DYNAMIC DISCOUNTED COUNTERFACTUAL REGRET MINIMIZATION

Hang Xu[1,2,*] **Kai Li**[1,2,*,†] **Haobo Fu**[6] **Qiang Fu**[6] **Junliang Xing**[5,†] **Jian Cheng**[1,3,4]
[1]Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]School of Future Technology, University of Chinese Academy of Sciences
[4]AiRiA [5]Tsinghua University [6]Tencent AI Lab
{xuhang2020,kai.li,jian.cheng}@ia.ac.cn,
{haobofu,leonfu}@tencent.com, jlxing@tsinghua.edu.cn

ABSTRACT

Counterfactual regret minimization (CFR) is a family of iterative algorithms showing promising results in solving imperfect-information games. Recent novel CFR variants (*e.g.*, CFR+, DCFR) have significantly improved the convergence rate of the vanilla CFR. The key to these CFR variants' performance is weighting each iteration non-uniformly, *i.e.*, discounting earlier iterations. However, these algorithms use a *fixed, manually-specified* scheme to weight each iteration, which enormously limits their potential. In this work, we propose Dynamic Discounted CFR (DDCFR), the first equilibrium-finding framework that discounts prior iterations using a *dynamic, automatically-learned* scheme. We formalize CFR's iteration process as a carefully designed Markov decision process and transform the discounting scheme learning problem into a policy optimization problem within it. The learned discounting scheme dynamically weights each iteration on the fly using information available at runtime. Extensive experimental results demonstrate that DDCFR's dynamic discounting scheme has a strong generalization ability and leads to faster convergence with improved performance. The code is available at https://github.com/rpSebastian/DDCFR.

1 INTRODUCTION

Imperfect-information games (IIGs) model strategic interactions between players with hidden information. Solving such games is challenging since it requires reasoning under uncertainty about the opponents' private information. The hidden information is omnipresent in real-world decision-making problems, making the research on IIGs theoretically and practically crucial.

In this work, we focus on solving two-player zero-sum IIGs. For these games, the typical goal is to find an (approximate) Nash equilibrium (Nash, 1950) — a strategy profile in which no player can improve their utilities by unilaterally switching to a different strategy. One of the most successful approaches to computing Nash equilibrium in IIGs is the family of counterfactual regret minimization (CFR) algorithms (Zinkevich et al., 2007). CFR iteratively minimizes both players' regrets so that the time-averaged strategy profile gradually approaches the Nash equilibrium. Due to the sound theoretical guarantee and strong empirical performance, CFR-type algorithms are the basis of several breakthroughs (Bowling et al., 2015; Moravčík et al., 2017; Brown & Sandholm, 2018; 2019a).

Over the past decade, researchers have proposed various novel CFR variants (Tammelin, 2014; Brown & Sandholm, 2019b; Xu et al., 2022) with faster convergence rates. In particular, the development of CFR+ (Tammelin, 2014) was a milestone at least an order of magnitude faster than the vanilla CFR in practice, which was the key to solving the heads-up limit Texas Hold'em poker. The recently proposed Discounted CFR (DCFR) (Brown & Sandholm, 2019b) is one of the most efficient equilibrium-finding algorithms, outperforming other CFR variants in a diverse set of IIGs.

[*]Equal contribution. † Corresponding authors.

Unlike the vanilla CFR, which assigns equal weights to every iteration, the key to the performance of these new CFR variants is that they use different discounting schemes to weight each iteration non-uniformly. For example, CFR+ uses a linear discounting scheme in which iteration t's contribution to the average strategy is proportional to t. DCFR is a family of algorithms that discounts prior iterations when determining both regrets and average strategy, *i.e.*, assigning smaller weights to earlier iterations using three hyperparameters (α, β, and γ). Although these CFR variants have shown that faster convergence is achievable by discounting, they all rely on a fixed and manually-specified discounting scheme, which significantly limits their potential. Specifically, since there are a limitless number of discounting schemes in theory, it is impractical and even impossible to manually find one that performs well in most games, which leads to the manually-specified scheme being suboptimal. Additionally, the fixed discounting scheme is pre-determined before the start of the iteration process, which is excessively restrictive. It is desirable to design a more flexible scheme that can dynamically weight each iteration on the fly using the information available at runtime.

To this end, we propose a novel Dynamic Discounted CFR (DDCFR) framework that weights each iteration using a dynamic, automatically-learned discounting scheme. We formulate CFR's iteration process as a Markov decision process (MDP) and transform the discounting scheme learning problem into a policy optimization problem within it. Specifically, we regard the dynamic discounting scheme as an *agent* interacting with the MDP. The agent receives the current state of the iteration process and outputs an action consisting of the discounting weights for the next iteration. The agent's goal is to find an approximate Nash equilibrium promptly by choosing appropriate weights.

Our ultimate goal is to learn a discounting policy that generalizes across many different IIGs, not limited to some particular games. To this end, we carefully design the training games, the MDP, and the training algorithm. Specifically, we train the discounting policy on a diverse set of IIGs to improve its generalization ability. We design a game-agnostic state space for the MDP, so the trained policy naturally applies to different games. We further exploit a highly scalable training algorithm to optimize the discounting policy based on evolution strategies (Salimans et al., 2017). We evaluate the learned dynamic discounting scheme on new games not seen during training, including large-scale games, such as heads-up no-limit Texas hold'em (HUNL) subgames. The results demonstrate that it has a strong generalization ability and outperforms the fixed, manually-specified discounting scheme on both training and testing games.

This paper makes three novel contributions: (i) We propose DDCFR, the first equilibrium-finding framework that discounts prior iterations using a dynamic, automatically-learned scheme. (ii) We formulate CFR's iteration process as an MDP and transform the discounting scheme learning problem into a policy optimization problem in this MDP. (iii) We design an effective training algorithm to optimize the discounting policy. The learned discounting scheme exhibits strong generalization ability, achieving competitive performance on new games not seen during training.

2 PRELIMINARY

2.1 NOTATIONS

Extensive-form games (Osborne & Rubinstein, 1994) are a tree-based formalism commonly used to describe IIGs. In these games, there is a finite set $\mathcal{N}$ of **players** and a unique player c called **chance** with a fixed, known stochastic strategy. **History** h consists of all actions taken by players, including private knowledge known to only a subset of players. All possible histories in the game tree form the set $\mathcal{H}$. Players take turn to take actions and, $\mathcal{A}(h)=\{a : ha \in \mathcal{H}\}$ denotes the **actions** available at a given history h. $\mathcal{Z} \subseteq \mathcal{H}$ are **terminal histories** for which no actions are available. Each player $i \in \mathcal{N}$ has a **utility function** $u_i(z):\mathcal{Z}\rightarrow [u_{\text{min}}, u_{\text{max}}] \subset \mathbb{R}$, and $\Delta=u_{\text{max}}-u_{\text{min}}$ is the **range of utilities**.

In IIGs, the lack of information is represented by **information sets** $\mathcal{I}_i$ for each player $i \in \mathcal{N}$. If h, h' are in the same information set $I \in \mathcal{I}_i$, player i cannot distinguish between them. For example, in poker, all histories within an information set differ only in the private cards of other players. Thus, $\mathcal{A}(I) = \mathcal{A}(h)$ for arbitrary $h \in I$. We define $|\mathcal{I}| = \sum_{i\in\mathcal{N}} |\mathcal{I}_i|$ and $|\mathcal{A}| = \max_{i\in\mathcal{N}} \max_{I\in\mathcal{I}_i} |\mathcal{A}(I)|$.

A **strategy** $\sigma_i(I)$ assigns a probability distribution over the actions available for acting player i in an information set I. Specifically, $\sigma_i(I, a)$ denotes the probability of player i taking action a. Since all histories in an information set belonging to player i are indistinguishable for that player, the strategies in each must be identical. Therefore, for any $h_1, h_2 \in I$, we have $\sigma_i(I) = \sigma_i(h_1) = \sigma_i(h_2)$. A **strategy profile** $\sigma = \{\sigma_i | \sigma_i \in \Sigma_i, i \in \mathcal{N}\}$ is a specification of strategies for all players, where Σ_i denotes the set of all possible strategies for player i, and σ_{-i} refers to the strategies of all

players other than player i. $u_i(\sigma_i, \sigma_{-i})$ represents the **expected utility** for player i if player i plays according to σ_i and the others play according to σ_{-i}.

2.2 Best Response and Nash Equilibrium

The **best response** to σ_{-i} is any strategy $\text{BR}(\sigma_{-i})$ that maximizes player i's expected utility, such that $u_i(\text{BR}(\sigma_{-i}), \sigma_{-i}) = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i})$. The **Nash equilibrium** is a strategy profile $\sigma^* = (\sigma^*_i, \sigma^*_{-i})$ in which every player plays a best response to the other players' strategies. Formally, $\forall i \in \mathcal{N}, u_i(\sigma^*_i, \sigma^*_{-i}) = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma^*_{-i})$. The **exploitability** of strategy σ_i is a measure of how far that strategy is from optimal. It is defined as $e_i(\sigma_i) = u_i(\sigma^*_i, \sigma^*_{-i}) - u_i(\sigma_i, \text{BR}(\sigma_i))$. In an **$\epsilon$-Nash equilibrium**, no player has exploitability higher than ϵ. The exploitability of a strategy profile σ is $e(\sigma) = \sum_{i \in \mathcal{N}} e_i(\sigma_i)/|\mathcal{N}|$. We can interpret it as the approximation error to the Nash equilibrium.

2.3 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) is one of the most popular equilibrium-finding algorithms for solving extensive-form IIGs (Zinkevich et al., 2007). The algorithm iteratively refines the players' average strategies by minimizing regrets. CFR frequently uses **counterfactual value** $v_i^\sigma(I)$, which is the expected utility of a information set $I \in \mathcal{I}_i$ for a player i under a given strategy profile σ, assuming that the player tries to reach it. The counterfactual value of a specific action a in I is $v_i^\sigma(I, a)$. The rigorous definitions of $v_i^\sigma(I)$ and $v_i^\sigma(I, a)$ can be found in Appendix A.

CFR typically proceeds with a uniform random strategy σ^1. On each iteration t, CFR first recursively traverses the game tree using the current strategy σ^t to calculate the **instantaneous regret** $r_i^t(I, a)$ of not choosing action a in an information set I for player i, which is defined as:

$$r_i^t(I, a) = v_i^{\sigma^t}(I, a) - v_i^{\sigma^t}(I). \tag{1}$$

CFR then calculates the **cumulative regret** $R_i^t(I, a) = \sum_{k=1}^t r_i^k(I, a)$ and computes a strategy for the next iteration using regret-matching (Hart & Mas-Colell, 2000; Cesa-Bianchi & Lugosi, 2006):

$$\sigma_i^{t+1}(I, a) = \begin{cases} \frac{R_i^{t,+}(I,a)}{\sum_{a' \in \mathcal{A}(I)} R_i^{t,+}(I,a')}, & \text{if } \sum_{a' \in \mathcal{A}(I)} R_i^{t,+}(I, a') > 0 \\ \frac{1}{|\mathcal{A}(I)|}, & \text{otherwise,} \end{cases} \tag{2}$$

where $R_i^{t,+}(I, a) = \max(R_i^t(I, a), 0)$. In two-player zero-sum IIGs, if both players use CFR on each iteration, their **average strategies** $\bar{\sigma}^t$ will converge to an ϵ-Nash equilibrium in $\mathcal{O}\left(|\mathcal{I}|^2|\mathcal{A}|\Delta^2/\epsilon^2\right)$ iterations (Zinkevich et al., 2007). The average strategy $\bar{\sigma}^t$ is calculated as:

$$C_i^t(I, a) = \sum_{k=1}^t \left(\pi_i^{\sigma^k}(I)\sigma_i^k(I, a)\right), \ \bar{\sigma}_i^t(I, a) = \frac{C_i^t(I, a)}{\sum_{a' \in \mathcal{A}(I)} C_i^t(I, a')},$$

where $\pi_i^{\sigma^k}(I)$ is information set I's reach probability (Appendix A) and $C_i^t(I, a)$ is player i's **cumulative strategy** for action a in information set I on iteration t.

2.4 The Discounting Variants of CFR

Since the birth of CFR, many new CFR variants have been proposed and greatly improved the convergence rate of the vanilla CFR. CFR+ (Tammelin, 2014) is like CFR with three small but effective modifications and converges an order of magnitude faster than CFR. First, CFR+ uses a linear discounting scheme that weights iteration t by t rather than using a uniformly-weighted average strategy as in CFR. Second, to immediately reuse an action when it shows promise of performing well instead of waiting for the cumulative regret to become positive, CFR+ sets any action with negative cumulative regret to zero on each iteration. Finally, CFR+ uses the alternating-updates technique.

The recently proposed DCFR (Brown & Sandholm, 2019b) algorithm investigates the discounting schemes in regret minimization algorithms more systematically. DCFR is a family of algorithms that discounts cumulative regrets and the cumulative strategy, dramatically accelerating the convergence rate. Specifically, on each iteration t, DCFR multiplies positive cumulative regrets by $\frac{(t-1)^\alpha}{(t-1)^\alpha+1}$, negative cumulative regrets by $\frac{(t-1)^\beta}{(t-1)^\beta+1}$, and the cumulative strategy by $\left(\frac{t-1}{t}\right)^\gamma$. Formally,

$$R_i^t(I, a) = \begin{cases} R_i^{t-1}(I, a)\frac{(t-1)^\alpha}{(t-1)^\alpha+1} + r_i^t(I, a), & \text{if } R_i^{t-1}(I, a) > 0 \\ R_i^{t-1}(I, a)\frac{(t-1)^\beta}{(t-1)^\beta+1} + r_i^t(I, a), & \text{otherwise,} \end{cases} \tag{3}$$

$$C_i^t(I,a) = C_i^{t-1}(I,a)\left(\frac{t-1}{t}\right)^{\gamma} + \pi_i^{\sigma^t}(I)\sigma_i^t(I,a). \tag{4}$$

The three hyperparameters (α, β, γ) determine DCFR's discounting scheme. For example, we can recover the linear discounting scheme by setting $\alpha = 1$, $\beta = 1$, and $\gamma = 1$. Although there are a limitless number of discounting schemes that converge in theory, DCFR's authors empirically set $\alpha = 1.5$, $\beta = 0$, and $\gamma = 2$ since they found this setting performs the best in the games they tested.

3 Learning Dynamic Discounting Scheme

3.1 The DDCFR Framework

As mentioned earlier, the existing discounting CFR variants have obtained remarkable performance in solving IIGs. However, they all exploit a fixed and manually-specified discounting scheme. These pre-determined schemes are not flexible enough, thus inevitably limiting the convergence performance. We argue that an ideal discounting schema should fulfill two criteria. First, the scheme should be automatically learned rather than manually designed. For example, in DCFR, designing the discounting scheme by manually setting the values of α, β, and γ is unscalable and suboptimal since finding one setting that can efficiently solve a wide variety of IIGs is challenging. Second, the scheme should be able to adjust the weights dynamically instead of using fixed weights. For regret minimization algorithms, the discounting weights often need to change dynamically during learning. This is particularly important in solving different types of IIGs, where the distribution of regret values and the cost of taking suboptimal actions vary significantly during the iteration process.

To achieve this goal, we propose a novel Dynamic Discounted CFR (DDCFR) framework that weights each iteration using a dynamic, automatically-learned discounting scheme. DDCFR learns to adjust the discounting weights dynamically using information available at runtime, such as iterations and exploitability. By adapting to changing conditions in real time, we believe that DDCFR can lead to more efficient and effective learning.

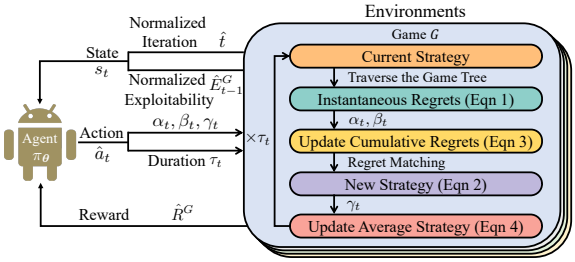

Figure 1: The DDCFR framework.

The high-level idea of DDCFR is to encapsulate CFR's iteration process into an *environment* and regard the discounting scheme as an *agent* interacting with it. Specifically, the agent receives the current state (*i.e.*, the current status of CFR's iteration process) and outputs an action consisting of the discounting weights for the next iteration. This process continues until the maximum number of iterations. The agent's goal is to find an optimal discounting policy that can dynamically choose suitable weights for each iteration and thus minimize the exploitability of the obtained average strategies. Formally, the interaction process between the agent and the environment in game G constitutes a Markov decision process (MDP), represented as $(G, S, A, P^G, \hat{R}^G)$.

The game G. Each game G represents a different environment. Our ultimate goal is to train an agent whose discounting policy performs well on the training games and generalizes to new games.

The state space S. The state $s_t \in S$ contains information that the agent observes at each iteration t in game G. s_t should contain as much general and transferable information as possible, helping the agent make good decisions on choosing discounting weights for each iteration and making the learned scheme applicable to different games. We propose a game-agnostic state representation that meets these requirements, *i.e.*, s_t consisting of two parts: 1) the normalized iteration $\hat{t}$, which is the ratio of the current iteration t to the total number of iterations; 2) the normalized exploitability $\hat{E}^G_{t-1}$, which is calculated as $(\log E^G_{t-1} - \log E_{\text{min}})/(\log E^G_1 - \log E_{\text{min}})$, where E^G_1 and E^G_{t-1} are the exploitability of the average strategies in game G at iteration 1 and $t-1$, respectively, and E_{min} is the minimum reachable exploitability set to 10^{-12}. The logarithmic operation results in a more even distribution of normalized exploitability, thereby facilitating agent training (Appendix I).

The action space A. The agent's action $\hat{a}_t \in A$ at iteration t consists of four numbers, *i.e.*, $\hat{a}_t = [\alpha_t, \beta_t, \gamma_t, \tau_t]$. Similar to DCFR in Equations 3 and 4, $\alpha_t, \beta_t, \gamma_t$ are used to determine the

discounting weights, *i.e.*, $\frac{(t-1)^{\alpha_t}}{(t-1)^{\alpha_t}+1}$, $\frac{(t-1)^{\beta_t}}{(t-1)^{\beta_t}+1}$, and $\left(\frac{t-1}{t}\right)^{\gamma_t}$ for positive cumulative regrets, negative cumulative regrets, and the cumulative strategy, respectively. The key difference between our DDCFR and DCFR is that DCFR uses a pre-determined discounting scheme, *i.e.*, DCFR empirically sets α=1.5, β=0, and γ=2, which is fixed not only during the iteration process but also in different games. In contrast, our proposed DDCFR can utilize $\hat{a}_t$ to dynamically adjust the discount weights at runtime and adaptively control its iteration process for each game. τ_t represents the duration for how long to use these discounting weights. Incorporating τ_t enhances the interpretability of the learned scheme, indicating whether the hyperparameters need to be adjusted frequently.

The state transition P^G. $P^G : S \times A \to S$ describes how the state changes during the iteration process of DDCFR in game G. Specifically, the agent observes the current state s_t at each iteration t and outputs an action $\hat{a}_t$. DDCFR uses the discounting weights calculated by α_t, β_t, and γ_t for τ_t iterations, and finally, the state transitions to the next state $s_{t+\tau_t}$. This process allows for the consecutive use of each set of discounting weights for a certain number of iterations.

The reward function $\hat{R}^G$. $\hat{R}^G$ evaluates the agent's performance and guides DDCFR's iteration process. Since the agent's ultimate goal is to find a strategy profile with the lowest exploitability, we use the sparse reward setting where the agent only receives a reward at the end of the iteration process. Specifically, the reward is $\hat{R}^G = \log E_1^G - \log E_T^G$, where E_T^G is the exploitability of the average strategies in game G at the maximum number of iterations T.

The agent is typically represented by a neural network with parameters $\boldsymbol{\theta}$, *i.e.*, $\pi_{\boldsymbol{\theta}} : S \to A$. Figure 1 shows DDCFR's overall iteration process. Given a game G, at iteration t, the agent receives the current state s_t and outputs an action $\hat{a}_t = (\alpha_t, \beta_t, \gamma_t, \tau_t) = \pi_{\boldsymbol{\theta}}(s_t)$. DDCFR then uses the discounting weights calculated by α_t, β_t, and γ_t to update cumulative regrets and the average strategy for τ_t iterations. This process continues until the maximum number of iterations, and the agent eventually receives a reward $\hat{R}^G$. In each game G, the objective is to maximize the final reward $\hat{R}^G$, represented as $f^G(\boldsymbol{\theta}) = \hat{R}^G$. DDCFR's *overall objective* is to maximize the average sum of the rewards across the training games $\mathbb{G}$, *i.e.*, $f(\boldsymbol{\theta}) = \frac{1}{|\mathbb{G}|}\sum_{G\in\mathbb{G}} f^G(\boldsymbol{\theta})$. By optimizing $f(\boldsymbol{\theta})$, our ultimate goal is to learn a generalizable discounting policy that applies to new games.

3.2 Theoretical Analysis

Although DDCFR's dynamic discounting scheme is highly flexible, a question arises regarding its convergence properties. Here we theoretically demonstrate that DDCFR is guaranteed to converge to a Nash equilibrium as long as $\alpha_t, \beta_t, \gamma_t$ are within a certain range.

Theorem 1. *Assume that conduct DDCFR T iterations in a two-player zero-sum game. If DDCFR selects hyperparameters as follows: $\alpha_t \in [0,5]$ for $t < \frac{T}{2}$ and $\alpha_t \in [1,5]$ for $t \geq \frac{T}{2}$, $\beta_t \in [-5,0]$, $\gamma_t \in [0,5]$, the weighted average strategy profile is a $6|\mathcal{I}|\Delta\left(\frac{8}{3}\sqrt{|\mathcal{A}|} + \frac{2}{\sqrt{T}}\right)/\sqrt{T}$-Nash equilibrium.*

The proof is in the Appendix B, mainly rescaling inequalities based on the range of $\alpha_t, \beta_t, \gamma_t$ following DCFR's proof. This theorem signifies that numerous dynamic discounting schemes converge in theory. We then describe how to efficiently optimize $\pi_{\boldsymbol{\theta}}$ to find a well-performing scheme in practice.

3.3 Optimization through Evolution Strategies

When training the discounting policy $\pi_{\boldsymbol{\theta}}$, we encounter the challenges of sparse reward (the agent receives a reward at the end of the iteration), long time horizons (CFR requires thousands of iterations), and multi-task learning (the agent maximizes rewards across various games). Off-the-shelf reinforcement learning (RL) algorithms like DQN (Mnih et al., 2015) and PPO (Schulman et al., 2017) struggle to cope with these challenges, requiring the integration of more sophisticated components. Additionally, RL is notably sensitive to hyperparameters. Evolution Strategies (ES) (Wierstra et al., 2014; Salimans et al., 2017) has demonstrated its efficacy as a scalable alternative to RL in tackling these challenges. As a black box optimization technique, ES is indifferent to the distribution of rewards and tolerant of arbitrarily long time horizons. Besides, ES is easy to implement and is highly scalable and efficient to use on distributed hardware.

Inspired by the effectiveness of ES in optimizing complex objective functions and addressing RL-related problems, we design an ES-based algorithm to train the discounting policy $\pi_{\boldsymbol{\theta}}$. The ES approach proposed in (Salimans et al., 2017) states that for an objective function $f(\boldsymbol{\theta})$ over a network

Algorithm 1: DDCFR's training procedure.

Input: Trainig games $\mathbb{G}$, epochs M, learning rate lr, population size N, standard deviation δ

Randomly initialize the agent's parameters $\boldsymbol{\theta}^1$;
for $m = 1$ ***to*** M **do**
  Initialize population $\mathbb{P}$ with an empty list;
  for $i = 1, \ldots, \frac{N}{2}$ **do**
    Sample $\boldsymbol{\epsilon_i} \sim \mathcal{N}(0, I)$;
    Add $\boldsymbol{\epsilon_i}, -\boldsymbol{\epsilon_i}$ to the population $\mathbb{P}$;
  foreach $\boldsymbol{\epsilon} \in \mathbb{P}$ **do**
    foreach $G \in \mathbb{G}$ **do**
      Compute $f^G(\boldsymbol{\theta}^m + \delta\boldsymbol{\epsilon})$, ▷(Algorithm 2);
    $f(\boldsymbol{\theta}^m + \delta\boldsymbol{\epsilon}) \leftarrow \frac{1}{|\mathbb{G}|} \sum_{G \in \mathbb{G}} f^G(\boldsymbol{\theta}^m + \delta\boldsymbol{\epsilon})$
  Compute $f'(\boldsymbol{\theta}^m + \delta\boldsymbol{\epsilon})$ using Eqn. 5;
  $\boldsymbol{\theta}^{m+1} \leftarrow \boldsymbol{\theta}^m + \frac{lr}{\delta \cdot N} \sum_{\epsilon \in \mathbb{P}} f'(\boldsymbol{\theta}^m + \delta\boldsymbol{\epsilon})\boldsymbol{\epsilon}$

Output: The trained agent $\pi_{\boldsymbol{\theta}^{M+1}}$

Algorithm 2: The calculation process of $f^G(\boldsymbol{\theta})$.

Input: Training game G, agent $\pi_{\boldsymbol{\theta}}$, total itertations T

$t \leftarrow 1$;
while $t \leq T$ **do**
  $[\alpha_t, \beta_t, \gamma_t, \tau_t] = \hat{a}_t = \pi_{\boldsymbol{\theta}}(s_t)$;
  Use $[\alpha_t, \beta_t, \gamma_t]$ to calculate discounting weights and perform τ_t numbers of CFR iterations;
  $t \leftarrow t + \tau_t$;
Compute exploitability E_1^G, E_T^G;

Output: $f^G(\boldsymbol{\theta}) \leftarrow \log E_1^G - \log E_T^G$

parameterized by $\boldsymbol{\theta}$, its gradient estimation can be obtained by applying Gaussian perturbations to $\boldsymbol{\theta}$, *i.e.*, $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0,I)} f(\boldsymbol{\theta} + \delta\boldsymbol{\epsilon}) = \frac{1}{\delta} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0,I)} f(\boldsymbol{\theta} + \delta\boldsymbol{\epsilon})\boldsymbol{\epsilon}$, where δ denotes the noise standard deviation, and $\mathcal{N}(0, I)$ is a Gaussian distribution with mean 0 and covariance I. This gradient estimation can be approximated with samples. Specifically, we generate a population of perturbed network parameters at each epoch and evaluate the resulting parameters under all training games. We then combine the results to calculate the gradient and update the original parameter using stochastic gradient ascent.

To improve the efficiency of the training process, we employ three acceleration techniques: antithetic estimator, fitness shaping, and parallel evaluation. Algorithm 1 outlines the complete training procedure of the discounting policy. (i) We use the antithetic estimator (Nesterov & Spokoiny, 2017) to reduce variance. It is an unbiased estimator using control variates (Liu et al., 2019), given by the symmetric difference, *i.e.*, $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0,I)} f(\boldsymbol{\theta} + \delta\boldsymbol{\epsilon}) = \frac{1}{2\delta} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0,I)} \left[f(\boldsymbol{\theta} + \delta\boldsymbol{\epsilon}) - f(\boldsymbol{\theta} - \delta\boldsymbol{\epsilon}) \right] \boldsymbol{\epsilon}$. (ii) We apply fitness shaping (Wierstra et al., 2014) by performing a rank transformation on the objective function $f(\boldsymbol{\theta})$ before computing the gradient. For parameter $\boldsymbol{\theta_k}$ with the kth largest value in the population of size N, the shaped result is as follows:

$$f'(\boldsymbol{\theta_k}) = \frac{\max\left(0, \log\left(\frac{N}{2} + 1\right) - \log(k)\right)}{\sum_{j=1}^{P} \max\left(0, \log\left(\frac{N}{2} + 1\right) - \log(j)\right)} - \frac{1}{N}. \tag{5}$$

This approach makes the gradient only relevant to the relative ordering of parameters' fitness values rather than their absolute magnitudes. By doing so, our algorithm becomes more robust to variations in the original fitness values. (iii) To further speed up training, we evaluate the performance of each perturbed parameter under each game in parallel.

3.4 Additional Computation Cost Analysis

Compared with DCFR, DDCFR introduces additional computation costs, which mainly includes three parts: feature calculations, network inference, and policy training. However, these additional

time costs are very small compared to the overall time costs. (i) Feature calculations. The state representation requires the iteration number and the exploitability. The iteration number is effortlessly obtainable. Since the exploitability calculation and the instantaneous regret calculation are completely independent, multiprocessing can be used to concurrently execute both processes without increasing the wall-clock time (Appendix C). (ii) Network inference. The time overhead of π is negligible compared to the overall iteration time. To support this claim, we compare the wall-clock time between DCFR and DDCFR in Appendix D. DDCFR exhibits only a 0.22% increase in time on average. (iii) Policy training. The effort put into training is worthwhile, as the trained discounting policy can be directly applied to numerous new games without modification. This means that the additional work during training is amortized across the various games to which the policy is applied.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Training and testing games: We use several commonly used IIGs in the research community. Here, we provide brief descriptions of these games; please refer to the Appendix J for more details. ***Kuhn Poker*** (Kuhn, 1950) is a simplified form of poker with three cards in a deck and one chance to bet for each player. ***Leduc Poker*** (Southey et al., 2005) is a larger game with a 6-card deck and two rounds. ***Big Leduc Poker*** uses a deck of twenty-four cards with twelve ranks, allowing a maximum of six raises per round. In ***Liar's Dice-x*** (Lisý et al., 2015), each player has an x-sided dice, which they roll at the start. Players then take turns placing bets on the outcome. ***Goofspiel-x*** (Ross, 1971) is a card game where each player has x cards and tries to score the most points by making sealed bids in x rounds. In ***Battleship-x*** (Farina et al., 2019), players secretly place a 1×2 ship on separate $2 \times x$ grids and then take turns firing at the opponent's ship three times. ***HUNL Subgame-x*** is a HUNL subgame generated by the top poker agent Libratus.

We select eight relatively large IIGs as testing games, *i.e.*, *Battleship-2*, *Battleship-3*, *Goofspiel-4*, *Liar's Dice-4*, *Leduc Poker*, *Big Leduc Poker*, *HUNL Subgame-3*, and *HUNL Subgame-4*. These games are diverse in size and challenging to solve, making them suitable for testing the learned discounting scheme's generalization performance.

The training games should be relatively small to speed up the training process while still exhibiting the typical characteristics of the testing games. We select four training games: *Kuhn Poker*, *Goofspiel-3*, *Liar's Dice-3*, and *Small Matrix*. The first three games are simplified versions of the testing games, while *Small Matrix* simulats common situations in games where irrational actions lead to large losses. Our training games have varying rules and scales, exhibiting a strong diversity, crucial for improving generalization ability.

Training details: We set a fixed noise standard deviation of $\delta{=}0.5$ and a population size of $N{=}100$. We distribute the evaluation of perturbed parameters across 200 CPU cores. For the action space, we set the range of α and γ to $[0, 5]$, β to $[-5, 0]$ following Theorem 1, and choose τ in $[1, 2, 5, 10, 20]$. We employ a network consisting of three fully-connected layers with 64 units and ELU activation functions to represent the discounting policy $\pi_{\boldsymbol{\theta}}$. The network takes the current state of the environment as input and outputs values for α, β, γ, and the probability of selecting each τ. We use Adam with a learning rate lr of 0.01 to optimize the network and trained the agent for $M = 1000$ epochs, which took roughly 24 hours. We set the number of CFR iterations T to 1000, typically sufficient for convergence to a small exploitability. All CFR variants used the alternating-updates technique.

4.2 COMPARISON TO DISCOUNTING CFR VARIANTS

In Figures 2 and 3, we compare DDCFR's learned discounting scheme with two discounting CFR variants, *i.e.*, CFR+ and DCFR. Since the iteration process of DDCFR is deterministic (Appendix F), we only test it once in each game. Due to the rapid decline in exploitability in the early iterations of different CFR algorithms, their differences are primarily manifested in the later iterations. For a better comparison of the algorithms, we focus on their performance in the later iterations, *i.e.*, from 500 to 1000. Please refer to Appendix E for all iterations. As shown in Figure 2, DDCFR converges faster to a lower exploitability on the training games, which is expected since we are optimizing DDCFR's performance on these games. However, even in unseen testing games shown in Figure 3, DDCFR still achieves competitive performance against the other CFR variants due to the fact that DDCFR is designed with generalization in mind. Specifically, it yields an average reduction in exploitability

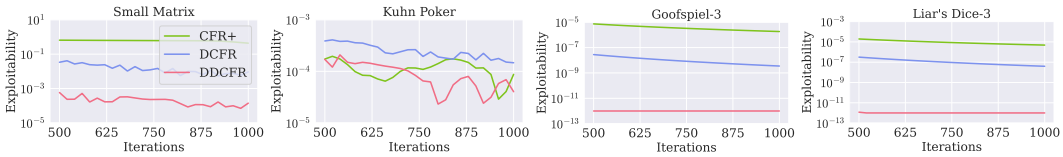

Figure 2: Comparison of DDCFR against two discounting CFR variants on four training games.

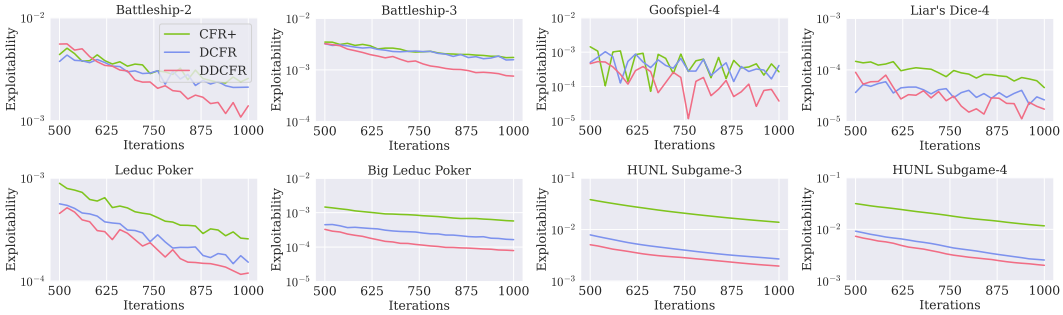

Figure 3: Comparison of DDCFR against two discounting CFR variants on eight testing games.

of 42% compared to DCFR, which is remarkable given DCFR's already strong performance. Unlike CFR+ and DCFR with fixed and manually-specified discounting schemes, our DDCFR achieves better performance on all games thanks to the learned dynamic discounting scheme's ability to adjust the discounting weights on the fly using information available at runtime.

Greedy Weights (Zhang et al., 2022) is a recently proposed algorithm specifically designed for normal-form games by greedily weighing iterates based on regrets observed at runtime. Although it can be adapted for extensive-form games as well, we find that its performance is notably inferior to that of DDCFR which is designed explicitly for extensive-form games (Appendix G). We suspect that when faced with a large number of information sets, the computed weights might not be appropriate, resulting in poor performance in extensive-form games.

To further understand the behaviors of the learned dynamic discounting scheme $\pi_{\boldsymbol{\theta}}$, we visualize its actions (*i.e.*, $\alpha_t, \beta_t, \gamma_t, \tau_t$) during the iteration process. Figure 4 shows that the actions outputted by $\pi_{\boldsymbol{\theta}}$ have different values in different games but exhibit a similar trend. Specifically, α_t increases with iterations in all games while β_t and γ_t decrease. Compared with DCFR's fixed discounting scheme (*i.e.*, α=1.5, γ=2), the learned scheme is more aggressive in earlier iterations. Smaller α and bigger γ lead to smaller weights $\frac{(t-1)^\alpha}{(t-1)^\alpha+1}$ and $\left(\frac{t-1}{t}\right)^\gamma$ when discounting positive cumulative regrets and the cumulative strategy. This may be because the optimal strategy based on the opponent's strategy frequently changes in earlier iterations; thus, earlier iterations need to be given smaller weights. As the iteration progresses, the scheme becomes more moderate, which may aid in the convergence of the average strategy. Additionally, the scheme consistently uses more aggressive strategies when discounting negative cumulative regrets. This is beneficial as it allows for more rapid reuse of promising actions instead of waiting for cumulative regrets to become positive. The change in τ_t suggests that discounting weights should be adjusted more frequently in earlier iterations.

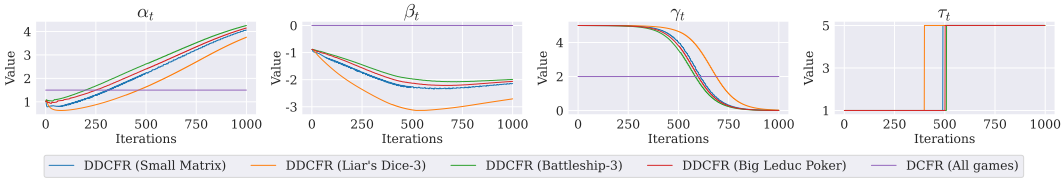

Figure 4: The learned discounting scheme behaves differently in various games yet exhibits a similar trend. Compared with DCFR's fixed discounting scheme (we can view DCFR as a special case of DDCFR, where $\alpha_1 = 1.5, \beta_1 = 0, \gamma_1 = 2$, and the duration $\tau_1 = \infty$), it is more aggressive in the earlier iterations and becomes more moderate as the iteration progresses.

4.3 Ablation Studies

To verify the effectiveness of DDCFR's components, we conduct extensive ablation studies and report the results in Table 1. The results of each column are obtained by replacing one component of DDCFR, and the rest remains unchanged. We use the exploitability on three testing games to compare the learned discounting schemes' performance.

The number of training games. We first investigate how the number of training games affects the learned discounting scheme. DDCFR-x means using the first x training games from *Kuhn Poker*, *Goofspiel-3*, *Liar's Dice-3*, and *Small Matrix*. DDCFR-1 performs relatively poorly since the learned discounting scheme is prone to overfitting using only one training game. DDCFR-2 and DDCFR-3 obtain better results because they need to balance their performance under multiple games and improve the generalization ability of the learned discounting schemes. With the addition of *Small Matrix*, DDCFR achieves a much lower exploitability in *HUNL Subgame-3*, probably because both games exhibit the same property of having high-loss actions.

The state representation. We consider how the state representation affects DDCFR's final performance. DDCFR-it and DDCFR-ex only use the normalized iteration and the normalized exploitability as the state representation, respectively. As shown in Table 1, DDCFR-it achieves better performance than DDCFR-ex, indicating iterations provide the agent with more information for decision-making. However, both algorithms perform worse than DDCFR, showing that these two state features are complementary and that using them together can obtain the best performance.

The action space design. DDCFR adopts a continuous action space design, *i.e.*, α and γ are continuous values between 0 and 5, β is a continuous value between -5 and 0. In contrast, DDCFR-dis uses a discrete action space design, *i.e.*, it discretizes the range of α, β, γ to consecutive integers. The results in Table 1 show that continuous actions help the learned dynamic discounting scheme perform more fine-grained control and obtain better performance.

The optimization algorithm. We compare DDCFR's optimizing algorithm ES with RL. DDCFR-rl uses PPO to optimize the discounting policy, underperforming in all testing games. We attribute this outcome to several factors commonly encountered during training, such as sparse rewards, long time horizons, and muli-task learning.

Table 1: Ablation analyses of DDCFR.

	DDCFR-1	DDCFR-2	DDCFR-3	DDCFR-it	DDCFR-ex	DDCFR-dis	DDCFR-rl	DDCFR
LeducPoker	1.566e-4	1.490e-4	1.213e-4	1.249e-4	1.289e-4	1.156e-4	1.229e-4	**1.198e-4**
Subgame-3	2.533e-3	2.481e-3	2.393e-3	2.292e-3	5.320e-3	5.795e-3	4.987e-3	**1.963e-3**
Subgame-4	2.517e-3	2.175e-3	2.122e-3	2.714e-3	3.804e-3	5.803e-3	5.198e-3	**2.005e-3**

4.4 Combination with Other CFR Variants

Our proposed dynamic discounted regret minimization is a general framework that can be universally applied in any CFR variants that employ fixed and manually specified discounting rules. The DDCFR is just an example application of this framework to DCFR. The predictive CFR+ (PCFR+) (Farina et al., 2021) is a recently proposed state-of-the-art CFR variant based on predictive Blackwell approachability. Nonetheless, it also uses a fixed quadratic discounting scheme. To showcase the generalizability of our framework, we apply it to PCFR+, resulting in the Dynamic Predictive CFR+ (DPCFR+). We find that DPCFR+ can further unlock the potentials of PCFR+ and enhance its performance (Appendix H). These results demonstrate the general applicability of our dynamic discounting framework. It can be regarded as a *performance booster* that can be applied in a plug-and-play manner to various CFR variants that employ fixed discounting rules.

5 Conclusion

We present DDCFR, the first equilibrium-finding framework that discounts prior iterations using an automatically-learned dynamic scheme. We first formulate CFR's iteration process as a carefully designed MDP and transform the discounting scheme learning problem into a policy optimization problem. We then design a scalable ES-based algorithm to optimize the discounting policy efficiently. The learned discounting policy exhibits strong generalization ability, achieving competitive performance on both training games and new testing games.

6 ACKNOWLEDGEMENTS

This work is supported in part by the National Science and Technology Major Project (2022ZD0116401); the Natural Science Foundation of China under Grant 62076238, Grant 62222606, and Grant 61902402; the Jiangsu Key Research and Development Plan (No. BE2023016); and the China Computer Federation (CCF)-Tencent Open Fund.

REFERENCES

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305, 2012.

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

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456): 885–890, 2019a.

Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *AAAI Conference on Artificial Intelligence*, pp. 1829–1836, 2019b.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Gabriele Farina, Chun Kai Ling, Fei Fang, and Tuomas Sandholm. Correlation in extensive-form games: Saddle-point formulation and benchmarks. In *Advances in Neural Information Processing Systems*, pp. 9229–9239, 2019.

Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Faster game solving via predictive blackwell approachability: Connecting regret matching and mirror descent. In *AAAI Conference on Artificial Intelligence*, pp. 5363–5371, 2021.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

Viliam Lisỳ, Marc Lanctot, and Michael H Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 27–36, 2015.

Hao Liu, Richard Socher, and Caiming Xiong. Taming maml: Efficient unbiased meta-reinforcement learning. In *International Conference on Machine Learning*, pp. 4061–4071, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

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

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

Sheldon M Ross. Goofspiel—the game of pure strategy. *Journal of Applied Probability*, 8(3): 621–625, 1971.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Conference on Uncertainty in Artificial Intelligence*, pp. 550–558, 2005.

Oskari Tammelin. Solving large imperfect information games using CFR+. *arXiv preprint arXiv:1407.5042*, 2014.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

Hang Xu, Kai Li, Haobo Fu, Qiang Fu, and Junliang Xing. Autocfr: Learning to design counterfactual regret minimization algorithms. In *AAAI Conference on Artificial Intelligence*, pp. 5244–5251, 2022.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2396–2407, 2018.

Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

Hugh Zhang, Adam Lerer, and Noam Brown. Equilibrium finding in normal-form games via greedy regret minimization. In *AAAI Conference on Artificial Intelligence*, pp. 9484–9492, 2022.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, pp. 1729–1736, 2007.

A Rigorous Definitions of Reach Probabilities and Counterfactual Values

The **history reach probability** of a history h is the joint probability of reaching that history if all players play according to σ, denoted by $\pi^\sigma(h) = \prod_{h'a \sqsubseteq h} \sigma_{\mathcal{P}(h')}(h', a)$ where the relationship $g \sqsubseteq h$ indicates that g is equal to or a prefix of h, and $\mathcal{P}(h)$ is the unique player who takes action at that history h. It can be decomposed into each player's contribution, *i.e.*, $\pi^\sigma(h) = \pi_i^\sigma(h)\pi_{-i}^\sigma(h)$, where $\pi_i^\sigma(h)$ refers to player i's contribution, and $\pi_{-i}^\sigma(h)$ denotes all players' contributions except player i. We define the **information set reach probability** as $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$, and the **interval history reach probability** from history h' to h as $\pi^\sigma(h', h) = \pi^\sigma(h)/\pi^\sigma(h')$ if $h' \sqsubseteq h$. $\pi_i^\sigma(I), \pi_{-i}^\sigma(I), \pi_i^\sigma(h, h'), \pi_{-i}^\sigma(h, h')$ are defined similarly.

For player i at an information set $I \in \mathcal{I}_i$ under a given strategy profile σ, the counterfactual value of I is $v_i^\sigma(I) = \sum_{h \in I}(\pi_{-i}^\sigma(h) \sum_{z \in \mathcal{Z}}(\pi^\sigma(h, z)u_i(z))$. The counterfactual value of a specific action a in I is $v_i^\sigma(I, a) = \sum_{h \in I}(\pi_{-i}^\sigma(h) \sum_{z \in \mathcal{Z}}(\pi^\sigma(ha, z)u_i(z))$.

B Proof of Theorem 1

Proof. Since the lowest amount of instantaneous regret on any iteration is $-\Delta$ and DDCFR multiplies negative regrets by $\frac{(t-1)^{\beta_t}}{(t-1)^{\beta_t}+1} \leq \frac{(t-1)^0}{(t-1)^0+1} = \frac{1}{2}$ on iteration t, the regret for any action at any point is greater than -2Δ.

Consider the weighted sequence of iterates $\sigma'^1, \ldots, \sigma'^T$, where σ'^t is identical to σ^t, but weighted by $w_{a,t} = \Pi_{k=t+1}^T \left(\frac{k-1}{k}\right)^{\gamma_k}$ rather than $w_t = \Pi_{k=t+1}^T \frac{(k-1)^{\alpha_k}}{(k-1)^{\alpha_k}+1}$. The regret of action a in information set I on iteration t of this new sequence is $R'^t(I, a)$.

From Lemma 5 we know that $R^T(I, a) \leq \frac{8}{3}\Delta\sqrt{|\mathcal{A}|}\sqrt{T}$ for player i for action a in information set I. Since $w_{a,t}$ is an increasing sequence, we can apply Lemma 1 using weight $w_{a,t}$ for iteration t with $B = \frac{8}{3}\Delta\sqrt{|\mathcal{A}|}\sqrt{T}$ and $C = -2\Delta$. This yields $R'^T(I, a) \leq \left(\frac{8}{3}\Delta\sqrt{|\mathcal{A}|}\sqrt{T} + 2\Delta\right)$. As the weights sum is $\sum_{t=1}^T w_{a,t} \geq \sum_{t=1}^T \Pi_{k=t+1}^T \frac{(k-1)^5}{k^5} = \sum_{t=1}^T \frac{t^5}{T^5} = \frac{T^2(T+1)^2(2T^2+2T-1)}{12T^5} \geq \frac{T}{6}$, the weighted average regret $R_i'^{w,T} = \max_{a \in \mathcal{A}} \frac{R'^T(I,a)}{\sum_{t=1}^T w_{a,t}} \leq 6\Delta(\frac{8}{3}\sqrt{|\mathcal{A}|} + \frac{2}{\sqrt{T}})/\sqrt{T}$. Applying Theorem 3 in CFR (Zinkevich et al., 2007), we get overall weighted average regret for player i is at most $6\Delta|\mathcal{I}_i|\left(\frac{8}{3}\sqrt{|\mathcal{A}|} + \frac{2}{\sqrt{T}}\right)/\sqrt{T}$. Since $|\mathcal{I}_1| + |\mathcal{I}_2| = |\mathcal{I}|$, the weighted average strategies form a $6\Delta|\mathcal{I}|\left(\frac{8}{3}\sqrt{|\mathcal{A}|} + \frac{2}{\sqrt{T}}\right)/\sqrt{T}$-Nash equilibrium after T iterations. □

Lemma 1. *(Brown & Sandholm, 2019b, Lemma 1) Call a sequence $x_1, \ldots, x_T$ of bounded real values BC-plausible if $B > 0$, $C \leq 0$, $\sum_{k=1}^t x_k \geq C$ for all $t \leq T$, and $\sum_{t=1}^T x_t \leq B$. For any BC-plausible sequence and any sequence of non-decreasing weights $w_t \geq 0$, $\sum_{t=1}^T (w_t x_t) \leq w_T(B - C)$.*

Lemma 2. *Given a set of actions $\mathcal{A}$, any sequence of rewards v^t, and any sequence of weights w^t such that $\sum_{t=1}^\infty w_t = \infty$ and $|v^t(a) - v^t(b)| \leq \Delta$ for all t and all $a, b \in \mathcal{A}$, after playing a sequence of strategies determined by regret matching but every reward v^t is weighted by the weight w^t, the weighted average regret $R_i^{w,T} = \max_{a \in \mathcal{A}} \frac{\sum_{t=1}^T \left(w_t r^t(a)\right)}{\sum_{t=1}^T w^t}$ is bounded by $R_i^{w,T} \leq \frac{\Delta\sqrt{|\mathcal{A}|}\sqrt{\sum_{t=1}^T w_t^2}}{\sum_{t=1}^T w_t}$.*

Proof. The proof is identical to that of Theorem 2.8 in (Cesa-Bianchi & Lugosi, 2006), with $p = 2$ for the polynomially weighted average forecaster. □

Lemma 3. *(Brown & Sandholm, 2019b, Lemma 2) Given a sequence of strategies $\sigma^1, \ldots, \sigma^T$, each defining a probability distribution over a set of actions $\mathcal{A}$, consider any definition for $Q^t(a)$ satisfying the following conditions:*

1. $Q^0(a) = 0$
2. $Q^t(a) = Q^{t-1}(a) + r^t(a)$ *if* $Q^{t-1}(a) + r^t(a) > 0$

3. $0 \geq Q^t(a) \geq Q^{t-1}(a) + r^t(a)$ *if* $Q^{t-1}(a) + r^t(a) \geq 0$.

The regret-like value $Q^t(a)$ *is then an upper bound on the regret* $R^t(a)$ *and* $Q^t(a) - Q^{t-1}(a) \geq r^t(a) = R^t(a) - R^{t-1}(a)$.

Lemma 4. *(Brown & Sandholm, 2019b, Lemma 3) Given a set of actions* $\mathcal{A}$ *and any sequence of rewards* v^t *such that* $|v^t(a) - v^t(b)| \leq \Delta$ *for all* t *and all* $a, b \in \mathcal{A}$*, after playing a sequence of strategies determined by regret matching but using the regret-like value* $Q^t(a)$ *in place of* $R^t(a)$*,* $Q^T(a) \leq \Delta\sqrt{|\mathcal{A}|T}$ *for all* $a \in \mathcal{A}$.

Lemma 5. *Assume that player* i *conducts* T *iterations of DDCFR. Then the weighted regret for the player is at most* $\Delta\,|\mathcal{I}_i|\,\sqrt{|\mathcal{A}|}\sqrt{T}$ *and the weighted average regret for the player is at most* $\frac{8}{3}\Delta\,|\mathcal{I}_i|\,\sqrt{|\mathcal{A}|}/\sqrt{T}$.

Proof. The weight of iteration $t < T$ is $w_t = \Pi_{k=t+1}^{T} \frac{(k-1)^{\alpha_k}}{(k-1)^{\alpha_k}+1}$ and $w_T = 1$. Thus, $w_t \leq \Pi_{k=t+1}^{T} \frac{(k-1)^5}{(k-1)^5+1} \leq 1$ for all t and therefore $\sum_{t=1}^{T} w_t^2 \leq T$.

Additionally, $w_t \geq \Pi_{k=t+1}^{T} \frac{(k-1)^1}{(k-1)^1+1} = \frac{t}{T}$ for $\frac{T}{2} \leq t < T$ and $w_T = 1$. Thus, $\sum_{t=1}^{T} w_t \geq \sum_{t=\frac{T}{2}}^{T} w_t \geq \sum_{t=\frac{T}{2}}^{T} \frac{t}{T} = \frac{1}{2} * \frac{3}{2}T\left(\frac{T}{2}+1\right) * \frac{1}{T} > \frac{3}{8}T$.

Applying Lemma 2 and 4, we see that $Q_i^{w,T}(I,a) \leq \frac{\Delta\sqrt{|\mathcal{A}|}\sqrt{\sum_{t=1}^{T} w_t^2}}{\sum_{t=1}^{T} w_t} \leq \frac{8\Delta\sqrt{|\mathcal{A}|}\sqrt{T}}{3T}$. Applying Theorem 3 in CFR (Zinkevich et al., 2007), we see that $Q_i^{w,T} \leq \frac{8\Delta|\mathcal{I}_i|\sqrt{|\mathcal{A}|}\sqrt{T}}{3T}$. Since $R_i^{w,T} \leq Q_i^{w,T}$, so $R_i^{w,T} \leq \frac{8\Delta|\mathcal{I}_i|\sqrt{|\mathcal{A}|}\sqrt{T}}{3T}$. □

C Feature Calculations in DDCFR

The features in DDCFR contain the normalized iteration $\hat{t}$ and the normalized exploitability $\hat{E}_{t-1}^G$. The iteration number is effortlessly obtainable. The normalized exploitability calculation can be done with the instantaneous regret calculation in parallel. We provide a detailed explanation below.

On each iteration t, our DDCFR first recursively traverses the game tree using the strategy σ^t to compute the instantaneous regret r^t. Then, DDCFR updates the cumulative regrets R^t, the average strategy $\bar{\sigma}^t$, and the new strategy σ^{t+1} using the weights $(\alpha_t, \beta_t, \gamma_t)$ generated by the discounting policy $\pi_{\boldsymbol{\theta}}(s_t)$. Specifically, $\alpha_t, \beta_t, \gamma_t = \pi_{\boldsymbol{\theta}}(s_t) = \pi_{\boldsymbol{\theta}}([\hat{t}, \hat{E}_{t-1}^G])$, where $\hat{t}$ is the normalization of iteration t and $\hat{E}_{t-1}^G$ is the normalized exploitability of the average strategy $\bar{\sigma}^{t-1}$ at iteration $t-1$.

Calculating $\hat{E}_{t-1}^G$ requires traversing the game tree using the average strategy $\bar{\sigma}^{t-1}$, which leads to additional time overhead under naive implementation. However, it's crucial to note that we can calculate $\hat{E}_{t-1}^G$ and compute the instantaneous regret r^t in parallel since these two processes depend on two different strategies, $\bar{\sigma}^{t-1}$ and σ^t, and each requires traversing the game tree once. Consequently, there is no additional wall-clock time overhead associated with the calculation of exploitability in our parallel implementation.

D Running Time Comparison between DCFR and DDCFR

To further support claim in Section 3.4, we compare the running time between DCFR and DDCFR. Since the running time is influenced by the available hardware resources of the system, we execute the algorithms on various games one bye one, but distributed across multiple servers. In addition, to mitgate the time overhead associated with writing to disk, we remove the logging module and only print the final running time. Table 2 presents the running time of the two algorithms on eight testing games. For example, completing 1000 iterations of DCFR on Battleship-2 takes roughly 14 minutes, 54 seconds, and 558 milliseconds. In comparison to DCFR, DDCFR exhibits only a 0.22% increase in time on average. Therefore, the additional computation time incurred by DDCFR is negligible compared to the overall iteration time.

Table 2: Running time comparsion between DCFR and DDCFR.

Game	DCFR_time	DDCFR_time	(DDCFR_time - DCFR_time) / DCFR_time
Battleship-2	0:14:54.558	0:14:55.814	0.140%
Battleship-3	11:10:46.895	11:12:05.156	0.194%
Goofspiel-4	00:01:03.259	00:01:03.518	0.409%
Liar's Dice-4	0:06:57.896	0:06:58.577	0.163%
Leduc Poker	0:08:13.140	0:08:13.921	0.158%
Big Leduc Poker	14:55:48.347	14:58:07.357	0.259%
HUNL Subgame-3	0:02:54.634	0:02:55.024	0.223%
HUNL Subgame-4	0:02:14.379	0:02:14.648	0.200%

E ALL ITERATIONS OF DDCFR AND DISCOUNTING CFR VARIANTS

Due to the rapid decline in exploitability in the early iterations of different CFR algorithms, their differences are primarily manifested in the later iterations. Consequently, for a better comparison of the algorithms, we focus on the later iterations in the main paper. For the sake of completeness, we show the all iterations of DDCFR and discounting CFR variants in Figures 5 and 6.

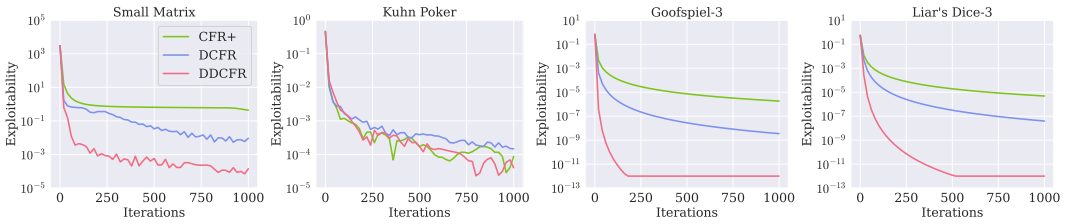

Figure 5: All iterations of DDCFR and discounting CFR variants on four training games.

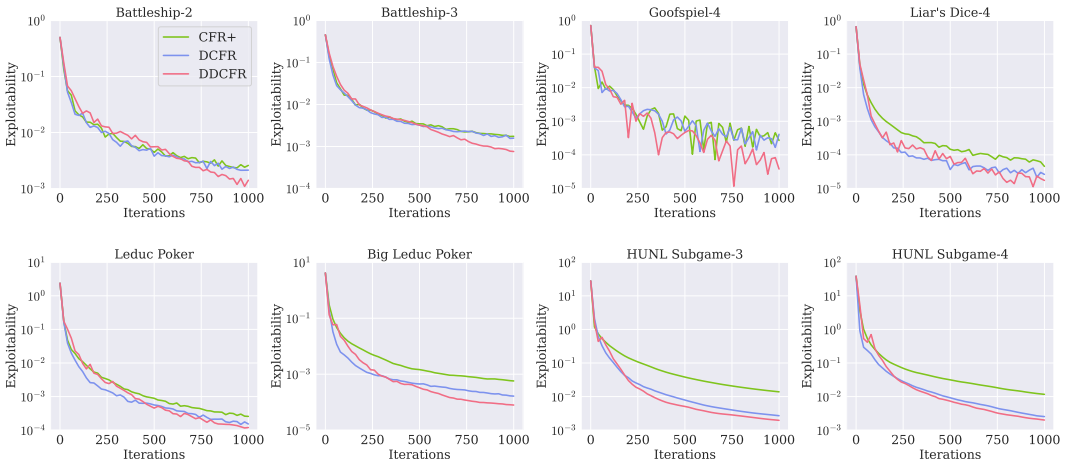

Figure 6: All iterations of DDCFR and discounting CFR variants on eight testing games.

F DETERMINISTIC NATURE OF DDCFR'S ITERATION PROCESS

We use ES to learn the discounting policy and select one of the best performing policy on the training games for testing. It is worth noting that there is no randomness throughout the iteration process of DDCFR when using the learned discounting policy on the testing games. Specifically, DDCFR always initializes with the uniform random average strategy, and the policy network consistently

produces the same fixed action when inputs are determined. Put simply, each run of DDCFR with the learned model consistently generates the same exploitability curve.

G Comparison with Greedy Weights

We conduct experimental evaluations comparing DDCFR and GreedyWeights (Zhang et al., 2022) on extensive-form games. We integrate GreedyWeights with CFR by extending upon the officially released code for random norm-form games. We run a grid search within the range of 0.1, 0.5, 1.0, 2.0, 5.0 to tune the hyperparameter "weight floor" for each game. The experimental results shown in Figure 7 demonstrate that DDCFR consistently converges faster to a lower exploitability on four extensive-form games. While GreedyWeights is designed to efficiently solve normal-form games and shows promising results on extensive-form games, our DDCFR designed explicitly for extensive-form games outperforms GreedyWeights significantly. One key distinction between normal-form games and extensive-form games lies in the quantity of information sets. We suspect that when faced with a large number of information sets, the computed weights in GreedyWeights might not be appropriate, resulting in poor performance in extensive-form games.

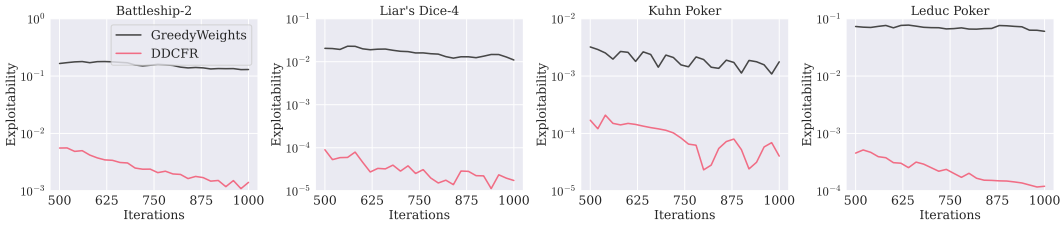

Figure 7: Performance comparison between DDCFR and GreedyWeights on extensive-form games.

H Comparison and Combination with Predictive CFR+

The predictive CFR+ (PCFR+) (Farina et al., 2021) is a recently proposed state-of-the-art CFR variant based on predictive Blackwell approachability. However, it is not a consistently well-performing algorithm. The authors of PCFR+ have already found in their paper that PCFR+ is faster than DCFR on non-poker games, whereas on poker games DCFR is the fastest. We present a comparison between our DDCFR and PCFR in Figure 8. The experimental results align with expectations, i.e., PCFR+ converges faster in non-poker games while DDCFR outperforms PCFR+ in poker games, given that our DDCFR is built upon DCFR. Nonetheless, DDCFR can improve the performance of DCFR to narrow the gap with PCFR+ on non-poker games, while amplifying the advantage over PCFR+ on poker games.

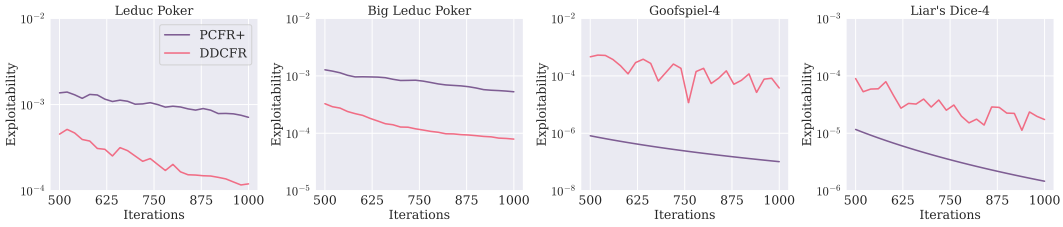

Figure 8: Performance comparison between DDCFR and PCFR+ on four testing games.

It is worth emphasizing that our main contribution is not to propose a specific algorithm, but to propose a general dynamic and trainable discounting regret minimization framework. This framework can be universally applied in any CFR variants that employ fixed and manually specified discounting rules. The DDCFR algorithm presented in the paper is just an example application of our framework to the DCFR algorithm.

PCFR+ also uses a fixed quadratic discounting scheme, formalized as $(\frac{t-1}{t})^{\gamma}$, where gamma is set to 2 in PCFR+. We apply our framework to learn a dynamic discounting averaging strategy, resulting in the Dynamic Predictive CFR+ (DPCFR+) algorithm. Figure 9 illustrates that this integration can further unlock PCFR+'s potential for solving non-poker games without sacrificing its performance on poker games.

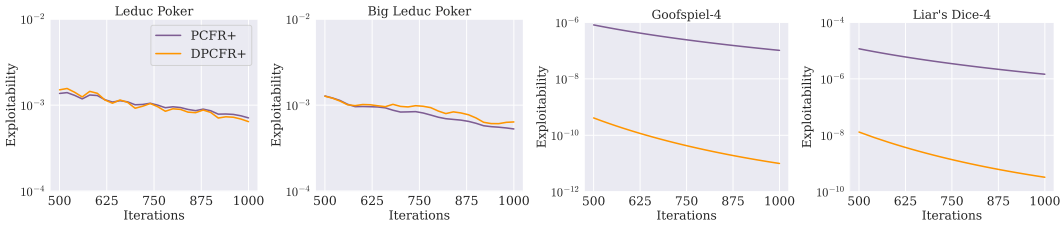

Figure 9: Performance comparison between DPCFR+ and PCFR+ on four testing games.

Besides PCFR+, we consider another algorithm PDCFR in the PCFR+ paper, which can be seen as a combination of PCFR+ and DCFR. PDCFR also uses a fixed discounting scheme and performs well on both non-poker and poker games, benefiting from the characteristics inherited from PCFR+ and DCFR. We aim to enhance PDCFR using our framework, resulting in the Predictive Dynamic Discounted CFR (PDDCFR) algorithm. As illustrated in Figure 10, our framework significantly enhances PDCFR's performance in both non-poker and poker games.

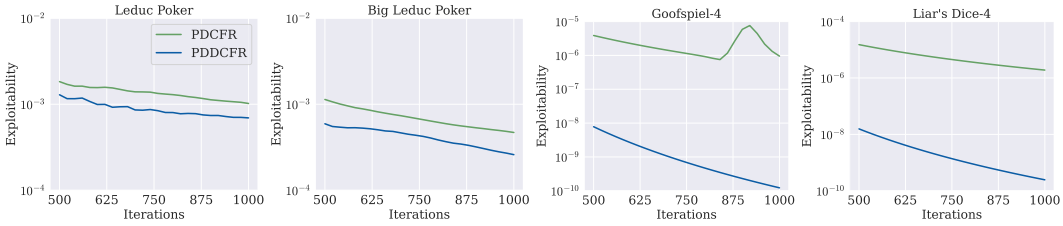

Figure 10: Performance comparison between PDDCFR and PDCFR on four testing games.

All of the above results demonstrate the general applicability of our dynamic discounting framework. Our framework can be regarded as a performance booster that can be applied in a plug-and-play manner to various CFR variants that employ fixed discounting rules.

I About Logarithmic Operation

The state in DDCFR consists of the normalized iteration and the normalized exploitability. When calculating the normalized exploitability, we employ logarithmic operation. We use Kuhn Poker as an example to demonstrate how normalized exploitability changes with the number of iterations. We compare three different normalization methods in Table 3.

Table 3: Normalized exploitability variation in Kuhn Poker with three normalization methods.

t	1	10	20	100	500	1000
E_t	4.583e-1	3.302e-2	1.599e-2	1.767e-3	1.699e-4	4.021e-5
$(\log E_t - \log E_{\min})/(\log E_1 - \log E_{\min})$	1.000	0.902	0.875	0.793	0.706	0.652
$(\sqrt{E_t} - \sqrt{E_{\min}})/(\sqrt{E_1} - \sqrt{E_{\min}})$	1.000	0.268	0.187	0.062	0.019	0.009
$(E_t - E_{\min})/(E_1 - E_{\min})$	1.000	0.072	0.034	0.004	0.000	0.000

CFR variants have an upper bound on the regret values at a square root scale, but they converge much faster in practice. To capitalize on this characteristic, we adopt logarithmic normalization, which

results in a more even distribution of the normalized exploitability. This normalization technique facilitates neural network training, allowing the agent to make better decisions at each iteration.

J DESCRIPTION OF THE GAMES

Kuhn Poker is a simplified version of poker proposed by Harold W. Kuhn (Kuhn, 1950). It uses a deck of three cards (J, Q, K), and players begin by betting one chip into the pot and receiving a private card from the shuffled deck. In the game, players can take four different actions: 1) fold, in which they forfeit the current game, and the other player wins the pot, 2) call, in which they match the other player's bet, 3) bet, in which they add more chips to the pot, and 4) check, in which they decline to place a bet when not facing one from the other player. In *Kuhn Poker*, each player has a chance to bet one chip. If neither player folds, both players reveal their cards, and the player with the higher card takes all chips in the pot. The utility for each player is determined by the difference in the number of chips they possess before and after playing.

Leduc Poker is a larger poker game first introduced in (Southey et al., 2005). It uses six cards divided into two suits, each with three ranks (Js, Qs, Ks, Jh, Qh, Kh). Similar to *Kuhn Poker*, each player starts by betting one chip, receiving a private card, and has the same action options. In *Leduc Poker*, there are two betting rounds. During the first round, each player has the opportunity to bet two chips, and in the second round, they can bet four chips. After the first round, one public card is revealed. If a player's private card matches the public card, that player wins the game. Otherwise, the player with the highest private card wins the game.

Big Leduc Poker is a more complex version of poker. It has rules similar to *Leduc Poker* but uses twenty-four cards divided into twelve ranks. In addition to the increased number of cards, it also allows a maximum of six raises per round instead of the two raises allowed in *Leduc Poker*.

HUNL Subgame-x is a heads-up no-limit Texas hold'em(HUNL) sub-game generated and solved in real-time by the state-of-the-art poker agent Libratus (Brown & Sandholm, 2018)[1]. In HUNL, the two players, P1 and P2, start each hand with \$20,000 and are dealt two private cards from a standard 52-card deck. P1 places \$100 in the pot, and P2 places \$50 in the pot. P2 starts the first round of betting, and players take turns choosing to fold, call, check or raise. A round ends when a player calls if both players have acted. After the first round, three community cards are dealt face-up for all players to observe, and P1 starts a similar betting round. In the third and fourth rounds, one additional community card is dealt, and betting starts again with P1. Unless a player has folded, the player with the best five-card poker hand, constructed from their two private cards and the five community cards, wins the pot. In the case of a tie, the pot is split evenly. *HUNL Subgame-3* begins at the start of the final betting round with \$500 in the pot, and *HUNL Subgame-4* begins at the start of the final betting round with \$3,750 in the pot. In the first betting round, we use bet sizes of 0.5x, 1x the size of the pot, and an all-in bet. In other betting rounds, we use 1x the pot and all-in.

Liar's Dice-x (Lisý et al., 2015) is a dice game where each player receives an x-sided dice and a cup used for concealment. At the start of the game, each player rolls the dice under their cup and examines the dice secretly. The first player starts by bidding on the form p-q, announcing that there are at least p dices with the number of q under both cups. The highest dice numbered x can be treated as any number. Players then have two choices during their turn: 1) bidding of the form p-q, with p or q being greater than the previous player's bidding, 2) calling 'Liar', ending the game immediately and revealing all the dices. If the last bid is not satisfied, the player called 'Liar' wins the game. The winner's utility is 1, and the loser's -1.

Goofspiel-x (Ross, 1971) is a bidding card game where players start with x cards numbered $1 \dots x$. A shuffled point card deck, also numbered $1 \dots x$, is used in the game. The game proceeds in x rounds. Players select a card from their hand in each round to make a sealed bid for the top revealed point card. When both players have chosen their cards, they show their cards simultaneously. The player who makes the highest bid wins the point card. If the bids are equal, the point card will be discarded. The game concludes after x rounds. The player who holds the most point cards wins the game and receives a utility of 1, while the losing player has a utility of -1. We use a fixed deck of decreasing points and an imperfect information variant where players are only informed whether they have won or lost the bid, not the card played by the other player.

[1] `https://github.com/CMU-EM/LibratusEndgames`

Battleship-x (Farina et al., 2019) is a classic board game where players secretly place a ship on their separate grids of size $2 \times x$ at the start of the game. Each ship is 1×2 in size and has a value of 2. Players take turns shooting at their opponent's ship, and the ship that has been hit at all its cells is considered sunk. The game ends when one player's ship is sunk or when each player has completed three shots. The utility for each player is calculated as the sum of the values of the opponent's sunk ship minus the sum of the values of their own lost ship.

Small Matrix is a game where player 1 can choose five actions, *i.e.*, rock, paper, scissors, A1, and A2, while player 2 can only choose rock, paper, and scissors. When both players choose rock, paper, and scissors, the game is a modified rock-paper-scissors game where the winner receives two points when either player chooses scissors; otherwise, the winner receives one point. However, when player 1 chooses the last two actions, *i.e.*, A1/A2, player 1 will lose 10,000/20,000.

We measure the sizes of the games in many dimensions and report the results in Table 4. In the table, *#Histories* measures the number of histories in the game tree. *#Infosets* measures the number of information sets in the game tree. *#Terminal histories* measures the number of terminal histories in the game tree. *Depth* measures the depth of the game tree, *i.e.*, the maximum number of actions in one history. *Max size of infosets* measures the maximum number of histories that belong to the same information set.

Table 4: Sizes of the games.

Game	#Histories	#Infosets	#Terminal histories	Depth	Max size of infosets
Small Matrix	21	2	15	3	5
Kuhn Poker	58	12	30	6	2
Goofspiel-3	67	16	36	5	4
Liar's Dice-3	1,147	192	567	10	3
Battleship-2	10,069	3,286	5,568	9	4
Battleship-3	732,607	81,027	552,132	9	7
Goofspiel-4	1,077	162	576	7	14
Liar's Dice-4	8,181	1024	4,080	12	4
Leduc Poker	9,457	936	5,520	12	5
Big Leduc Poker	6,178,561	100,800	3,953,424	20	23
HUNL Subgame-3	398,112,843	69,184	261,126,360	10	1,980
HUNL Subgame-4	244,005,483	43,240	158,388,120	8	1,980

K Related Work

DDCFR is closely related to hyperparameter optimization since we can regard the discounting weights as hyperparameters of the CFR algorithm. Hyperparameter optimization (Yu & Zhu, 2020) is a widely studied topic in machine learning, which aims to find the optimal hyperparameters for a learning algorithm. Perhaps the most popular hyperparameter optimizer is grid/random search (Bergstra & Bengio, 2012). More sophisticated techniques, such as Bayesian optimization (Snoek et al., 2012), have also proven effective on many different tasks. However, these existing methods suffer from a major problem, *i.e.*, they only estimate the best fixed hyperparameter values that are used throughout the model learning. In contrast, DDCFR's discounting scheme can dynamically adjust the discounting weights online, not just find the best fixed value. The recently proposed techniques, such as meta-gradients (Xu et al., 2018), can dynamically tune hyperparameters on the fly by computing the gradients of the objective function with respect to the hyperparameters directly. However, in our setting, computing the gradients of exploitability with respect to discounting policy's parameters directly using backpropagation is challenging since the forward pass contains many CFR iterations which involve complex tree-based recursive computations. To our knowledge, Greedy Weights (Zhang et al., 2022) is the only dynamic discounted regret minimization algorithm. However, it focuses on solving normal-form games, while DDCFR is designed explicitly for extensive-form games.