $\mu_I[s] \leftarrow \mu_I[s] + \pi_i \cdot \sigma^t(I,s)$

        r_sum $\leftarrow \sum_{a \in A(I)} r_I[a]$
        //Update player i's strategy using regret matching
        for $s \in A(I)$ **do**
            if r_sum > 0 **then**
                $\sigma^{t+1}(I,s) \leftarrow \frac{r_I[s]}{r\_sum}$
            else
                $\sigma^{t+1}(I,s) \leftarrow \frac{1}{|A(I)|}$

    return v_σ

function SOLVE():
    for $t \in \{1, 2, 3..., T\}$ **do**
        for $i \{1, 2\}$ **do**
            CFR($\emptyset, i, t, 1, 1$)

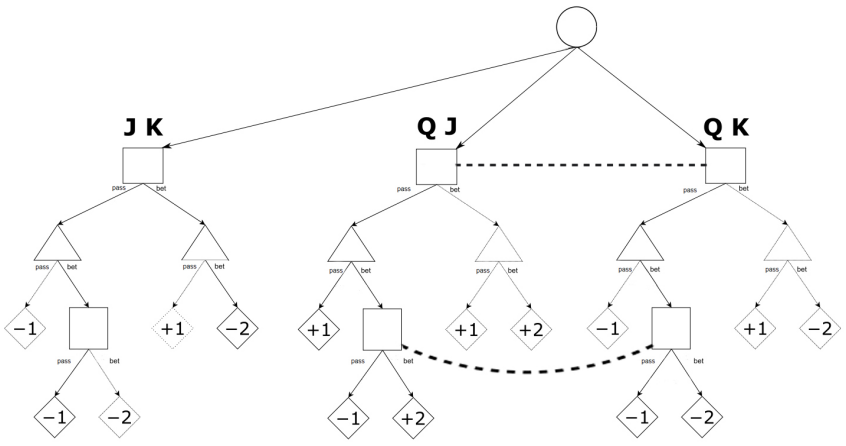

Fig. 1. If the outcome of the root chance node is $[QJ]$, then the three trees show all information sets for all players. The figure marks the information sets of player 1 with dashed lines.

for whom the function was called for on the iteration (the parameter i in the function call) will have their strategies, cumulative regrets table, and cumulative strategy tables updated for each information set.

3.1 Example: Kuhn Poker

Kuhn Poker is a simple 3-card poker game with 2 players [4]. At the beginning of the game, each player bets 1 chip into the pot. The 3 cards, which have strictly-increasing values (say J, Q, and K) are shuffled and each player is dealt a card which is held as private information. Play alternates starting with player 1. This section will illustrate the operation of CFR by running one iteration of the algorithm on this game for player 1. For the sake of this example, a few changes will be made to the game: first, for this iteration of interest, the assumption is made that the chance node will deal the cards Q and J to players 1 and 2 respectively. Second, the actions of the game will be abstracted into two categories in a manner similar to [8]: the action "pass" will corresponding to *check* and *fold*; "bet" will correspond to *bet* and *call*. The game tree will be as in Figure 1, middle branch. As in previous sections, a reference to ourselves is a reference to player 1. The terminal nodes in the figure show our payoffs, from which player 2's payoff may be directly inferred as $-u_1$.

The three game branches in Figure 1 are three out of 6 possible outcome of the root chance node (shuffling and dealing the cards) in the game of Kuhn Poker. The other branches are omitted for brevity and because they are not

relevant to this example. These 3 branches are shown to illustrate the concept of information sets. The information sets of player 1 are marked by dashed lines. The information sets of player two are not marked, but they can be inferred similarly from the definitions. The actual game tree concerning this example is the middle QJ branch. On our play turn, we receive the card Q, so our information set contains the histories $[QJ]$ and $[QK]$. Since Kuhn Poker is a 3-card game, we know that our opponent could be holding one of the cards J or K. The information partitions for this game for both players are listed below this paragraph, with the restriction that the chance outcome is QJ. History descriptions between square brackets denote a single history, e.g., "[QK]b" is a concatenation of the history [QK], an outcome of a chance node where the cards Q and K are dealt to players 1 and 2 respectively, and the action 'b', where player 1 chooses to bet. $P(h)$ in this case can be inferred algorithmically from the length of the history (odd length corresponding to player 1's turn; even length corresponds to player 2's).

$$\mathcal{I}_1 = \{\{[QJ], [QK]\}, \{[QJ]pb, [QK]pb\}\}$$

$$\mathcal{I}_2 = \{\{[QJp], [KJp]\}, \{[QJ]b, [KJ]b\}\}$$

Figure 2 shows the results of running one iteration of the algorithm for player 1, i.e. CFR($\emptyset, 1, 1, 1.0, 1.0$) with the assumption that the chance node outcome will be $[QJ]$. Rectangular boxes contain the parameters $\boxed{h, \pi_1, \pi_2}$ as the algorithm runs on the node the box is adjacent to in the figure. I is always a local variable, denoting the local information set as inferred from the history parameter h. π_i is the probability that player i will reach the local history h if all players play according to the current strategy profile.

4 Modern Champion Poker-playing Agents

Several variations of CFR have emerged since its introduction in 2008. The most notable of these are MCCFR[5] and CFR$^+$ [12,13]. These variants of CFR, combined with various methods of optimization, abstraction, and post-processing techniques are the main points of difference between different Poker-playing agents. Tartanian-7 from the Carnegie Mellon University uses MCCFR as its equilibrium-finding algorithm [2]. MCCFR differs in that it samples opponent actions along the tree, in addition to sampling chance nodes. Tartanian-7 uses an abstraction algorithm that is tailored for a distributed system with separate memory blades. Since accessing memory from another memory blade is slower than accessing local memory, the abstraction algorithm aims to minimize such operations by clustering histories into separate buckets at early stages during the game, with each bucket corresponding to a separate memory blade. Tartanian-7 participated in the 2014 Annual Computer Poker Competition (ACPC), winning against all opponents. The game played was no-limit Texas Hold'em, which has 100^{165} nodes in its game tree [6]. Training Tartanian-7 for ACPC 2014 took 1,200

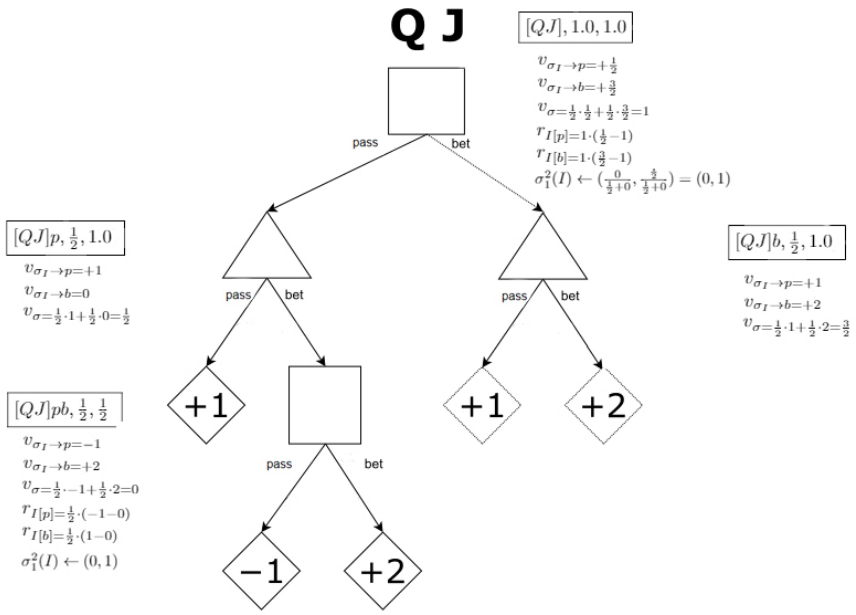

Fig. 2. Results of calling CFR$(\emptyset, 1, 1, 1.0, 1.0)$ assuming the chance node outcome is $[QJ]$

hours on a supercomputer. A thorough overview of Tartanian-7 can be found in [2].

Cepheus from the University of Alberta is another notable Poker-playing agent, and it uses a variant of CFR called CFR$^+$ [12]. CFR$^+$ differs from the original CFR algorithm in several respects. CFR$^+$ calculates its average strategies using weighted averages, as opposed to the uniform averages calculated by CFR. In addition to this, it does not use sampling as in MCCFR and CFR. The original CFR algorithm updates the regrets table for both players simultaneously, while CFR$^+$ alternates this operation for both players. Most importantly, CFR$^+$ uses an alternative regret-matching algorithm called regret-matching$^+$, with one of the key differences being in resetting negative accumulated regrets back to zero whenever a negative regret is encountered for that action, while CFR simply ignores negative regret values. Readers interested in playing against Cepheus can do so at: http://poker-play.srv.ualberta.ca/

References

1. Bowling, Burch, Johanson, Tammelin.: Heads-up limit hold'em poker is solved. Science Magazine. 2015

2. Brown, Ganzfried, Sandholm. Hierarchical Abstraction, Distributed Equilibrium Computation, and Post-Processing, with Application to a Champion No-Limit Texas Hold'em Agent. AAMAS 2015. 2015
3. Hart, Mas-Colell. A Simple Adaptive Procedure Leading to Correlated Equilibrium. 2000.
4. Kuhn. Simplified two-person poker. 1950.
5. Lanctot, Waugh, Zinkevich, and Bowling. Monte Carlo sampling for regret minimization in extensive games. 2009
6. M. Johanson. Measuring the size of large no-limit poker games. Technical report, University of Alberta. 2013.
7. Neller, Hnath. Approximating Optimal Dudo Play with Fixed-Strategy Iteration Counterfactual Regret Minimization. 2011.
8. Neller, Lanctot. Intro. to Counterfactual Regret Minimization. 2013.
9. Osborne, Rubinstein. A Course in Game Theory.
10. Sandholm, Gilpin. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. 2006
11. Sandholm. The State of Solving Large Incomplete-Information Games, and Application to Poker. AI Magazine. 2010.
12. Tammelin Solving Large Imperfect Information Games Using CFR+. 2014.
13. Tammelin,Burch, Johanson, Bowling. Solving Heads-up Limit Texas Hold'em. 2015.
14. Zinkevich, Johanson, Bowling, Piccione. Regret Minimization in Games with Incomplete Information. 2008.