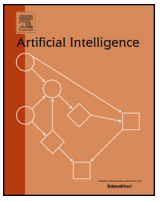
Artificial Intelligence 337 (2024) 104232

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Automatically designing counterfactual regret minimization algorithms for solving imperfect-information games

Kai Li [a,b], Hang Xu [a,b], Haobo Fu [c], Qiang Fu [c], Junliang Xing [d,*]

[a] Institute of Automation, Chinese Academy of Sciences, Beijing, China
[b] School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
[c] Tencent AI lab, Shenzhen, China
[d] Department of Computer Science and Technology, Tsinghua University, Beijing, China

ARTICLE INFO

Keywords:
Imperfect-information game
Nash equilibrium
Regret minimization
Evolution algorithm

ABSTRACT

Strategic decision-making in imperfect-information games is an important problem in artificial intelligence. Counterfactual regret minimization (CFR), a family of iterative algorithms, has been the workhorse for solving these types of games since its inception. In recent years, a series of novel CFR variants have been proposed, significantly improving the convergence rate of vanilla CFR. However, most of these new variants are hand-designed by researchers through trial and error, often based on different motivations, which generally requires a tremendous amount of effort and insight. This work proposes AutoCFR, a systematic framework that meta-learns novel CFR algorithms through evolution, easing the burden of manual algorithm design. We first design a search language that is rich enough to represent various CFR variants. We then exploit a scalable regularized evolution algorithm with a set of acceleration techniques to efficiently search over the combinatorial space of algorithms defined by this language. The learned novel CFR algorithm can generalize to new imperfect-information games not seen during training and performs on par with or better than existing state-of-the-art CFR variants. In addition to superior empirical performance, we also theoretically show that the learned algorithm converges to an approximate Nash equilibrium. Extensive experiments across diverse imperfect-information games highlight the scalability, extensibility, and generalizability of AutoCFR, establishing it as a general-purpose framework for solving imperfect-information games.

1. Introduction

From its inception, artificial intelligence (AI) research has been focusing on building agents that can play games like humans. For more than half a century, games have continued to be AI testbeds for novel ideas, and the resulting achievements have marked important milestones in the history of AI. Notable examples include the checkers-playing bot Chinook winning a world championship against top humans [1], Deep Blue beating Kasparov in chess [2], and AlphaGo defeating Lee Sedol [3] in the complex ancient Chinese game Go. Although substantial progress has been made in solving these perfect-information games, in which all players know the exact state of the game at every decision point, solving imperfect-information games presents a much more difficult challenge. Imperfect-

* Corresponding author.
E-mail addresses: kai.li@ia.ac.cn (K. Li), xuhang2020@ia.ac.cn (H. Xu), haobofu@tencent.com (H. Fu), leonfu@tencent.com (Q. Fu), jlxing@tsinghua.edu.cn (J. Xing).

https://doi.org/10.1016/j.artint.2024.104232
Received 8 February 2023; Received in revised form 19 August 2024; Accepted 27 September 2024

Available online 11 October 2024

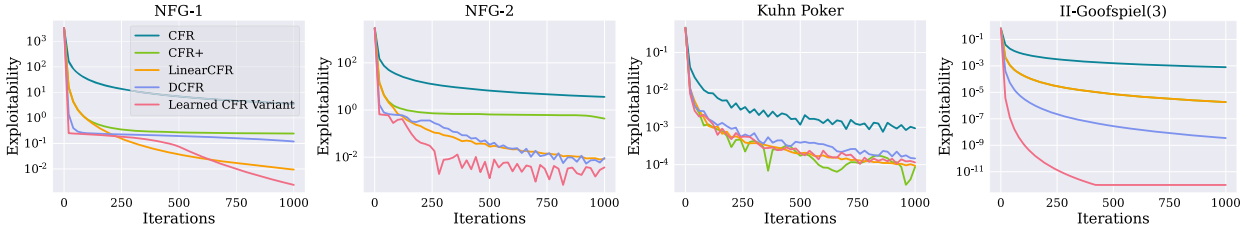


Fig. 1. Convergence speed of four CFR-type algorithms and our learned one in four games. Exploitability measures how well a strategy profile approximates a Nash equilibrium. The closer it is to zero, the closer the policy is to Nash equilibrium. NFG-1 and NFG-2 are two-player zero-sum normal-form games. Kuhn Poker is a simplified form of poker proposed by [12]. II-Goofspiel(3) represents the imperfect information variant of Goofspiel played with 3 cards. Please refer to Section 4.1 for detailed rules of each game. Different CFR-type algorithms perform distinctively in these games. Our framework learns a new CFR variant performing consistently well.

information games model strategic interactions between players with hidden information. Solving this type of games is challenging since it requires reasoning under uncertainty about the opponents' private information. Meanwhile, hidden information is omnipresent in real-world decision-making problems, such as negotiation, business, and security, making the research on imperfect-information games crucial both theoretically and practically.

In this work, we focus on solving two-player zero-sum imperfect-information games. For these games, the common goal is to find a Nash equilibrium [4] in which no player can improve by deviating from this equilibrium. Playing a strategy from a Nash equilibrium in a two-player zero-sum game is guaranteed not to lose in expectation even if the opponent uses the best response strategy. As a popular method of computing Nash equilibrium, counterfactual regret minimization (CFR) [5] has attracted extensive attention due to its sound theoretical guarantee and strong empirical performance. CFR iteratively minimizes the regrets of both players so that the time-averaged strategy profile gradually approximates the Nash equilibrium. Over the past decade, many novel CFR variants have been proposed [6–9] with faster convergence than the vanilla CFR. For example, CFR+ [6,10] was the key to solve the heads-up limit Texas Hold'em poker. Discounted CFR (DCFR) [7] is a family of algorithms that assigns more weight to the regrets and strategies in later iterations, which achieves competitive performance compared with other CFR variants. Linear CFR [7,11] is a simplified version of DCFR and performs well in practice.

Despite the great success of CFR and its improved variants, all of them are hand-designed by researchers based on different motivations, which usually requires a lot of efforts and insights. CFR-type algorithms have many design choices, *e.g.*, new strategy calculation, regret accumulation, average strategy calculation, *etc*. Therefore, it is difficult to systematically consider the space of all CFR variants to design effective ones that can efficiently solve across a wide variety of games. As shown in Fig. 1, it is clear that CFR variants perform differently in different games, and no one performs consistently well in all cases. Moreover, the actual convergence rates of CFR-type algorithms are sometimes different from their theoretical properties. Some variants converge much faster in practice despite having worse theoretical bounds (*e.g.*, CFR+). These gaps between theoretical properties and practical performance further increase the difficulty of manually designing effective CFR variants, as theory sometimes does not offer substantial guidance for creating algorithms with good practical performance. Consequently, designing an algorithm that excels in real applications still necessitates a trial-and-error approach.

To ease the burden and limitation of manual algorithm design, we propose AutoCFR, a framework that learns to design better CFR variants than researchers could design manually. Specifically, AutoCFR formulates the problem of designing new CFR variants as one of meta-learning: an *outer loop* searches over the space of CFR-type algorithms, and an *inner loop* performs equilibrium finding using the learned algorithm on the meta-training games. The objective of the outer loop is to minimize the distance between the strategy obtained by the inner loop and the Nash equilibrium in each meta-training game. Since the No Free Lunch theorem posits that no learning algorithm can excel across all domains, it is more pragmatic to develop CFR algorithms suitable for a class of games. Our AutoCFR framework precisely adheres to this principle by meta-learning CFR algorithms tailored to specific distributions of games. Our ultimate goal is to discover novel CFR variants capable of generalizing to new testing games, which are similar but not identical to the meta-training games.

To define the space of CFR-type algorithms, we formalize an domain-specific language for representing CFR algorithms as computational graphs. This language is expressive enough to represent many existing hand-designed CFR variants as well as other potential alternatives. Since efficiently searching over the space of algorithms defined by this language is generally difficult, we exploit a scalable regularized evolution [13] algorithm with a bag of carefully designed acceleration techniques for the outer loop optimization. Regularized evolution can scale with the number of compute nodes and has been shown effective for searching supervised learning and reinforcement learning algorithms [14,15]. We adapt this method to design algorithms for equilibrium finding in imperfect-information games automatically. We believe that by performing meta-learning in such a rich, combinatorial, open-ended space of algorithms, we will discover highly general, efficient CFR-type equilibrium-finding algorithms. To summarize, this paper makes three contributions:

- We propose AutoCFR, the first framework to meta-learn novel CFR-type imperfect-information game equilibrium-finding algorithms.
- We design an expressive language to describe the space of CFR-type algorithms and exploit an efficient and scalable evolutionary algorithm to make the search feasible.

- We automatically discover new CFR variants with theoretical convergence guarantees, which outperform other state-of-the-art CFR variants across multiple imperfect-information games.

This paper is a systematic extension of our preliminary conference version [16] published at AAAI 2022. In particular, the main changes to the conference version are detailed as follows. We provide a theoretical analysis of the convergence properties of the algorithms learned by our AutoCFR framework, demonstrating that the learned algorithms theoretically converge to an approximate Nash equilibrium. We conduct more experimental analysis and ablation studies to verify the effectiveness of our AutoCFR framework. We also expand the related work, the method, and the experiment sections in more detail to make the paper more self-contained.

2. Preliminary

In this section, we first provide some notations to formulate imperfect-information games. Next, we introduce some important concepts like best response, Nash equilibrium, and exploitability. Finally, we discuss the vanilla CFR algorithm and its typical variants.

2.1. Notations

Imperfect-information games are usually described by a tree-based formalism called extensive-form games. In an extensive-form game, there is a finite set $\mathcal{N} = \{1, 2, \dots, N\}$ of **players**, and there is also a special player c called **chance** with a fixed known stochastic strategy. **History** h consists of all actions taken by players and all possible histories in the game tree form the set $\mathcal{H}$. $\mathcal{Z} \subseteq \mathcal{H}$ are **terminal histories** for which no actions are available. $g \sqsubseteq h$ refers to the fact that g is equal to or a **prefix** of h. $\mathcal{A}(h) = \{a : ha \in \mathcal{H}\}$ denotes the **actions** available in the history, and $\mathcal{P}(h)$ is the unique **player** who takes action in the history. For each player $i \in \mathcal{N}$, there is a **utility function** $u_i(z) : \mathcal{Z} \to \mathbb{R}$. Δ_i is the **range of payoffs** reachable by player i, *i.e.*, $\Delta_i = \max_{z\in\mathcal{Z}} u_i(z) - \min_{z\in\mathcal{Z}} u_i(z)$ and $\Delta = \max_{i\in\mathcal{N}} \Delta_i$.

In imperfect-information games, imperfect 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 $I_i \in \mathcal{I}_i$, player i cannot distinguish between them. Take poker as an example, all histories in an information set differ only in the private card of other players. So we can define $\mathcal{A}(I_i) = \mathcal{A}(h)$ and $\mathcal{P}(I_i) = \mathcal{P}(h)$ for arbitrary $h \in I_i$. We define $|\mathcal{I}| = \max_{i\in\mathcal{N}} |\mathcal{I}_i|$ and $|\mathcal{A}| = \max_{i\in\mathcal{N}} \max_{I_i\in\mathcal{I}_i} |\mathcal{A}(I_i)|$.

A **strategy** $\sigma_i(I_i)$ assigns a distribution over $\mathcal{A}(I_i)$. $\sigma_i(I_i, a)$ is the probability of player i taking action a. Since all histories in an information set belonging to player i are indistinguishable, the strategies in each of them are identical. Therefore, for any $h_1, h_2 \in I_i$, we have $\sigma_i(I_i) = \sigma_i(h_1) = \sigma_i(h_2)$. 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 refers to the set of all possible strategies for player i, and σ_{-i} denotes the strategies of all players other than player i. $u_i(\sigma_i, \sigma_{-i})$ is player i's **expected payoff** if player i plays according to σ_i and the other players play according to σ_{-i}.

$\pi^\sigma(h)$ denotes the **history reach probability** of h if all players play according to σ. It can be decomposed into each player's contribution, *i.e.*, $\pi^\sigma(h) = \pi_i^\sigma(h)\pi_{-i}^\sigma(h)$, where $\pi_i^\sigma(h) = \prod_{h' a \sqsubseteq h, \mathcal{P}(h')=i} \sigma_i(h', a)$ is player i's contribution and $\pi_{-i}^\sigma(h) = \prod_{h' a \sqsubseteq h, \mathcal{P}(h')\neq i} \sigma_{\mathcal{P}(h')}(h', a)$ is all players' contribution except player i. The **information set reach probability** is defined as $\pi^\sigma(I_i) = \sum_{h\in I_i} \pi^\sigma(h)$. The **interval history reach probability** from h' to h is defined as $\pi^\sigma(h', h) = \pi^\sigma(h)/\pi^\sigma(h')$ if $h' \sqsubseteq h$. $\pi_i^\sigma(I_i), \pi_{-i}^\sigma(I_i), \pi_i^\sigma(h, h'), \pi_{-i}^\sigma(h, h')$ are defined similarly.

2.2. Best response and Nash equilibrium

The **best response** to σ_{-i} is any strategy $\mathrm{BR}(\sigma_{-i})$ such that $u_i(\mathrm{BR}(\sigma_{-i}), \sigma_{-i}) = \max_{\sigma_i'\in\Sigma_i} u_i(\sigma_i', \sigma_{-i})$. The **Nash Equilibrium** is a strategy profile $\sigma^* = (\sigma_i^*, \sigma_{-i}^*)$ where everyone plays a best response: $\forall i \in \mathcal{N}, u_i(\sigma_i^*, \sigma_{-i}^*) = \max_{\sigma_i'\in\Sigma_i} u_i(\sigma_i', \sigma_{-i}^*)$. The **exploitability** of a strategy σ_i is defined as $e_i(\sigma_i) = u_i(\sigma_i^*, \sigma_{-i}^*) - u_i(\sigma_i, \mathrm{BR}(\sigma_i))$. In an **$\epsilon$-Nash equilibrium**, no player has exploitability higher than ϵ. The exploitability of a strategy profile σ is $e(\sigma) = \sum_{i\in\mathcal{N}} e_i(\sigma_i)/|\mathcal{N}|$. It can be interpreted as the approximation error to the Nash equilibrium.

2.3. Counterfactual regret minimization

CFR is an iterative regret minimization algorithm for computing Nash equilibrium in extensive-form imperfect-information games [5]. CFR frequently uses **counterfactual value**, which is the expected payoff of an information set given that player i tries to reach it. Formally, for player i at an information set $I \in \mathcal{I}_i$ given a 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 an action a in I is $v_i^\sigma(I, a) = \sum_{h\in I}(\pi_{-i}^\sigma(h) \sum_{z\in\mathcal{Z}}(\pi^\sigma(ha, z)u_i(z))$.

CFR typically starts with a uniform random strategy σ^1. On each iteration T, CFR first recursively traverses the game tree using the 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, *i.e.*, $r_i^T(I, a) = v_i^{\sigma^T}(I, a) - v_i^{\sigma^T}(I)$. Then CFR accumulates the instantaneous regret to obtain the **cumulative regret** $R_i^T(I, a) = \sum_{t=1}^T r_i^t(I, a)$ and uses regret-matching [17,18] to compute the new strategy for the next iteration:

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

Table 1
Comparison of CFR and its typical variants (α, β, γ are hyperparameters).

Algorithms	Cumulative Regret $R_i^T(I,a)$	New Strategy $\sigma_i^{T+1}(I,a)$	Cumulative Strategy $C_i^T(I,a)$
CFR	$R_i^{T-1}(I,a)+r_i^T(I,a)$	$R_i^{T,+}(I,a)/\sum_{a'\in\mathcal{A}(I)}R_i^{T,+}(I,a')$	$C_i^{T-1}(I,a)+\pi_i^{\sigma^T}(I)\sigma_i^T(I,a)$
CFR+	$\max(0,R_i^{T-1}(I,a)+r_i^T(I,a))$	$R_i^{T,+}(I,a)/\sum_{a'\in\mathcal{A}(I)}R_i^{T,+}(I,a')$	$C_i^{T-1}(I,a)+T*\pi_i^{\sigma^T}(I)\sigma_i^T(I,a)$
Linear CFR	$R_i^{T-1}(I,a)+T*r_i^T(I,a)$	$R_i^{T,+}(I,a)/\sum_{a'\in\mathcal{A}(I)}R_i^{T,+}(I,a')$	$C_i^{T-1}(I,a)+T*\pi_i^{\sigma^T}(I)\sigma_i^T(I,a)$
DCFR	$R_i^{T-1}(I,a)*\frac{(T-1)^\alpha}{(T-1)^\alpha+1}+r^T(I,a)$, if $R_i^{T-1}(I,a)>0$ $R_i^{T-1}(I,a)*\frac{(T-1)^\beta}{(T-1)^\beta+1}+r_i^T(I,a)$, otherwise	$R_i^{T,+}(I,a)/\sum_{a'\in\mathcal{A}(I)}R_i^{T,+}(I,a')$	$C_i^{T-1}(I,a)*(\frac{T-1}{T})^\gamma+\pi_i^{\sigma^T}(I)\sigma_i^T(I,a)$
ECFR	$R_i^{T-1}(I,a)+w(I,a)*r_i^T(I,a)$, if $r_i^T(I,a)>0$ $R_i^{T-1}(I,a)+w(I,a)*\beta$, otherwise	$\frac{R_i^{T,+}(I,a)*w(I,a)}{\sum_{a'\in\mathcal{A}(I)}R_i^{T,+}(I,a')*w(I,a)}$	$C_i^{T-1}(I,a)+\pi_i^{\sigma^T}(I)\sigma_i^T(I,a)w(I,a)$

where $R_i^{T,+}(I,a)=\max(R_i^T(I,a),0)$. In two-player zero-sum imperfect-information games, if both players play according to CFR on each iteration then their **average strategies** $\bar{\sigma}^T$ converge to an ϵ-Nash equilibrium in $\mathcal{O}(|\mathcal{I}|^2|\mathcal{A}|\Delta^2/\epsilon^2)$ iterations [5]. $\bar{\sigma}^T$ is calculated as:

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

where $C_i^T(I,a)$ denotes player i's **cumulative strategy** for action a in information set I on iteration T.

2.4. CFR variants

Since the birth of CFR, many novel CFR variants have been proposed based on different motivations and greatly improved the convergence rate of the vanilla CFR. CFR+ [6,10] is like CFR with three small but effective modifications and converges an order of magnitude faster than CFR. First, 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. Second, CFR+ uses a weighted average strategy where iteration T is weighted by T rather than using a uniformly-weighted average strategy as in CFR. Third, CFR+ incorporates alternating updates. In each iteration, one player updates first, and the other player uses those updated results as input during its update. This ensures that the second updater employs information that is more current than in CFR. DCFR [7] is a family of algorithms which discounts prior iterations' cumulative regrets and dramatically accelerates convergence especially in games where some actions are very costly mistakes. Specifically, on each iteration T, DCFR multiplies positive cumulative regret by $T^\alpha/(T^\alpha+1)$, negative cumulative regret by $T^\beta/(T^\beta+1)$, and cumulative strategy by $(T/(T+1))^\gamma$. We choose the hyperparameters α=1.5, β=0, and γ=2, as suggested by the authors. Linear CFR [7] is a special case of DCFR where iteration T's contribution to cumulative regrets and cumulative strategy is proportional to T. ECFR [8] is based on the motivation that instantaneous regret reflects the advantage of one action over other actions, and actions with higher instantaneous regrets should be given higher weights. In practice, ECFR weights action a by $w(I,a)=\exp(r_i(I,a)-1/|\mathcal{A}(I)|\sum_{a\in\mathcal{A}(I)}r_i(I,a))$. The comparison of CFR and its variants is shown in Table 1.

3. Automatically design CFR algorithms

In this section, we first describe the overall framework of our proposed AutoCFR. We then describe the search language which enables the learning of general CFR-type algorithms and the tailored evolution algorithm, which can efficiently search over the algorithm space defined by this language.

3.1. The AutoCFR framework

As mentioned earlier, CFR and its variants have obtained remarkable performance in solving imperfect-information games. This success was possible due to decades of persistent efforts by researchers in the game theory and machine learning communities. However, as shown in Table 1, there are so many design choices in CFR-type algorithms, making it difficult to consider all of them systematically. Manual algorithm design requires many insights and efforts, and we believe that there are better CFR variants that humans have not discovered.

Based on the above considerations, we propose AutoCFR, a meta-learning framework that learns to design novel CFR algorithms. We use $\mathbb{A}$ to denote the space of CFR-type algorithms. Given a training set of games $\mathbb{G}=\{G_i\}_{i=1}^N$, the goal of AutoCFR is to explore this large space of algorithms for an optimal and generalizable $A^*\in\mathbb{A}$, which not only performs well on $\mathbb{G}$ but also generalizes to the unknown testing games $\hat{\mathbb{G}}$. The testing games $\hat{\mathbb{G}}$ and the training games $\mathbb{G}$ are distinct but share similar characteristics. These similarities are crucial for the generalization performance of the meta-learned CFR algorithm. Formally, AutoCFR's training objective function is:

$$A^* = \arg\max_{A \in \mathbb{A}} \left[\sum_{G \in \mathbb{G}} W_G \, \text{Eval}(A, G) \right], \tag{1}$$

where $\text{Eval}(A, G)$ is the inner loop procedure that evaluates algorithm A's performance in the training game G. In AutoCFR, we meta-learn the optimal CFR variant on multiple meta-training games simultaneously, each of which can be seen as a learning task. Therefore, this is essentially a multi-task learning problem. The game weights W_G are used to balance the learning of different tasks, which are specified by cross-validation. To calculate $\text{Eval}(A, G)$, we first use A to iterate M times in G and calculate the exploitability E_A^G of the obtained average strategy. We then use the normalized exploitability to summarize the performance (score) of A, *i.e.*,

$$\text{Eval}(A, G) = \min\left(S_G, \frac{\log E_{\text{CFR}}^G - \log E_A^G}{\log E_{\text{CFR}}^G - \log E_{\text{DCFR}}^G} \right), \tag{2}$$

where E_{CFR}^G is the baseline vanilla CFR's exploitability, E_{DCFR}^G is the state-of-the-art DCFR's exploitability in G. S_G is the predefined maximum score under game G.

Here, we provide a detailed explanation of the motivation behind Equation (2). In Equation (2), we employ two commonly used normalization techniques, namely log normalization and min-max normalization, to obtain a better numerical distribution of scores. Considering the wide range of exploitability which spans multiple orders of magnitude, we first employ log normalization to normalize E_{A}^G as $\log E_{\text{A}}^G$. Log normalization is particularly useful when dealing with data that has a wide range of values. We then utilize min-max normalization to rescale $\log E_{\text{A}}^G$ within a specific range, *i.e.*, $\frac{\text{MAX} - \log E_{\text{A}}^G}{\text{MAX} - \text{MIN}}$, where MAX and MIN are the maximum and minimum values. The better A performs, the lower its log exploitability $\log E_{\text{A}}^G$, and the higher its score. The performance of vanilla CFR is generally the worst, with relatively high log exploitability $\log E_{\text{CFR}}^G$; thus, $\log E_{\text{CFR}}^G$ can be considered an approximation of MAX. DCFR, currently a strong variant of CFR, exhibits lower log exploitability $\log E_{\text{DCFR}}^G$ and can be regarded as an approximation of MIN. So, after min-max normalization, the score of A is calculated as $\frac{\log E_{\text{CFR}}^G - \log E_A^G}{\log E_{\text{CFR}}^G - \log E_{\text{DCFR}}^G}$. Finally, to avoid algorithm A overfitting to G, we have imposed a limit on the maximum score that A can achieve in G. So, the final score of A on G is $\min\left(S_G, \frac{\log E_{\text{CFR}}^G - \log E_A^G}{\log E_{\text{CFR}}^G - \log E_{\text{DCFR}}^G} \right)$ which recovers Equation (2).

In summary, AutoCFR's outer loop searches over the space of CFR-type algorithms (*i.e.*, $\mathbb{A}$). Its inner loop performs equilibrium finding using the algorithm $A \in \mathbb{A}$ proposed by the outer loop on the meta-training games $\mathbb{G}$. The objective is to find algorithm A^* with a maximal weighted score over the set of training games. We believe that by performing meta-learning in a rich space of algorithms and with diverse training games, we will automatically discover novel, efficient, and generalizable CFR variants. Next, we will formally define the search space of the CFR algorithm in Section 3.2. Then, we will introduce the adopted search algorithm in Section 3.3, and finally, discuss the selection of training and testing games in Section 4.1.

3.2. Search language

Each iteration T of the CFR-type algorithms consists of two steps, *i.e.*, policy evaluation and policy update. In the first step, the algorithm traverses the game tree using the current strategy σ^T to collect the instantaneous regrets $r^T(I, a)$ and some auxiliary information such as the reach probabilities, *etc*. The second step exploits the collected data to obtain a new strategy σ^{T+1} for the next iteration. For example, in vanilla CFR, the second step accumulates regrets and computes a new strategy using regret-matching. As shown in Table 1, the main difference among CFR variants is mostly in the second step, *i.e.*, calculating the cumulative regret, the new strategy, and the cumulative strategy.

To better describe the space of CFR-type algorithms, the search language should be rich enough to represent existing CFR variants while enabling the learning of new algorithms that generalize to a wide range of games. Similar to [19,15], we describe the CFR-type algorithms as general computer programs with a domain-specific language. The programs are comprised of two-component functions, *i.e.*, `PE` (policy evaluation), and `PU` (policy update). More specifically, we express $A \in \mathbb{A}$ as a computational graph, *i.e.*, a directed acyclic graph of nodes. There are three kinds of nodes:

- **Input nodes** represent the input to the program A and include the current strategy σ^T, the cumulative regret R^{T-1}, the cumulative strategy C^{T-1}, the current iteration T, constant numbers, *etc*.
- **Operation nodes** define the mathematical operations which compute outputs given inputs from parent nodes. This includes operators from basic math, linear algebra, probability, and statistics. Inputs and outputs to nodes in the computational graph have two different data types, *i.e.*, vector $\mathbb{V}$ and scalar $\mathbb{R}$. For example, the current strategy $\sigma_i^T(I)$, cumulative regret $R_i^{T-1}(I)$ are vectors, and the current iteration T, constant numbers are scalars. Table 2 shows the full list of operation nodes.
- **Output nodes** are the outputs of program A which includes the new strategy σ^{T+1}, the updated cumulative regret R^T, and the updated cumulative strategy C^T.

Fig. 2 visualizes the computational graphs of CFR, CFR+, and DCFR. Our search language is highly flexible and can represent many state-of-the-art CFR variants, as well as many other potential alternatives, which lays the foundation for discovering better CFR variants. To limit the search space and prioritize more interpretable and computational-efficient algorithms, we limit the total number of operation nodes of the computation graph to 30.

Table 2
The complete list of operations nodes. $\mathbb{V}/\mathbb{R}$ indicates the variable's type is vector/scalar. Operations will broadcast, *i.e.*, adding a scalar to a vector means adding the scalar to each element of the vector.

Operation	Inputs	Input Types	Output	Output Type	Description
Add	$\vec{a}, \vec{b}$	$\mathbb{V}/\mathbb{R}, \mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \vec{a} + \vec{b}$
Minus	$\vec{a}, \vec{b}$	$\mathbb{V}/\mathbb{R}, \mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \vec{a} - \vec{b}$
Mul	$\vec{a}, \vec{b}$	$\mathbb{V}/\mathbb{R}, \mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \vec{a} * \vec{b}$
Max	$\vec{a}, \vec{b}$	$\mathbb{V}/\mathbb{R}, \mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \max(\vec{a}, \vec{b})$
Min	$\vec{a}, \vec{b}$	$\mathbb{V}/\mathbb{R}, \mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \min(\vec{a}, \vec{b})$
Div	$\vec{a}, b$	$\mathbb{V}/\mathbb{R}, \mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \vec{a}/b$
Pow	$\vec{a}, b$	$\mathbb{V}/\mathbb{R}, \mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \vec{a}^b$
LT	$\vec{a}, b$	$\mathbb{V}/\mathbb{R}, \mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$c = \mathbf{I}_{\vec{a}<b}$
GE	$\vec{a}, b$	$\mathbb{V}/\mathbb{R}, \mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = \mathbf{I}_{\vec{a}\geq b}$
Exp	$\vec{a}$	$\mathbb{V}/\mathbb{R}$	$\vec{c}$	$\mathbb{V}/\mathbb{R}$	$\vec{c} = e^{\vec{a}}$
Sum	$\vec{a}$	$\mathbb{V}$	c	$\mathbb{R}$	$c = \text{sum}(\vec{a})$
Mean	$\vec{a}$	$\mathbb{V}$	c	$\mathbb{R}$	$c = \text{mean}(\vec{a})$
Normalize	$\vec{a}$	$\mathbb{V}$	$\vec{c}$	$\mathbb{V}$	$\vec{c} = \begin{cases} \vec{a}/\text{sum}(\vec{a}) & \text{sum}(\vec{a}) > 0 \\ \vec{1}/\text{len}(\vec{a}) & \text{otherwise} \end{cases}$

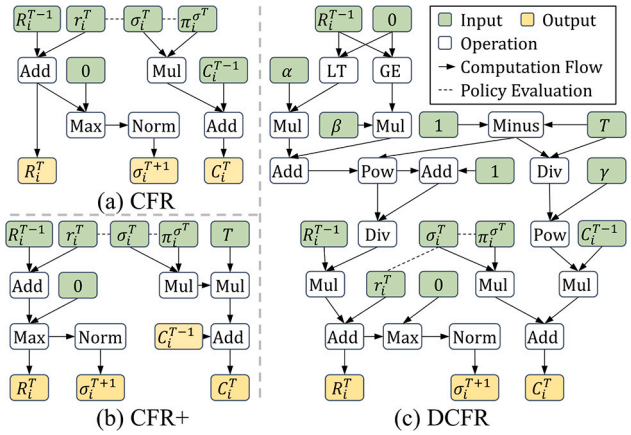


Fig. 2. Our search language can represent existing CFR variants. (a)(b)(c) visualize the computational graphs of CFR, CFR+, and DCFR. Moreover, this language enables the description of potentially better CFR variants.

3.3. Evolutionary search algorithm

The outer loop of AutoCFR is to find CFR variants A^* that work effectively in the training games $\mathbb{G}$. However, evaluating thousands of algorithms from the space $\mathbb{A}$ over a wide range of games in $\mathbb{G}$ is prohibitively expensive. Moreover, changing a single node in the computational graph can drastically change an algorithm's behavior, making the objective function in Equation (1) non-smooth. We use the regularized evolution algorithm [13] as the search method due to its simplicity and efficiency for this type of search problems, which has made remarkable breakthroughs [20,21,15,22,23] in the AutoML community recently. Regularized evolution uses a queue to maintain a population of P programs which can be randomly initialized or initialized by several known programs. The population is improved through cycles. In each cycle, $T < P$ programs are first selected, and the program with the highest score is chosen as the parent program. Then the parent program is mutated to obtain the child program. The child program is added to the queue while the oldest program in the queue is removed. We use a simple type of mutation, *i.e.*, randomly select a node for replacement, randomly select an operation with the same output type as that node, and finally choose the inputs for this operation randomly.

There exists a combinatorially large number of algorithms in $\mathbb{A}$. Furthermore, the inner loop of evaluating a single algorithm A in a game G, *i.e.*, calculating $\text{Eval}(A, G)$, requires multiple CFR-type iterations, which can take up a significant amount of time. Avoiding needless computation and parallelism is essential to make the outer loop more tractable. By taking inspiration from efforts in the AutoML community [24], we extend regularized evolution with a bag of tailored acceleration techniques to make the outer loop optimization more efficient. The complete training procedure is outlined in Algorithm 1.

Program validity check. We perform basic checks to rule out and skip evaluating invalid mutated programs. Specifically, we randomly generate 100 valid samples and input them into the mutated program A. If A fails to satisfy the following rules, we discard it and mutate the parent program again. For example, illegal values (*e.g.*, nan, inf) and exceptions should not be generated when executing A; the action probabilities of the current and average strategies produced by A should be greater than zero and sum to one, *etc*.

Functional equivalence check. Since our search language is highly flexible, there are many non-obvious ways of getting functionally equivalent programs. To find duplicates, we generate a hash code for each program. Specifically, we input 20 random samples into the program and concatenate the outputs as its code. If A's code is the same as the code of the previously evaluated program A',

Algorithm 1: AutoCFR's training procedure.

Input: training games $\mathbb{G}$, game weights W_G, hurdle game G_h, CFR variants $\{\hat{A}\}$, cycle number N, population size P, tournament size T

Initialize population $|\mathbb{P}| = P$ with an empty queue;
Initialize history set $\mathbb{H} \leftarrow \emptyset$;
for *algorithm* $\hat{A}$ ***in*** $\{\hat{A}\}$ **do**
  $\hat{A}.score \leftarrow \sum_{G \in \mathbb{G}} W_G \, \text{Eval}(\hat{A}, G)$ ▷(Algorithm 2);
  Add $\hat{A}$ to $\mathbb{H}$ and $\mathbb{P}$;
for $n = 0$ ***to*** N **do**
  tournament set $\mathbb{T} \leftarrow \emptyset$;
  while $|\mathbb{T}| < T$ **do**
    randomly pick a candidate A^c from $\mathbb{P}$;
    add A^c to $\mathbb{T}$;
  parent algorithm $A \leftarrow$ highest-scored one in $\mathbb{T}$;
  child algorithm $A' \leftarrow \text{Mutate}(A)$;
  $A'.valid \leftarrow \text{ValidityCheck}(A')$;
  $A'.hurdle\_score \leftarrow \text{Eval}(A', G_h)$;
  $A'.hash \leftarrow \text{HashEncoding}(A')$;
  hurdle threshold $\eta \leftarrow \text{Percentile}(\mathbb{P}, 75^{th})$;
  if $A'.valid$ ***and*** $A'.hurdle\_score >= \eta$ **then**
    if $\exists A^{temp} \in \mathbb{H}, A^{temp}.hash == A'.hash$ **then**
      $A'.score \leftarrow A^{temp}.score$
    else
      $A'.score \leftarrow \sum_{G \in \mathbb{G}} W_G \, \text{Eval}(A', G)$
    Add A' to $\mathbb{H}$ and the circular queue $\mathbb{P}$

Output: A^* with the highest score

Algorithm 2: Inner loop procedure $\text{Eval}(A, G)$.

Input: Candidate algorithm A, training game G, iterations M, exploitability $E^G_{\text{CFR}}, E^G_{\text{DCFR}}$, maximum score S_G

Initialize strategy $\sigma^1(I, a) \leftarrow 1/|\mathcal{A}(I)|$;
Initialize cumulative regret $R^0(I, a) \leftarrow 0$;
Initialize cumulative strategy $C^0(I, a) \leftarrow 0$;
for $T = 1$ ***to*** M **do**
  $\pi^{\sigma^T}, r^T \leftarrow A.\texttt{PE}(G, \sigma^T)$;
  $inputs = \{\sigma^T, R^{T-1}, \pi^{\sigma^T}, C^{T-1}, r^T, T, \dots\}$;
  $\sigma^{T+1}, R^T, C^T \leftarrow A.\texttt{PU}(inputs)$;
$\bar{\sigma}^M \leftarrow \text{Normalize}(C^M)$;
$E^G_A \leftarrow \bar{\sigma}^M$'s exploitability on G;

Output: $\text{Eval}(A, G) \leftarrow \min\left(S_G, \frac{\log E^G_{\text{CFR}} - \log E^G_A}{\log E^G_{\text{CFR}} - \log E^G_{\text{DCFR}}}\right)$

we no longer evaluate A and use A' saved score as A's. Since programs with the same code may have different structures, we still add A to the population to potentially mutate into functionally different programs in the future.

Early hurdles. AutoCFR's ultimate goal is to find programs that perform well on many different imperfect-information games, both simple and complex. If the program performs poorly in small simple games, there is no need to evaluate it in large complex games. We use Kuhn poker as an early hurdle game G_h and maintain the 75^{th} percentile η of the scores of all algorithms in the population on G_h. If $\text{Eval}(A, G_h) < \eta$, we early-stop evaluation A on other games and discard it immediately.

Learning from bootstrapping. AutoCFR can learn from scratch by initializing the population with random algorithms or bootstrap the population with known algorithms. Learning from scratch is less biased toward human-designed algorithms and is more likely to discover completely different algorithms. However, it may take a long time to converge to practical algorithms. Bootstrapping from existing algorithms can make the search start from a good starting point and reduce the time required for convergence. We initialize the population with CFR and its typical variants, including CFR+, Linear CFR, and DCFR.

Parallelism. In our actual implementation, the outer and inner loop are executed in parallel. We use a distributed generator to implement the outer loop, which inputs the parent programs and outputs the mutated programs. Similarly, we implement the inner loop as a distributed evaluator, which inputs programs and training games and outputs the scores. These tasks are distributed among multiple processes on multiple machines, communicating through queues.

4. Results and analysis

4.1. Training and testing games

The choice of training games $\mathbb{G}$ (*e.g.*, game sizes, payoff ranges, *etc.*) dramatically affects the learned algorithm and its performance. The more diverse $\mathbb{G}$ is, the better the generalization performance of the resulting algorithm. Besides, the games in $\mathbb{G}$ should not be too large to solve as AutoCFR will evaluate thousands of candidate algorithms during training.

We use some commonly used extensive-form games in the imperfect-information game research community. ***Poker*** has a long history as a benchmark for developing algorithms that deal with imperfect-information. The poker game involves all players being dealt with some private cards visible only to themselves, with players taking structured turns to make actions. Players usually have the following options: 1) fold, giving up the current game, the other player gets all the pot, 2) call, increasing his/her bet until both players have the same chips, 3) bet, putting more chips to the pot, 4) raise, putting more chips into the pot than is required to call the current bet. and 5) check, declining to wager any chips when not facing a bet. ***Kuhn Poker*** is a simplified form of poker proposed by Harold W. Kuhn [12]. There is a deck of three cards often denoted by J, Q, K. At the beginning of the game, each player gets a private card from the shuffled deck and bets one chip into the pot. Throughout the game, players have four options: folding, calling, betting, or checking. 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 defined as the number of chips after playing minus the number of chips before playing.

Leduc Poker is a larger poker game first introduced in [25]. The game uses six cards that include two suites, each with three ranks (Js, Qs, Ks, Jh, Qh, Kh). Like *Kuhn Poker*, each player initially bets one chip, receives a single private card, and has the same action options. In *Leduc Poker*, there are two betting rounds. Each player has a chance to bet two chips in the first round and a chance to bet four chips in the second round. After the first round, one public card is revealed. If a player's private card is paired with the public card, that player wins the game; otherwise, the player with the highest private card wins the game.

Liar's Dice(x) is a dice game where each player gets an x-sided dice and a cup used for concealment. At the beginning of the game, each player rolls the dice under their cup and looks at their hand, keeping it concealed from the other player. The first player begins bidding of the form p-q, announcing that there are at least p dices with the number of q under all of the cups. The highest dice number x can be treated as any number. Then players take turns to take action: 1) bidding of the form p-q, p or q must be 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 calling 'Liar' wins the game. The winner's utility is 1 and the loser -1.

II-Goofspiel(x) is a bidding card game. At the beginning of the game, each player receives x cards numbered $1 \dots x$, and there is a shuffled point card deck containing cards numbered $1 \dots x$. The game proceeds in x rounds. In each round, players select a card from their hand 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. After x rounds, the player with the most point cards wins the game. The winner's utility is 1 and the loser -1. We use a fixed deck of decreasing points and an imperfect information variant where players are only told whether they have won or lost the bid, but not what the other player played.

HUNL Subgames introduced in [7] are heads-up no-limit Texas hold'em (HUNL) subgames generated by and solved in real-time by the state-of-the-art poker agent Libratus [26]. In HUNL, the two players (P1 and P2) start each hand with 20,000, and both players 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. The players alternate in 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 now starts a similar round of betting. 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. The authors of [7] have released a total of four subgames which begin on different betting rounds, named Subgame 1, Subgame 2, Subgame 3, and Subgame 4, respectively. We have chosen two games, Subgame 3 and Subgame 4, for testing. Specifically, *HUNL Subgame 3* begins at the start of the final betting round with \$500 in the pot. *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.

In addition, we manually design some two-player zero-sum normal-form games (***NFG-{1-4}***) with different characteristics and payoff ranges for training. Specifically, ***NFG-1*** is a simple two-action game where players decide between two actions. Player 1's payoff is 2 if takes the first action, and is 20,000 or 1 if takes the second action, depending on the action of player 2. ***NFG-2*** 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, 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. But when player 1 chooses the last two actions, *i.e.*, A1/A2, player 1 will lose 10,000/20,000. *NFG-2* is an abstraction of some situations in real-world games that include highly sub-optimal actions, *e.g.*, all-in irrationally leads to huge losses in poker. ***NFG-3*** is a game with small utility values where player 1 has three actions, and player 2 has two actions. If player 2 chooses the first action, player 1 will receive 0.001, 0.002, -0.1 for the three actions, respectively. If player 2 chooses the second action, player 1 will receive -0.001, -0.003, -0.002 for the three actions, respectively. ***NFG-4*** is a game with many actions where player 1 decides between 21 actions and player 2 only has one choice. The utilities of 21 actions range from -1,000 to 1,000, with an interval of 100. Although these norm-form games seem trivial, some of them are very challenging to solve efficiently, *e.g.*, the vanilla CFR requires 15,000 iterations to solve *NFG-1*. The payoff matrices of the four normal-form games are shown in Table 3.

In particular, the training games $\mathbb{G}$ include four normal-form games (*NFG-{1-4}*) and four small extensive-form games, *i.e.*, *Kuhn Poker*, *II-Goofspiel(3)*, *Liar's Dice(3)*, and *Liar's Dice(4)*. These training games are computationally inexpensive to solve but cover a diverse set of problems. The testing games include four relatively large extensive-form games, *i.e.*, *II-Goofspiel(4)*, *Leduc Poker*, *HUNL Subgame 3*, and *HUNL Subgame 4*. These testing games are diverse in size and nontrivial to solve, which are very suitable for testing the generalization performance of the learned algorithm.

Table 3
Payoff matrices of four normal-form games (***NFG-{1-4}***). The values in the matrix represent Player 1's payoffs, while Player 2's payoffs are the negative of Player 1's payoffs.

(a) NFG-1

	B1	B2
A1	2	2
A2	20000	1

(b) NFG-3

	B1	B2
A1	0.001	-0.001
A2	0.002	-0.003
A3	-0.1	-0.002

(c) NFG-2

	Rock	Paper	Scissors
Rock	0	-1	2
Paper	1	0	-2
Scissors	-2	2	0
A1	-10000	-10000	-10000
A2	-20000	-20000	-20000

(d) NFG-4

	B1
A1	-1000
A2	-900
A3	-800
...	...
A19	800
A20	900
A21	1000

Table 4
Sizes of the games.

Game	#Histories	#Infosets	#Terminal histories	Depth	Max size of infosets
NFG-1	7	2	3	3	3
NFG-2	21	2	15	3	5
NFG-3	10	2	6	3	3
NFG-4	43	2	21	3	21
Kuhn Poker	58	12	30	6	2
II-Goofspiel (3)	67	16	36	5	4
Liar's Dice (3)	1,147	192	567	10	3
Liar's Dice (4)	8,181	1,024	4,080	12	4
II-Goofspiel (4)	1,077	162	576	7	14
Leduc Poker	9,457	936	5,520	12	5
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

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.

4.2. Training details

We search over a program space containing a maximum of 30 operation nodes. The population size P is 300, and the tournament size T is 25, the same as those used in [15]. The parent program mutates with 0.95 probability and remains the same otherwise. We do not employ the crossover operation based on the observation that it could generate numerous invalid algorithms, thereby reducing search efficiency. We train AutoCFR on a distributed server with 250 CPU cores and run for about 8 hours, at which point around 10,000 algorithms have been evaluated. For the inner loop evaluation procedure $\text{Eval}(A, G)$, we set iteration M to 1,000 in all games, except for in *Liar's Dice(4)*, where M is 100 since it is a relatively large game.

4.3. Learned CFR variant: DCFR+

We focus on one particularly interesting new CFR variant, *i.e.*, DCFR+, that was learned by our AutoCFR framework, and that has good generalization performance on different imperfect-information games:

$$\begin{aligned} R_i^T(I,a) &= \max\left(0, R_i^{T-1}(I,a) * \frac{(T-1)^{1.5}}{(T-1)^{1.5}+1.5} + r_i^T(I,a)\right), \\ C_i^T(I,a) &= C_i^{T-1}(I,a) * \frac{T-1}{T} + \pi_i^{\sigma^T} * T^3 * \sigma_i^T(I,a), \\ \sigma_i^{T+1}(I,a) &= \frac{R_i^{T,+}(I,a)}{\sum_{a' \in \mathcal{A}(I)} R_i^{T,+}(I,a')}. \end{aligned} \tag{3}$$

DCFR+'s improvement over existing CFR variants shown in Figs. 3 and 4 is due to two core enhancements: the maximum function and a new discounting method. Here, we provide some intuitive explanations of why they improve the performance: 1) The most prominent feature of DCFR+ is the use of $\max(0, \cdot)$ to rectify the cumulative regrets, which is similar to regret-matching$^+$ in CFR+. When the best action suddenly changes, CFR may take a long time to overcome the accumulated negative regret. In contrast, DCFR+ will play the best action immediately since its accumulated negative regret is forgotten thanks to the $\max(0, \cdot)$ operator. 2) Similar to DCFR in Table 1, DCFR+ also discounts the previous iterations and gives higher weights to the later iterations when accumulating strategies and regrets, albeit in a very different way. This discounting mechanism is beneficial when encountering highly suboptimal

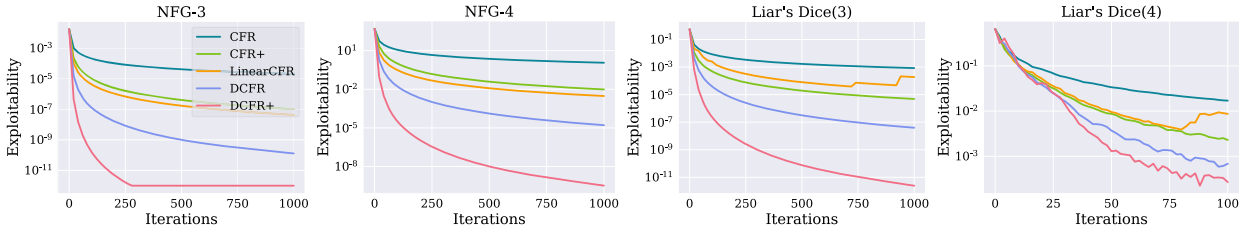


Fig. 3. Comparison of DCFR+ against four CFR variants on eight training games. Another four training games are in Fig. 1.

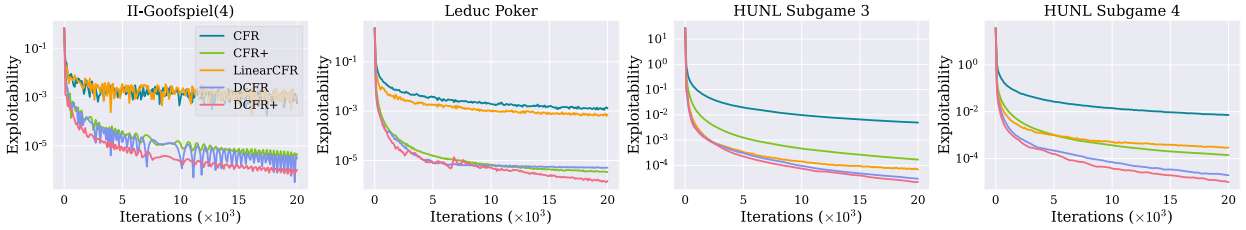


Fig. 4. Comparison of DCFR+ against four CFR variants on four testing games.

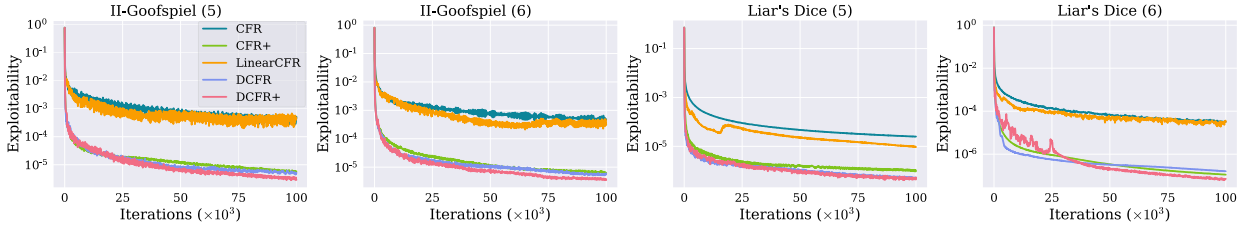


Fig. 5. Comparison of DCFR+ against four CFR variants for a large number of iterations (100,000).

actions, *i.e.*, actions that cause huge mistakes. It is worth noting that DCFR's authors have tried to combine CFR+ with DCFR, but they found that the combined algorithm resulted in poor performance. Our AutoCFR framework can automatically discover DCFR+ through evolutionary search without manual algorithm design, which finds a new way to combine the key insights of CFR+ and DCFR effectively.

Consider the simple two-action game *NFG-1*. The Nash equilibrium of this game is to choose the first action with 100% probability. The CFR variants use the uniform random strategy in the first iteration and result in cumulative regrets of $R1 = -4,999, R2 = 4,999$. CFR will take a long time to obtain the optimal strategy, where $R1 > 0$ and $R2 < 0$. In contrast, DCFR+ directly sets $R1$ to zero in the first iteration and discounts $R2$ in the later iterations. As a result, it will take CFR 15,000 iterations, CFR+ 10,001 iterations, DCFR 1,217 iterations, and DCFR+ only 540 iterations to approach the Nash equilibrium. It demonstrates that DCFR+ can quickly eliminate the negative effects of suboptimal actions.

We conduct additional experiments on larger games to further assess the scalability of DCFR+. Specifically, we test DCFR+ on II-Goofspiel (5), II-Goofspiel (6), Liar's Dice (5), and Liar's Dice (6). Additionally, we increase the number of iterations to 100,000 to observe long-term behavior. The results are presented in Fig. 5. The exploitability of DCFR+ decreases faster than or at least as fast as other CFR variants, indicating that DCFR+ converges to the Nash equilibrium more efficiently. Our findings consistently demonstrate that DCFR+ outperforms other CFR variants in large games, affirming the scalability of DCFR+ in such contexts.

4.4. Convergence analysis of DCFR+

Besides the superior empirical performance of DCFR+, here we demonstrate that it converges to approximate Nash equilibrium after enough iterations in two-player zero-sum imperfect-information games theoretically.

Theorem 1. *Assume that T iterations of DCFR+ are conducted in a two-player zero-sum game. Then the weighted average strategy profile is a $5|\mathcal{I}|\Delta((\sqrt{|A|}+1.5)\sqrt{T}+1.5\sqrt{|A|}(\ln T+1))/T$-Nash equilibrium.*

Prior work [27] has shown that, in two-player zero-sum games, if both players' weighted average regret is ϵ, then their weighted average strategies are a 2ϵ-Nash equilibrium. Therefore, the key idea to prove Theorem 1 is to show that each player's weighted average regret is upper bounded by ϵ, which approaches to zero as T goes to infinity. We provide the detailed proof in appendix A, which incorporates and extends the proof for the vanilla CFR [5], CFR+ [10] and DCFR [7].

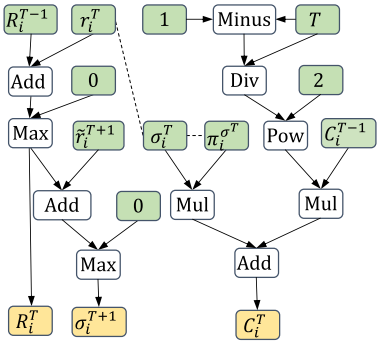


Fig. 6. The computational graph of PCFR+ using our domain-specific language.

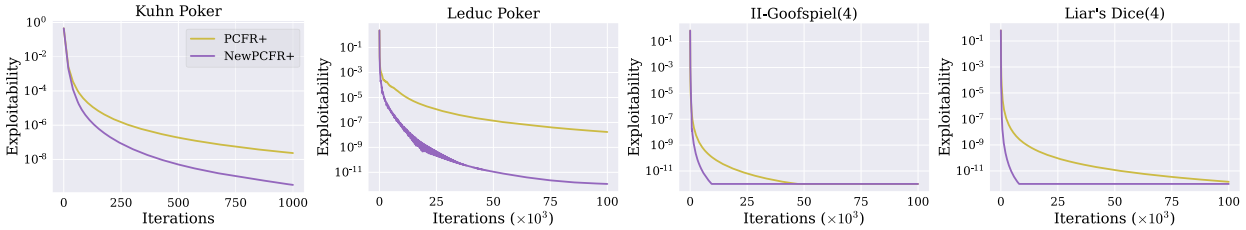


Fig. 7. Comparison of NewPCFR+ against PCFR+.

Theorem 1 shows that DCFR+ achieves similar asymptotic guarantees as CFR, albeit with somewhat larger convergence bound. However the extensive empirical results in Figs. 3, 4 and 5 demonstrate that in practice our DCFR+ dramatically outperforms CFR. These observations are similar to the behaviors of CFR+, whose convergence bound is higher than CFR but typically converges much faster than CFR. These results also validate our motivation that the gaps between theoretical properties and practical performance increase the difficulty of manually designing effective CFR variants only from the theoretical point of view, since the actual convergence rates of CFR-type algorithms are usually different from their theoretical properties. In contrast, our AutoCFR framework can search for novel CFR variants with strong generalization ability and practical performance from the rich space of equilibrium-finding algorithms *directly*.

4.5. The extensibility of AutoCFR in designing novel CFR algorithms

AutoCFR can automatically discover new CFR algorithms through meta-learning in the computational graph space, which are represented by a carefully designed domain-specific language. This language is highly extensible; it can not only represent classic CFR variants (e.g., CFR+) but also can be extended to incorporate the latest developments in CFR (e.g., predictive CFR+ [28]) to find variants better than the current state-of-the-art algorithms.

Specifically, Predictive CFR+ (PCFR+) is a recently proposed state-of-the-art CFR variant which extends Blackwell approachability [29] to regret minimization to form Predictive Regret Matching+ (PRM+). As shown in Fig. 6, our domain-specific language can be easily extended by incorporating PRM+ to represent PCFR+. To demonstrate the extensibility of AutoCFR, we have integrated these latest developments into the search space and automatically discovered a new CFR variant as follows, which we named NewPCFR+.

$$
\begin{aligned}
R_i^T(I,a) &= \begin{cases} R_i^{t-1}(I,a) + 4 * r_i^t(I,a), & \text{if } R_i^{t-1}(I,a) > 0 \\ r_i^T(I,a) & \text{otherwise,} \end{cases} \\
C_i^T(I,a) &= \left(C_i^{T-1}(I,a) * \left(\frac{T-1}{T} \right)^2 + \pi_i^{\sigma^T} * \sigma_i^T(I,a) \right) * \frac{T-1}{T} \\
\sigma_i^{T+1}(I,a) &= \frac{\max\left(R_i^T(I,a) + r_i^T(I,a), 0 \right)^2}{\sum_{a' \in \mathcal{A}(I)} \max\left(R_i^T(I,a) + r_i^T(I,a), 0 \right)^2}
\end{aligned}
\tag{4}
$$

As depicted in Fig. 7, NewPCFR+ outperforms PCFR+ in most games. A noteworthy aspect of NewPCFR+ is its utilization of a square function in computing the new strategy, a feature we believe has not been previously explored in the CFR literature. These results demonstrate that our AutoCFR framework can stand on the shoulders of researchers, leveraging their latest insights and discoveries to automatically discover more effective algorithms. We believe this is also why AutoCFR is important for the advancement of game-solving algorithms.

Table 5
Performance comparison of the best algorithms learned by three AutoCFR variants.

	Kuhn Poker	II-Goofspiel(4)	Leduc Poker	HUNL Subgame 3	HUNL Subgame 4
AutoCFR	$114.89 * 10^{-6}$	$\mathbf{1.04 * 10^{-6}}$	$\mathbf{1.46 * 10^{-6}}$	$\mathbf{22.02 * 10^{-6}}$	$\mathbf{9.80 * 10^{-6}}$
AutoCFR-4	$\mathbf{57.80 * 10^{-6}}$	$2.74 * 10^{-6}$	$2.88 * 10^{-6}$	$43.29 * 10^{-6}$	$43.93 * 10^{-6}$
AutoCFR-S	$144.92 * 10^{-6}$	$332.71 * 10^{-6}$	$11363.57 * 10^{-6}$	$4806.02 * 10^{-6}$	$7590.12 * 10^{-6}$

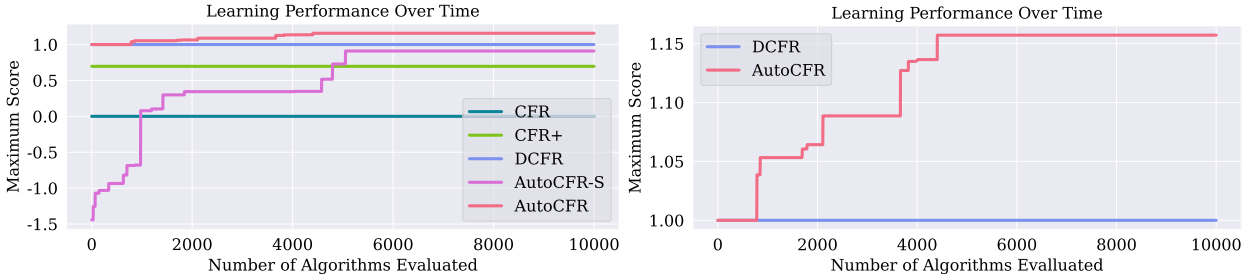


Fig. 8. Performance curves for learning from scratch and bootstrapping as the number of evaluated algorithms increases. The performance represents the maximum score within the population, which aligns with the training objective function A^* in Equation (1).

4.6. Ablation studies of AutoCFR

4.6.1. Varying the number of training games

The choice of training games dramatically affects the performance of the algorithm learned by our AutoCFR framework. Here, we specifically consider how the number of training games affects the learned algorithm, and the results are shown in Table 5. AutoCFR-4, *i.e.*, the learned algorithm using four games (*i.e.*, *Kuhn Poker*, *II-Goofspiel(3)*, *Liar's Dice(3)*, and *Liar's Dice(4)*), performs best in the training game *Kuhn Poker*. However, it underperforms AutoCFR, *i.e.*, the learned algorithm DCFR+ using eight games (*i.e.*, *Kuhn Poker*, *II-Goofspiel(3)*, *Liar's Dice(3)*, *Liar's Dice(4)* and *NFG-{1-4}*), in the testing games. These results clearly demonstrate that training with four games suffers from some overfitting, and the additional four normal-form games increase the learned algorithm's generalization performance.

4.6.2. Learning from scratch versus bootstrapping

As discussed previously, a crucial step to accelerate AutoCFR's training process is learning from bootstrapping. We compare learning from bootstrapping (AutoCFR) with learning from scratch (AutoCFR-S) using the same eight training games (*i.e.*, *Kuhn Poker*, *II-Goofspiel(3)*, *Liar's Dice(3)*, *Liar's Dice(4)* and *NFG-{1-4}*). As shown in Table 5, bootstrapping from existing CFR variants significantly improves the learning performance of AutoCFR over AutoCFR-S without bootstrapping. In Fig. 8, we further demonstrate the effectiveness of our search algorithm. AutoCFR substantially enhances the performance (from 1.00 to 1.15) over the state-of-the-art DCFR algorithm. Meanwhile, AutoCFR-S efficiently finds an algorithm that surpasses CFR by a large margin (from 0.0 to 0.9) even by learning from scratch. Although there is still room for improvement in learning from scratch, we believe this is primarily due to AutoCFR-S exploring only a tiny proportion of the vast search space, constrained by the limited budget available for training.

AutoCFR is biased toward existing CFR algorithms through learning with bootstrapping. In contrast, learning from scratch is less biased toward human-designed algorithms and is more likely to uncover completely different algorithms. Indeed, AutoCFR-S finds many "unconventional" CFR algorithms by learning from scratch. One top-performing example is as follows:

$$
\begin{aligned}
R_i^T(I,a) = &\min\left(1, (\sigma_i^T + 1.5) * \sigma_i^T(I,a) * 0.01\right. \\
&+ \max\left(T * r_i^T(I,a), \frac{\left(\sigma_i^T(I,a) + 1.5\right) * \sigma_i^T(I,a) * 0.01}{T}\right) \\
&\left. * \frac{\max\left(C_i^T(I,a), 1\right)}{\sum_{a' \in \mathcal{A}(I)} \max\left(C_i^T(I,a'), 1\right)}\right) \\
C_i^T(I,a) = &\max\left(\min\left(T, \min\left(C_i^{T-1}(I,a), \frac{R_i^{T-1}(I,a)}{1.5}\right)\right.\right. \\
&\left.\left. + \max\left(C_i^{T-1}(I,a) + r_i^T(I,a), 0\right)\right), -0.01\right) \\
\sigma_i^{T+1}(I,a) = &\frac{R_i^T(I,a) + C_i^T(I,a)}{\sum_{a' \in \mathcal{A}(I)} \left(R_i^T(I,a') + C_i^T(I,a')\right)}
\end{aligned}
\tag{5}
$$

This algorithm is radically different from the existing CFR algorithms. For example, it uses the cumulative strategy to calculate the cumulative regret, the instantaneous regret to calculate the cumulative strategy, and the cumulative strategy to calculate the new

strategy. Although this algorithm is somewhat peculiar, we have found that its performance can surpass CFR+ and approach DCFR in same games. However, it exhibits a more complex form and poor interpretability, which needs further investigation. We plan to release the computation graphs for the top algorithms for both learning from scratch and learning from bootstrapping to enable further theoretical and empirical analysis of the discovered algorithms. We believe that this computational graph dataset is of independent interest to the community and will inspire the design of more effective equilibrium-finding algorithms.

5. Related work

5.1. Counterfactual regret minimization

CFR [5] is the workhorse for solving two-player zero-sum imperfect-information games. Since its birth, many different variants have been proposed, which can be broadly classified into three categories. The first category tries to make CFR more efficient, and the typical representatives are CFR+ [6] and DCFR [7]. Besides, Monte Carlo counterfactual regret minimization (MCCFR) [30,31] is a family of sample-based algorithms which reduces the cost per iteration of vanilla CFR. Some pruning [32,33] and warm starting [34] algorithms are also used to improve the efficiency of CFR. The second category is to extend CFR for imperfect-information subgame solving [35,36] and has made a series of breakthroughs in the poker AI community. For example, DeepStack [37], which exploits a continual re-solving algorithm, defeated ten professional poker players. Libratus [26] defeated four top poker professionals by using a nested safe subgame solving algorithm [36]. Student of Games [38] is a general-purpose algorithm that combines guided search, self-play learning, and game-theoretic reasoning, achieves strong empirical performance in both large perfect and imperfect information games. The third category tries to combine CFR with neural networks to improve CFR's generalization ability, and the typical examples are DeepCFR [11], Single DeepCFR [39], and Double Neural CFR [40]. Our AutoCFR belongs to the first category and is complementary to other CFR variants. For example, DeepCFR can use the algorithm learned by our AutoCFR framework to further improve its performance.

5.2. Automated machine learning

To make machine learning techniques easier to apply and reduce the demand for experienced human experts, automated machine learning (AutoML) [41,42], including meta-learning, which aims to automate the machine learning process has recently become a hot topic in both academia and industry. One notable example is neural architecture search [43,44,13] which automates network architecture engineering and aims to learn a network topology that outperform hand-designed architectures. Besides, there are some other AutoML methods which try to automatically design the learning rules used during backpropagation [45,46], the data augmentation policies in supervised learning [47] or the intrinsic curiosity reward in reinforcement learning [19]. More recently, AutoML-Zero [14] automatically finds machine learning algorithms from scratch using basic mathematical operations. AutoCFR shares similar ideas but is applied to search equilibrium-finding algorithms in imperfect-information games for the first time. To our best knowledge, the most similar work to our AutoCFR is the "Learning not to Regret" framework proposed in [48], which enhances the efficiency of regret minimization algorithms through meta-learning a regret prediction network. However, AutoCFR and the methods presented in [48] exhibit clear differences. AutoCFR can be considered a meta-learning algorithm operating in the computational graph space. In contrast, [48] presents a meta-learning algorithm in the neural network space. AutoCFR and [48] provide two distinct perspectives on using meta-learning for game-solving, with AutoCFR's advantage lying in stronger interpretability due to its use of computational graphs. The algorithm in [48], utilizing gradient descent, may have advantages in terms of training efficiency. Exploring how to synergize the strengths of both approaches to obtain more effective game-solving algorithms is an intriguing avenue for future work.

5.3. Evolutionary algorithm

Evolutionary algorithm [49] is a generic population-based metaheuristic optimization algorithm. It uses mechanisms inspired by biological evolution, such as mutation and selection, where candidate solutions play the role of individuals in the population and the fitness function determines the quality of the solutions. Evolutionary algorithm can effectively address complex optimization problems that traditional optimization algorithms struggle to solve. Recent progress using evolutionary algorithms has achieved impressive results in playing games [50], optimizing neural networks [51,52], and finding novel reinforcement learning algorithms [22,15]. In contrast, we use evolutionary algorithm with lots of acceleration techniques to search for novel CFR variants for solving imperfect-information games. Recently, many differentiable search algorithms [53] have been proposed and have been shown to be faster by several orders of magnitude than evolutionary algorithms. We believe this to be a highly intriguing area for future work to enhance the efficiency of AutoCFR.

6. Conclusion and future work

This paper introduces AutoCFR, a systematic framework for the automatic design of novel CFR algorithms. By introducing a specially designed domain-specific language to represent CFR-type algorithms and utilizing a regularized evolution algorithm for efficient search within the combinatorial space defined by this language, we have successfully discovered new CFR variants that outperform existing state-of-the-art CFR variants. AutoCFR exhibits excellent scalability and generalizability, capable of handling

unseen games of different scales and types. It also demonstrates flexibility, allowing for leveraging the latest insights and discoveries to automatically uncover more effective algorithms.

Our main contributions and insights are as follows: 1) Our work demonstrates that efficient game-solving algorithms can be automatically discovered through meta-learning, without manual trial and error. This opens up a new paradigm for future algorithm design. 2) Our domain-specific language is sufficiently rich to represent existing and potential CFR variants. The expressiveness of this language is pivotal for the successful discovery of new algorithms. 3) The regularized evolution algorithm we adopted, combined with a series of acceleration techniques, can effectively navigate the vast algorithmic space. This highlights the potential of evolutionary algorithms in addressing complex combinatorial optimization problems.

In conclusion, AutoCFR offers a new paradigm for solving imperfect-information games, showcasing the immense potential of automatic algorithm design in enhancing computational efficiency and discovering innovative solutions. We anticipate that this approach will stimulate more research on the automation of algorithm design, ultimately contributing to the development of more efficient and universally applicable artificial intelligence systems. Looking ahead, there are numerous avenues for future research and development. Expanding the search space to encompass Monte Carlo CFR types and incorporating a broader range of imperfect-information games could further enhance AutoCFR. Another promising direction is to restrict the search space of AutoCFR to algorithms with regret minimization guarantees. This approach could not only strengthen the theoretical guarantees of the obtained algorithms but also reduce the size of the search space. Additionally, investigating the integration of neural networks and other machine learning techniques with the evolved CFR algorithms could lead to even more scalable and efficient game-solving algorithms.

CRediT authorship contribution statement

Kai Li: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Hang Xu:** Visualization, Validation, Software, Data curation. **Haobo Fu:** Resources. **Qiang Fu:** Resources. **Junliang Xing:** Supervision, Investigation, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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 (No. RAGR20200104).

Appendix A. Proof of Theorem 1

We first introduce some notations and equations to facilitate the proof, where the information set I is omitted for clarity.

- $v_t = \frac{t^{1.5}}{t^{1.5}+1.5}$, $w_t = \Pi_{i=t}^{T-1} v_i$ for $t < T$, and $w_T = 1$.
- $u_t = t^3 \Pi_{i=t}^{T-1} \frac{i}{i+1} = \frac{t^4}{T}$ for $t < T$, and $u_T = T^3$.
- $Q^T(a) = \max\left(0, Q^{T-1}(a) v_{T-1} + r^T(a)\right)$.
- $P^T(a) = P^{T-1}(a) v_{T-1} + r^T(a) = \sum_{t=1}^{T} w_t r^t(a)$.
- $\Delta Q^t(a) = Q^t(a) - Q^{t-1}(a) \geq Q^{t-1}(a) v_{t-1} + r^t(a) - Q^{t-1}(a) = r^t(a) - (1 - v_{t-1}) Q^{t-1}(a)$.
- $\Delta P^t(a) = P^t(a) - P^{t-1}(a) = w_t r^t(a) = r^t(a) - (1 - v_{t-1}) P^{t-1}(a)$.

Base on these notations, DCFR+ can be reformulated as:

$$\begin{aligned} &Q^T(a) = \max\left(0, Q^{T-1}(a) v_{T-1} + r^T(a)\right). \\ &C^T(a) = C^{T-1}(a) \frac{T-1}{T} + \pi_i^{\sigma^T} T^3 \sigma^T(a) = \sum_{t=1}^{T} u_t \pi_i^{\sigma^t} \sigma^t(a). \\ &\sigma^{T+1}(a) = Q_i^{T,+}(a) / \sum_{a' \in \mathcal{A}(I)} Q_i^{T,+}(a'). \end{aligned} \tag{A.1}$$

Lemma 1. *Given a sequence of strategies $\sigma^1, \ldots, \sigma^T$, each defining a probability distribution over a set of actions $\mathcal{A}$, $Q^t(a) = \max(0, Q^{t-1}(a) * v_{t-1} + r^t(a))$, $P^t(a) = P^{t-1}(a) * v_{t-1} + r^t(a)$ and $Q^0(a) = P^0(a) = 0$ for all actions $a \in \mathcal{A}$. $Q^t(a)$ is then an upper bound of $P^t(a)$ for all $0 \leq t \leq T$.*

Proof. We prove this inductively. When $t = 0$, $P^0(a) = Q^0(a) = 0$, so $P^t(a) \le Q^t(a)$ is true for $t = 0$. Suppose $P^t(a) \le Q^t(a)$ is true for $t = k$. Then $Q^{k+1}(a) = \max(0, Q^k(a) * v_k + r^{k+1}(a)) \ge Q^k(a) * v_k + r^{k+1}(a) \ge P^k(a) * v_k + r^{k+1}(a) = P^{k+1}(a)$. Thus, $P^t(a) \le Q^t(a)$ holds for $t = k + 1$. By the principle of induction, $P^t(a) \le Q^t(a)$ is true for all $0 \le t \le T$. □

Lemma 2. $\sum_{a\in\mathcal{A}} Q^T(a) r^{T+1}(a) = 0$.

Proof. The player selects actions $a \in \mathcal{A}$ on iteration $T + 1$ according to regret-matching, *i.e.*, $\sigma^{T+1}(a) = \frac{Q_+^T(a)}{\sum_{b\in\mathcal{A}} Q_+^T(b)} = \frac{Q^T(a)}{\sum_{b\in\mathcal{A}} Q^T(b)}$. The expected value v^{T+1} on iteration $T + 1$ is $v^{T+1} = \sum_{a\in\mathcal{A}} \sigma^{T+1}(a) v^{T+1}(a) = \frac{\sum_{a\in\mathcal{A}} Q^T(a) v^{T+1}(a)}{\sum_{b\in\mathcal{A}} Q^T(b)}$. Then $\sum_{a\in\mathcal{A}} Q^T(a) v^{T+1} = \frac{\sum_{a\in\mathcal{A}} Q^T(a) * \sum_{a\in\mathcal{A}} Q^T(a) v^{T+1}(a)}{\sum_{b\in\mathcal{A}} Q^T(b)} = \sum_{a\in\mathcal{A}} Q^T(a) v^{T+1}(a)$. So $\sum_{a\in\mathcal{A}} Q^T(a) r^{T+1}(a) = \sum_{a\in\mathcal{A}} Q^T(a)(v^{T+1}(a) - v^{T+1}) = 0$ □

Lemma 3. *Given a set of actions $\mathcal{A}$, and any sequence of T value functions $v^t : \mathcal{A} \to \mathbb{R}$ with a bound Δ such that $|v^t(a) - v^t(b)| \le \Delta$ for all t and $a, b \in \mathcal{A}$, then $Q^T(a) \le \Delta\sqrt{|A|T}$ for all $a \in \mathcal{A}$.*

Proof.

$$
\begin{aligned}
&(\max_a Q^T(a))^2 = \max_a Q^T(a)^2 \le \sum_a Q^T(a)^2 \\
&= \sum_a \left(\max\left(0, Q^{T-1}(a) v_{T-1} + r^T(a)\right)\right)^2 \\
&\le \sum_a \left(Q^{T-1}(a) v_{T-1} + r^T(a)\right)^2 \\
&= \sum_a Q^{T-1}(a)^2 v_{T-1}^2 + \sum_a 2 Q^{T-1}(a) r^T(a) v^{T-1} + \sum_a r^T(a)^2 \\
&= \sum_a Q^{T-1}(a)^2 v_{T-1}^2 + \sum_a r^T(a)^2 \quad \text{(Lemma 2)} \\
&\le v_{T-1}^2 \sum_a Q^{T-1}(a)^2 + |A|\Delta^2 \\
&\le v_{T-1}^2 (v_{T-2}^2 \sum_a Q^{T-2}(a)^2 + |A|\Delta^2) + |A|\Delta^2 \le \dots \\
&\le \sum_{t=1}^{T-1} \{\Pi_{i=t}^{T-1} v_i\}^2 |A|\Delta^2 + |A|\Delta^2 = \sum_{t=1}^{T} w_t^2 |A|\Delta^2 \le T|A|\Delta^2,
\end{aligned}
$$

which gives us $Q^T(a) \le \Delta\sqrt{|A|T}$. □

Lemma 4. *Given a set of actions $\mathcal{A}$, and any sequence of T value functions $v^t : \mathcal{A} \to \mathbb{R}$ with a bound Δ such that $|v^t(a) - v^t(b)| \le \Delta$ for all t and $a, b \in \mathcal{A}$, then $-\Delta T/2 \le P^T(a) \le \Delta\sqrt{|A|T}$ for all $a \in \mathcal{A}$.*

Proof. From Lemma 3, we have $Q^T(a) \le \Delta\sqrt{|A|T}$. By Lemma 1, we can conclude that $P^T(a) \le Q^T(a) \le \Delta\sqrt{|A|T}$. $w_t = \Pi_{i=t}^{T-1} \frac{i^{1.5}}{i^{1.5}+1.5} \ge \Pi_{i=t}^{T-1} \frac{i}{i+1} = \frac{t}{T}$ ($\frac{i^{1.5}}{i^{1.5}+1} \ge \frac{i}{i+1}$ is not true when $i = 1, 2$, but can be ignored when T is large). $\sum_{t=1}^{T} w_t \ge \frac{T(T+1)}{2T} > \frac{T}{2}$, $P^T(a) = \sum_{t=1}^{T} w_t r^t(a) \ge -\frac{T}{2}\Delta$. So $-\Delta T/2 \le P^T(a) \le \Delta\sqrt{|A|T}$ for all $a \in \mathcal{A}$. □

Lemma 5. *Call a sequence $x_1, ..., x_T$ of bounded real values BC-plausible if $B > 0, C \le 0, \sum_{t=1}^{i} x_t \ge C$ for all i, and $\sum_{t=1}^{T} x_t \le B$. For any BC-plausible sequence and any sequence of non-decreasing weights $w_t \ge 0$, $\sum_{t=1}^{T}(w_t x_t) \le w_T(B - C)$.*

Proof. The proof is identical to that of Lemma 1 in [7]. □

Lemma 6. *Let $\mathcal{A}$ be a set of actions, $v^t : \mathcal{A} \to \mathbb{R}$ be a sequence of T value functions over $\mathcal{A}$ with a bound Δ such that $|v^t(a) - v^t(b)| \le \Delta$ for all t and $a, b \in \mathcal{A}$, σ^t be the sequence of strategies generated by Equation (A.1). Construct a weighted sequence σ'^t in which σ'^t is identical to σ^t, but weighted by u_t. Then the weighted regret $R^T(a)$ of this new sequence is bounded by $T^3(\Delta(\sqrt{|A|}+1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$, and the weighted average regret is bounded by $5(\Delta(\sqrt{|A|}+1.5)/\sqrt{T} + (1.5\Delta\sqrt{|A|}(\ln T + 1))/T)$.*

Proof. We know that $w_{t+1} = \Pi_{i=t+1}^{T-1} v_i = w_t/v_t = w_t * \frac{t^{1.5}+1.5}{t^{1.5}}$, and $u_{t+1} = \frac{(t+1)^4}{T} = u_t * (\frac{t+1}{t})^4$. Then $\frac{u^{t+1}}{w_{t+1}} = \frac{u_t}{w_t} * (\frac{t+1}{t})^4 * \frac{t^{1.5}}{t^{1.5}+1.5}$. Since $(\frac{t+1}{t})^4 * \frac{t^{1.5}}{t^{1.5}+1.5} \ge (\frac{t+1}{t})^4 * \frac{t}{t+1.5} = (\frac{t+1}{t})^2 * \frac{(t+1)^2}{t(t+1.5)} \ge 1$, so $\frac{u^{t+1}}{w^{t+1}} > \frac{u^t}{w^t}$, and $\frac{u^t}{w^t}$ is non-decreasing. The weighted regret $R^T(a) = \sum_{t=1}^{T} u_t r^t(a) = \sum_{t=1}^{T} \frac{u_t}{w_t}(w_t r^t(a)) = \sum_{t=1}^{T} \frac{u_t}{w_t} \Delta P^t(a)$.

$$\Delta P^t(a) - \Delta Q^t(a) \le [r^t(a) - (1-v_{t-1})P^{t-1}(a)]$$
$$- [r^t(a) - (1-v_{t-1})Q^{t-1}(a)]$$
$$= (Q^{t-1}(a) - P^{t-1}(a))(1 - v_{t-1}(a))$$
$$= (Q^{t-1}(a) - P^{t-1}(a))\frac{1.5}{(t-1)^{1.5}+1.5}$$
$$\le \left(\Delta\sqrt{|A|(t-1)} + \frac{(t-1)}{2}\Delta\right)\frac{1.5}{(t-1)^{1.5}+1.5} \quad \text{(Lemma 3, 4)}$$
$$\sum_{t=1}^{T}\frac{t-1}{2}\Delta\frac{1.5}{(t-1)^{1.5}+1.5} = \frac{1.5}{2}\Delta\sum_{t=1}^{T}\frac{t-1}{(t-1)^{1.5}+1.5}$$
$$\le \frac{1.5}{2}\Delta\sum_{t=2}^{T}\frac{1}{(t-1)^{0.5}} \le \frac{1.5}{2}\Delta\left(1 + \int_{t=2}^{T}\frac{1}{\sqrt{t-1}}\right)dt$$
$$\le \frac{1.5}{2}\Delta\left(1 + 2\sqrt{t-1}\Big|_{t=2}^{T}\right) \le 1.5\Delta\sqrt{T}$$
$$\sum_{t=1}^{T}\frac{1.5\Delta\sqrt{|A|(t-1)}}{(t-1)^{1.5}+1.5} = 1.5\Delta\sqrt{|A|}\sum_{t=1}^{T}\frac{\sqrt{t-1}}{(t-1)^{1.5}+1.5}$$
$$\le 1.5\Delta\sqrt{|A|}\sum_{t=2}^{T}\frac{1}{(t-1)} \le 1.5\Delta\sqrt{|A|}\left(1 + \int_{t=2}^{T}\frac{1}{t-1}dt\right)$$
$$\le 1.5\Delta\sqrt{|A|} * \left(1 + \ln(t-1)\Big|_{t=2}^{T}\right) \le 1.5\Delta\sqrt{|A|}(\ln T + 1).$$

So $\sum_{t=1}^{T}\Delta P^t(a) - \Delta Q^t(a) \le 1.5\Delta\sqrt{|A|}(\ln T + 1) + 1.5\Delta\sqrt{T}$. Since $\Delta P^t(a) \le \Delta Q^t(a) + \max(0, \Delta P^t(a) - \Delta Q^t(a))$, so

$$\sum_{t=1}^{T}\Delta P^t(a) \le \sum_{t=1}^{T}\Delta Q^t(a) + \max(0, \Delta P^t(a) - \Delta Q^t(a))$$
$$= Q^T(a) + \sum_{t=1}^{T}\max(0, \Delta P^t(a) - \Delta Q^t(a))$$
$$\le \Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1).$$

Since the weight sequence $\frac{u_t}{w_t}$ is non-decreasing, we can apply Lemma 5 using weight $\frac{u_t}{w_t}$ with $B = \Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1)$, $C = 0$, and $\frac{u^T}{w_T} = T^3$. Then we can conclude that $R^T(a) \le T^3(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$. The sum of weights $\sum_{t=1}^{T} u_t = \frac{1}{T}\sum_{t=1}^{T} t^4 = \frac{1}{T} * \frac{T(T+1)(2T+1)(3T^2+3T-1)}{30} = \frac{(T+1)(2T+1)(3T^2+3T-1)}{30} \ge \frac{T^4}{5}$. So the weighted average regret $\frac{R^T(a)}{\sum_{t=1}^{T} u_t} \le \frac{5}{T}(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$. □

Proof of Theorem 1. From Lemma 6, we know that for any information set I and action a, $R^T(I,a) \le \frac{5}{T}(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$. Because this holds for arbitrary a, we have $R^T(I) \le \frac{5}{T}(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$. From the original CFR convergence proof [5], we have $R_i^T \le \sum_{I_i \in \mathcal{I}_i} R^T(I_i) \le \frac{5|\mathcal{I}_i|}{T}(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$. Prior work [27] has shown that, in two-player zero-sum games, if weighted average regret of player 1 and 2 is a and b, then the weighted average strategy is a $(a+b)$-Nash equilibrium. Since $|\mathcal{I}_1| + |\mathcal{I}_2| = |\mathcal{I}|$, so the weighted average strategy profile of DCFR+ form a $\frac{5|\mathcal{I}|}{T}(\Delta(\sqrt{|A|} + 1.5)\sqrt{T} + 1.5\Delta\sqrt{|A|}(\ln T + 1))$-Nash equilibrium. □

Data availability

Data will be made available on request.

References

[1] J. Schaeffer, One jump ahead: challenging human supremacy in checkers, ICGA J. 20 (2) (1997) 93.
[2] M. Campbell, A.J. Hoane Jr., F.-h. Hsu, Deep blue, Artif. Intell. 134 (1–2) (2002) 57–83.
[3] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[4] J.J.F. Nash, Equilibrium points in n-person games, Proc. Natl. Acad. Sci. USA 36 (1) (1950) 48–49.

[5] M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, Regret minimization in games with incomplete information, in: Advances in Neural Information Processing Systems, 2007, pp. 1729–1736.
[6] O. Tammelin, Solving large imperfect information games using CFR+, arXiv:1407.5042, 2014.
[7] N. Brown, T. Sandholm, Solving imperfect-information games via discounted regret minimization, in: AAAI Conference on Artificial Intelligence, 2019, pp. 1829–1836.
[8] H. Li, X. Wang, S. Qi, J. Zhang, Y. Liu, Y. Wu, F. Jia, Solving imperfect-information games via exponential counterfactual regret minimization, arXiv:2008.02679, 2020.
[9] G. Farina, C. Kroer, N. Brown, T. Sandholm, Stable-predictive optimistic counterfactual regret minimization, in: International Conference on Machine Learning, 2019, pp. 1853–1862.
[10] M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved, Science 347 (6218) (2015) 145–149.
[11] N. Brown, A. Lerer, S. Gross, T. Sandholm, Deep counterfactual regret minimization, in: International Conference on Machine Learning, 2019, pp. 793–802.
[12] H.W. Kuhn, 9. a simplified two-person poker, in: Contributions to the Theory of Games (AM-24), vol. I, Princeton University Press, 2016, pp. 97–104.
[13] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: AAAI Conference on Artificial Intelligence, 2019, pp. 4780–4789.
[14] E. Real, C. Liang, D. So, Q. Le, AutoML-zero: evolving machine learning algorithms from scratch, in: International Conference on Machine Learning, 2020, pp. 8007–8019.
[15] J.D. Co-Reyes, Y. Miao, D. Peng, E. Real, Q.V. Le, S. Levine, H. Lee, A. Faust, Evolving reinforcement learning algorithms, in: International Conference on Learning Representations, 2020, pp. 1–15.
[16] H. Xu, K. Li, H. Fu, Q. Fu, J. Xing, AutoCFR: learning to design counterfactual regret minimization algorithms, in: AAAI Conference on Artificial Intelligence, 2022.
[17] S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium, Econometrica 68 (5) (2000) 1127–1150.
[18] N. Cesa-Bianchi, G. Lugosi, Prediction, Learning, and Games, Cambridge University Press, 2006.
[19] F. Alet, M.F. Schneider, T. Lozano-Perez, L.P. Kaelbling, Meta-learning curiosity algorithms, in: International Conference on Learning Representations, 2019, pp. 1–21.
[20] A. Piergiovanni, A. Angelova, A. Toshev, M.S. Ryoo, Evolving space-time neural architectures for videos, in: International Conference on Computer Vision, 2019, pp. 1793–1802.
[21] A. Faust, A. Francis, D. Mehta, Evolving rewards to automate reinforcement learning, arXiv:1905.07628, 2019.
[22] J.K. Franke, G. Koehler, A. Biedenkapp, F. Hutter, Sample-efficient automated deep reinforcement learning, in: International Conference on Learning Representations, 2020, pp. 1–23.
[23] Y. Ci, C. Lin, M. Sun, B. Chen, H. Zhang, W. Ouyang, Evolving search space for neural architecture search, in: International Conference on Computer Vision, 2021, pp. 6659–6669.
[24] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer Nature, 2019.
[25] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, C. Rayner, Bayes' bluff: opponent modelling in poker, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2005, pp. 550–558.
[26] N. Brown, T. Sandholm, Superhuman AI for heads-up no-limit poker: libratus beats top professionals, Science 359 (6374) (2018) 418–424.
[27] N. Brown, T. Sandholm, Regret transfer and parameter optimization, in: AAAI Conference on Artificial Intelligence, 2014.
[28] G. Farina, C. Kroer, T. Sandholm, Faster game solving via predictive Blackwell approachability: connecting regret matching and mirror descent, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2021, pp. 5363–5371.
[29] D. Blackwell, An analog of the minimax theorem for vector payoffs, Pac. J. Math. 6 (1) (1956) 1–8.
[30] M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, Monte Carlo sampling for regret minimization in extensive games, Adv. Neural Inf. Process. Syst. 22 (2009) 1078–1086.
[31] M. Schmid, N. Burch, M. Lanctot, M. Moravčík, R. Kadlec, M. Bowling, Variance reduction in Monte Carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines, in: AAAI Conference on Artificial Intelligence, 2019, pp. 2157–2164.
[32] N. Brown, T. Sandholm, Regret-based pruning in extensive-form games, Adv. Neural Inf. Process. Syst. 28 (2015) 1972–1980.
[33] N. Brown, C. Kroer, T. Sandholm, Dynamic thresholding and pruning for regret minimization, in: AAAI Conference on Artificial Intelligence, 2017, pp. 421–429.
[34] N. Brown, T. Sandholm, Strategy-based warm starting for regret minimization in games, in: AAAI Conference on Artificial Intelligence, vol. 30, 2016.
[35] S. Ganzfried, T. Sandholm, Endgame solving in large imperfect-information games, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2015, pp. 37–45.
[36] N. Brown, T. Sandholm, Safe and nested subgame solving for imperfect-information games, in: Advances in Neural Information Processing Systems, 2017, pp. 689–699.
[37] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, Deepstack: expert-level artificial intelligence in heads-up no-limit poker, Science 356 (6337) (2017) 508–513.
[38] M. Schmid, M. Moravčík, N. Burch, R. Kadlec, J. Davidson, K. Waugh, N. Bard, F. Timbers, M. Lanctot, G.Z. Holland, et al., Student of games: a unified learning algorithm for both perfect and imperfect information games, Sci. Adv. 9 (46) (2023) eadg3256.
[39] E. Steinberger, Single deep counterfactual regret minimization, arXiv preprint, arXiv:1901.07621.
[40] H. Li, K. Hu, S. Zhang, Y. Qi, L. Song, Double neural counterfactual regret minimization, in: International Conference on Learning Representations, 2020.
[41] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, Y. Yu, Taking human out of learning applications: a survey on automated machine learning, arXiv:1810.13306, 2019.
[42] X. He, K. Zhao, X. Chu, Automl: a survey of the state-of-the-art, Knowl.-Based Syst. 212 (2021) 106622.
[43] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations, 2017, pp. 1–16.
[44] A. Brock, T. Lim, J.M. Ritchie, N.J. Weston, SMASH: one-shot model architecture search through hypernetworks, in: International Conference on Learning Representations, 2018, pp. 1–21.
[45] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, in: Advances in Neural Information Processing Systems, 2016, pp. 3981–3989.
[46] S. Ravi, H. Larochelle, Optimization as a model for few-shot learning, in: International Conference on Learning Representations, 2016.
[47] E.D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, Q.V. Le, AutoAugment: learning augmentation strategies from data, in: IEEE Conference on Computer Vision and Pattern Recognition, 2019.
[48] D. Sychrovský, M. Šustr, E. Davoodi, M. Bowling, M. Lanctot, M. Schmid, Learning not to regret, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2024, pp. 15202–15210.
[49] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, Found. Genet. Algorithms 1 (1991) 69–93.
[50] D.G. Wilson, S. Cussat-Blanc, H. Luga, J.F. Miller, Evolving simple programs for playing atari games, in: Genetic and Evolutionary Computation Conference, 2018, pp. 229–236.

[51] X. Cui, W. Zhang, Z. Tüske, M. Picheny, Evolutionary stochastic gradient descent for optimization of deep neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 6051–6061.
[52] S.R. Young, P. Devineni, M. Parsa, J.T. Johnston, B. Kay, R.M. Patton, C.D. Schuman, D.C. Rose, T.E. Potok, Evolving energy efficient convolutional neural networks, in: International Conference on Big Data, 2019, pp. 4479–4485.
[53] A. Heuillet, A. Nasser, H. Arioui, H. Tabia, Efficient automation of neural network design: a survey on differentiable neural architecture search, ACM Comput. Surv. 56 (11) (2024) 1–36.